Using Moldability to Improve Scheduling Performance of Parallel Jobs on Computational Grid

Kuo-Chan Huang[1], Po-Chi Shih[2], and Yeh-Ching Chung[2]

[1] Department of Computer and Information Science
National Taichung University
No. 140, Min-Shen Road, Taichung, Taiwan
kchuang@mail.ntcu.edu.tw
[2] Department of Computer Science
National Tsing Hua University
101, Section 2, Kuang-Fu Road, Hsinchu, Taiwan
shedoh@sslab.cs.nthu.edu.tw, ychung@cs.nthu.edu.tw

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 methods based on the moldable property of parallel jobs to deal with such situations in a heterogeneous computational grid environment. The proposed methods are evaluated through a series of simulations using real workload traces. The results indicate that adaptive processor allocation methods can further improve the system performance of a load sharing computational grid.

1 Introduction

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 allocating 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.

Multi-site parallel execution [7~12] is a possible approach to this issue. Previous research on homogeneous and heterogeneous grids has shown significant performance improvement. However, multi-site parallel execution in heterogeneous grid environments

S. Wu, L.T. Yang, and T.L. Xu (Eds.): GPC 2008, LNCS 5036, pp. 116–127, 2008.

might lead to inefficient resource usage because the portion of computation on faster sites would finish earlier than those on slower sites but the faster sites' resources wouldn't be released until the entire parallel computation comes to the end. This inefficiency could in turn degrade the overall system performance. This paper develops adaptive processor allocation methods based on the moldable property of parallel jobs. The proposed methods are evaluated through a series of simulations using real workload traces. The results indicate that the adaptive processor allocation method outperforms the multi-site parallel execution approach and can further improve the system performance of a heterogeneous computational grid.

2 Related Work

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 [1~4].

England and Weissman in [5] 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 [1, 6] 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 [7] the authors addressed the scheduling of parallel jobs in a heterogeneous multi-site environment. They also evaluated a scheduling strategy that uses multiple simultaneous requests. However, although dealing with a multi-site environment, the parallel jobs in their studies were not allowed for multi-site parallel execution. Each job was allocated to run within a single site.

The support of multi-site parallel execution [8~12] on a computational grid has been examined in previous works, concerning the execution of a job in parallel at different sites. Under the condition of a limited communication overhead, the results from [1, 3, 4, and 6] all showed that multi-site parallel execution can improve the overall average response time. The overhead for multi-site parallel execution mainly results from the slower communication between different sites compared to the intra-site communication. This overhead has been modeled by extending the execution time of a job by a certain percentage [2, 3, and 6].

In [2] the authors further examined the multi-site scheduling behavior by applying constraints for the job fragmentation during the multi-site scheduling. Two parameters were introduced for the scheduling process. The first parameter *lower bound* restricted the jobs that can be fragmented during the multi-site scheduling by a minimal number of necessary requested processors. The second parameter was implemented as a vector describing the maximal number of job fragments for certain intervals of processor numbers.

However, the simulation studies in the previous works are performed based on a homogeneous computational grid model in which nodes on different sites all run at the same speed. In this paper we explore possible multi-site selection policies for a heterogeneous computational grid. In [13] the authors proposed job scheduling algorithms which allow multi-site parallel execution, and are adaptive and scalable in a heterogeneous computational grid. However, the introduced algorithms require predicted execution time for the submitted jobs. In this paper, we deal with the site selection problem for multi-site parallel execution, requiring no knowledge of predicted job execution time.

In the literature [19~25] 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 [18] 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. In this paper adaptive processor allocation is viewed as an alternative to multi-site parallel execution for improving system utilization as well as shortening waiting time for user jobs.

3 Computational Grid Model and Experimental Setting

In the computational grid model, there are several independent computing sites with their own local workload and management system. 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 [14]. 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. 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 [15].

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. The local site which a job is submitted from will be called the *home site* of the job

henceforward in this paper. We assume the ability of jobs to run in multi-site mode. That means a job can run in parallel on a node set distributed over different sites when no single site can provide enough free processors for it due to a portion of resources are occupied by some running jobs. In addition, 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 *slowdown* attribute is generated by the simulation program according to a specified statistical distribution.

- **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. However, in our job scheduling methods the job scheduler does not know the job runtime prior to a job's execution. Therefore, they do not use this information to guide the determination process of job scheduling and allocation.
- **Slowdown.** It is a value indicating how much longer a job will take to finish its execution if it conducts multi-site parallel execution, compared to the runtime required when running in its home site. The runtime for multi-site parallel execution is equal to the runtime within its home site multiplied by the slowdown value.

Our simulation studies were based on publicly downloadable workload traces [16]. We used the SDSC's SP2 workload logs[1] and LANL's CM5 workload logs[2] on [16] as the input workload in the simulations. The detailed workload characteristics are shown in Tables 1 and 2.

In the SDSC's SP2 and LANL's CM5 systems the jobs in the logs are put into different queues and all these queues share the same pool of processors on the system. The SDSC's SP2 system has 128 processors and the LANL's CM5 has 1024 processors. In the following simulations these workload logs 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. Tables 3 and 4 show the corresponding configurations of the computational grid according to the respective workload logs under study. 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.

[1] The JOBLOG data is Copyright 2000 The Regents of the University of California All Rights Reserved.

[2] The workload log from the LANL CM-5 was graciously provided by Curt Canada, who also helped with background information and interpretation.

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*.

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. Characteristics of the workload log on LANL's CM5

	Number of jobs	Maximum execution time (sec.)	Average execution time (sec.)	Maximum number of processors per job	Average number of processors per job
Group 1	79076	66164	158.90	1024	57
Group 2	85358	239892	2027.81	128	55
Group 3	22515	170380	3625.65	1024	210
Group 4	14394	239470	3815.42	1024	238
Total	201343				

Table 3. 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

Table 4. Configuration of the computational grid according to LANL's CM5 workload.

	total	site 1	site 2	site 3	site 4
Number of processors	3200	1024	128	1024	1024

4 Multi-site Parallel Execution

In this paper we use the average turnaround time of all jobs as the comparison criterion in all simulations, which is defined as:

$$Average\ Turnaround\ \ Time = \frac{\sum_{j \in All\ Jobs} (endTime_{\ j} - submitTime_{\ j})}{TotalNumbe\ \ rofJobs} \quad (1)$$

Multi-site parallel execution is traditionally regarded as a mechanism to enable the execution of such jobs requiring large parallelisms that exceed the capacity of any single site. This is a major application area in grid computing called distributed supercomputing [17]. However, multi-site parallel execution could be also beneficial for another application area in grid computing: high throughput computing [17]. In our high throughput computing model in this paper, each job's parallelism is bound by the total capacity of its home site. That means multi-site parallel execution is not inherently necessary for these jobs. However, for high throughput computing a computational grid is used in the space-sharing manner. It is therefore not unusual that upon a job's submission its requested number of processors is not available from any single site due to the occupation of a portion of system resources by some concurrently running jobs. In such a situation, splitting the job up into multi-site parallel execution is promising in shortening the turnaround time of the job through reducing its waiting time. However, in multi-site parallel execution the impact of bandwidth and latency has to be considered as wide area networks are involved. In this paper we summarize the overhead caused by communication and data migration as an increase of the job's runtime [2, 6]. The magnitude of this overhead greatly influences the achievable turnaround time reduction for a job which is allowed to perform multi-site parallel execution.

If a job is performing multi-site parallel execution, the runtime of the job is extended by the overhead which is specified by a parameter p [2]. Therefore the new runtime r^* is:

$$r^* = (1 + p) \times r \quad (2)$$

Where r is the runtime for the job running on a single site. As for the site selection issue in multi-site parallel execution, previous works in [1, 6] suggested the *larger-first* policy for a homogeneous grid environment, which repeatedly picks up a site with the largest number of free processors until all the selected sites together can fulfill the requirement of the job to be allocated. As a heterogeneous grid being considered, the speed difference among participating sites should be taken into account. An intuitive heuristic is called the *faster-first* policy, which each time picks up the site with the fastest computing speed instead of the site having the most amount of free processors. In [26] we developed an *adaptive* site selection policy which dynamically changes between the *larger-first* and the *faster-first* policies based on a calculation of which policy can further accommodate more jobs for immediate single-site execution.

Figure 1 is an example under the SDSC's SP2 workload, which demonstrates that supporting multi-site parallel execution can further improve the performance of a heterogeneous load sharing computational grid with the multi-site overhead p=2. Moreover, our proposed *adaptive* site selection policy outperforms the *larger-first* and the *faster-first* policies significantly. Actually in all the 120 simulations we performed for different speed configurations the *adaptive* policy performs better than the other two policies for each case.

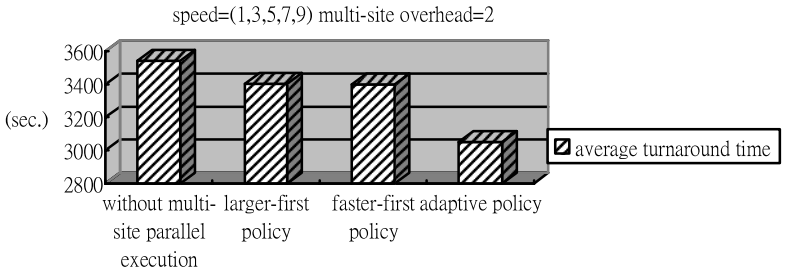

Fig. 1. Performance evaluation of adaptive site selection in multi-site parallel execution

5 Adaptive Processor Allocation Using Moldability

When a job can not fit in any single site in a computational grid, in addition to multi-site parallel execution, adaptive processor allocation is another choice which allocates a smaller number of processors than specified upon submission to a job, allowing it to fit in a single site for immediate execution. This would improve system utilization and shorten the waiting times for user jobs at the cost of enlarged job execution time. The combined effects of enlarged execution time and reduced waiting time for adaptive processor allocation on a homogeneous single-site parallel computer have been evaluated in previous work [18] and shown to be promising in improving average turnaround time for user jobs. In this section an adaptive processor allocation policy for a heterogeneous grid environment is developed. The major difference between the adaptive processors allocation procedures for a single-site parallel computer and for a heterogeneous grid environment is the site selection process regarding the calculation 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 upon job submission multiplied by the computing speed of a single processor on its home site. A configurable threshold parameter, *power*, with its value ranging from zero to one is defined in the adaptive processor allocation procedure. A site will be selected to allocate the job only when the site's free computing power is equal to or larger than the job's required computing power multiplied by the predefined threshold value and it provides the largest available computing power among all sites in the grid. Figure 2 is an example under the SDSC's SP2 workload, which demonstrates adaptive processor allocation can further improve system performance in a heterogeneous grid.

Figures 3 and 4 show that the value of the *power* parameter greatly affects the performance of the adaptive processor allocation method. Therefore, selection of an appropriate value for the *power* parameter becomes a critical issue when applying the adaptive processor allocation method to a heterogeneous grid. We conducted a series of 120-case simulations corresponding to all possible permutations of the site speed vector (1,3,5,7,9) and found that 0.5 is the best value for the *power* parameter under the SDSC's SP2 workload. Another series of 24-case simulations for all possible

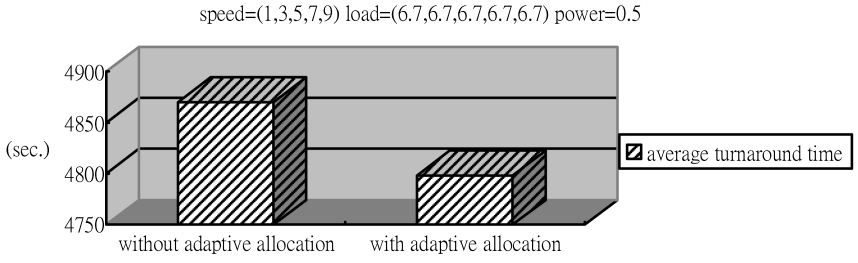

Fig. 2. Performance comparison of loading sharing with/without adaptive processor allocation

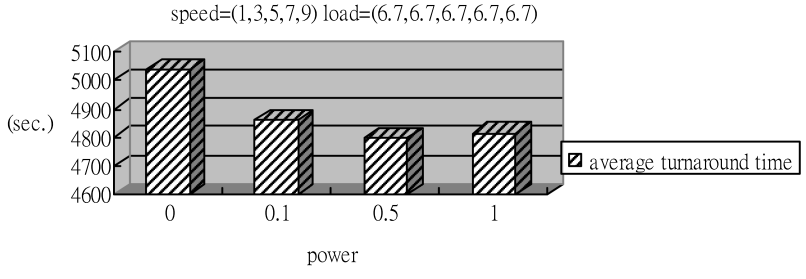

Fig. 3. Adaptive processor allocation with different power values under SDSC SP2 workload

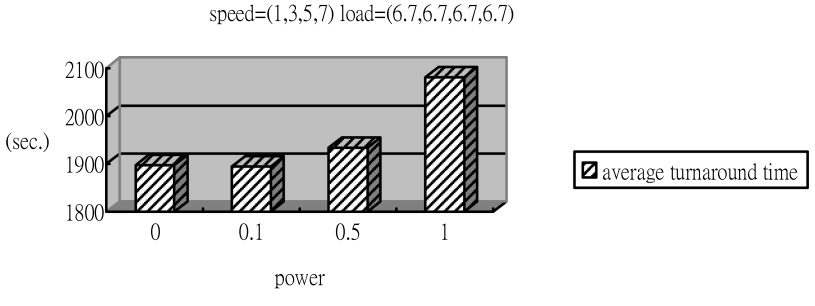

Fig. 4. Adaptive processor allocation with different power values under LANL CM5 workload

permutations of the four-site speed vector (1,3,5,7) indicate that 0.1 is the best value for *power* under the LANL's workload. 0.5 and 0.1 are then used for *power* throughput the following simulation studies in this section for the SDSC's SP2 and LANL's CM5 workloads, respectively.

Figures 5 and 6 compare multi-site parallel execution and adaptive processor allocation under the two different workloads. In our job model, each job is associated with an attribute, *slowdown*, which indicates how long its runtime would be extended to when performing multi-site parallel execution in the grid. In the simulations, the *slowdown* values for these jobs are generated according to specified statistical distributions and upper limits. The upper limits are denoted by *p* in figures 5 and 6.

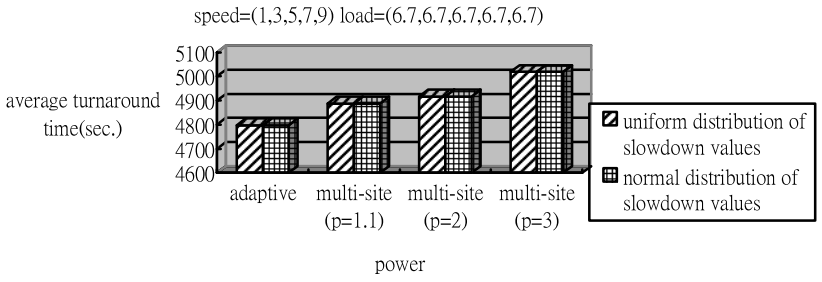

Fig. 5. Comparison under SDSC's SP2 workload for uniformly and normally distributed slowdown values

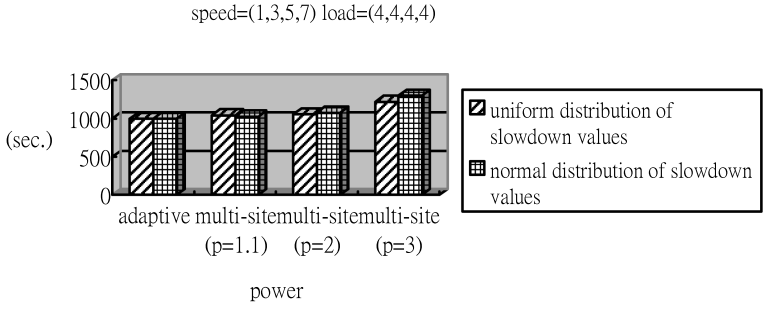

Fig. 6. Comparison under LANL's CM5 workload for uniformly and normally distributed slowdown values

Two types of statistical distributions, uniform and normal distributions, are evaluated in the simulations. Results in figures 5 and 6 show that the performance of multi-site parallel execution is greatly affected by the *slowdown* value which is determined by both the parallel program characteristics and underlying interconnection speed. On the other hand, performance of adaptive processor allocation is irrelative to the *slowdown* values and the results also indicate that adaptive processor allocation outperforms multi-site parallel execution in the simulations.

To further compare these two approaches for all possible permutations of speed vectors, we conducted a series of 120-case simulations under the SDSC's SP2 workload. The results are shown in figure 7. Adaptive processor allocation outperforms multi-site parallel execution in all cases and in average produces more than five times of performance improvement. Although, for a single job, multi-site parallel execution might outperform adaptive processor allocation, *e.g.* reducing the number of processors from 5 to 3 in adaptive processor allocation and the slowdown value being just 1.1 for multi-site parallel execution. The simulation results indicate that adaptive processor allocation is better considering overall performance. This might be because multi-site parallel execution would enlarge the total occupied time period of processor resources, *i.e.* execution time multiplied by the number of processors, while adaptive processor allocation would not. These results shed some light on how to handle the situation where a parallel job can not fit in any single site in a heterogeneous computational grid. Adaptive processor allocation might be a more

promising solution than multi-site parallel execution when the parallel jobs have the moldable property.

Figure 8 is an example demonstrating how much performance improvement a load-sharing computational grid with adaptive processor allocation can bring under the SDSC's SP2 workload. Compared with the non-grid architecture, five independent clusters, the load-sharing grid with adaptive processor allocation leads to more than 4 times of performance improvement.

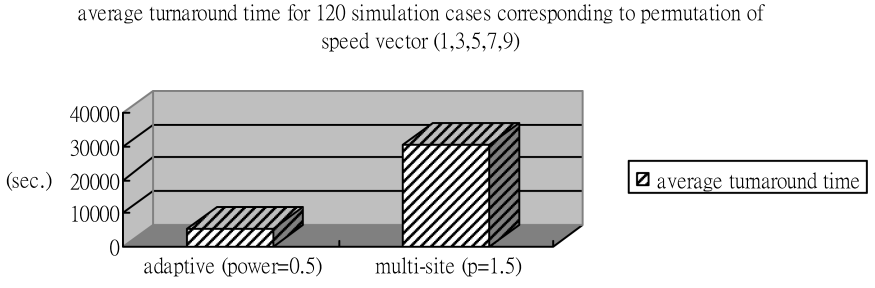

Fig. 7. Thorough comparison under SDSC's SP2 workload

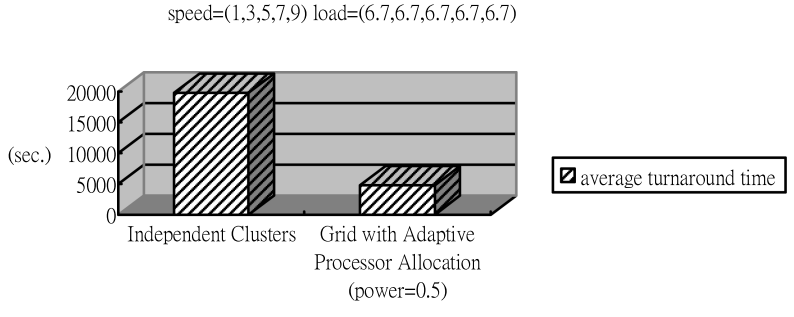

Fig. 8. Performance improvement with load-sharing grid using adaptive processor allocation

6 Conclusion

A grid environment is usually heterogeneous in nature in the real world 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 explores the job scheduling and allocation issue in heterogeneous computational grids when a parallel job, during the scheduling activities, cannot fit in any single site in the grid. Multi-site parallel execution is a possible approach to this issue. However, in heterogeneous grid environments it might lead to inefficient resource usage. This inefficiency could in turn degrade the overall system performance. This paper develops adaptive processor allocation methods based on the moldable property of parallel jobs. The proposed method is evaluated through a series of simulations using real workload traces. The results indicate that the adaptive

processor allocation method outperforms the multi-site parallel execution approach and can further improve the system performance of a heterogeneous computational grid when parallel jobs have the moldable property.

References

1. Hamscher, V., Schwiegelshohn, U., Streit, A., Yahyapour, R.: Evaluation of Job-Scheduling Strategies for Grid Computing. In: Proceedings of the 7th International Conference on High Performance Computing, HiPC 2000, Bangalore, India, pp. 191–202 (2000)
2. Ernemann, C., Hamscher, V., Yahyapour, R., Streit, A.: Enhanced Algorithms for Multi-Site Scheduling. In: Proceedings of 3rd International Workshop Grid 2002, in conjunction with Supercomputing 2002, Baltimore, MD, USA, pp. 219–231 (2002)
3. Ernemann, C., Hamscher, V., Schwiegelshohn, U., Streit, A., Yahyapour, R.: On Advantages of Grid Computing for Parallel Job Scheduling. In: Proceedings of 2nd IEEE International Symposium on Cluster Computing and the Grid (CC-GRID 2002), Berlin, Germany, pp. 39–46 (2002)
4. Ernemann, C., Hamscher, V., Streit, A., Yahyapour, R.: On Effects of Machine Configurations on Parallel Job Scheduling in Computational Grids. In: Proceedings of International Conference on Architecture of Computing Systems, pp. 169–179 (2002)
5. England, D., Weissman, J.B.: Costs and Benefits of Load Sharing in the Computational Grid. In: Feitelson, D.G., Rudolph, L., Schwiegelshohn, U. (eds.) JSSPP 2004. LNCS, vol. 3277, pp. 160–175. Springer, Heidelberg (2005)
6. Huang, K.C., Chang, H.Y.: An Integrated Processor Allocation and Job Scheduling Approach to Workload Management on Computing Grid. In: Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, USA, pp. 703–709 (2006)
7. Sabin, G., Kettimuthu, R., Rajan, A., Sadayappan, P.: Scheduling of Parallel Jobs in a Heterogeneous Multi-Site Environment. In: Proceedings of 9th Workshop on Job Scheduling Strategies for Parallel Processing (2003)
8. Brune, M., Gehring, J., Keller, A., Reinefeld, A.: Managing Clusters of Geographically Distributed High-Performance Computers. Concurrency – Practice and Experience 11, 887–911 (1999)
9. Bucur, A.I.D., Epema, D.H.J.: The Performance of Processor Co-Allocation in Multicluster Systems. In: Proceedings of the Third IEEE International Symposium on Cluster Computing and the Grid (2003)
10. Bucur, A.I.D., Epema, D.H.J.: The Influence of Communication on the Performance of Co-allocation. In: Feitelson, D.G., Rudolph, L. (eds.) JSSPP 2001. LNCS, vol. 2221, pp. 66–86. Springer, Heidelberg (2001)
11. Bucur, A.I.D., Epema, D.H.J.: Local versus Global Schedulers with Processor Co-Allocation in Multicluster Systems. In: Feitelson, D.G., Rudolph, L., Schwiegelshohn, U. (eds.) JSSPP 2002. LNCS, vol. 2537, pp. 184–204. Springer, Heidelberg (2002)
12. Banen, S., Bucur, A.I.D., Epema, D.H.J.: A Measurement-Based Simulation Study of Processor Co-allocation in Multicluster Systems. In: Feitelson, D.G., Rudolph, L., Schwiegelshohn, U. (eds.) JSSPP 2003. LNCS, vol. 2862, pp. 105–128. Springer, Heidelberg (2003)

13. Zhang, W., Cheng, A.M.K., Hu, M.: Multisite Co-allocation Algorithms for Computational Grid. In: Proceedings of the 20th International Parallel and Distributed Processing Symposium (2006)
14. Feitelson, D., Rudolph, L.: Parallel Job Scheduling: Issues and Approaches. In: Proceedings of IPPS 1995 Workshop: Job Scheduling Strategies for Parallel Processing, pp. 1–18 (1995)
15. Ernemann, C., Hamscher, V., Yahyapour, R.: Benefits of Global Grid Computing for Job Scheduling. In: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing, pp. 374–379 (2004)
16. Parallel Workloads Archive (2008), http://www.cs.huji.ac.il/labs/parallel/workload/
17. Foster, I., Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers, Inc., San Francisco (1999)
18. Huang, K.C.: Performance Evaluation of Adaptive Processor Allocation Policies for Moldable Parallel Batch Jobs. In: Proceedings of the Third Workshop on Grid Technologies and Applications, Hsinchu, Taiwan (2006)
19. Srinivasan, S., Krishnamoorthy, S., Sadayappan, P.: A Robust Scheduling Strategy for Moldable Scheduling of Parallel Jobs. In: Proceedings of the Fifth IEEE International Conference on Cluster Computing (2003)
20. Cirne, W., Berman, F.: Using Moldability to Improve the Performance of Supercomputer Jobs. Journal of Parallel and Distributed Computing 62(10), 1571–1601 (2002)
21. Srinivasan, S., Subramani, V., Kettimuthu, R., Holenarsipur, P., Sadayappan, P.: Effective Selection of Partition Sizes for Moldable Scheduling of Parallel Jobs. In: Sahni, S.K., Prasanna, V.K., Shukla, U. (eds.) HiPC 2002. LNCS, vol. 2552, pp. 174–183. Springer, Heidelberg (2002)
22. Cirne, W., Berman, F.: Adaptive Selection of Partition Size for Supercomputer Requests. In: Feitelson, D.G., Rudolph, L. (eds.) IPDPS-WS 2000 and JSSPP 2000. LNCS, vol. 1911, pp. 187–208. Springer, Heidelberg (2000)
23. Sabin, G., Lang, M., Sadayappan, P.: Moldable Parallel Job Scheduling Using Job Efficiency: An Iterative Approach. In: Proceedings of the 12th Workshop on Job Scheduling Strategies for Parallel Processing (2006)
24. Barsanti, L., Sodan, A.C.: Adaptive Job Scheduling via Predictive Job Resource Allocation. In: Proceedings of the 12th Workshop on Job Scheduling Strategies for Parallel Processing (2006)
25. Turek, J., Ludwig, W., Wolf, J.L., Fleischer, L., Tiwari, P., Glasgow, J., Schwiegelshohn, U., Yu, P.S.: Scheduling Parallelizable Tasks to Minimize Average Response Time. In: Proceedings of the Sixth Annual ACM Symposium on Parallel Algorithms and Architectures, pp. 200–209 (1994)
26. Huang, K.C., Shih, P.C., Chung, Y.C.: Towards Feasible and Effective Load Sharing in a Heterogeneous Computational Grid. In: Proceedings of the Second International Conference on Grid and Pervasive Computing, France (2007)