Moldable Job Scheduling for HPC as a Service

Kuo-Chan Huang[1], Tse-Chi Huang[1], Mu-Jung Tsai[1], and Hsi-Ya Chang[2]

[1] Department of Computer Science
National Taichung University of Education
No. 140, Min-Shen Road, Taichung, Taiwan
kchuang@mail.ntcu.edu.tw,
{rogevious,amy29605}@gmail.com
[2] National Center for High-Performance Computing
No. 7, R&D 6th Rd., Hsinchu Science Park, Hsinchu, Taiwan
9203117@nchc.narl.org.tw

Abstract. As cloud computing emerges and gains acceptance, more and more software applications of various domains are transforming into the SaaS model. Recently, the concept of HPC as a Service (HPCaaS) was proposed to bring the traditional high performance computing field into the era of cloud computing. One of its goals aims to allow users to get easier access to HPC facilities and applications. This paper deals with related job submission and scheduling issues to achieve such goal. Traditional HPC users in supercomputing centers are required to specify the amount of processors to use upon job submission. However, we think this requirement might not be necessary for HPCaaS users since most modern parallel jobs are moldable and they usually could not know how to choose an appropriate amount of processors to allow their jobs to finish earlier. Therefore, we propose a moldable job scheduling approach which relieves HPC users' burden of selecting an appropriate number of processors and can achieve even better system performance than existing job scheduling methods. The experimental results indicate that our approach can achieve up to 75% performance improvement than the traditional rigid processor allocation method and 3% improvement than previous moldable job scheduling methods.

Keywords: moldable job, HPC as a Service, processor allocation.

1 Introduction

High performance computing (HPC) has long been a very important field for solving large-scale and complex scientific and engineering problems. However, accessing and running applications on HPC systems remains tedious, limiting wider adoption and user population [1]. As cloud computing emerges, which emphasizes easier and efficient access to IT infrastructure, recently the concept of *HPC as a Service* [1] was proposed to transform HPC facilities and applications into a more convenient and accessible service model.

Traditional HPC users at supercomputing centers are required to specify an amount of processors to use upon job submission. This requirement might be reasonable in

James J. (Jong Hyuk) Park et al. (eds.), *Future Information Technology*,
Lecture Notes in Electrical Engineering 276,
DOI: 10.1007/978-3-642-40861-8_7,

earlier days for the following two reasons. Firstly, some parallel jobs might be rigid jobs [14] which can only be executed with a specific amount of processors. Secondly, developers of parallel programs want to conduct performance benchmarking, e.g. drawing the speedup curve. However, the situation has changed. Most modern parallel applications are moldable [14] and written in a way allowing them to run with different number of processors as required, such as MPI [17] parallel programs. Moreover, most end users just want to get their jobs done faster, but don't care and even don't know how many processors is the best amount to use. Therefore, it seems that it is no longer necessary to require users to specify the amount of processors to use when they submit parallel jobs, especially for the end users of HPC applications as a Service.

Information about parallel program behavior is crucial for job schedulers to automatically choose effective amounts of processors for applications. In this paper, we consider two commonly used parallel speedup models: Amdahl's law [15] and Downey's speedup model [6][7], which have been shown capable of representing many applications' parallel behavior effectively. Based on these two parallel speedup models, we developed an effective moldable job scheduling approach to relieving HPC users' burden of selecting an appropriate number of processors upon job submission. A series of simulation experiments were conducted for performance evaluation. The experimental results show that in addition to relieving users' burden our approach can achieve even better system performance than existing job scheduling methods, up to 75% performance improvement than the traditional rigid processor allocation method and 3% improvement than previous moldable methods.

2 Related Work

Parallel job scheduling and allocation has long been an important research topic [3][4][13]. For rigid jobs [14], backfilling job scheduling approaches have been proposed to improve system performance [2][5]. For moldable jobs [14], previous research [11] has shown potential performance improvement achieved by adaptive processor allocation. The proposed adaptive processor allocation methods in [11] dynamically determine the number of processors to allocate just before job execution according to the amount of current available resources and job queue information.

In [8][9], Srinivasan *et al.* proposed a schedule-time aggressive fair-share strategy for moldable jobs, which adopts a profile-based allocation scheme. This strategy thus needs to have the knowledge of job execution time. On the other hand, our approach does not require the information of job execution time. Sun *et al.* proposed an adaptive scheduling approach for malleable jobs with periodic processor reallocations based on parallelism feedback of the jobs and allocation policy of the system in [10].

In [1], AbdelBaky *et al.* proposed the concept of HPC as a Service, aiming to transform traditional HPC resources into a more convenient and accessible service. They focused on the issues related to elastic provisioning and dynamic scalability, which are concerned in malleable jobs [14]. In this paper, we take advantage of the moldable property [14] in most modern parallel applications to develop an effective

moldable job scheduling approach for HPCaaS, aiming to relieve users' burden of specifying appropriate numbers of processors and improve overall system performance.

3 Processor Allocation for Moldable Job Scheduling

This section deals with the issues on processor allocation for moldable job scheduling. The job scheduler has to make processor allocation decisions on two kinds of events: *job arrival* and *job finish*. In general, there are two possible philosophies: running as many jobs in queue simultaneously as possible or giving the first job as many processors as possible. We call these two philosophies *parallel policy* and *serial policy*, respectively, in this paper. Which policy is better would largely depend on the parallel behavior of applications.

In the following, we explore the potential of the two policies on three common parallel speedup models which cover the behavior of most parallel applications. The first is the model usually introduced in the textbook of parallel processing, where *speedup* is defined by $S_p = T_1/T_p$, with p the number of processors, T_1 the execution time of the sequential run, T_p the execution time of parallel processing with p processors. Based on the definition of speedup, *efficiency* is another performance metric defined as $E_p = S_p/p = T_1/pT_p$. Efficiency is a value, typically between zero and one, estimating how well-utilized the processors are in solving the problem. The second model is Amdahl's law [15], which states that if P is the proportion of a program that can be made parallel, then the maximum speedup that can be achieved by using N processors is $S(N) = 1 / ((1-P) + P/N)$. The third is Downey's speedup model of parallel programs, which has been shown capable of representing the parallelism and speedup characteristics of many real parallel applications [6][7]. Downey's model is a non-linear function of two parameters. The first parameter σ (*sigma*) is an approximation of the coefficient of variance in parallelism within the job. It determines how close to linear the speedup is. A value of zero indicates linear speedup and higher values indicate greater deviation from the linear curve. Another parameter is A, denoting the average parallelism of a job and is a measure of the maximum speedup that the job can achieve.

Based on the speedup models, the resultant average turnaround time of the two allocation policies can be derived. For example, the following two equations represent the average turnaround time achieved by the parallel and serial allocation policies, respectively, for applications of the Amdahl's law model, where t is the job's sequential runtime, x is the parallel proportion between 0 and 1, n is the number of free processors, and d is the number of jobs in queue, assuming n to be a multiple of d.

$$\text{Average turnaround time}_{\text{parallel}} = t \cdot \left((1-x) + \frac{x}{\frac{n}{d}} \right) \cdot d \cdot \frac{1}{d} \qquad \text{Average turnaround time}_{\text{serial}} = t \cdot \left((1-x) + \frac{x}{n} \right) \cdot (1+d) \cdot d \cdot \frac{1}{d}$$

Figures 1 to 4 compare the performance of parallel and serial allocation policies, in terms of average turnaround time, on different application speedup models. The comparison indicates that job scheduler has to adopt different processor allocation policies for applications of different speedup models. For example, the serial allocation policy is superior for applications of the first model. Based on this analysis, we developed a moldable job scheduling approach for HPC as a Service, which can automatically determine the amount of processors to use for HPC users and would not only relieve users' burden of specifying appropriate numbers of processors but also achieve even better system performance than existing job scheduling methods. The proposed approach will be evaluated in the following section.

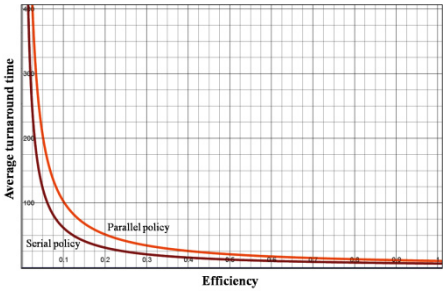

Fig. 1. The first model (Efficiency)

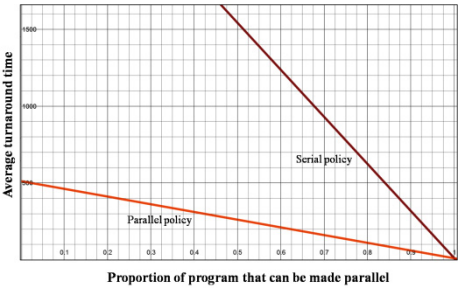

Fig. 2. The second model (Amdahl'law)

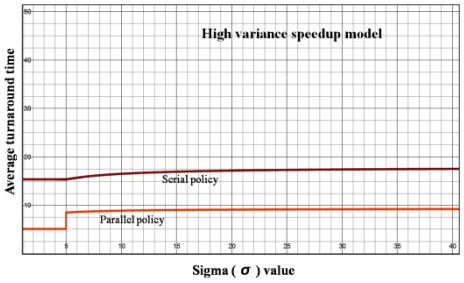

Fig. 3. Downey's high-variance model

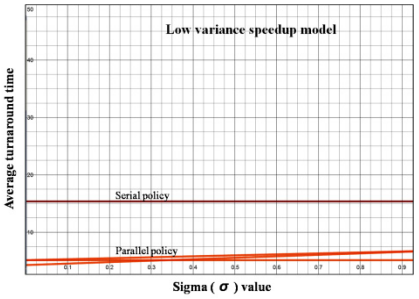

Fig. 4. Downey's low-variance model

4 Experiments and Performance Evaluation

This section evaluates the proposed approach and compares it with four other methods: rigid, adaptive scaling up and down protected [16], restricted scaling up and down protected [16], and random. The *rigid* method is commonly used in most current HPC systems, which can only allocate a fixed amount of processors, specified by the user, to a job. The two scaling up and down allocation methods are previous moldable job scheduling approaches shown to achieve good performance [16]. The random approach is a simple policy for the job scheduler to perform automatic processor amount determination, randomly choosing the amount. The performance

evaluation was conducted through a series of simulation experiments, assuming a 128-processor cluster, based on a public workload log on SDSC's SP2 [12]. The two parameters, σ and A, for Downey's speedup models were generated randomly.

Figures 5 and 6 show the experimental results based on the Downey's low variance model and Amdahl's law, respectively. The results indicate that our approach achieve the best overall performance, up to 75% performance improvement than the traditional rigid method and 3% improvement than previous moldable methods.

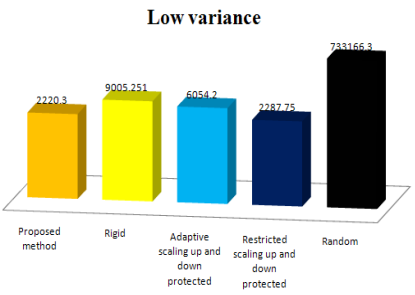

Fig. 5. Downey's low variance speedup model

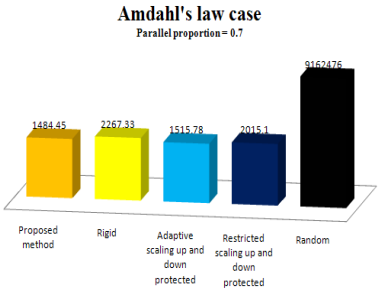

Fig. 6. Amdahl's law model

5 Conclusions

HPC as a Service is a future trend for high-performance computing, aiming to provide a more convenient and accessible HPC resources and applications. To achieve that goal, one potential issue to resolve is relieving users' burden of choosing an appropriate amount of processors to use upon job submission, when the submitted jobs have the moldable property which is common in most modern parallel programs. This paper proposes a moldable job scheduling approach for HPC as a Service, which not only relieves users' burden but also achieves even better system performance than existing methods, up to 75% performance improvement than the traditional rigid method and 3% improvement than previous moldable methods.

References

1. AbdelBaky, M., Parashar, M., Kim, H., JordanKirk, E.J., Sachdeva, V., Sexton, J., Jamjoom, H., Shae, Z.Y., Pencheva, G., Tavakoli, R., Wheeler, M.F.: Enabling High Performance Computing as a Service. IEEE Computer 45, 72–80 (2012)
2. Feitelson, D.G., Weil, A.M.: Utilization and Predictability in Scheduling the IBM SP2 with Backfilling. In: 12th Int'l Parallel Processing Symp., pp. 542–546 (April 1998)
3. Gibbons, R.: A Historical Application Profiler for Use by Parallel Schedulers. In: Feitelson, D.G., Rudolph, L. (eds.) IPPS-WS 1997 and JSSPP 1997. LNCS, vol. 1291, pp. 58–77. Springer, Heidelberg (1997)

4. Lifka, D.: The ANL/IBM SP Scheduling System. In: Feitelson, D.G., Rudolph, L. (eds.) IPPS-WS 1995 and JSSPP 1995. LNCS, vol. 949, pp. 295–303. Springer, Heidelberg (1995)
5. Mu'alem, A.W., Feitelson, D.G.: Utilization, Predictability, Workloads, and User Runtime Estimate in Scheduling the IBM SP2 with Backfilling. IEEE Transactions on Parallel and Distributed Systems 12(6), 529–543 (2001)
6. Downey, A.B.: A Model for Speedup of Parallel Programs. UC Berkeley EECS Technical Report, No. UCB/CSD-97-933 (January 1997)
7. Downey, A.B.: A Parallel Workload Model and Its Implications for Processor Allocation. In: The 6th International Symposium on High Performance Distributed Computing (1997)
8. Srinivasan, S., Krishnamoorthy, S., Sadayappan, P.: A Robust Scheduling Strategy for Moldable Scheduling of Parallel Jobs. In: 5th IEEE International Conference on Cluster Computing, pp. 92–99 (2003)
9. 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)
10. Sun, H., Cao, Y., Hsu, W.J.: Efficient Adaptive Scheduling of Multiprocessors with Stable Parallelism Feedback. IEEE Transactions on Parallel and Distributed System 22(4) (April 2011)
11. Huang, K.C.: Performance Evaluation of Adaptive Processor Allocation Policies for Moldable Parallel Batch Jobs. In: 3th Workshop on Grid Technologies and Applications (2006)
12. Parallel Workloads Archive, http://www.cs.huji.ac.il/labs/parallel/workload/
13. Feitelson, D.G.: A Survey of Scheduling in Multiprogrammed Parallel Systems, Research Report RC 19790 (87657), IBM T. J. Watson Research Center (October 1994)
14. Feitelson, D.G., Rudolph, L., Schweigelshohn, U., Sevcik, K., Wong, P.: Theory and Practice in Parallel Job Scheduling. In: Feitelson, D.G., Rudolph, L. (eds.) IPPS-WS 1997 and JSSPP 1997. LNCS, vol. 1291, pp. 1–34. Springer, Heidelberg (1997)
15. Kleinrock, L., Huang, J.H.: On parallel processing systems: Amdahl's law generalized and some results on optimal design. IEEE Trans. Softw. Eng. 18(5) (1992)
16. Huang, K.C., Huang, T.C., Tung, T.H., Shih, P.Z.: Effective Processor Allocation for Moldable Jobs with Application Speedup Model. In: Proceedings of the International Computer Symposium, ICS 2012, Taiwan (2012)
17. The Message Passing Interface (MPI) standard, http://www.mcs.anl.gov/research/projects/mpi/