Moldable Job Scheduling for HPC as a Service with Application Speedup Model and Execution Time Information

Kuo-Chan Huang[a],
[a]Department of Computer Science
National Taichung University of Education
Taichung, Taiwan
kchuang@mail.ntcu.edu.tw {rogevious, amy29605}@gmail.com

Tse-Chi Huang[a] Mu-Jung Tsaia Hsi-Ya Chang[b]
[b]National Center for High-Performance Computing
Hsinchu, Taiwan
9203117@nchc.narl.org.tw

Yuan-Hsin Tung[c]
[c]Chunghwa Telecommunication Laboratories
Taoyuan, Taiwan
yhdong@cht.com.tw

***Abstract*—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. Traditionally, HPC users in supercomputing centers are required to specify the amount of processors to use upon job submission. However, we think this requirement might no longer be necessary for HPCaaS users since most modern parallel jobs are moldable and users usually could not know how to choose an appropriate amount of processors to allow their jobs to finish earlier. With moldable property, parallel programs can choose to exploit different parallelisms for execution just before their start of running. Therefore, we propose two moldable job scheduling approaches which take advantage of information of application speedup model to not only relieve HPC users' burden of selecting an appropriate number of processors but also achieve even better system performance than existing methods. The experimental results indicate that the proposed approaches can achieve up to 78% and 89% performance improvement in terms of average turnaround time.**

Keywords- moldable property, adaptive processor allocation, application speedup model, HPC as a Service.

I. INTRODUCTION

Parallel job scheduling and allocation has long been an important research topic[1][2][3]. Users at traditional supercomputing centers usually need to specify the amount of processors to use when submitting a parallel job. The workload management system and job scheduler will allocate computing resources to each job according to the specified processor requirement. If the specified amount of processors is larger than current available resources, the job would have to wait while the available resources are kept idle, resulting in degraded resource utilization and job turnaround time.

The above situation might be unavoidable when running rigid jobs [4], which can only run with a specific amount of processors, or conducting performance benchmarking, e.g. drawing the speedup curve. However, most modern parallel applications, e.g. those written with MPI [5], usually have the moldable property [4], which allows them to choose to exploit different parallelisms for execution just before their start of running. In such cases, system performance could be improved through adaptive processor allocation taking advantage of the moldable property. Here, adaptive processor allocation means users and job schedulers have the flexibility to select a suitable amount of processors for job execution, considering both jobs' parallelism characteristics and the amount of available resources at that moment. Some existing High-Performance Computing (HPC) workload management systems already support moldable job submission. For example, Load Sharing Facility (or simply LSF) [6], one of the most famous commercial HPC workload management system owned by IBM, allows users to specify a range of processor requirement, instead of a specific amount of processors, when submitting a moldable job. However, its scheduling mechanism for moldable jobs is quite primitive, simply adopting a greedy method to allocate as many processors as possible within the range specified by a moldable job upon submission.

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 [7]. As cloud computing emerges [8], which emphasizes easier and efficient access to IT infrastructure, recently the concept of HPC as a Service (HPCaaS) [7] was proposed to transform HPC facilities and applications into a more convenient and accessible service model. Unlike users at traditional supercomputing centers, who usually run parallel programs developed by themselves, most HPCaaS users run parallel applications developed by others and do not have a clear picture about the parallelism characteristics of the underlying applications. In such cases, users simply want to get their jobs done as soon as possible and don't want or even have no idea on how to specify an appropriate amount of processors for application's execution

Therefore, moldable job scheduling becomes a crucial research issue for HPCaaS in the following two aspects. Firstly, it relieves users' burden of selecting an appropriate number of processors upon job submission, leading to a much easier and convenient access model for HPCaaS. Secondly, moldable job scheduling has the potential to improve the average turnaround time of parallel applications and the overall resource utilization, benefiting both HPCaaS users and providers. Moreover, moldable job scheduling is also more feasible in HPCaaS than in traditional HPC systems since the parallelism characteristics and speedup model of the parallel applications prepared by

HPCaaS are more likely to be available than those of home-made parallel programs on traditional supercomputers.

Logically, moldable job scheduling approaches can be classified into four categories, as shown in Figure 1, according to two important aspects: submit-time or schedule-time decision and having job execution time information or not. Several moldable job scheduling approaches has been proposed in previous research [9][10][11], which fall into the categories in Figure 1 except quadrant IV, the one without job execution time information and making moldable decisions at submission time. In general, the approaches at schedule-time have potential to outperform the approaches at submit-time since job schedulers would have more accurate job execution time information and greater flexibility in choosing an appropriate amount of processors for a specific job at schedule-time.

Therefore, in this paper, we focus on the two schedule-time categories and propose two moldable job scheduling approaches which take advantage of the application speedup models to make appropriate processor allocation decisions at schedule time, falling into categories II and III in Figure 1. The proposed approaches were evaluated through a series of simulation experiments. The experimental results indicate that the proposed approaches can achieve up to 78% and 89% performance improvement in terms of average turnaround time.

The remainder of this paper is organized as follows. Section 2 discusses related works on moldable job scheduling. Section 3 presents a moldable job scheduling approach without job execution time information, while section 4 proposes an advanced approach to scheduling moldable jobs with execution time information. Section 5 presents the experiments and the results of performance evaluation. Section 6 concludes this paper.

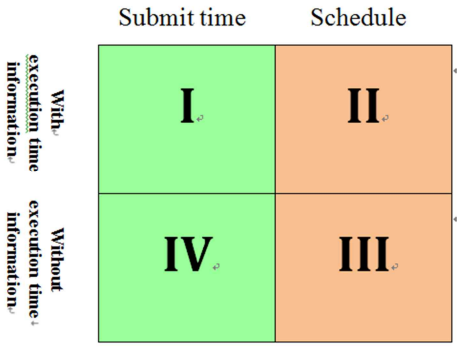

Fig.1. Moldable job scheduling quadrants

II. Related Work

Parallel jobs, according to the flexibility in parallelism, can be divided into four different classes [4]:(1) Rigid, (2) Moldable (3) Evolving, and (4) Malleable. A rigid job [4] can only run with a specific number of processors. Backfilling job scheduling approaches [19][20] are usually adopted to deal with the resource fragmentation problem and improve the overall system performance when scheduling rigid jobs in parallel systems. Moldable jobs [4] are flexible in the number of processors to use at the time the job starts, but cannot be reconfigured during execution. This is the type of parallel jobs we deal with in this paper. Malleable jobs [4] are similar to moldable jobs in that they both can run with different parallelisms in contrast to rigid jobs. However, malleable jobs are even more flexible in that they can change the amount of processors used dynamically during execution, while moldable jobs must determine the number of processors to use before execution and then fix the amount of processors throughout the entire execution period. 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 [21]. Both Evolving and Malleable jobs can change their processor requirements during execution [22][23]. For evolving jobs [4] changes are application initiated, while the changes in malleable jobs are system initiated.

According to the classification in Figure 1, in the following we discuss related work on moldable job scheduling in parallel computing systems. For category I, Cirne and Berman proposed an application-level scheduling approach for moldable jobs in [24][25], where users provide a set of candidate requests with different processor requirements, and the application scheduler is used to adaptively select the most suitable request based on current system configuration and workload status. For category II, in [26], Sabin et al. proposed an iterative algorithm which utilizes job efficiency information for scheduling moldable jobs. The proposed algorithm has high computational complexity since it is an iterative approach. In [9][10], Srinivasan et al. proposed a schedule-time aggressive fair-share strategy and a combined moldable scheduling strategy for moldable jobs, which adopt a profile-based allocation scheme. The strategies keep track of the information about all the free-time blocks available in current schedule and scan all the blocks to find the most suitable one for a moldable job at each scheduling activity, considering the effects of partition size on the performance of the application. The strategies thus need to have the knowledge of job execution time.

For category III, Huang proposed and evaluated four adaptive processors allocation heuristics for moldable jobs in [11]. The heuristics determine the amount of processors to use for each moldable job when it becomes the first job in the waiting queue. Only current available resources and job queue information are considered when making processor allocation decisions in the heuristics. The proposed approaches do not require the information of job execution time and the processor allocation decision is made at job starting time instead of submission time. Most of the previous approaches for moldable job scheduling require users to provide a candidate amount or range of amounts of processor requirement upon job submission. The moldable job scheduling approaches then tries to find the most appropriate amount of processors for job execution based on application characteristics and workload conditions. In this paper, we aim to develop moldable job scheduling approaches which can not only relieve users' burden of specifying processor requirement upon job submission, but also improve overall system performance by reducing the average turnaround time of all jobs. The proposed approaches can contribute to the realization of the HPC as a Service (HPCaaS) model [7].

III. Moldable Job Scheduling Without Job Execution Time Information

This section explores the issues of schedule-time moldable job scheduling without job execution time information, falling into category III in Figure 1. We propose a moldable job scheduling approach for the HPCaaS scenarios, where users do

not have to specify an amount of processors to use upon job submission. The scheduler will automatically select a most appropriate amount of processors for a job according to its speedup model and the workload condition at the moment. The automatic processor allocation decision is expected to benefit both HPCaaS users and providers through reduced average turnaround time of jobs and increased resource utilization. The proposed moldable job scheduling approaches will be evaluated and compared to existing methods, including the combined moldable scheduling strategy in [9][10] and the auxiliary moldable job scheduling approach in [18], through a series of simulation experiments in section 5.

Our approach adopts Downey's speedup model of parallel programs [14][15] to take into consideration of both single job speedup and entire system performance. The speedup model developed by Downey has been shown capable of representing the parallelism and speedup characteristics of real parallel applications [14][15]. The speedup of a job on n processors is defined as the ratio of the job's execution time on a single processor to the job's execution time on n processors:

$$S(n) = \frac{L}{T(n)} \quad (1)$$

Here, S is the speedup function, L is the effective sequential execution time and T(n) is the execution time of the job on n processors. Downey's model is a non-linear function of the following two parameters [14]:

- **σ** 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.
- **A** denotes the average parallelism of a job and is a measure of the maximum speedup that the job can achieve.

Downey proposed two speedup models with low and high variances, respectively, in [14]. Figure 2 is a hypothetical parallelism profile for a program with low variance in degree of parallelism. The parallelism is equal to A, the average parallelism, for all but some fraction σ of the duration ($0 \le \sigma \le 1$). The remaining time is divided between a sequential component and a high-parallelism component (with parallelism chosen such that the average parallelism is A). The run time and speedup of a parallel program, as functions of processor number, with the low-variance model are described in equations (2) and (3), respectively.

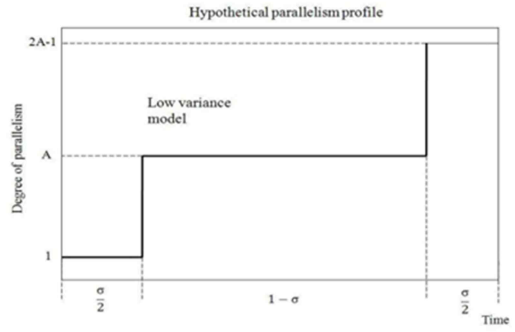

Fig.2. The parallelism profile for low-variance speedup model

$$T(n)=\begin{cases} \frac{A-\sigma/2}{n}+\sigma/2 & 1 \le n \le A \\ \frac{\sigma(A-1/2)}{n}+1-\sigma/2 & A \le n \le 2A-1 \\ 1 & n \ge 2A-1 \end{cases} \quad (2)$$

$$S(n)=\begin{cases} \frac{An}{A+\frac{\sigma}{2(n-1)}} & 1 \le n \le A \\ \frac{An}{\sigma\left(A-\frac{1}{2}\right)+n(1-\sigma/2)} & A \le n \le 2A-1 \\ A & n \ge 2A-1 \end{cases} \quad (3)$$

Figure 3 shows a hypothetical parallelism profile for a program with high variance in parallelism. The profile consists of a sequential component of duration σ and a parallel component of duration 1 with the parallelism of A + Aσ-σ. A program with this profile would have the following execution time and speedup functions of processor number, described in equations (4) and (5), respectively.

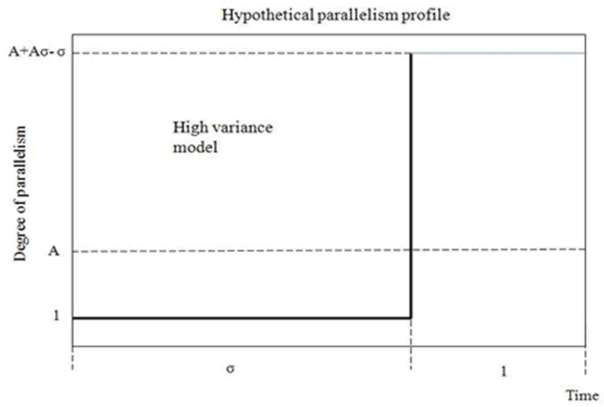

Fig.3. The parallelism profile for high-variance speedup model

$$T(n)=\begin{cases} \sigma+\frac{A+A\sigma-\sigma}{n} & 1 \le n \le A+A\sigma-\sigma \\ \sigma+1 & n \ge A+A\sigma-\sigma \end{cases} \quad (4)$$

$$S(n)=\begin{cases} \frac{nA(\sigma+1)}{\sigma(n+A-1)+A} & 1 \le n \le A+A\sigma-\sigma \\ A & n \ge A+A\sigma-\sigma \end{cases} \quad (5)$$

Figure 4 is an example illustrating the advantage of moldable job scheduling over traditional rigid job scheduling. In the left part of the figure, since rigid job scheduling would try to allocate exactly the amount of processors specified upon job submission to each parallel job, task III cannot get enough processors to start its execution in the beginning, resulting in degraded average turnaround time of all three jobs and worse resource utilization rate. On the other hand, in the right part of the figure, moldable job scheduling would scale the parallelism of task III down to use only 20 processors for execution, instead of 30 specified originally. This arrangement allows task III to start its execution earlier with less number of processors. Although task III thus needs longer time to finish its execution in this arrangement, the total turnaround time is actually reduced since the waiting time of task III decreases to zero. The comparative example demonstrates the potential advantage of moldable job scheduling.

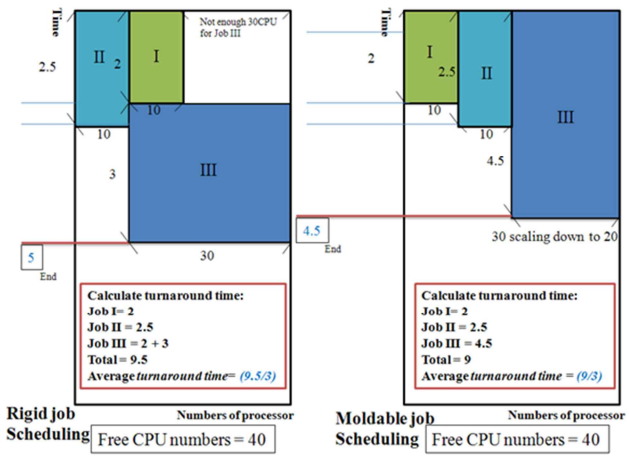

Fig.4. Advantage of moldable job scheduling

The usage scenario on traditional supercomputers requires users to specify preferred amounts of processors upon job submission. This would be tedious and even difficult for HPCaaS users who just want to run their application services as quickly as possible and do not have enough knowledge of the underlying parallel structure of the application services developed by the HPCaaS providers. In the following, we propose a moldable job scheduling approach for HPCaaS which can automatically select a most appropriate amount of processors to use for each job, benefiting both the job's turnaround time and the resource utilization cared by HPCaaS providers.

Without the processor requirement information provided by users, the job scheduler, in general, has two possible directions in making processor allocation decisions. The first direction is to allow as many jobs to run immediately as possible. Therefore, each job would get only a small portion of total free processors for its execution that results a higher degree of concurrency among jobs. The alternative choice is to give the first job in queue as many processors as possible, allowing the job to run faster, but decreasing the degree of concurrency among jobs. We call these two directions parallel policy and serial policy, respectively. Which policy is better would largely depend on the parallel behavior of applications. Therefore, in the following we investigate the effects of these two allocation policies under three commonly used parallel speedup models.

The three parallel speedup models considered in this section cover the behavior of most parallel applications. The first is the model usually introduced in the textbook of parallel processing [16], called linear speedup model hereafter in this paper, where speedup is defined by Sp = T1/Tp, with p the number of processors, T1 the execution time of the sequential run, Tp the execution time of parallel processing with p processors. Based on the definition of speedup, efficiency is another performance metric defined as Ep=Sp/p = T1/pTp. Efficiency is a value, typically ranging between zero and one, estimating how well-utilized the processors are in solving the problem.

The second model considered in this section is Amdahl's law [13], which states that if P is the proportion of a program that can be made parallel, then the maximum speedup value 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 [14][15]. Downey's model is a non-linear function of two parameters which has been described in details in the previous section.

Based on the speedup models, the resultant average turnaround time of a set of parallel jobs using the two allocation policies can be derived. For example, equations (6) and (7) are for the linear speedup model and the average turnaround time for Amdahl's law is equations (8) and (9), where t is the job's sequential execution time, 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. For simplicity, n is assumed to be d's multiple and all jobs have the same sequential execution time, t.

$$\text{Average turnaround time}_{\text{parallel}} = \frac{t}{\frac{n}{d}\cdot x}\cdot d\cdot\frac{1}{d} \quad (6)$$

$$\text{Average turnaround time}_{\text{serial}} = \frac{t}{n\cdot x}\cdot(1+d)\cdot\frac{d}{2}\cdot\frac{1}{d} \quad (7)$$

$$\text{Average turnaround time}_{\text{parallel}} = t\cdot\left((1-x)+\frac{x}{\frac{n}{d}}\right)\cdot d\cdot\frac{1}{d} \quad (8)$$

$$\text{Average turnaround time}_{\text{serial}} = t\cdot\left((1-x)+\frac{x}{n}\right)\cdot(1+d)\cdot d\cdot\frac{1}{d} \quad (9)$$

Figures 5 to 8 compare the performance of the parallel and serial allocation policies, in terms of average turnaround time, on different application speedup models. The comparison indicates that a 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 linear speedup model, while the parallel allocation policy can achieve better performance for other models. Based on this analysis, we developed a moldable job scheduling approach for HPCaaS, 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 through a series of simulation experiments in section 5.

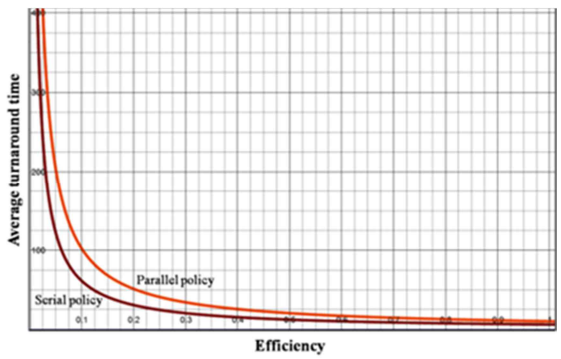

Fig.5. Linear speedup model

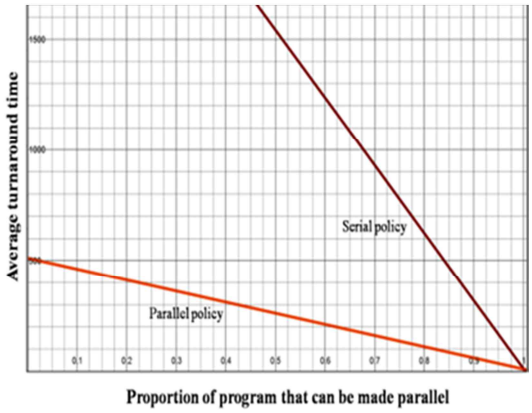

Fig.6. Amdahl'law

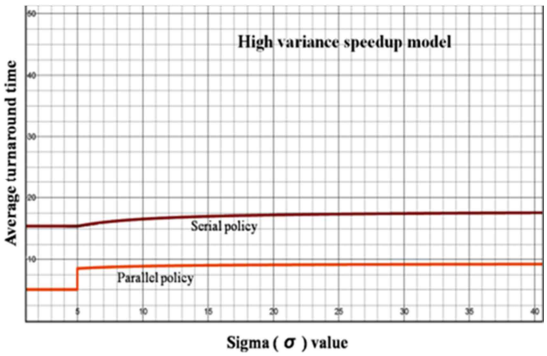

Fig.7. Downey's high-variance model

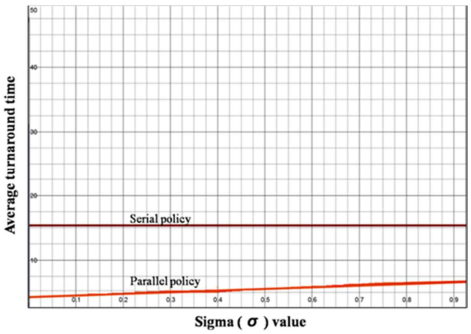

Fig.8. Downey's low-variance model

Figure 9 is an example based on Downey's high-variance model, illustrating the benefits of the proposed moldable job scheduling approach for HPCaaS by comparing it with the auxiliary moldable job scheduling approach proposed in [18]. In the lower part of figure, the auxiliary moldable job scheduling approach tends to give task I as many processors as possible, provided that the amount of processors allocated does not exceed its knee value. However, this arrangement would delay the start time of task III, resulting in a longer average turnaround time. On the other hand, in the upper part of the figure, the moldable job scheduling approach for HPCaaS tends to run the three tasks simultaneously since parallel allocation policy is a better choice for Downey's high-variance model as shown in Figure 8, leading to a shorter average turnaround time compared to the lower part. This comparative example demonstrates the potential superiority of the proposed moldable job scheduling approach for HPCaaS.

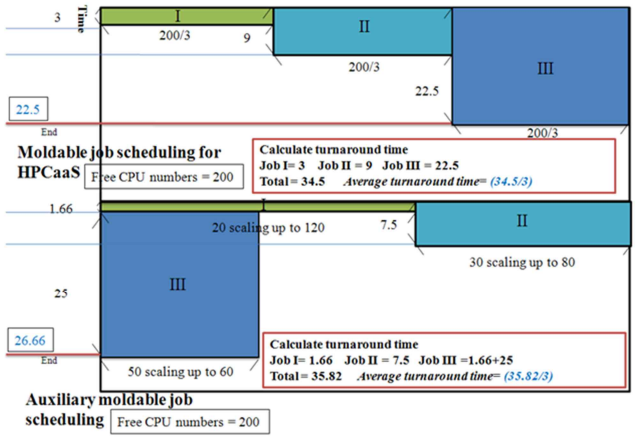

Fig.9. Comparison of auxiliary moldable job scheduling and moldable job scheduling for HPCaaS

IV. Moldable Job Scheduling with Job Execution Time Information

The approach discussed in section 3 falls into the category III in Figure 1, which do not require the job execution time information of submitted jobs. This section explores the issues of moldable job scheduling in the category II of Figure 1 and proposes a new moldable job scheduling approach, which takes advantage of the job execution time information of submitted jobs to make better processor allocation decisions. Jobs' execution time information has been used in many parallel job scheduling methods, e.g. backfilling job scheduling methods for rigid jobs [19][20]. In such scheduling systems, users are required to provide job execution time information upon job submission. The information is then used to guide the advanced non-FCFS (First-Come-First-Serve) job scheduling through job backfilling. In this section, the job execution time information is used to help making better processor allocation decisions for moldable job scheduling through increasing resource utilization as more as possible.

4.1 Previous Moldable Job Scheduling Approaches with Job Execution Time Information

Several previous submit-time or schedule-time moldable job scheduling methods [9][10] have adopted the job execution time information to improve job scheduling performance. In these methods, job execution time information is mainly used to derive a maximum number of processors to use for each job based on the notion of fair sharing. The proposed fair-sharing schemes believe in that each job should be given a limit on the maximum allowable amount of processors to use and the limit should depend on the job's fraction of the total weight of the jobs currently in the system, including both running and waiting jobs. Therefore, several schemes for calculating the limit have been proposed in previous research [9][10].

In [10], a scheme was proposed for calculating the maximum allowable amount of processors to use for each job as follows.

Weight fraction of job i=

$$\frac{\text{Weight of job i}}{\sum_{\text{all jobs}} \text{Weight of job currently in system}}$$

where Weight refers to the effective sequential execution time of a job, and all currently running and queued jobs in the

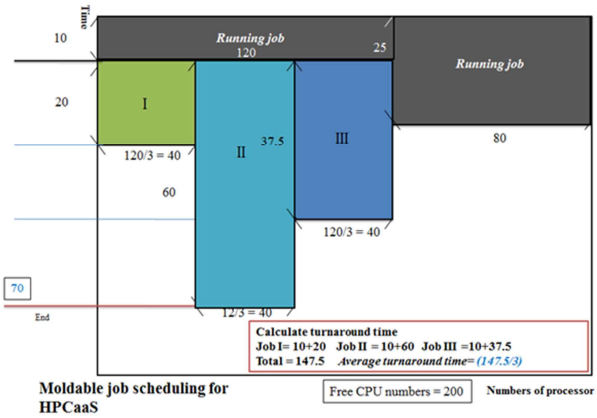

Fig.10. The original moldable job scheduling for HPCaaS approach when applying the parallel allocation policy

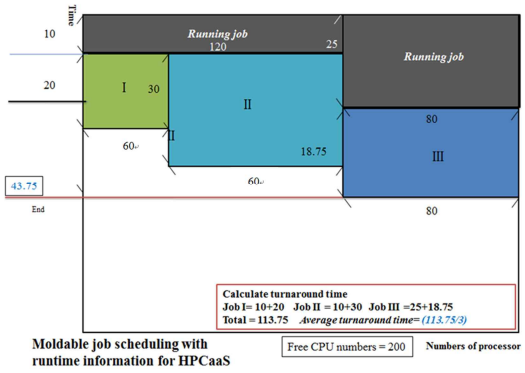

Fig.11. Advantage of the moldable job scheduling with job execution time information for HPCaaS

system are considered. In [9], Srinivasan et al. proposed another modified weight-fraction formula as follows.

Modified weight fraction of job i=

$$\frac{\sqrt{\text{Weight of job i}}}{\sum \sqrt{\text{Weight of job currently in system}}}$$

Given two jobs, with one having 4 times the weight of the other, the idea is to give the large job twice as many processors as the small job, with the effect that it would also have twice the execution time as the small job. Thus relative to the small job, the large job is reshaped to spread out in a roughly equal way along both the space and time dimensions. However, even though each sequential job can only take at most 1 processor, the modified weight-fraction formula would set aside a proportional amount of processors even for such jobs. To correct this inconsistency, they ignored the sequential jobs while calculating the weight fraction and the corrected modified strategy can improve much more performance in their study [9]. The corrected modified weight-fraction formula is:

Corrected modified weight fraction of job i=

$$\frac{\sqrt{\text{Weight of job i}}}{\sum_{\text{all jobs}} \sqrt{\text{Weight of pallel jobs currently in system}}}$$

Based on the above corrected modified weight faction calculation, a schedule-time combined moldable job scheduling strategy was proposed and shown to outperform previous submit-time and schedule-time moldable job scheduling approaches in [10][24].

4.2 Our Moldable Job Scheduling Approaches with Job Execution Time Information for HPCaaS

In this section, we extend the moldable job scheduling for HPCaaS approach, described in section 3, to take advantage of job execution time information to further improve the overall system performance. Figure 10 is an example showing the potential inefficiency of the original moldable job scheduling for HPCaaS approach when applying the parallel allocation policy. As shown in the figure, the parallel allocation policy tends to run all the jobs in queue, tasks I, II, and III, simultaneously. However, if the waiting queue remains empty till some of the running jobs finish their execution, the released resources will become idle and result in degraded resource utilization. This arrangement also leads to longer execution time for tasks I, II, and III since each of them is allocated a smaller portion of processors than available after time 30.

To avoid the potential inefficiency illustrated in Figure 10, we propose an improved moldable job scheduling approach for HPCaaS, which takes the advantage of job execution time information to improve overall system performance through increasing resource utilization. The proposed approach works as follows. It maintains two queue structures recording information about running and waiting jobs, respectively. On each processor allocation decision, it first scans the running queue to collect the time instants of future resource releases resulting from the finishes of running jobs based on the job execution time information. Then, it counts the number of jobs in the waiting queue. If the number of waiting jobs is less than the numbers of future resource releases, it simply changes the parallel allocation policy into serial allocation policy, and allocates only the first job in the waiting queue for execution.

On the other hand, if the number of waiting jobs is larger than the numbers of future resource releases, it keeps the parallel allocation policy, but only allocates the first n jobs in the waiting queue for execution. The value of n is calculated by subtracting the number of running jobs from the number of waiting jobs. In this way, the improved moldable job scheduling approach can raise the resource utilization rate as much as possible, and is expected to result in shorter average turnaround time for all jobs.

Figure 11 is another example showing the benefits of the improved moldable job scheduling approach. Compared to Figure 10, now task III is not allocated immediately at the same time as task I. Instead, its allocation is delayed until the running job finishes its execution. In this way, although the waiting time of task III is increased, all three tasks are allocated more processors for execution, compared to Figure 10, resulting in shorter turnaround time in average. Figure 12 shows the detailed algorithm of the improved moldable job scheduling approach.

V. EXPERIMENTS AND PERFORMANCE EVALUATION

This section evaluates the proposed moldable job scheduling approaches in sections 3 and 4, and compares them with previous methods in [9][10][18].The performance evaluation was conducted through a series of simulation experiments, assuming a 128-processor cluster, based on a

Algorithm: Improved moldable job scheduling for HPCaaS

Attributes:

waitQueue.Length: number of waiting jobs
runQueue.Length: number of running jobs
numRD:number of jobs to be allocated

```
1:  numRD = waitQueue.Length–runQueue.Length;
2:  if (numRD> 0)
3:   {
4:    waitQueue.deQueue(first numRD jobs);
5:      allocate the numRD jobs using the parallel allocation
policy;
6:      runQueue.enQueue(these numRD allocated jobs);
7:          }
8:  else
9:   if (waitQueue.Length> 0)
10:   {
11:    waitQueue.deQueue(first job);
12:     allocate the job with all current free processors
13:      runQueue.enQueue(the allocated job);
14:    }
```

Fig.12. Algorithm of the improved moldable job scheduling for HPCaaS

public workload log on SDSC's SP2 [17]. The workload log contains 73496 records collected on a 128-node IBM SP2 machine at San Diego Supercomputer Center (SDSC) from May 1998 to April 2000. After excluding some problematic records based on the completed field [17] in the log, the simulation experiments in this paper use 56490 job records as the input workload. The two parameters, σ and A, for Downey's speedup models were generated randomly. Moreover, we introduced an integer load factor in the simulation experiments, which was used to generate different levels of system load by simply multiplying it with the original job execution time recorded in the workload log.

Figures 13 and 14 evaluate the proposed moldable job scheduling for HPCaaS approach and compare it with the auxiliary moldable job scheduling approach [18] using Downey's two parallel speedup models. The experimental results indicate that the proposed moldable job scheduling for HPCaaS approach not only can relieve users' burden of specifying an appropriate amount of processors upon job submission, but also outperforms the previous method significantly across different levels of system load, achieving up to 78% performance improvement in terms of average turnaround time.

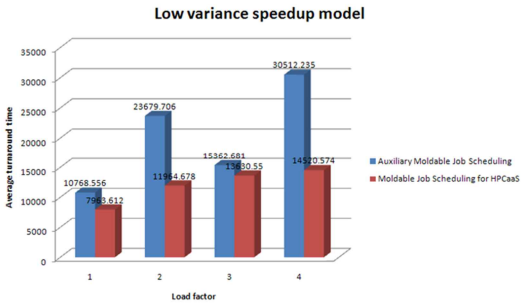

Fig.13. Evaluation of moldable job scheduling for HPCaaS with Downey's low-variance speedup model.

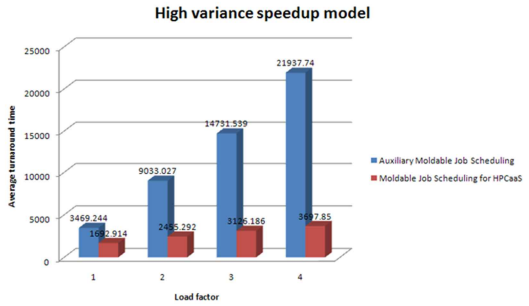

Fig.14. Evaluation of moldable job scheduling for HPCaaS with Downey's high-variance speedup model.

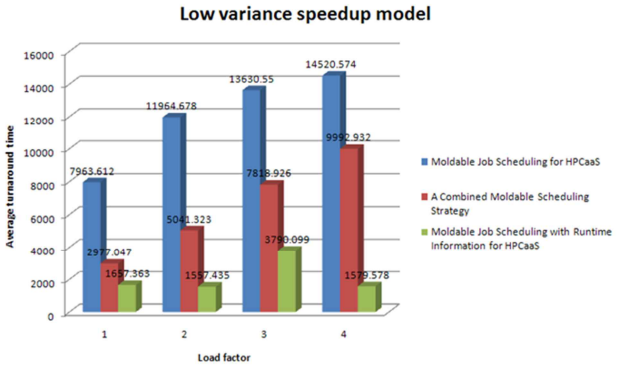

Fig.15. Evaluation of moldable job scheduling with job execution time information using Downey's low-variance speedup model.

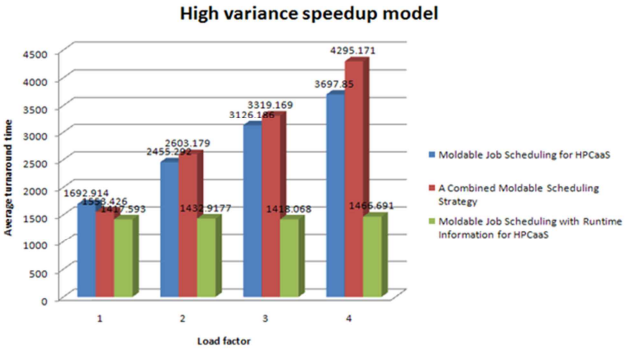

Fig.16. Evaluation of moldable job scheduling with job execution time information using Downey's low-variance speedup model.

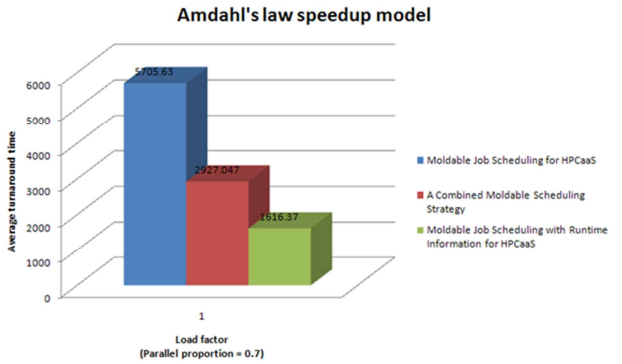

Fig.17. Evaluation of moldable job scheduling with job execution time information using Amdahl's law speedup model.

Figures 15, 16, and 17 evaluate our moldable job scheduling approach with job execution time information, comparing it with the combined moldable job scheduling

strategy in [9] and the previous moldable job scheduling for HPCaaS approach based on Downey's two parallel speedup models and the Amdahl's law speedup model. The experimental results indicate that the our moldable job scheduling approach with job execution time information outperforms the previous methods significantly across different levels of system load, achieving up to 89% performance improvement in terms of average turnaround time.

VI. CONCLUSIONS

Traditional supercomputing centers usually adopt rigid job scheduling, which requires users to specify the amount of processors to use upon job submission and then allocates computing resources to each job according to the specified processor requirement. However, if the specified amount of processors is larger than current available resources, the job would have to wait while the available resources are kept idle, resulting in degraded resource utilization and job turnaround time. Most modern parallel applications, e.g. those written with MPI, usually have the moldable property, which allows them to exploit different parallelisms for execution just before starting to run. In such cases, the traditional rigid job scheduling is inappropriate. Therefore, moldable job scheduling becomes an important research topic which aims to improve the overall system performance through adaptive processor allocation taking advantage of the moldable property.

Moreover, as cloud computing emerges, recently the concept of HPC as a Service (HPCaaS) was proposed to transform HPC facilities and applications into a more convenient and accessible service model. For HPCaaS, users simply want to get their jobs done as soon as possible and don't want to or even have no idea on how to specify an appropriate amount of processors for application's execution. Therefore, moldable job scheduling can contribute to HPCaaS in two important aspects. Firstly, it relieves users' burden of selecting an appropriate number of processors upon job submission, leading to a much easier and convenient service model for HPCaaS. Secondly, moldable job scheduling has the potential to improve the average turnaround time of parallel applications and the overall resource utilization, benefiting both HPCaaS users and providers.

In this paper, we classify previous research on moldable job scheduling into four quadrants according to two aspects: submit-time or schedule-time decision and having job execution time information or not. Based on this classification, we make two contributions to moldable job scheduling by taking advantage of the information of applications' speedup models. In the first contribution, we propose a moldable job scheduling approach for HPCaaS, which can automatically select a most appropriate amount of processors for a job's execution based on applications' speedup models and workload conditions at the moment, relieving users' burden of selecting an appropriate number of processors upon job submission. The proposed approach in the first contribution does not require job execution time information. In the second contribution, we propose an advanced moldable job scheduling approach, taking advantage of job execution time information to further improve the overall system performance. The proposed moldable job scheduling approaches were evaluated through a series of simulation experiments, and compared to previous methods in the literature. The experimental results indicate that our approaches outperform existing methods significantly, achieving up to 78% and 89% performance improvement in terms of average turnaround time, respectively.

REFERENCES

[1] D. G. Feitelson, "A Survey of Scheduling in Multiprogrammed Parallel Systems", Proc. Research Report RC 19790 (87657), IBM T. J. Watson Research Center, Oct. 1994.

[2] R. Gibbons, "A Historical Application Profiler for Use by Parallel Schedulers", Proc. Job Scheduling Strategies for Parallel Processing, pp. 58-77, Springer-Verlag, 1997.

[3] D. Lifka, "The ANL/IBM SP Scheduling System", Proc. Job Scheduling Strategies for Parallel Processing, pp. 295-303, Springer-Verlag, 1995.

[4] D. G. Feitelson, L. Rudolph, U. Schweigelshohn, K. Sevcik, and P. Wong, "Theory and Practice in Parallel Job Scheduling". Proc. Job Scheduling Strategies for Parallel Processing, D. G. Feitelson and L. Rudolph (Eds.), pp. 1-34, Springer-Verlag, 1997. Lecture Notes in Computer Science Vol. 1291.

[5] The Message Passing Interface standard, http://www.mcs.anl.gov/research/projects/mpi/ (June 2013)

[6] Load sharing facility, http://www-03.ibm.com/systems/technicalcomputing/platformcomputing/products/lsf/ (June 2013)

[7] M. AbdelBaky,M.Parashar, H. Kim,E. J.JordanKirk,V.Sachdeva, J. Sexton, H. Jamjoom, Z.Y. Shae, G. Pencheva, R. Tavakoli and M. F. Wheeler, "Enabling High Performance Computing as a Service", Proc. IEEE Computer, Vol. 45, pp. 72-80. IEEE Press, Oct. (2012).

[8] Cloud computing, http://www.infoworld.com/d/cloud-computing/what-cloud-computing-really-means-031 (Mar 2012)

[9] S. Srinivasan, S. Krishnamoorthy and P. Sadayappan, "A Robust Scheduling Strategy for Moldable Scheduling of Parallel Jobs", Proc. 5th IEEE International Conference on Cluster Computing, pp. 92-99, 2003.

[10] S. Srinivasan, V. Subramani, R. Kettimuthu, P. Holenarsipur and P. Sadayappan, "Effective Selection of Partition Sizes for Moldable Scheduling of Parallel Jobs", Proc. 9th International Conference on High Performance Computing, Springer, Lecture Notes in Computer Science, Bangalore, India, Vol. 2552, pp.174-183, 2002.

[11] K. C. Huang, "Performance Evaluation of Adaptive Processor Allocation Policies for Moldable Parallel Batch Jobs", Proc. 3th Workshop on Grid Technologies and Applications, Dec 2006.

[12] D. L. Eager, J. Zahorjan, E. D. Lozowska,"Speedup versus efficiency in parallel systems". IEEE Transactions on Computers archive Vol.38, Issue 3, pp. 408-423, March 1989.

[13] L. Kleinrock, J.H. Huang: On parallel processing systems, "Amdahl's law generalized and some results on optimal design", Proc. IEEE Transactions Softw. Eng. 18(5) (1992)

[14] A. B. Downey, "A Model for Speedup of Parallel Programs", Proc. UC Berkeley EECS Technical Report, No. UCB/CSD-97-933, January 1997.

[15] A. B. Downey, "A Parallel Workload Model and Its Implications for Processor Allocation", Proc. the 6th International Symposium on High Performance Distributed Computing, 1997.

[16] T. G. Lewis and H. E. Rewini, Introduction to Parallel Computing, Prentice-Hall International, 1992.

[17] Parallel Workloads Archive, http://www.cs.huji.ac.il/labs/parallel/workload/ (Jan 2012)

[18] K. C. Huang, T. C. Huang, Y. H. Tung, P. Z. Shih, "Effective Processor Allocation for Moldable Jobs with Application Speedup Model", Proc. International Computer Symposium, Hualien, Taiwan, December 12-14 2012, pp 563-572.

[19] D. G. Feitelson and A. M. Weil, "Utilization and Predictability in Scheduling the IBM SP2 with Backfilling", Proc. 12th Int'l Parallel Processing Symp., pp. 542-546, Apr. 1998

[20] A. W. Mu'alem and 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, Issue 6, pp. 529-543, June 2001.

[21] H. Sun, Y. Cao and W. J. Hsu, "Efficient Adaptive Scheduling of Multiprocessors with Stable Parallelism Feedback", Proc. IEEE Transactions on Parallel and Distributed System, Vol. 22, No. 4, April 2011.

[22] S. Ioannidis, U. Rencuzogullari, R. Stets, and S. Dwarkadas, "CRAUL: Compiler and run-time integration for adaptation under load", Journal of Scientific Programming, Aug. 1999.

[23] J. Pruyne and M. Livny. Parallel Processing on Dynamic Resources with CARMI. In D. G. Feitelson and L.Rudolph, editors, Proc. Job Scheduling Strategies for Parallel Processing, Vol 949, pp. 259–278. Springer, 1995.

[24] W. Cirne and F. Berman, "Using Moldability to Improve the Performance of Supercomputer Jobs", Journal of Parallel and Distributed Computing, Vol. 62, pp. 1571-1601, Oct 2002

[25] W. Cirne and F. Berman, "Adaptive Selection of Partition Size for Supercomputer Requests", Proc. Job Scheduling Strategies for Parallel Processing Lecture Notes in Computer Science, Vol. 1911, pp. 187-207 , 2000

[26] G. Sabin, M. Lang and P Sadayappan, "Moldable Parallel Job Scheduling Using Job Efficiency: An Iterative Approach", Proc. Job Scheduling Strategies for Parallel Processing, Saint Malo, France, June 2006.

BIOGRAPHIES

Kuo-Chan Huang received his B.S. and Ph.D. degrees in Computer Science and Information Engineering from National Chiao-Tung University, Taiwan, in 1993 and 1998, respectively. He is currently an Associate Professor in Computer Science Department at National Taichung University of Education, Taiwan. He is a member of ACM and IEEE Computer Society. He has served as workshop chair, publicity chair, and program committee member for several international conferences. His research areas include parallel processing, cluster, grid, and cloud computing, workflow computing, and services computing.

Tse-Chi Huang received his B.S. and M.S. degrees in Computer Science from National Taichung University of Education, Taiwan, in 2013. His main research interests include parallel job scheduling, cluster computing, parallel processing.

Mu-Jung Tsai received her B.S. degrees in 2012 from and is currently a graduate student in the department of Computer Science at National Taichung University of Education, Taiwan. Her main research interests include SOA, cloud and services computing.

Hsi-Ya Chang received his Ph.D. degree from Duke University, USA in 1986. He had taught at University of Miami, Florida, USA for several years before he returned to his homeland Taiwan and became a research scientist in the National Center for High-Performance Computing. His research interests include parallel processing, cluster/Grid/Cloud computing and HPC applications. He has published over 70 scholarly papers. He was a founding member of UniGrid, a Grid platform connecting major universities and research institutes in Taiwan. In 2010 Dr. Chang initiated the Alliance of Cloud Computing Technologies and Applications (ACCTA) to promote cloud computing development/activities/research in Taiwan. He is now the Secretary General of ACCTA and also a council member of the Taiwan Association of Cloud Computing (TACC). He has served as general chair, program chair/vice chair, publicity chair, committee member, etc., in over 60 international conferences.

Yuan-Hsin Tung received his Ph.D. degree in the Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan in 2013. He received a BS degree in Transportation Communication Management from the National Cheng-Kung University in Taiwan, and an MS in Information Management from the National Sun Yat-Sen University in Taiwan. Currently, he has been project manager, since 2007, of Test Center of Telecommunication Labs. of Chunghwa Telecom in Taoyuan, Taiwan. His main researches are software testing, cloud computing, knowledge engineering, data-mining technology, and software engineering.