J. Parallel Distrib. Comput. 73 (2013) 1661–1672

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

TLA: Temporal look-ahead processor allocation method for heterogeneous multi-cluster systems

Po-Chi Shih [a,*], Kuo-Chan Huang [b], Che-Rung Lee [a], I-Hsin Chung [c], Yeh-Ching Chung [a]

[a] Department of Computer Science, National Tsing-Hua University, Hsinchu, Taiwan
[b] Department of Computer and Information Science, National Taichung University of Education, Taichung, Taiwan
[c] IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, USA

HIGHLIGHTS

- The allocation simulation process used in TLA is novel and effective.
- TLA directly utilizes the performance metric to make allocation decisions.
- Extensive simulation has been carrying out to evaluate the performance of TLA.
- With precise runtime, TLA has up to an 87% performance improvement.

ARTICLE INFO

Article history:
Received 16 October 2012
Received in revised form
4 May 2013
Accepted 19 July 2013
Available online 14 August 2013

Keywords:
Parallel job scheduling
Multi-cluster
Heterogeneity
Processor allocation
Look-ahead

ABSTRACT

In a heterogeneous multi-cluster (HMC) system, processor allocation is responsible for choosing available processors among clusters for job execution. Traditionally, processor allocation in HMC considers only resource fragmentation or processor heterogeneity, which leads to heuristics such as Best-Fit (BF) and Fastest-First (FF). However, those heuristics only favor certain types of workloads and cannot be changed adaptively. In this paper, a temporal look-ahead (TLA) method is proposed, which uses an allocation simulation process to guide the decision of processor allocation. Thus, the allocation decision is made dynamically according to the current workload and system configurations. We evaluate the performance of TLA by simulations, with different workloads and system configurations, in terms of average turnaround time. Simulation results indicate that, with precise runtime information, TLA outperforms traditional processor allocation methods and has up to an 87% performance improvement.

1. Introduction

This paper focuses on the issue of processor allocation for parallel jobs in heterogeneous multi-cluster (HMC) systems. An HMC system consists of multiple clusters, whose computational power, memory size, and communication capability can be varied for different clusters. Such systems are becoming more and more popular in grid computing and cloud computing [20,24,23].

In an HMC system, a central job scheduler (also known as a meta-scheduler) is commonly used to dispatch all submitted jobs [20,23]. A central queue, called the waiting queue, is used to accommodate those submitted jobs. The central job scheduler usually involves two operations: job scheduling and processor allocation. Job scheduling decides the execution order of the jobs, while processor allocation decides which cluster(s) to allocate the job to [20,23].

The issues of processor allocation in a HMC system are more complicated than those in other parallel systems. In a homogeneous cluster system or a supercomputer, all processors have equal computation capability. Therefore, it makes no significant difference to allocate a job on different processors. For a homogeneous multi-cluster system, processor allocation methods can be divided into two categories, single-site allocation and multi-site co-allocation, depending on whether the system supports a job to be executed across different clusters [12]. The single-site allocation algorithms need to take care of the resource fragmentation problem, which means the entire system has a sufficient amount of available processors for a job but no single cluster alone has enough free processors to accommodate it. For an HMC system, the heterogeneity of resources adds another dimension of complexity to the allocation problem [23].

In this paper, we focus on the case of single-site allocation on HMC, since multi-site co-allocation is rarely seen in production systems [21]. In addition, we assume the heterogeneity is only for computing speed (it is called *speed heterogeneity* hereafter in this paper). Because of speed heterogeneity, allocating a job onto

* Corresponding author.
E-mail address: shedoh@sslab.cs.nthu.edu.tw (P.-C. Shih).

0743-7315/$ – see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jpdc.2013.07.018

different clusters could lead to a different job execution time. Moreover, different allocation decisions for a job may also affect the waiting time of the jobs behind it in the waiting queue because of the resource fragmentation. Therefore, processor allocation becomes a crucial issue in a HMC system because it affects both the waiting time and execution time of the job.

Traditional processor allocation heuristics on a HMC system aim to resolve the *resource fragmentation* problem or to leverage the *speed heterogeneity* property to improve system performance, which leads to heuristics such as Best-Fit (BF) [10] and Fastest-First (FF) [12]. As reported in [23], their performance largely depends on the input workload and system configurations since both methods consider only a single performance factor.

This paper proposes a novel single-site allocation method called temporal look-ahead (TLA), which uses an allocation simulation process to guide the decision of processor allocation. Instead of focusing on resource fragmentation and speed heterogeneity directly, TLA tries to optimize the performance of entire system based on the specified performance metric. The idea behind TLA is that by considering the performance metric directly it will naturally take into account all relevant performance factors to that metric. This design gives TLA potential to optimize other performance metric, which may have completely different performance factors.

TLA works as follows. For each job j to be allocated, TLA evaluates all possible allocations and picks the one that could result in best system performance. Each possible allocation, say allocate to cluster c, is evaluated through a simulation, which simulates the job scheduling, allocation, and execution of all subsequent jobs in the waiting queue, under the assumption that job j is allocated to cluster c, and evaluates the consequent system performance of such an allocation.

The realization of TLA needs to know which job scheduling algorithm is used since it decides the execution order of jobs in the waiting queue. TLA is a general processor allocation algorithm which can work with different job scheduling algorithms. To simplify the presentation, this paper only demonstrate the capability of TLA with the well-known First-Come–First-Served (FCFS) job scheduling algorithm [20,23,15].

To show the effectiveness of TLA, we compared TLA with existing processor allocation heuristics with the metric of average job turnaround time. Simulation experiments were conducted for various input workloads and system configurations. Simulation results show that the peak performance improvements made by TLA can be up to 87%, when the runtime estimation is accurate.

The rest of this paper is organized as follows. Section 2 presents the system model and reviews the related work. Section 3 illustrates the idea of TLA. Section 4 presents the results of experiments which are based on precise job runtime estimation. Section 5 presents and discusses the performance issue of TLA. Conclusions and future work are given in Section 6.

2. Background and related works

2.1. System model

The system in discussion is a heterogeneous multi-cluster (HMC) architecture, which consists of a collection of interconnected clusters. Each cluster is a computer system with homogeneous processors, while the number and the speed of processors can be varied for different clusters. Here we assume the speed difference among the clusters can be perfectly reflected by job runtime. For example, if the speed of processor in cluster A is twice as fast as that in cluster B, running a job on cluster A would take half of the time by running the job on cluster B. This assumption is optimistic, but widely used in literature [28,29,25].

In HMC, we assume there is a global waiting queue to accommodate all the submitted jobs, which are then scheduled by a central job scheduler. Theoretically, a central job scheduler could be a critical limitation concerning scalability 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 job manager [6].

Each submitted job can only be allocated within a single cluster and we assume no dependency among the jobs. The HMC system in discussion runs jobs in an exclusive fashion, i.e. time-sharing of jobs is not allowed on each processor. Once a job starts execution, it runs to finish without interruption. Neither migration nor preemption is allowed. Job submissions are in an on-line manner, i.e. no knowledge of further job submission can be obtained. Each submitted job needs to specify the required number of processors and the estimated running time. All jobs are assumed to be rigid [16], i.e. the required number of processors of each job cannot be changed after job submission. The model described above is commonly used in literature [20,23,21].

2.2. Scheduling model

We use the same online scheduling model as presented in [23,12]. In the model, the scheduling session is activated by the scheduling event. A scheduling event is triggered when a new job is submitted or when a running job finishes. The entire scheduling flow in a scheduling session is depicted in Fig. 1. For each scheduling session, it will go through the following phases:

- Job scheduling: picks one target job from the waiting queue and pass it to Processor Allocation phase.
- Processor Allocation: chooses a cluster with enough free processors to execute the target job. If such cluster exists, allocate the target job to the selected cluster immediately for execution, then go back to Job Scheduling phase to select the next target job. Otherwise, end this scheduling session.

Based on the flow, each scheduling session will try to allocate as many jobs as possible for immediate execution, until there are no jobs left in the waiting queue or there are no clusters with enough free processors for the target job.

2.3. Job allocation algorithms

The problem of parallel job allocation had been widely investigated for various computational architectures. On the early hypercube-based parallel computers, processor allocation [18,22] is critical for performance, because allocating a job to different sub-cubes, might lead to diverse system performance owing to resource fragmentation. Later, for switch-based parallel computers and cluster-based computing systems, job scheduling [9,17,8,27] becomes a more important problem, which stemmed from the fact that on such systems an allocation can be made with any portion of the system without causing any performance difference. However, things have changed again for the emergence of grid and cloud systems [6,1], in which usually multiple clusters are interconnected. The resource fragmentation problem appears once more, and receives increasing research attention.

The multi-cluster system can be further classified into two categories:

- Single-site allocation system [20,3]: Since an inter-cluster communication network is usually much slower than an intra-cluster high-speed network, a parallel job is restricted to run within a single cluster for better performance than running across different clusters. However, such an allocation policy could result in resource fragmentation.
- Multi-site co-allocation system [24,28,3]: A parallel job is allowed to be allocated across different clusters. This allocation policy does not suffer the resource fragmentation problem. However, the execution performances of co-allocated jobs are usually worse than executing them in single cluster due to the much slower inter-cluster network.

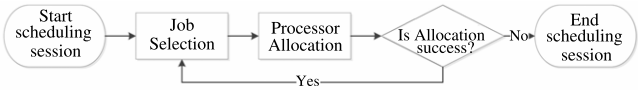


Fig. 1. The scheduling flow in a scheduling session.

This paper focuses on single site allocation, as multi-site co-allocation is rarely seen in production systems [21]. In single site allocation, Huang and Chang [13] studied many classical methods, such as First-Fit, Best-Fit, Worse-Fit, Median-Fit, and Random Fit. Simulation results showed that the Best-Fit (BF) heuristic, which allocates a job to the cluster resulting in the least left-over processors, outperforms all the other methods in the homogeneous multi-cluster environments. Similar results can also be found in [11]. In the heterogeneous multi-cluster environment, the Fastest-First (FF) method, which tries to minimize the job execution time through allocating each job onto the fastest available cluster, is shown to surpass BF when speed heterogeneity is high [12]. However, these two methods have the drawback that they consider one job at a time when making allocation decision. Such allocation scheme though flexible but hard to consider the effect of resource fragmentation, which largely depends on the size (required number of processors) of the subsequent jobs to be allocated. For this, Shih et al. [23] consider all waiting jobs together when making each allocation decision. Several intelligent allocation approaches are proposed which dynamically change the allocation policy between BF and FF. Among all the proposed algorithms, the adaptive intelligent 2 (AI2) [23] method will be used for performance comparison since it shows the best performance.

Recently, J. Ramírez-Alcaraz et al. [20] proposed several allocation strategies on a multi-cluster Grid. The goal is to balance the load among different clusters. However, the used scheduling model differs from ours in that for each new submitted job it will firstly be allocated to one of the cluster by the allocation strategy. Local scheduling is then applied on all jobs submitted to that cluster.

The approach proposed in this paper further improves Shih's AI2 method [23] in three aspects. First, TLA considers all possible allocation decision, rather than the one produced by BF or FF. Second, TLA can directly optimize the performance of entire system based on the specified performance metric. Third, TLA further utilizes the execution time information of the waiting jobs to consider more precise effects of each allocation.

The following briefly describes how AI2 works. In each scheduling session it conducts a simulation process on two allocation branches. Branch A allocates the first target job using BF while branch B allocates the first target job using FF. The rest of the target jobs (if any) in the same scheduling session are allocated using FF. The final allocation decision (for the first target job only) is made based on which branch consumes more computing power in the simulation. The consumed computing power is defined as the used number of processors multiplied by the speed of that cluster. For example, in the simulation, branch A totally allocates 3 target jobs in the same scheduling session and yields 12 units of computing power to be consumed. Branch B allocates totally 2 target jobs and yields 13 units of computing power to be consumed. The final decision would be allocating the first target job using FF (branch B).

2.4. Performance metric

There are two commonly used metrics to evaluate the performance of an on-line job scheduling algorithm: the Average Turnaround Time (ATT) [17,8] and the Bounded Slowdown [7]. Both metrics are considered in evaluating the performance of TLA. However, in the simulation results, no significant performance differences are found between these two metrics, and therefore only the case of average turnaround time is presented.

Scoring Function $f(j, c)$

I. Simulate allocating job j to cluster c.

II. Simulate the allocation and execution of all subsequent waiting jobs.

III. Return the calculated score, in terms of ATT, based on the simulation results.

Fig. 2. Scoring function of TLA.

The turnaround time (TT) of a job i is defined as

$$\mathrm{TT}_i = \mathrm{endTime}_i - \mathrm{submitTime}_i \tag{1}$$

and the ATT of a group of jobs is defined as

$$\mathrm{ATT} = \left(\sum_{\forall \text{ job } i} TT_i\right) \Big/ N \tag{2}$$

where N is the total number of jobs.

3. TLA method

For the job in the waiting queue to be allocated, say job j, TLA is asking the following question:

> If jobs in the current waiting queue are what the system will have ultimately, which allocation of job j will achieve the best overall system performance?

To answer that, TLA evaluates all possible allocations for job j, in terms of a desired performance metric, such as ATT introduced in Section 2.4. For a possible allocation, say cluster c, TLA utilizes a simulation procedure for all the subsequent waiting jobs in queue to see the outcome of allocating job j to cluster c. The simulation of the allocation and execution of the waiting jobs is based on the information provided by the waiting jobs, namely the number of requested processors and the runtime estimation. Fig. 2 outlines the procedure of allocation evaluation, denoting the *Scoring Function*.

The obstacle of TLA is Step II, which simulates the allocation of the subsequent waiting jobs. The most brute-force but precise way is to invoke TLA again inside the simulation. However, this recursive invocation will make the entire calculation time exponential, which is inefficient and unnecessary, because the waiting jobs could be changed when their real allocation and execution happen. What we want is an estimation to the effect of allocating job j to cluster c, based on some reasonable calculations. We will show how this can be done later.

It should be noted that when there is no job in the waiting queue, the allocation decision made by TLA will be the same as that produced by simple heuristics, such as BF and FF. Therefore, we will not address the algorithms for the case of empty waiting queue in the following discussion. In all the simulation experiments, we use FF as our default policy when facing this situation.

Before moving on, we define the used notations:

- J: a list of jobs in the waiting queue with required information, such as the number of required processors, the estimated runtime, etc.
- C: a list of clusters with its profile, such as the speed of processors, total number of processors, and the number of currently available processors, etc.

3.1. TLA algorithm

The detailed scoring function of TLA is shown in Fig. 3, in which only Step II is highlighted because other steps are the same as in Fig. 2. In Step II, the term "sim-allocate" represents "simulated allocation", which means all the actions are taking place just by simulations inside the scoring function. In Step II-(c), instead of

Scoring Function $f(j, c)$

I. Simulate allocating job j to cluster c.

II. While there exists any job in the waiting queue not yet sim-allocated

(a). Pick up the first job k in queue.

(b). Search the earliest time that some cluster(s) in C can run job k.

(c). If there is more than one candidate, select the cluster according to FF policy.

(d). Sim-allocate required processors in the selected cluster to job k.

Fig. 3. The detailed scoring function of TLA.

Table 1
Score calculated by TLA in allocation simulation (the lower the better).

Job ID	Number of required processors	Submit time[a]	Estimated runtime	Score for allocating job 1 to cluster A	Score for allocating job 1 to cluster B
1	3	−5	10	10	15
2	6	−4	2	10	5
3	2	−2	8	11	6
4	7	−2	2	9	6
Ave.				10	**8**

[a] The submit time is shown in negative value to represent the time prior to current time 0.

recursively invoking TLA, a simple heuristic FF is used to decide the allocation of subsequent waiting jobs. This trick greatly reduces the time complexity from exponential to polynomial. The use of FF is because it usually results in a better turnaround time for individual tasks. We will discuss the case of using other heuristics in allocation simulation in Section 5.3.

Fig. 4 provides an example to illustrate how TLA works. Fig. 4(a) shows the snapshot of current waiting jobs and the status of clusters before TLA begins. There are four waiting jobs represented by rectangles, in which the height is the required number of processors (RNP), the width is the estimated runtime (ERT), and the number in the rectangle is the job ID, as well as the execution order. There are two clusters. Each cluster has nine processors, and the ones in cluster A are two times faster than those in cluster B. Job $r1$ and $r2$ are two current running jobs.

To find an allocation for job 1, TLA has to calculate $f(1, A)$ and $f(1, B)$. The calculation of $f(1, A)$ first runs a simulation to allocate all subsequent waiting jobs, as shown in Fig. 4(b). After that, the score is calculated by averaging the turnaround time of all four waiting jobs, as shown in Table 1, whose value is 10. The same procedure is applied to compute $f(1, B)$, whose simulation result is shown in Fig. 4(c), and the computed score is 8. In this example, TLA will allocate job 1 to cluster B, since it has a smaller ATT.

3.2. Time complexity

This section analyzes the worst-case time complexity of TLA. Assume a HMC system has m clusters, and there are q waiting jobs and r running jobs before performing TLA.

In TLA, the complexity of the scoring function is dominated by step II (see Fig. 3), which checks if there are enough free processors to accommodate the subsequent waiting jobs at the end time of each running job. If a job can be allocated immediately, it requires $\log m$[1] steps to decide its allocation using FF policy and r steps to put its end time to the right place in the end of the event queue, which is a link-list data structure sorted by the ascending order of job end time so that one can find the nearest job end time in constant step. If a job cannot be allocated immediately, it traverses the end event queue and accumulates the processors released by each running job until sufficient processors has been reclaimed to allocate the job. The traverse of the entire waiting jobs takes at most $r+q$ steps. In the worst case it requires at most $q \log m+r+q$ operations to go through step II. Since the allocatable cluster of a job is at most m, the total operation required is $qm \log m+rm+qm$ and the time complexity of TLA is $O(qm \log m + rm)$.

In grid or cloud environments, the number of clusters may be large, which may influence the scalability of the algorithms. However, we should point out those algorithms can be embarrassingly parallelized and are strongly scalable with the number of clusters m.

Table 2 summarizes the time complexity of TLA and its competitors: Best-Fit, Fastest-First, and Adaptive Intelligent 2. Additionally, we collect the time required for TLA to compute the score in the simulation procedure on a general core2 dual PC with 2 GB memory, as displayed in Fig. 5. The number of clusters is five, the same as most of our experimental setting presented in Section 4. From the figure one can find that the overhead of TLA is in general negligible (less than 1 s) compared to the time required to execute a job (thousands of seconds on average), even with an extremely large number of waiting jobs.

Table 2
Time complexity of Best-Fit, Fastest-First, adaptive intelligent 2, and TLA.

Processor allocation algorithms	Time complexity
Best-Fit (BF) and Fastest-First (FF)	$O(m)$
Adaptive Intelligent 2 (AI2)	$O(qm)$
TLA	$O(qm \log m+rm)$

Used notations:
m: total number of clusters in HMC.
q: number of current waiting jobs.
r: number of current running jobs.

4. Experiments

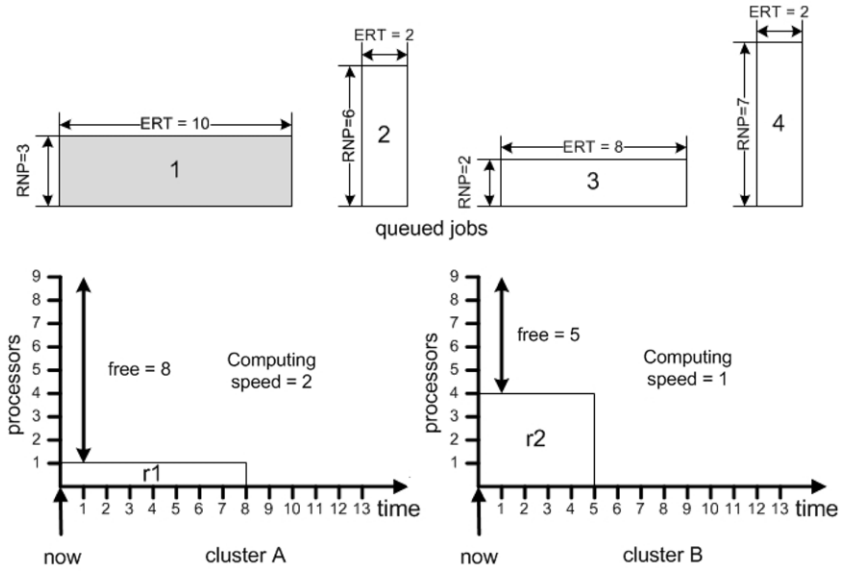
In this section we present the simulation results based on precise job runtime estimation. The experimental settings, including the used input workloads and the methods to model different system configurations, are presented in Section 4.1. Section 4.2 present the simulation results as well as the discussion.

4.1. Experimental settings

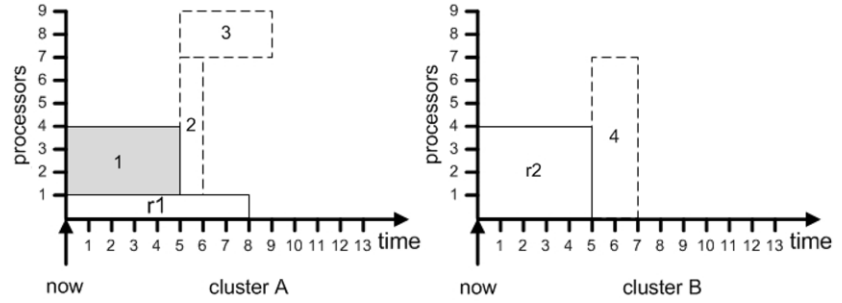
It has been reported that workload characteristic and system configuration can have significant impact when evaluating the performance of a scheduling algorithm [9,2,5]. In order to consider those effects, the simulation configuration includes various workload source, system configuration, degree of speed heterogeneity and degree of system loading.

For workload source, we use several public downloadable workload logs and models from parallel workload archive [19] to avoid bias caused by specific workload characteristic. The workload logs used include SDSC's SP2, DAS2 5-cluster grid, and ANL Intrepid. The SDSC's SP2 log contains 73,496 records collected on a 128-node IBM SP2 machine at San Diego Supercomputer Center (SDSC) from May 1998 to April 2000. The DAS2 5-cluster grid log contains in total 432,987 records collected on five Pentium/Linux clusters (one cluster with 144 CPUs and the rest with 64) from Jan 2003 to Dec 2003. The ANL Intrepid log contains several months' records (in total 68,936 jobs from Jan to Sep 2009) collected on a large Blue Gene/P system called Intrepid, which is a 557 Teraflops, 40-rack Blue Gene/P system deployed at Argonne Leadership Computing Facility (ALCF) at Argonne National Laboratory. Excluding

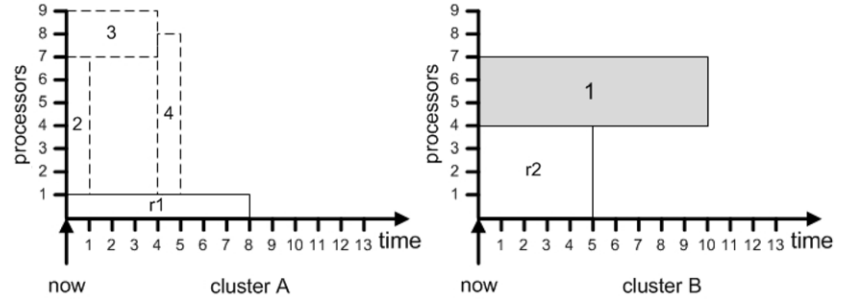
[1] R.P. Brent in his work "Efficient implementation of the first-fit strategy for dynamic storage allocation" has proposed an $O(\log m)$ mechanism to implementation the first-fit method. The proposed mechanism can also be applied to fastest-first since it is a special case of first-fit where the clusters are sorted from faster to slower.



(a) Before allocation.



(b) Simulated allocation results if job 1 is allocated to cluster A.



(c) Simulated allocation results if job 1 is allocated to cluster B.

Fig. 4. Illustration of how TLA works.

problematic records such as non-positive job runtime and processor requirement, the actual number of jobs used in simulation for each workload is shown in Table 3. The used synthetic workload models include Lublin99 [16], Feitelson96 [4], and Jann97 [14]. These three models are chosen because their job type (rigid) fits our assumption. For each model, 50,000 jobs were generated for simulation.

To have various system configurations, the HMC system used in the simulations is modeled according to the used workload and for each workload we prepare two sets of system configuration, which differ either in the number of clusters or in the number of processors in each cluster. For example, when using SDSC's SP2 log the number of clusters is set to five to match the number of job submission queues in the SDSC's SP2 system and the number of processors for each cluster is set either equal to the maximum number of required processors in all queues, which are 128 for each cluster, or equal to the maximum number of required processors of the jobs in the corresponding queue, which are 8, 128, 128, 128, and 50 for each cluster. The detailed system configuration is also shown in Table 3. One thing that needs to be mentioned is the workload C2 (refer Table 3 for the workload ID). To have a case with a large number of clusters, the total 163,840 processors are divided into 20 clusters with 8192 processors each. For jobs requiring more than 8192 processors, they are divided into several small jobs with 8192 processors and the last one job takes the leftover processors. For example, a job requiring 20,000 processors will be divided into two jobs with 8192 processors plus one job with 3616 processors. Note that job splitting may not be a good modeling approach as

Table 3
Summary of the characteristic of the used workloads and corresponding system configurations.

Type	Name	Workload ID	Jobs	Clusters	System configuration	Ori. SL
Log	SDSC's SP2	A1	54,041	5	$128 * 5 = 640$	0.17
		A2	54,041	5	$128 * 3 + 50 + 8 = 442$	0.24
Log	DAS2	B1	416,836	5	$144 * 5 = 720$	0.08
		B2	416,836	5	$144 + 64 * 4 = 400$	0.14
Log	ANL Intrepid	C1	68,936	5	$163,840 * 5 = 819,200$	0.11
		C2	88,205	20	$8192 * 20 = 163,840$	0.59
Model	Lublin 99	D1	50,000	5	$128 * 5 = 640$	0.34
		D2	50,000	10	$128 * 10 = 1280$	0.17
Model	Feitelson 96	E1	50,000	5	$128 * 5 = 640$	0.09
		E2	50,000	3	$128 * 3 = 384$	0.15
Model	Jann 97	F1	50,000	5	$322 * 5 = 1610$	0.13
		F2	50,000	8	$322 * 8 = 2576$	0.08

it changes the characteristic of original workload. It just serves as a special case to see if a different modeling method affects the performance of TLA.

Speed heterogeneity (SH) controls the variance of the computing speed among different clusters. To simulate the differences in computing speed we define a speed vector, $(cs_1, cs_2, \ldots, cs_m)$, to describe the relative computing speeds of all the clusters [23,12]. The value 1 represents the computing speed resulting in the job runtime in the original workload log. For other values, the runtime of jobs allocated to cluster i is divided by cs_i. With which, speed heterogeneity is defined as

$$\mathrm{SH} = \frac{\sum_{i=1}^{m}(cs_i - 1)^2}{m}, \tag{3}$$

in which m is the total number of clusters. Three levels of speed heterogeneity, 0, 0.1, and 0.2 are considered in the simulation configurations. In addition, for fairness, we require the generated cs_i satisfying

$$\sum_{i=1}^{m} cs_i \times NP_i = \text{System Service Rate}, \tag{4}$$

in which NP_i is the total number of processors in cluster i and the *SystemServiceRate* is a constant to represent the total computing capability of the system. In this paper it is set to the total number of processors in the clusters. For example, in the workload B1 the *SystemServiceRate* is set to 720 while in B2 it is set to 400. To fulfill both Eqs. (3) and (4) simultaneously, $cs_1, cs_2, \ldots, cs_{m-2}$ are generated first by normal distribution with mean equal to 1 and variance equal to SH, while cs_{m-1} and cs_m are derived from Eqs. (3) and (4). To avoid bias in a particular speed setting, ten speed vectors for each *SH* value were randomly generated. All presented results with non zero speed heterogeneity are the average value of ten experiments with those ten speed vectors.

System loading (SL) represents the heaviness of the input workload and is modeled by

$$\mathrm{SL} = \frac{\mathrm{AverageWorkloadInputRate}}{\mathrm{SystemServiceRate}}, \tag{5}$$

in which the *AverageWorkloadInputRate* is defined as

$$\mathrm{AverageWorkloadInputRate} = \frac{\sum_{\forall \mathrm{job}\, j} RT_j \times RNP_j}{\mathrm{WorkloadDuration}}. \tag{6}$$

The RT_j and RNP_j are the runtime and the required number of processors of job j, respectively. The workload duration is the time period between the submission of the first job to the submission of the last job in the workload. Three levels of system loading, namely

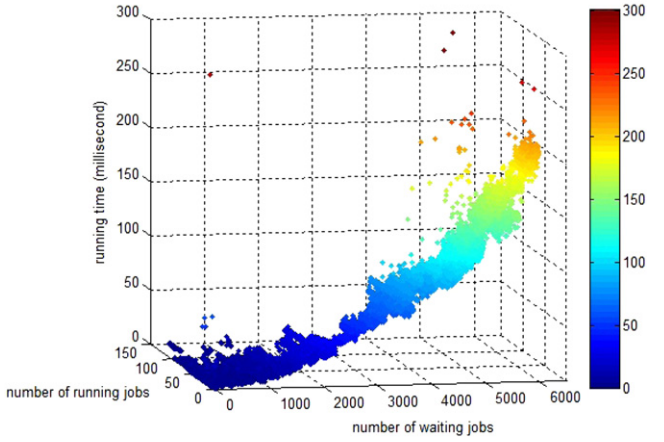


Fig. 5. The time required for TLA to calculate the score in the scoring function under a different number of running jobs (r) and waiting jobs (q).

low (SL = 0.5), medium (SL = 0.75), and high (SL = 1), are considered in the simulations. The last column of Table 3 shows the original SL value calculated by Eq. (5) with respect to each set of workload and system configuration. To generate the desired SL for each workload set, the runtime of each job in the workload will be multiplied by a constant value according to its original SL, ex. to generate medium system loading for workload B2 the runtime of each job is multiplied by 5.35.

All experiments presented in the following subsections are with various simulation configurations, including the combinations of the following parameters:

- Speed heterogeneity: 0, 0.1, 0.2.
- System loading: low, medium, high.
- Workload source and system configuration: twelve pairs as shown in Table 3.

4.2. Experimental results

Experiments in this subsection compare the performance of TLA with BF, FF, and AI2. Table 4 summarizes the performance improvement of TLA with respect to the best result of BF, FF, and AI2, under all 108 cases of simulation configurations. From the table one can find that TLA outperforms BF, FF, and AI2 for almost all the simulation configurations (106/108), especially for the case with medium system loading, which shows up to an 87% performance improvement compared to the best of BF, FF, and AI2 in terms of average turnaround time.

The two cases where TLA shows negative improvement reveal the weakness of TLA, that is, when facing zero speed heterogeneity

Table 4
Performance improvement of TLA with respect to the best result of BF, FF and AI2 under various simulation configurations.

Workload ID	SL = low			SL = medium			SL = high		
	SH = 0	SH = 0.1	SH = 0.2	SH = 0	SH = 0.1	SH = 0.2	SH = 0	SH = 0.1	SH = 0.2
A1	1%	1%	2%	45%	49%	45%	41%	35%	34%
A2	−2%	4%	4%	75%	79%	74%	33%	32%	27%
B1	48%	37%	45%	54%	71%	70%	32%	47%	43%
B2	60%	66%	59%	48%	54%	43%	24%	36%	28%
C1	5%	11%	6%	47%	50%	48%	28%	26%	24%
C2	1%	2%	2%	13%	13%	12%	13%	13%	12%
D1	19%	28%	28%	74%	71%	68%	34%	33%	31%
D2	6%	14%	20%	81%	82%	78%	42%	41%	35%
E1	4%	14%	21%	**87**%	85%	79%	30%	31%	27%
E2	22%	18%	19%	55%	52%	48%	24%	22%	19%
F1	1%	5%	5%	81%	79%	77%	48%	40%	38%
F2	−3%	2%	2%	80%	80%	77%	43%	45%	42%

Table 5
Average utilization observed for workload A2 with zero speed heterogeneity.

System loading	AI2	BF	FF	TLA
Low (SL = 0.5)	0.50	0.50	0.50	0.50
Medium (SL = 0.75)	0.74	0.74	0.73	**0.75**
High (SL = 1)	0.79	0.79	0.78	**0.83**

in conjunction with low system loading. Fig. 6 provides the detailed performance result of workload A2, one of these two cases. Zero speed heterogeneity makes the resource fragmentation the only performance factor. Low system loading further makes the effect of resource fragmentation much less significant, which results in little performance difference as shown in Fig. 6(a).

Additionally, low system loading results in fewer jobs waiting in the queue, providing less information for TLA to make a better allocation decision. This can be shown by the *average queue length* (AQL), the average number of jobs waiting in the queue. In these two cases the AQL is observed less than ten, which is quite small compared to the hundreds observed in medium system loading. This also explains the extraordinary result shown in workload B1 and B2 with low system loading. The AQL observed in these two cases are around 500 in B1 and 9000 in B2, respectively, which is caused by the original workload characteristic and the way we model the HMC system.

Fig. 6(d) illustrates the performance improvement made by TLA for different system loading and speed heterogeneity with workload A2. This sub-figure clearly shows that the higher the system loading the better the performance improvement of TLA. A similar phenomenon can also be found in other workloads. The reason why medium system loading shows a better improvement ratio is that, with high system loading, the turnaround time of each job is largely increased, which makes the improvement ratio less notable.

Another interesting observation is that a different speed vector does have a significant effect on the overall performance. Fig. 7 shows the box chart of the two workloads with the highest coefficient of variation (CV) of the ten speed vectors. The highest CV observed is 100.8%, as shown in Fig. 7(a) with BF heuristics. Another observation is that in most of the simulation configurations the TLA method generates a smaller CV compared to all the other methods, such as the example shown in Fig. 7(b).

The reason that TLA is superior to BF, FF, or AI2 is that the allocation made by TLA optimizes the system performance based on workload and resource information provided at runtime, not just a fixed policy. Additionally, with job runtime information, TLA can have more precise estimation on the effect of an allocation on the performance of the subsequent waiting jobs. Table 5 further compares the average resource utilization of different allocation methods. The result shows that TLA can achieve higher utilization than the other three methods, especially with high system loading. Such a result also appears in other workload sources and simulation configurations.

5. Discussion on the performance issue of TLA

In this section we further discuss some performance issues of TLA. The first thing to be verified is how TLA performs with inaccurate runtime estimation, since in the real world it is hard to precisely estimate the runtime of each parallel job. This issue will be discussed in Section 5.1. By default, TLA simulates the allocation of all waiting jobs in the score calculation. Actually, TLA can use an arbitrary number of jobs for allocation simulation. This number is called the *simulation depth*. Section 5.2 will discuss the performance effect with different simulation depth. Next, as described in Section 3.1, we use the FF policy to decide the allocation of subsequent waiting jobs in the simulation procedure. It is also possible to use other policies, such as best-fit or random. This effect will be discussed in Section 5.3. Last, by default, each simulation result of TLA is only used to make one allocation decision. TLA can also be configured to make multiple allocation decisions within a single allocation simulation. The number of allocation decisions is called the *allocation depth*. The effect of the allocation depth will be addressed in Section 5.4. Note that there is no relation between simulation depth and allocation depth. Simulation depth controls how many jobs in the waiting queue to be simulated to get a score. Once the score is calculated, it can be used to make multiple allocation decisions of the waiting jobs, in which the number is determined by allocation depth.

5.1. Effect of inaccurate runtime estimation

Before presenting the result we first show how the estimation inaccuracy is modeled. As far as we know there is no inaccuracy model proposed for the heterogeneous multi-cluster system as the user usually does not have enough information on the speed difference among all clusters to provide accurate job runtime estimation [20]. Therefore we turn to use the well-known *f-model* proposed in the single cluster system [17,26].

In the *f*-model, a variable named *inaccuracy factor* is used to model different levels of estimation error. There are two types of estimation error, over-estimation and under-estimation. We assume that each appears with probability 1/2. For the case of over-estimation the estimated runtime (ERT) of each job i is generated by Eq. (7), while under-estimation is generated by Eq. (8).

$$\mathrm{ERT}_i = \mathrm{ORT}_i * \mathrm{Random}(1, \text{inaccuracy factor}) \tag{7}$$

$$\mathrm{ERT}_i = \mathrm{ORT}_i * \mathrm{Random}\left(\frac{1}{\text{inaccuracy factor}}, 1\right). \tag{8}$$

The ORT_i is the original runtime of job i provided by the workload sources. The *Random* function generates a random value between the left number and the right number uniformly. Fourteen sets of inaccuracy factor are considered in the simulation, which

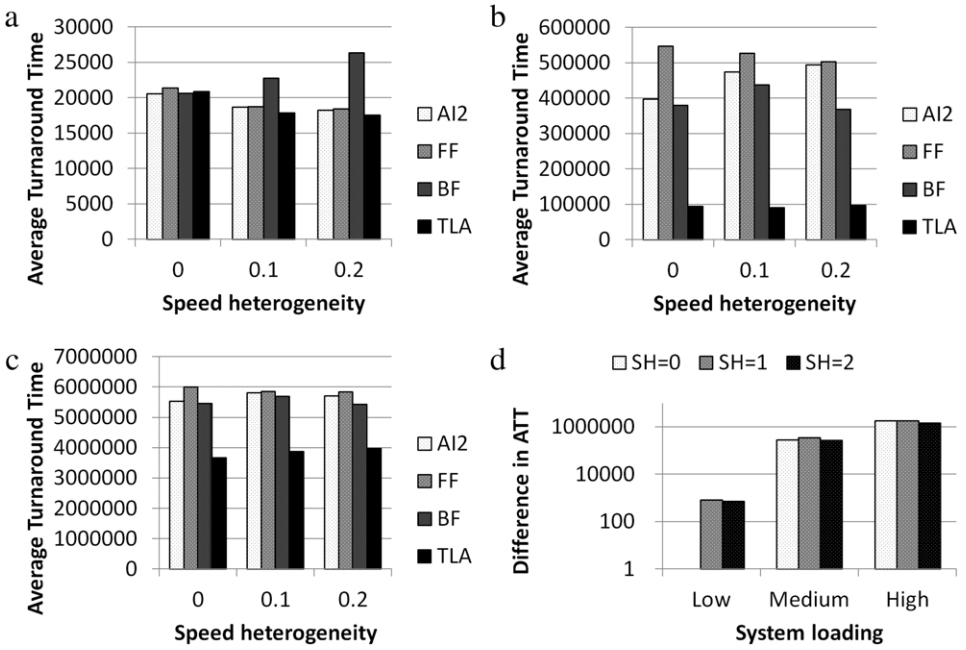


Fig. 6. Performance result of workload A2 with (a) low system loading, (b) medium system loading, (c) high system loading, and (d) performance improvement with respect to different system loading and speed heterogeneity (SH).

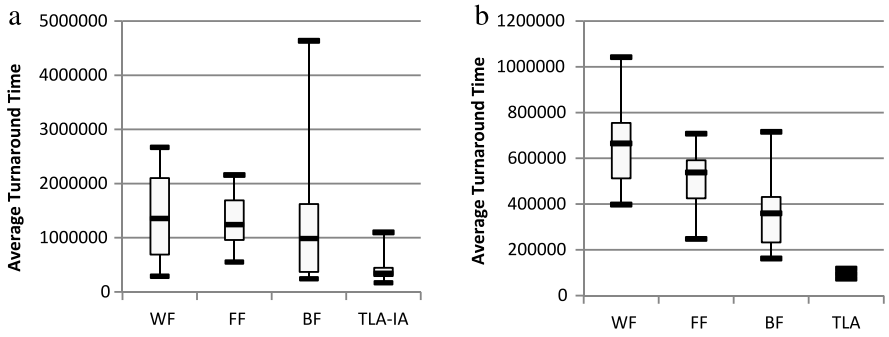


Fig. 7. Effect of speed vectors. (a) Low system loading with workload B2 and (b) Medium system loading with workload A2.

are 1 (100% accurate), 1.25, 1.5, 1.75, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024.

The most representative result among all the workloads is shown in Fig. 8. From the figure one can find that the results of BF, FF, and AI2 remain constant across different inaccuracy values. This is because they do not rely on job runtime to make allocation decisions. For TLA, the performance deteriorates as the inaccuracy factor increases. It is reasonable because the allocation decision made by TLA is based on the simulated allocation results, which provide a hint to the outcome of future allocations. Estimation error could make the simulated allocation results deviate from the actual allocations taking place in the future and hence deteriorate the overall performance.

We use the same example in Fig. 4 to illustrate this situation. Assuming that job $r2$ actually ends at $t = 1$ instead of at $t = 5$ due to over-estimation. In this case the practical allocation results are shown in Fig. 9, which deviate from the simulated allocation results shown in Fig. 4. The practical ATT if job 1 is allocated to cluster A and B are 7.75 and 8.25 respectively, which indicates that allocating job 1 to cluster A is a better choice. However, due to estimation error, the score calculated with simulated allocation results indicate the TLA to allocate job 1 to cluster B.

As a short summary, the performance of TLA could be seriously affected by runtime estimation inaccuracy. However, the inaccuracy model used in this paper is just one of the many models

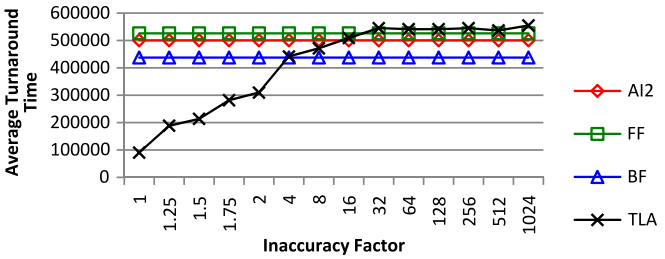
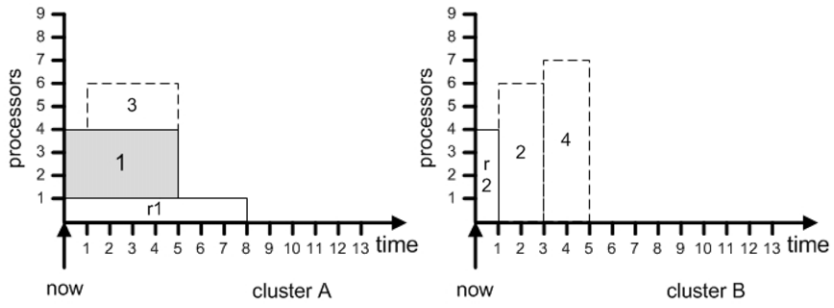


Fig. 8. Effect of inaccuracy in job runtime estimation.

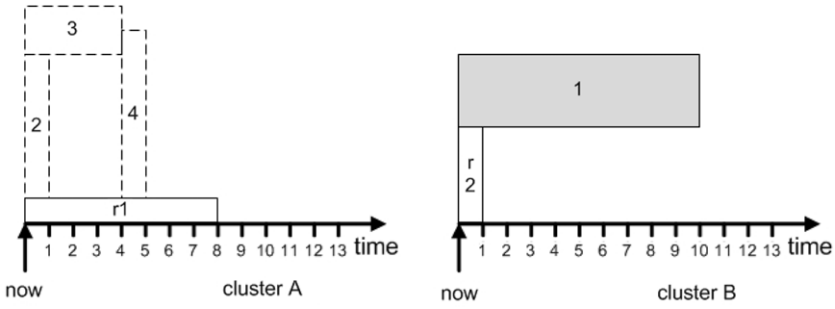
presented in [26] and none of which is designed for a heterogeneous multi-cluster system. It would be an interesting research topic to investigate the relations of other runtime estimation models and the performance of TLA.

5.2. Effect of simulation depth

Fig. 10 depicts the effect of different simulation depth using workload A1, which is the most representative case that results of workload A2, C2, D1, D2, E1, and E2 are all similar to A1. Since the effect of simulation depth depends on the number of jobs waiting in the queue, only the result with high system loading is presented.



(a) Practical allocation results if job $r2$ ends at $t = 1$ and job 1 is allocated to cluster A.



(b) Practical allocation results if job $r2$ ends at $t = 1$ and job 1 is allocated to cluster B.

Fig. 9. Illustration of the effect of runtime estimation inaccuracy on the performance of TLA.

The result shows that the higher the simulation depth the better the performance of TLA.

To uncover the reason why simulation depth helps, we profile the detailed behavior of TLA by collecting the number of times TLA changes the allocation decision, i.e. the cluster with the best score changes to another cluster as simulation depth increases. Under 54 041 jobs there are 23 952 times where TLA is activated, and among the activation times, only 1101 times has the allocation decision changed. That is to say in all the other 22 851 times, simulating one job in TLA results in the same allocation decision as simulating all the waiting jobs. An implication to this result is that only a few jobs' allocation decisions affect the overall performance. Furthermore, we plot the probability mass function and cumulative distribution function when allocation decision changes, as shown in Fig. 11. The result shows that 80% of allocation change happens below simulation depth 160 (see Fig. 11(b)). By calculating the Pearson product-moment correlation coefficient (PCC) on number of allocation changes and the performance of TLA, we found a very high negative correlation between them (PCC $= -0.97$). This finding indicates that it is not always necessary to use the entire waiting job in allocation simulations. In this case, setting the simulation depth to 160 can yield an 80% performance improvement and reduce the execution time of TLA to one-sixth. The finding also shows a hint that one might use the characteristic of allocation change to find the optimal simulation depth that balances the performance and time complexity of TLA.

5.3. Effect of the used allocation policy in allocation simulation

In addition to FF, the other three allocation policies considered in the allocation simulation are Best-Fit (BF), Worst-Fit (WF), and Random-Fit (RF). Worst-Fit chooses the cluster with the most left-over processors. Random-Fit randomly chooses the cluster among available ones. The simulation result with different allocation simulation policy is shown in Fig. 12 using workload D1. The result using other workloads leads to similar observations, and is therefore omitted.

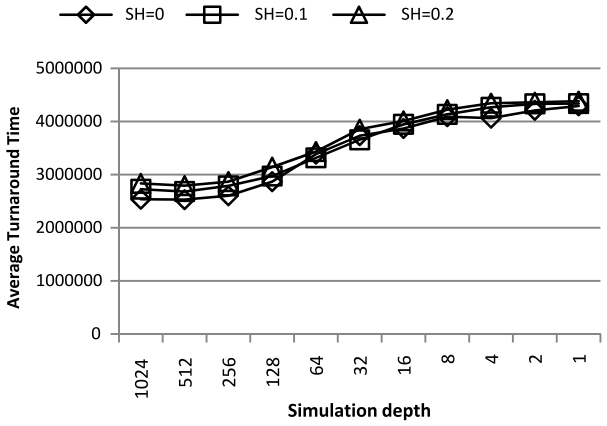
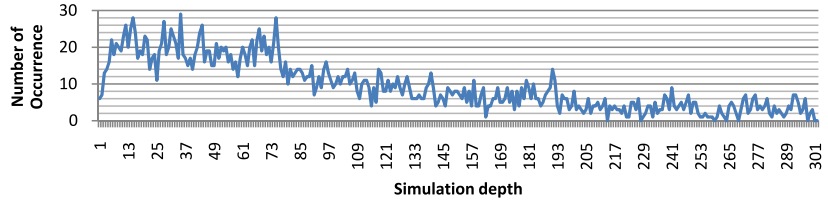


Fig. 10. Effect of different simulation depth.

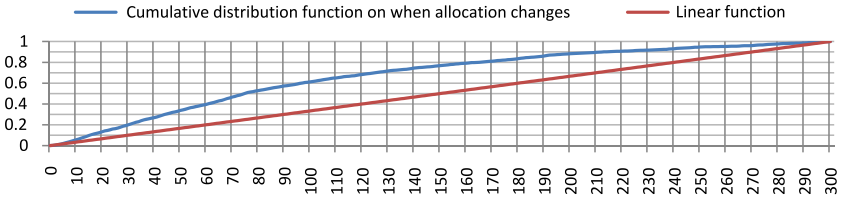
The simulation depth is contained in the simulation in order to show more detailed effects. From the figures one can find that RF shows the worst performance while the other three policies show similar performance. Although RF is the worst choose, however, it still achieves a notable performance improvement with the increase of the simulation depth. This result deviates our expectation that BF and WF will get much worse performance. One thing we can be certain of is that the effect of simulation depth is more significant than the choice of the allocation simulation policy. The result also indicates that simple allocation policy, even random, serves well in estimating the effect of an allocation decision.

5.4. Effect of allocation depth

The result with different allocation depth is shown in Fig. 13 using workload A2. The result with low system loading almost remains constant, and is therefore omitted. Speed heterogeneity is also hidden since it does not change anything that can be observed.

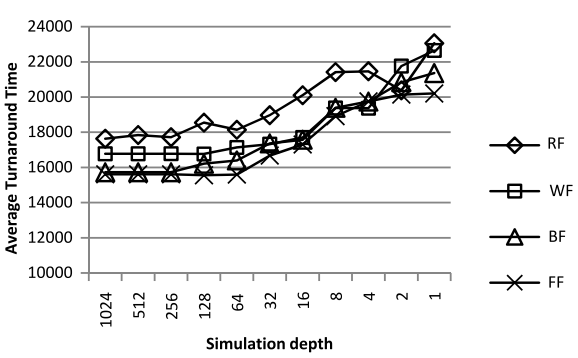


(a) Histogram.

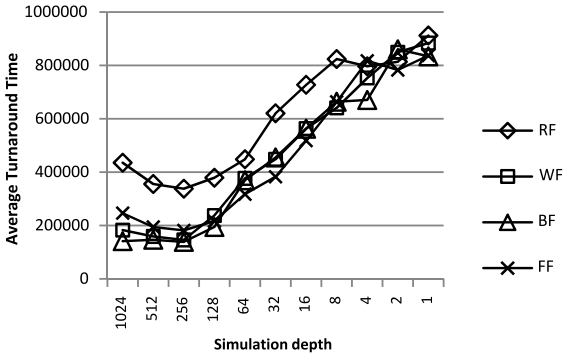


(b) Cumulative distribution function (CDF).

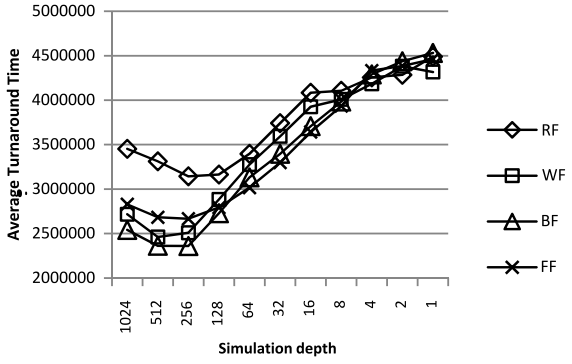
Fig. 11. Distribution when allocation changes with simulation depth.



(a) SL = low.

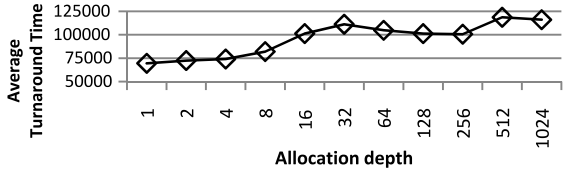


(b) SL = medium.

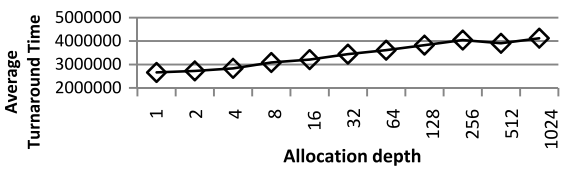


(c) SL = high.

Fig. 12. Result with different allocation simulation policy and simulation depth using workload D1.



(a) SL = medium.



(b) SL = high.

Fig. 13. Result with different allocation depth using workload A2.

The result shows a negative correlation, such that the larger the allocation depth the worse the performance. Despite the drawback, we should note that the increase in the allocation depth can help reduce the overall time complexity. In Fig. 13 one can find that setting the allocation depth to 4 could be a sweet spot where the performance degradation is acceptable while the execution time

of TLA can be reduced to one-fourth. However, this sweet spot changes with workload and other factors. How to find the sweet spot is a complicated issue which is beyond the scope of this paper.

6. Conclusion and future work

This paper investigates the issues of single-site processor allocation in heterogeneous multi-cluster (HMC) systems. Traditional processor allocation policies, such as Best-Fit (BF) and Fastest-First (FF), consider only speed heterogeneity or resource fragmentation. Their performance is not consistent for different input workload and system configurations. In this paper, we propose the temporal look-ahead (TLA) processor allocation method, which tries to find an allocation that can benefit the average turnaround time of all the jobs. Three characteristics of TLA's scoring function make TLA distinct from other processor allocation methods. First, it directly takes into account the specific performance metric to make an allocation decision. Since all performance factors regarding the given metric will naturally be considered through the simulation process, this allows TLA to consider other performance factors, given a suitable performance metric. Second, the scoring function is based on the simulation of current workload, which makes TLA adjustable to various situations. Third, it further utilizes the execution time information of the waiting jobs to consider more precise effects of each allocation.

Experiments on various workload and system configurations have been conducted. Simulation results show a different level of performance improvement with respect to the workload used and the system loading. In general, medium system loading results in the best performance improvement compared with low or high system loading. With accurate runtime estimation, TLA shows up to an 87% performance improvement over traditional methods. Simulation results also indicate that the performance of TLA can be seriously affected by the inaccuracy of runtime estimation. It would be an interesting research topic to investigate the relations of different runtime estimation methods and the performance of TLA.

As for future works, first, the root cause of the performance of TLA is not very clear. Some theoretical bases must be established to uncover the root cause, which is our major topic. Second, TLA can work with different job scheduling methods. This paper only considers the First-Come–First-Served method. The performance of TLA, while working with other job scheduling methods such as Shortest-Job-First and Narrowest-Job-First [12] requires further studies. Third, TLA could be generalized to optimize other performance metrics. In this paper we only consider the average turnaround time, but other performance metrics, such as fairness, can also be integrated in the design of the scoring function. The effectiveness of TLA under other performance metrics requires further studies.

Last but not least, the TLA method and the concept of allocation simulation provide a brand-new viewpoint to the processor allocation algorithms. We anticipate a further improvement can be made by utilizing those concepts in the design of new processor allocation algorithms.

References

[1] M. Armbrust, et al., Above the clouds: a Berkeley view of cloud computing, Tech. Rep. UCB/EECS-2009-28, EECS Department, University of California, Berkeley, 2009.

[2] A.I.D. Bucur, D.H.J. Epema, An evaluation of processor co-allocation for different system configurations and job structures, in: Proceedings of the 14th Symposium on Computer Architecture and High Performance Computing, 2002, pp. 195–203.

[3] E. Carsten, et al., On advantages of grid computing for parallel job scheduling, in: Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGrid'02, 2002, pp. 39–39.

[4] G.F. Dror, Packing schemes for gang scheduling, in: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing, 1996, pp. 89–110.

[5] C. Ernemann, et al., On effects of machine configurations on parallel job scheduling in computational grids, in: Proceedings of International Conference on Architecture of Computing Systems, ARCS, 2002, pp. 169–179.

[6] C. Ernemann, et al., Benefits of global grid computing for job scheduling, in: Proceedings of the IEEE/ACM International Workshop on Grid Computing, 2004, pp. 374–379.

[7] D.G. Feitelson, et al., Theory and practice in parallel job scheduling, in: Proceedings of the International Conference on Job Scheduling Strategies for Parallel Processing, 1997, pp. 1–34.

[8] D.G. Feitelson, Experimental analysis of the root causes of performance evaluation results: a backfilling case study, IEEE Transactions on Parallel and Distributed Systems 16 (2005) 175–182.

[9] D.G. Feitelson, et al., Parallel job scheduling—a status report, in: Proceedings of the International Workshop on Job Scheduling Strategies for Parallel Processing, 2005, pp. 1–16.

[10] V. Hamscher, et al., Evaluation of job-scheduling strategies for grid computing, in: Proceedings of the First IEEE/ACM International Workshop on Grid Computing, 2000, pp. 191–202.

[11] V. Hamscher, et al., Evaluation of job-scheduling strategies for grid computing, in: Proceedings of the 7th International Conference on High Performance Computing, HiPC-2000, 2000, pp. 191–202.

[12] K.-C. Huang, et al., Towards feasible and effective load sharing in a heterogeneous computational grid, in: Proceedings of the 2nd International Conference on Advances in Grid and Pervasive Computing, GPC'07, 2007, pp. 229–240.

[13] K.-C. Huang, H.-Y. Chang, 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, PDPTA'06, Las Vegas, USA, 2006, pp. 703–709.

[14] J. Jann, et al., Modeling of workload in MPPs, in: Proceedings of the Job Scheduling Strategies for Parallel Processing, 1997, pp. 95–116.

[15] K. Li, Job scheduling and processor allocation for grid computing on metacomputers, Journal of Parallel and Distributed Computing 65 (2005) 1406–1418.

[16] U. Lublin, D.G. Feitelson, The workload on parallel supercomputers: modeling the characteristics of rigid jobs, Journal of Parallel and Distributed Computing 63 (2003) 1105–1122.

[17] A.W. Mu'alem, D.G. Feitelson, Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling, IEEE Transactions on Parallel and Distributed Systems 12 (2001) 529–543.

[18] L.M. Ni, et al., Contention-free 2D-mesh cluster allocation in hypercubes, IEEE Transactions on Computers 44 (1995) 1051–1055.

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

[20] J. Ramírez-Alcaraz, et al., Job allocation strategies with user run time estimates for online scheduling in hierarchical grids, Journal of Grid Computing 9 (2011) 95–116.

[21] U. Schwiegelshohn, et al., Online scheduling in grids, in: Proceedings of the IEEE International Symposium on In Parallel and Distributed Processing, IPDPS'08, 2008, pp. 1–10.

[22] D.D. Sharma, D.K. Pradhan, Processor allocation in hypercube multicomputers: fast and efficient strategies for cubic and noncubic allocation, IEEE Transactions on Parallel and Distributed Systems 6 (1995) 1108–1122.

[23] P.-C. Shih, et al., Improving grid performance through processor allocation considering both speed heterogeneity and resource fragmentation, The Journal of Supercomputing (2010) 1–27.

[24] O. Sonmez, et al., On the benefit of processor coallocation in multicluster grid systems, IEEE Transactions on Parallel and Distributed Systems 21 (2010) 778–789.

[25] X. Tang, et al., List scheduling with duplication for heterogeneous computing systems, Journal of Parallel and Distributed Computing 70 (2010) 323–329.

[26] D. Tsafrir, et al., Modeling user runtime estimates, in: Proceedings of the 11th International Conference on Job Scheduling Strategies for Parallel Processing, Cambridge, MA, 2005, pp. 1–35.

[27] D. Tsafrir, et al., Backfilling using system-generated predictions rather than user runtime estimates, IEEE Transactions on Parallel and Distributed Systems 18 (2007) 789–803.

[28] Z. Weizhe, et al., Multisite co-allocation algorithms for computational grid, in: Proceedings of the 20th IEEE International Symposium on Parallel and Distributed Processing, IPDPS'06, 2006, pp. 335–335..

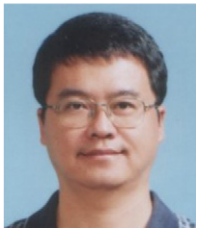
[29] Q. Xiao, An availability-aware task scheduling strategy for heterogeneous systems, IEEE Transactions on Computers 57 (2008) 188–199.

Po-Chi Shih was born on October 9, 1980 in Taipei, Taiwan, R.O.C. He received the B.S. and M.S. degrees in Computer Science and Information Engineering from Tunghai University in 2003 and 2005, and a Ph.D. degree in Computer Science from National Tsing Hua University in 2012. He is now doing a post-Doc in NTHU. His research interests include parallel processing, cloud computing and software defined network.

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 the Computer and Information Science Department at the National Taichung University, Taiwan. He is a member of ACM and the IEEE Computer Society. His research areas include parallel processing, cluster, grid and cloud computing and workflow computing.

Che-Rung Lee received a B.S. and M.S. degrees in Computer Science from National Tsing Hua University Taiwan in 1996 and 2000 respectively, and the Ph.D. degree in Computer Science from University of Maryland, College Park in 2007. He joined the Department of Computer Science at National Tsing Hua University as an Assistant Professor in 2008. His research interests include numerical algorithms, scientific computing, high-performance computation, and cloud computing. He is a member of IEEE and SIAM.

I-Hsin Chung is currently a Research Staff Member at the Thomas J. Watson Research Center, Yorktown Heights, NY, USA.



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