An integrated approach towards aggressive state-tracking migration for maximizing performance benefit in distributed computing

Yong-Hyuk Moon · Chan-Hyun Youn

Received: 15 September 2011 / Accepted: 27 December 2011 / Published online: 11 January 2012

Abstract This paper presents a new state-tracking migration scheme that is integrated with aggressive reservation strategies such as immediate restart, greedy backfilling and selective preemption. The main contribution of this paper is an analysis of the effects of three techniques that can be used beyond the conventional migration schemes. Our simulation results suggest that state-tracking migration with selective preemption entirely outperforms the others. We also observe that the overall performance of immediate restart strategy combining to migration can be stably maintained under various job lifetime distributions. Moreover, it is found that performance would be improved by fitting jobs ruled by the immediate restart strategy rather than queued jobs into the void-intervals under the state-tracking migration scheme.

Keywords State-tracking job migration · Job allocation · Resource reservation · Genetic algorithm · Multi-objective optimization · Distributed computing

Y.-H. Moon (✉)
Software Research Laboratory, Electronics and Telecommunications Research Institute (ETRI), Daejeon, Korea
e-mail: yhmoon@etri.re.kr

Y.-H. Moon
Department of Information and Communications Engineering, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea

C.-H. Youn
Department of Electrical Engineering, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea
e-mail: chyoun@kaist.ac.kr

1 Introduction

Currently a high level of interest is being generated through the development of a wide and varied range of distributed computing systems (DCS), such as Cloud, Grid, and Peer-to-Peer computing. In the general paradigm of job allocation under the DCS model, heterogeneous-capable resources are assigned to individual jobs with various requirements. Thus, load balancing plays a special role in the operation of extremely large resource-oriented computing applications [1]. One challenging issue is that resource failures are inevitable due to dynamic nature of DCS. To overcome this problem, most of distributed computing products or platforms [2] have adopted various types of migration based allocation as a leading risk-resilient policy.

So far, in order to reduce delays potentially imposed by inherited reallocation property of migration, void-filling algorithms, preemption schemes and check-pointing skills based migration mechanisms have been proposed as supplementary techniques. Unlike non-filling scheduling algorithms [3], the void-filling allocation schemes [4] usually provide better utilization, however they have a much longer scheduling time. The preemptive policies aim at reducing the high average slowdowns for the short jobs without significant degradation to long jobs. Hence, job preemption has been often assumed to prevent starvation and to improve the average turnaround time in many literatures [5–7]. However, most of these studies did not consider various lifetime distributions of jobs and limited the number of possible preemptions. Furthermore, it has been proved as NP-completeness to solve the deadline scheduling problem in non-preemptive disciplines even though all jobs have the same release time and deadline in offline mode (i.e., batch scheduling). The application-level check pointing techniques discussed in [8] were also used for maintaining jobs' states, such as a remaining execution length and a failure occurrence under the migratable and malleable job model. However, additional migration costs imposed by frequent job state traces and data movements have not been significantly investigated with various migration strategies.

From the results of the vast research addressing numerous variants of the migration policy in DCS, we observe a few key problems as follows. First, each resource reservation request for the job migration tends to be individually processed in a greedy manner. Besides, the effects of preemptive migration still remain uncertain, resulting in that the delayed job execution frequently occurs due to the difference of jobs' importance. More significantly, few works have focused on how to leverage the low complexity of offline algorithms and the high flexibility of online algorithms.

The primary contributions of this work are as follows:

- Proposal of novel space sharing strategies called 'aggressive reservation' for maximizing the efficiency of resource usage under the model of migratable and malleable jobs,
- Integrating a messy genetic algorithm based batch mode scheduling with each of aggressive reservation strategies for guaranteeing robustness and flexibility of schedule as well as for reduction of potential delays occurred by migrations,
- Evaluation of the impact of each strategy on overall performance metrics with increasing offered workloads, inter-job fairness under different job categories and cost effectiveness in average utilization.

The rest of the paper is organized as follows. Section 2 describes mainly observed problems in the traditional job migration and discusses how to predict the runtime under the state-tracking migration scheme. In Sect. 3, we propose three aggressive reservation strategies based on different technical disciplines. We present a messy genetic algorithm as a batch mode scheduling framework and propose how to merge it with our strategies in Sect. 4. Simulations are conducted and results are discussed with various performance aspects in Sect. 5. Finally, we conclude the paper in Sect. 6.

2 Problem description

In this section, we first describe main assumptions and problems fundamentally used throughout this paper. Then we propose how to estimate the expected runtime under the state-tracking migration scheme.

2.1 Resource failure prediction

Typically, resource availability can be unexpectedly fluctuated over time, so that a failure of resource is one crucial part in the distributed computing environments. Our prediction algorithm for failure probability is modeled by the intuition that a failure may occurs if a guaranteed duration in resource availability does not averagely satisfy a specific level of demanded completion time of a job. Moreover, we consider that a resource failure follows the exponential failure law having $e^{-\lambda t}$ as the hazard function [9]. Therefore, a failure probability of job i being assigned to resource j, $P_{i,j}$, is decided by the difference between service demand value (SDV_i) and resource availability level (RAL_j) as follows:

$$P_{i,j} = \begin{cases} 0 & \text{if } SDV_i \leq RAL_j \\ 1 - e^{-\lambda t(SDV_i - RAL_j)} & \text{if } SDV_i > RAL_j \end{cases}, \tag{1}$$

where t represents an instant in time and λ denotes a failure coefficient. We assume that SDV_i is given when a job i is submitted. The variation of RAL_j over some future time interval can be estimated by aggregated time series prediction. Hence, a failure probability between a job and a resource can be achieved at any time. Additionally, we suppose that a scheduler becomes aware of a failure of any jobs within a negligible amount of time. Therefore, each resource is characterized by the resource availability and a sequence of failure probabilities over time in the resource failure prediction model.

2.2 Delay model in job migration

The conventional job migration schemes include unavoidable delays owing to their inherited disadvantages of requiring additional migration overheads and of allowing the wasting time slots between jobs being assigned at resources. Figure 1 shows three major cases of migration delay such as job restart delay, inter-job delay, and delayed job execution where job i is terminated at resource x, R_x, due to an unexpected failure, so that it is migrated to resource y, R_y. Each of them seriously leads to an increase in makespan or a decrease in average utilization. In particular, one critical shortcoming is that the inter-job delays and delayed job executions can induce overestimation of utilization.

Problem 1 (Delays of migrant jobs) *The occurrence of these inevitable delays in the migration based job allocation results in further fragmentation of available power of resources. Moreover, it could potentially waste idle time slots in a given scheduling duration, due to unpredictable failures of any resources during executing jobs.*

Consequently, reducing such delays at each resource will be greatly beneficial to the performance improvement in aspect of overall scheduling quality.

2.3 State-tracking migration scheme

To reduce the job restart delay, we first adopt a state-tracking migration scheme (STMS) [10] which reactivates an only

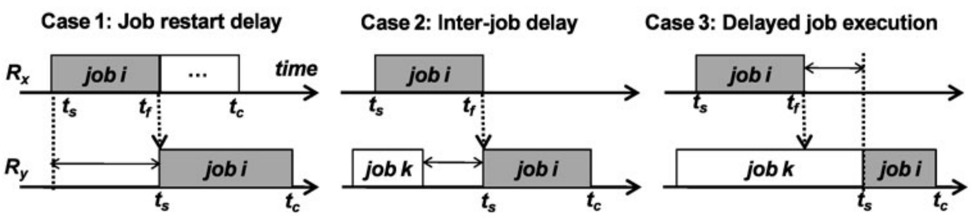


Fig. 1 Three delay cases of migration based re-dispatching: t_s, t_f, and t_c denote a start time, a failure time, and a completion time, respectively

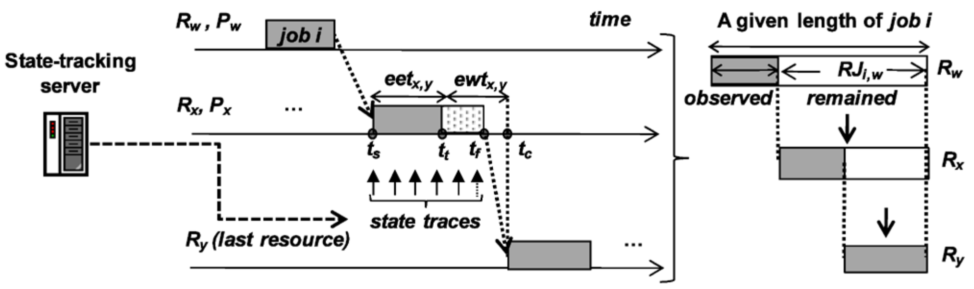


Fig. 2 High level description of PET estimation in STMS: t_t represents a termination time

Table 1 Notations

Symbol	Description
R_x	a resource x
RA_x	availability of R_x
P_x	a failure probability of R_x
$RJ_{i,x}$	a remaining size of job i failed at R_x
JD_i	a data size of job i
$ET_{i,x}$	the execution time of job i assigned to R_x
$SO_{i,x}$	state-tracking overheads of job i at R_x
$DM_{i,x,y}$	data movement delay of job i from R_x to R_y
$AB_{x,y}$	average bandwidth between R_x to R_y

residual job at a point of failure unlike the traditional stateless migration scheme [11]. In this approach, we assume that a job is re-dispatched to a more-capable backup resource if a failure occurs during job execution. Furthermore, the current states of job execution, such as an elapsed time and a failure time, are periodically traced by a state-tracking server using the application-level check pointing technique [12]. So, the dispatcher can instantly recognize a failure occurrence.

For clarity, main notations used in STMS's runtime estimation are summarized in Table 1.

2.3.1 Runtime estimation algorithm

To calculate makespan, average utilization, and the other performance metrics of each schedule generated by the dispatcher, firstly it is necessary to estimate the expected runtime of each job allocated by STMS under fluctuated resource availability.

Hence, we discuss how to estimate the probabilistic execution time (PET) of a job controlled by STMS as depicted in Fig. 2. Figure 2 shows that job i is reallocated from R_w to R_x and is then migrated again to R_y, when a failure is detected at t_f. Thus, R_x is responsible for executing the remaining size of job i from R_w, $RJ_{i,w}$, which is equal to '(a given length of job i—observed runtime by the state traces)'. By definition above, $ET_{i,x}$ is calculated by '$RJ_{i,w}/RA_x$'. We then assume that a failure is uniformly distributed, so that the average waste time is '$ET_{i,x}/2$' in case of one failure. The job's runtime is also associated with two failure probabilities of R_w and R_x denoted by P_w and P_x, respectively. As shown in Fig. 2, the resource failure impact on job execution in terms of PET can be considered as the sum of two parts: PET$(i, x; y)$:= *expected execution time* $(eet_{x,y})$ + *expected waste time* $(ewt_{x,y})$.

$$eet_{x,y} = (t_t - t_s) = P_w \cdot \{(1 - P_x) \cdot (ET_{i,x} + SO_{i,x})\}, \quad (2)$$

$$ewt_{x,y} = (t_c - t_t) = P_w \cdot \left\{P_x \cdot \left(\frac{ET_{i,x}}{2} + SO_{i,x} + DM_{i,x,y}\right)\right\}, \quad (3)$$

where $eet_{x,y}$ is particularly influenced by $SO_{i,x}$ that can be derived by '(number of state traces × its unit overhead)'. In case of $ewt_{x,y}$, it is also required to consider $DM_{i,x,y}$, which is computed as '$(t_c - t_f) = (RJ_{i,w} + JD_i)/AB_{x,y}$'.

2.3.2 Pessimistic approach in PET prediction

The proposed prediction algorithm is based on the pointwise approach where once a schedule is decided by a dispatcher, any degradation of service provisioning level should be tolerated although some performance related factors are unexpectedly changed. Therefore, an error margin is possibly added to the job's runtime prediction. Namely, this runtime estimation is based on the predicted allocation model; thus, it can be a type of pessimistic runtime anticipation of

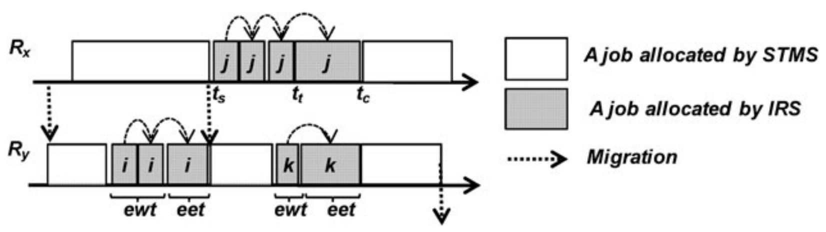


Fig. 3 Immediate restart with STMS

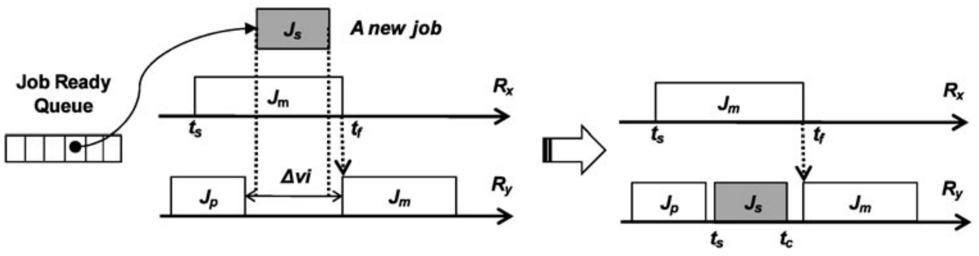


Fig. 4 Greedy backfilling for a new job in the queue

each job. To avoid unacceptable delays, potential inaccuracy of predicting the PET in STMS must be subject to the following equation:

Condition 1 *'pessimistically expected runtime* $\le \alpha \times$ *demanded flow time of a job'.*

Based on Condition 1, we can identify the acceptable number of failures or migrations by adjusting a value of α, an overestimation factor. Furthermore, we need to consider how this pessimistic approach influences to waiting jobs in a queue (future jobs) in terms of delays.

Problem 2 (Delays of queued jobs) *This limitation caused by the above approach may impose long ready time to future jobs in a queue. Namely, the STMS based scheduler would leave the free time slots of resources idle even though there were many waiting queued jobs.*

3 Aggressive reservation strategies

In this section, we propose three aggressive reservation strategies which are separately operational with the STMS based job allocation in order to resolve the Problems 1 and 2. The proposed methods are based on different technical approaches such as delay avoidance, greedy backfilling, and job preemption in order to assess job's eligibility.

3.1 Immediate restart strategy

In contrast to STMS, a failed job is re-dispatched to the same resource after resource availability increases to a sufficient level in this strategy. Here, this fault-tolerant re-dispatching rule is called an immediate restart strategy (IRS). Our first idea is to integrate STMS with IRS (IR-STMS). Figure 3 shows that any potential gaps in time between jobs can be filled with the short jobs executed by IRS. Hence, we expect that IR-STMS can be used as an inter-job delay guaranteed technique. As shown in Fig. 3, jobs i, j, and k are reallocated at the same resource, respectively as soon as they meet a failure. We assume that the amount of time of recovering resource availability from a specific failure is negligible in IRS.

In IRS, *eet* and *ewt* can be estimated as '$eet_{x,x} = (t_c - t_t) = ET_{i,x}$' and '$ewt_{x,x} = (t_t - t_s) = P_x \cdot (ET_{i,x}/2)$', respectively by the same manner as discussed in Sect. 2.3. Therefore, if there are n unexpected failures for processing a job i under IRS, we can obtain a PET for job i by

$$\mathrm{PET}(i,x;x) = ET_{i,x} + \sum_{f=1}^{n} \frac{P_x^f \cdot ET_{i,x}}{2}, \tag{4}$$

where P_x^f represents the fth failure probability of R_x.

We will present details of how to integrate STMS with IRS in Sect. 4.

3.2 Greedy backfilling strategy

In the conservative backfilling scheduling [13], after jobs are actually scheduled, they may be backfilled by a new job in the queue if some valid void-intervals are found. Thus, it has been expected that the backfilling algorithm guarantees start times of late jobs. However, rather it should be considered as a potential bonus according to our thorough observation. Due to the drawback of existing backfilling, we improve it with the discipline of weighted-shortest expected processing time (WSEPT) under STMS (GB-STMS). A main purpose of greedy backfilling strategy (GBS) is fitting as many small jobs as possible into void-intervals (inter-job delays) occurred by lots of migrations. Accordingly, we set the minimization of weighted flow time as the objective of WSEPT.

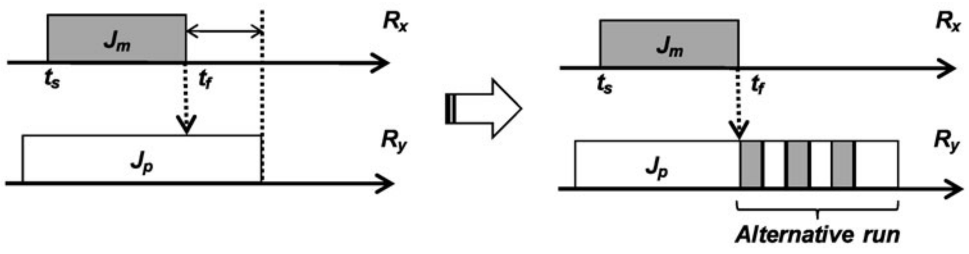


Fig. 5 Selective preemption for a migrant job

As depicted in Fig. 4, after a batch of jobs is allocated to some resources, the dispatcher scans the job ready queue for selecting a sufficiently small job according to WSEPT when some slacks are found. Then it immediately assigns a job one by one instead of aggregating small jobs, if such conditions are satisfied as follows:

Condition 2 *A length of void-interval (denoted as Δvi in Fig. 4) must be larger than or at least same as the length of selected job (J_s). Furthermore, J_s should not delay the previously running job (J_p) at a backup resource.*

Since IRS is assumed to be applied to estimate the expected runtime of J_s for selecting the mostly appropriate job to the objective of WSEPT, the PET of J_s can be estimated by (4). We also suppose that a demanded deadline of J_p is known when it is submitted. In this strategy, each reservation request for a new job is individually processed in a greedy and non-preemptive manner. So, it may be especially effective when the number of small jobs is dominant in the given workloads. Furthermore, it can proceed until the end of executing all jobs previously assigned to resources or before the next scheduling cycle starts. Then the rest of jobs in the job ready queue will be included in a new batch of the next scheduling cycle.

3.3 Selective preemption strategy

A basic concept of selective preemption strategy (SPS) is to avoid the delayed job executions (i.e., starvation) occurred by frequent migrations. More specifically, our approach extends the 'immediate service' [6] in three directions as follows: (1) We consider how to adaptively process a migrant job without compromising a previously running job's deadline requirement; while, the authors in [6] addressed how to reduce the number of late jobs by dealing with queued future jobs. (2) A job suspension criterion proposed here is based on the expected runtime which is calculated by the PET estimation method, instead of using the total accumulated runtime. (3) This strategy does not limit the number of suspensions; namely, we use a preemption factor to provide tunability in SPS by controlling a rate of suspension.

Figure 5 shows one example of how SPS can be applied in the STMS based job allocation (SP-STMS). Suppose that a migrant job, J_m, is aborted at time t_f due to a failure and then it is moved to a backup resource, R_y. Thus, J_m should compete with previously executed or reserved job, J_p, at R_y (i.e., race condition). At this time, two preemption priorities of J_m and J_p are compared. The preemption priority, PP, is computed by considering the expected remaining length and deadline of both jobs as follows:

$$PP(J_m) = \frac{t_w^m + t_r^m}{t_r^m}, \tag{5}$$

where t_w^m represents the time J_m has to be waited for re-allocation, when it is preempted by J_p. t_r^m is the expected remaining time of processing J_m and is calculated by the PET estimation. $PP(J_p)$ is also derived in the same way as $PP(J_m)$. Next, the dispatcher needs to decide whether J_m is suspended or preempts J_p by using a preemption factor (PF) which can be defined as '$PF = PP(J_m)/PP(J_p)$'. If the following condition is satisfied, then J_m preempts J_p.

$$PP(J_p) \cdot c < PP(J_m) = PF > c. \tag{6}$$

In (6), c should be a variable (termed 'preemption coefficient') since each job has a different remaining time to process as well as a different deadline. J_p should be waited for the next time slot until PF is equal to or smaller than c. In this strategy, if one is suspended then its preemption priority is increased; while, a preemption priority of running job is decreased with the same rate of the suspended job at a unit time. Therefore, execution of the two jobs can alternate as shown in Fig. 5, when a value of $PP(J_m)$ is close to $PP(J_p)$'s.

In order to identify the adaptive benefit from each strategy, we then propose a messy genetic algorithm (mGA) based allocation framework which operates under STMS.

4 mGA based allocation framework

We consider how to achieve the optimal or near-optimal mapping between jobs and resources under STMS in an efficient manner. Since STMS generally requires non-polynomial complexities, we use mGA [14] which can provide not only a special way of encoding a given problem but also very tractable search operations within extremely huge search space for satisfying specific performance criteria. In this section, we discuss three important parts of mGA: an encoding schema, search operations, and a fitness function.

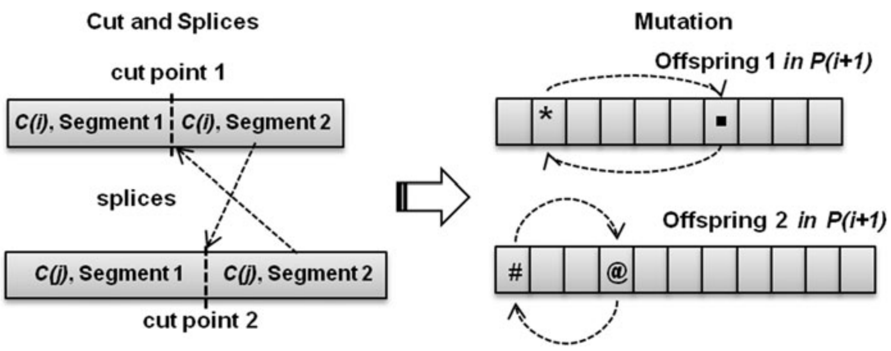


Fig. 6 Evolutionary search operations in mGA

4.1 Encoding schema used in mGA

In mGA, first and most importantly, an encoding schema should be designed before running genetic operations because it offers a way to define a problem that one focuses. In contrast to the traditional GAs using binary code, mGA provides a more tractable code comprising one dimensional string such as (position, attribute) [11]. This string is especially suitable for constructing the scheduling problem. Thus, the resource allocation to a job can be expressed as a gene such as (job index, resource index), regardless of the used scheme. We suppose that five jobs are waiting to be scheduled and three resources are available at this time. Thus, a schedule σ, referred to as a solution in the job assignment problem, can be defined by a set of 2-tuple strings as follows: $\sigma = \{(1,3); (4,2); (3,3); (1,2); (4,4); (5,2); (1,5)\}$. For example, job 1 has three resources indexed by 3, 2, and 5. This means that job 1 will be executed in order of appearance of resource index by STMS. Namely, job 1 is first assigned to resource 3 and will be migrated to resource 2. Then it will be finally completed at resource 5. In case of job 3 and 5, they have only one resource indexed by 3 and 2, respectively, so that they will be processed by IRS. Therefore, any resources can be randomly assigned to a particular job in this representation of schedule. In other words, various combinations of resource indices could exist for a particular job with different execution schemes such as STMS and IRS.

4.2 Messy genetic search operations

A set of schedules (called chromosomes in mGA) is referred to as a population. The messy genetic search technique evolves an initial population of possible chromosomes into the solution that can be obtained by recombination operations such as selection, cut-and-splices, and mutation, under the principle of the survival of the fittest. Algorithm 1 shows the overall procedure of mGA operations. This algorithm starts with a set of contending trial solutions called the initial population, $P(1)$. We then use the roulette wheel selection [1], where each individual chromosome $C(i)$ is selected

Algorithm 1 Messy genetic search operations

```
1.  Set $g = 1$ and generate an initial population $P(1)$
2.  **For** $g = 1$ until the stopping criterion is satisfied
3.      **Evaluate** $P(g)$
4.      If the stopping criterion is satisfied, then stop
5.      **Select** $C(i)$ from $P(g)$ by '$p_{sel(i)} = f_i / \sum_{q=1\sim k} f_q$'
6.      **Evolve** $C(i)$ to form $P(g+1)$ by using such
        Cut-and-Splices and Mutation operations
7.      Set $g = g + 1$
8.  **End**
```

with a probability, $p_{sel(i)}$, that is calculated by using a fitness value of each $C(i)$, f_i, as follows: $p_{sel(i)} = f_i / \sum_{q=1\sim k} f_q$, where k is the number of chromosomes in the current population. Therefore, a chromosome with a high selection probability can have an opportunity to be evolved to the next generation.

As illustrated in Fig. 6, with its predefined probability the cut and splices then attempt to select a cut point in two chromosomes (parents) one of which are chosen by the above selection method from the gth population, $P(g)$. So, each parent is divided into two parts. These operations then result in two child chromosomes (offspring) by exchanging the selected part with each other. In other words, they perform global search functionality. Next, the mutation operation swaps two genes selected by its given probability as well within the offspring in order to search the local optimum. Through these evolutionary search operations, the offspring are differently evolved in order to form the next generation, $P(g+1)$. Then, this algorithm proceeds until the stopping criterion is satisfied. Typically, the mGA is terminated when a predefined number of generations is achieved or when a fitness value is reached to a certain level. For more details on search operations, we refer to [14].

4.3 Designing of performance benefit

4.3.1 Fitness function

In the proposed allocation framework, we consider two objectives such as makespan *ms* and average utilization *au* as

Table 2 Parameter configuration adopted for the performance evaluations

Parameters	Values
Mean job arrival rate, Scheduling interval	1.67 jobs/sec, 1 hour
Job size, Batch size, Job data size	0.1–1 Mflops, 50, 300–1,000 MB
Resource availability, Average bandwidth	100–500 flops/sec, 10–50 Mbps
Overestimation factor, Preemption coefficient	1.75, 2.5
Population size, Maximum number of generations	30, 100
Cut-and-Splices probability, Mutation probability	0.18, 0.25

the performance benefit to be optimized. To achieve the bi-criteria optimization, we need to transform different quantities into the same domain in the form of combined fitness function. We thus normalize *ms* and *au* of chromosome *i* by their difference between the maximum value and the minimum value, respectively as follows:

$$\overline{ms_i} = \frac{ms_{\max} - ms_i}{ms_{\max} - ms_{\min}}, \qquad \overline{au_i} = \frac{au_{\min} - au_i}{au_{\max} - au_{\min}}, \tag{7}$$

where $\overline{ms_i}$ and $\overline{au_i}$ are normalized *ms* and *au* of chromosome i, respectively. Then a converged fitness for chromosome i, f_i, is derived from (7) by using the concept of weighted sum of objectives [9] as follows:

$$f_i = (\delta \cdot \overline{ms_i}) + (\omega \cdot \overline{au_i}) \quad \text{for } \delta + \omega = 1. \tag{8}$$

In (8), δ and ω specify scalar weights of each objective. Hence, it allows the equitable comparison through achieving the tuned fitness between the two objectives. A larger fitness value guarantees a better scheduling in terms of the performance benefit.

4.3.2 *Risk assessment factor for late jobs*

Each job has its deadline constraint, so that a degree of risk due to late jobs is generally evaluated by the job failure rate (JFR) that is the average percentage of missed deadlines. JFR assumes that all jobs are of equal criticality and require the same quality of services. However, the importance of each job is differentiated by its deadline. Accordingly, it is hard to expect that this factor could be suitable to represent risk (i.e., delay) resilience of a scheduling algorithm. Therefore, we define an inter-job fairness (IJF) as a new metric which describes how fairly a scheduling algorithm treats each job within a batch. In this context, IJF is defined as '1 − the standard deviation of JFR'.

$$\mathrm{IJF} = 1 - \sqrt{\frac{\sum_{i=1}^{n} (\xi_i \text{ } late\ jobs/\xi_i\ jobs - \overline{\mathrm{JFR}})^2}{n}}. \tag{9}$$

In (9), n represents the number of batches and ξ_i denotes the ith batch. $\overline{\mathrm{JFR}}$ is a mean value of JFR. To assess a degree of delays imposed by late jobs, we add the '1 − IJF' as a penalty function into the fitness function as follows:

$$f_i = (\delta \cdot \overline{ms_i}) + (\omega \cdot \overline{au_i}) + \kappa \cdot (1 - \mathrm{IJF}), \tag{10}$$

where κ represents a negative coefficient.

4.4 Flexible scheduling acceleration

To be granted exclusive access to a resource partition (i.e., space sharing), jobs should be submitted to a batch scheduler and then held in a job ready queue until a sufficient number of jobs become available to create a new schedule. Accordingly, it is not feasible to perform mGA on a small number of jobs. For this problem, we adopt the sliding window concept [1] as an alternative method that individually assigns remaining jobs to the lightest loaded resources in a manner of non-combinational scheduling algorithms, for example *Min-Min* [9].

5 Simulation and discussion

In this section, we present the results of a trace-driven simulation of four different STMS scheduling schemes discussed above. To compare their various aspects of performance, we consider six metrics such as makespan, average utilization, average throughput, JFR, IJF, and cost effectiveness. In the simulation, resource availability and job sizes are measured by million floating point operations (Mflops). Furthermore, every single point of metrics in the plots has been extensively computed with 30–50 resources and 2,000–10,000 jobs. Table 2 shows the parameter configuration mainly used in this simulation.

5.1 Overall performance comparison

In order to examine the overall trends of each scheduling algorithm as the number of offered workloads increases, we have implemented a trace-driven simulation of 50 resources and 2,000–10,000 jobs. As indicated in Fig. 7, SP-STMS averagely achieves better performance in its makespan, average throughput, and JFR than IR-STMS's as the job's burstiness increases. It means that preventing the delayed job executions is a more effective technique than reducing the inter-job delay in the migration based job allocation. However, IR-STMS shows the best case in the average utilization because sharing the inter-job delays with the IRS-ruled jobs can maximize the efficiency of resource usage. Moreover,

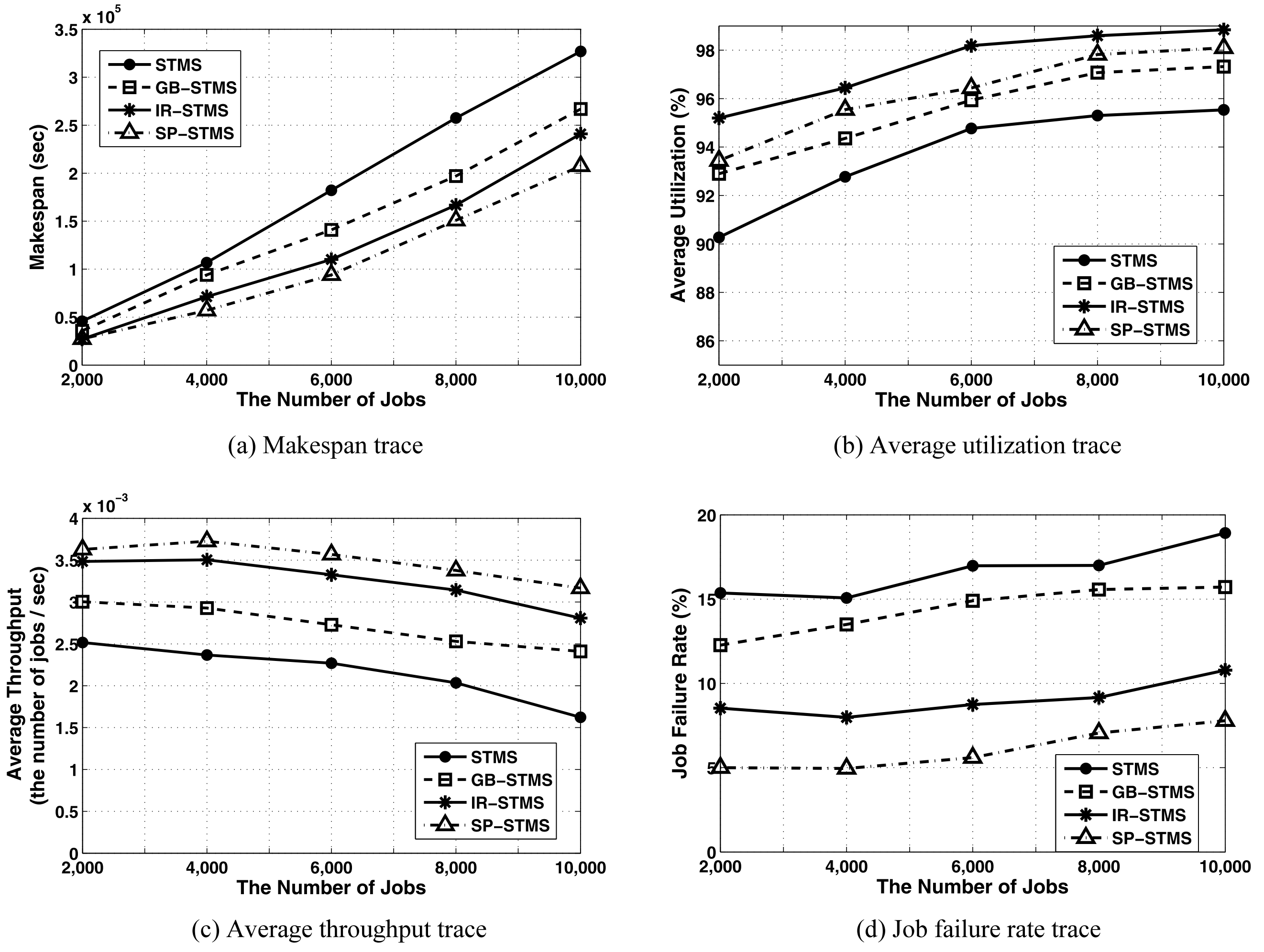


Fig. 7 Overall performance comparison

it is found that GB-STMS is entirely outperformed by IR-STMS. Namely, reserving inter-job delay for IRS-ruled jobs can be a more appropriate solution rather than fitting new jobs from a queue to the void-intervals for minimization of the completion time of jobs.

All schemes show that their average throughput gradually decreases; while, they have steady-state curves in JFR. In case of STMS, it demonstrates the worst performance due to its lack of flexibility to changes in resource availability as discussed above. Each trace of a particular migration scheme provides its fine-grained performance trend, so that it is very useful not only to comprehend entire system characteristics but also to anticipate system quality over some future time interval.

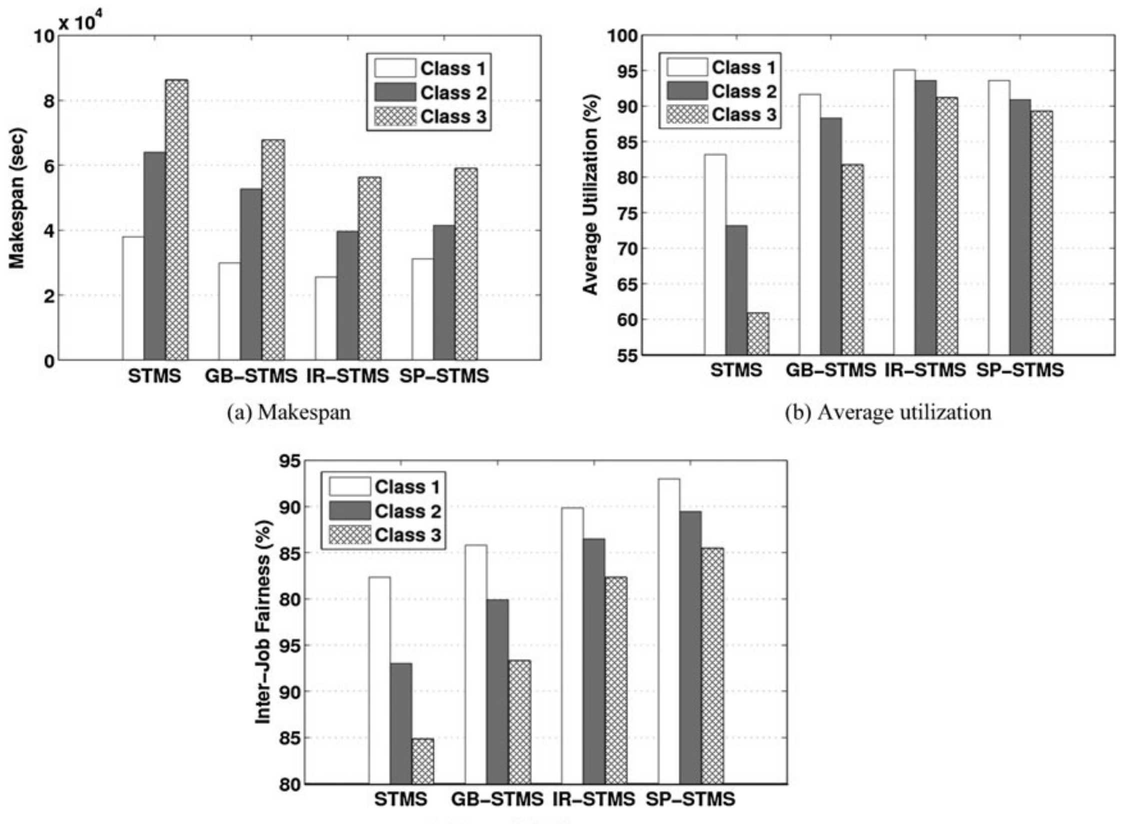
5.2 Behavior analysis under various job lifetime distributions

As presented in the archive of real workload logs [15], there are four types of jobs according to the length of a job: very short, short, long, and very long. However, in this simulation our objective is to investigate potential influences of different job lifetime distributions rather than individual job sizes on scheduling performance. As a reason of that, we classify the job lifetime distribution to 3 classes as shown in Table 3. Here, we suppose that the length of a short job is less than 0.15 Mflops and the size for a long one is in the range of 0.85 and 1 Mflops. The number of jobs and resources are considered to be fixed as 4,000 and 30, respectively.

Table 3 Classes of job lifetime distribution

	Short-lived : Long-lived
Class 1	8 : 2
Class 2	5 : 5
Class 3	2 : 8

Figure 8 demonstrates how strongly each scheme can respond to the different job lifetime distributions in aspects of three major metrics such as makespan, average utilization,



(a) Makespan

(b) Average utilization

(c) Inter-job fairness

Fig. 8 Analysis of scheduling behavior under three job lifetime distributions

and IJF. First, IR-STMS shows a much flexible ability in makespan compared to the other schemes. However, this result is slightly different from the overall trend on makespan shown in Fig. 7(a). This is due to the fact that IR-STMS possibly have more opportunities to fit the IRS-ruled jobs into potential void-intervals as the number of short jobs increases. On the other hand, SP-STMS has slightly higher makespan than IR-STMS does. Because of the frequent migrations, there is a certain degree of competition between jobs (J_m and J_p) to occupy the resource. So, this scheme can stay in a state of balance even though the number of long-lived jobs increases. With the same reason as IR-STMS, GB-STMS's makespan is better than STMS's. Next, three migration schemes except for STMS start from more than 90 percent average utilization in the class 1. However, all strategies generally suffer from degradation of average utilization as a ratio of large jobs increases from 20 to 80 percent. In particular, STMS experiences 27 percent reduction; while, this metric is stably maintained by IR-STMS and SP-STMS regardless of the classes. Although GB-STMS is designed to attempt to reserve some potential void-intervals for some future jobs, its effectiveness is less than IR-STMS's. In case of IJF, one interesting observation is that SP-STMS entirely outperforms the other schemes. Since a job to be executed is fairly selected by competition based on the preemption priority, SPS can guarantee IJF regardless of the classes. On the other hand, IR-STMS adaptively merges the IRS-ruled jobs and the STMS-ruled jobs in the form of batch to optimize the performance benefit. Owing to this inherited principle, IR-STMS seems less sensitive to changes in the job lifetime distribution.

As discussed, the average utilization can be overestimated by including unavoidable idle time slots. This problem is also induced by inaccurately estimating runtimes of jobs with insufficient information regarding scheduling conditions and requirements. To cope with the first issue, we propose to combine the aggressive reservation strategies with STMS. However, relation between the problem and the second reason still remains uncertain.

5.3 Impact of inaccurate PET estimation on the inter-job fairness

Although the migration schemes has been designed for mitigating harmful effects of late jobs, JFR may not be a suitable criterion to evaluate how jobs with various importance are fairly executed within a batch. Moreover, previous migration models tend to depend on an impractical assumption that a runtime generally would be estimated by users

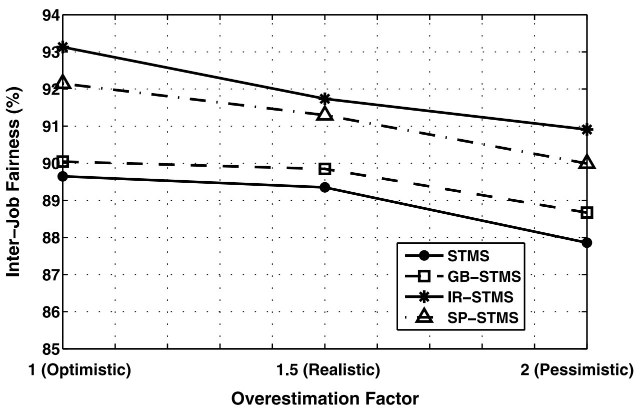


Fig. 9 Impact of inaccurate PET estimation

prior to scheduling. However, this simplification is overly optimistic in the sense that user's expected runtime may be almost close to a real processing time.

With 4,000 jobs and 30 resources, we have conducted a measurement to investigate how flexibly or stably each migration algorithm can maintain IJF according to a degree of inaccuracy of PET. To this end, we classify a degree of inaccuracy in the PET estimation into three types by introducing the over estimation factor α as follows:

- Optimistic prediction case: $1 \leq \alpha < 1.5$
- Realistic prediction case: $1.5 \leq \alpha < 2$
- Pessimistic prediction case: $2 \leq \alpha < 3$

As depicted in Fig. 9, we observe that IR-STMS and SP-STMS have a similar pattern of curves and they result in relatively high IJF compared to GB-STMS and STMS.

The result of IR-STMS represents an improvement 6.8 percent over STMS in case of optimistic prediction. In contrast to the result described in Fig. 8(c), IR-STMS always performs better than SP-STMS. To be precise, the pessimistic or realistic prediction case produces much more race conditions to be resolved by SPS. Thus, there are lots of negative effects on SP-STMS's IJF due to the inaccurate PET estimation. In particular, little improvement in the IJF is found in GB-STMS after α is smaller than 1.5. As a degree of inaccuracy in the PET estimation increases, lots of early completed jobs occur. So, a batch scheduling can be finished earlier in order to accommodate a new batch. Consequently, jobs in the ready queue lose an opportunity to occupy some idle time slots. For this reason, GB-STMS shows low flexibility in the IJF. From this result, we realize that more accurate prediction of PET leads to more robust IJF. However, the "quality" of runtime estimation is not defined by how close it is to the actual processing time. Rather its excellence can be evaluated by how much better it is compared to the average estimation.

5.4 Cost-effectiveness of resource usage

As seen in Figs. 7, 8, and 9, we discuss the performance behavior of user's perspective. In this section, we now turn our attention to resource provider's perspective. With the same simulation configuration as addressed in Sect. 5.3, we have evaluated the resource usage cost and the cost effectiveness of each migration scheme according to the following metrics:

$$c_i = \frac{\sum_{k=1}^{m}(RA_k \cdot F_c(t_c^k))}{m}, \tag{11}$$

where c_i and t_c^k represents the cost overhead of a particular migration scheme i and the completion time in resource k, respectively. The total number of assigned resources is assumed to be m. Moreover, we suppose that the usage cost of occupying one resource follows a concave function, $F_c(\cdot)$ that has a monotonically non-decreasing curve over time [16]. Thus, the cost overheads are proportional to accumulated resource occupation time and average provisioning level of resource availability. Accordingly, we then have the cost effectiveness of migration scheme i, ce_i as

$$ce_i = \left(\frac{\nu_i \cdot c_s}{c_i} - \nu_s\right) \cdot 100(\%), \tag{12}$$

where c_s denotes the cost overhead under STMS. ν_s and ν_i are and the average utilization when STMS and a migration scheme i are respectively applied.

Figure 10(a) is a plot of the average cost overheads that are required on each resource for different migration schemes. Since IRS-ruled jobs are repeatedly assigned to the same (faulty) resource when a failure occurs, they do not have any chance to utilize a more capable or stable resource. Hence, there is a tendency for the IRS-ruled jobs to be easily failed compared to the STMS-ruled jobs. Consequently, cost overheads of IR-STMS are higher than SP-STMS's and GB-STMS's. Even though STMS does not have any burdens imposed by IRS and is somewhat free from the job restart delays, it incurs unnecessary costs for migrations due to the absence of any reservation method. Figure 10(b) shows that how much improvement in average utilization per unit cost each migration scheme can guarantee compared to STMS's. Since the most expensive approach is STMS, the cost effectiveness of the other schemes is normalized by STMS's. As discussed in Figs. 7(b) and 8(b), IR-STMS demonstrates the best performance in average utilization. However, owing to its large cost overheads, the cost effectiveness with regard to the average utilization is not much higher than the other schemes. From this result, we observe that three migration schemes achieve 35, 54, and 49 percent improvements, respectively in the cost-effective resource usage.

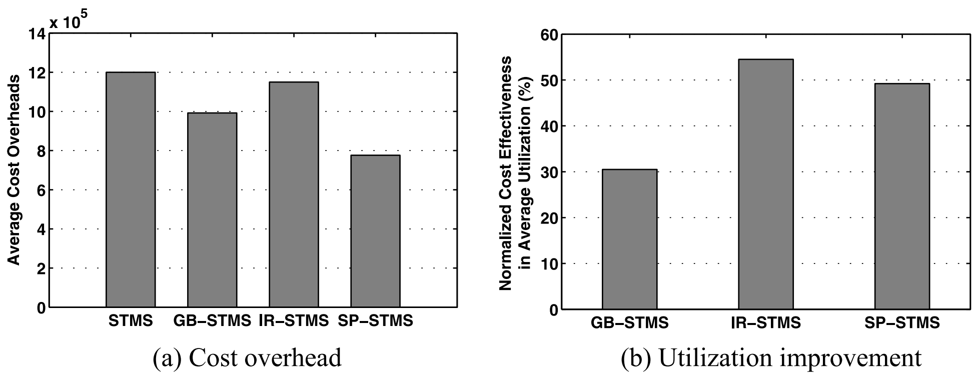


Fig. 10 Cost effectiveness of resource usage

6 Conclusion

This paper has reviewed four migration schemes we developed to enhance the job allocation performance in DCS. We started with an analysis of three commonly occurred problems: job restart delay, inter-job delay, and delayed job execution. Then we discussed how to estimate the expected runtime with our resource failure model and proposed three aggressive reservation strategies mentioned above. To combine each strategy into the batch mode scheduling, we also proposed the mGA based allocation framework. One key objective of this research is to investigate how effectively STMS with each strategy reactivates a failed job to reduce the three inevitable delays. As the second contribution of this paper, we observed how the proposed migration schemes take advantage of idle time slots for IRS-ruled jobs or new jobs to optimize the performance benefit without compromising IJF.

Summarized below are the major contributions and research results in this paper.

- SP-STMS entirely outperforms the other schemes when the amount of offered jobs increases. In general, preventing the delayed job executions with SP-STMS could be a better solution rather than reducing the inter-job delays by using IR-STMS or GB-STMS.
- IR-STMS has shown less sensitive behaviors than SP-STMS and GB-STMS when the long jobs are dominant in the given workload. Despite the inaccurate prediction of PETs, allocating the IRS-ruled jobs and the STMS-ruled jobs together has turned out to be the best approach in terms of IJF. On the whole, the performance would be also improved by fitting IRS-ruled jobs rather than new jobs into the void-intervals under STMS.

Acknowledgements This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (No. 2011-0020522).

References

1. Zomaya, A.Y., Teh, Y.-H.: Observations on using genetic algorithms for dynamic load-balancing. IEEE Trans. Parallel Distrib. Syst. **12**(9), 899–911 (2001)
2. Buyya, R., Abramson, D., Giddy, J.: Nimrod/g: an architecture for a resource management and scheduling system in a global computational grid. In: Proc. of the HPC ASIA 2000, pp. 283–289 (2000)
3. Xiong, Y., Vandenhoute, M., Cankaya, H.C.: Control architecture in optical burst-switched WDM networks. IEEE J. Sel. Areas Commun. **18**, 1838–1851 (2000)
4. Iizuka, M., Sakuta, M., Nishino, Y., Sasase, I.: A scheduling algorithm minimizing voids generated by arriving bursts in optical burst switched WDM network. In: Proc. of IEEE GLOBECOM 2000, pp. 2736–2740 (2000)
5. Zhao, W., Ramamritham, K., Stankovic, J.A.: Preemptive scheduling under time and resource constraints. IEEE Trans. Comput. **C-36**(8), 949–960 (1987)
6. Chiang, S.H., Vernon, M.K.: Production job scheduling for parallel shared memory systems. In: Proc. of Int. Parallel and Distributed Processing Symp. (2002)
7. Parsons, E.W., Sevcik, K.C.: Implementing multiprocessor scheduling disciplines. In: Feitelson, D.G., Rudolph, L. (eds.) Proc. of Job Scheduling Strategies for Parallel Processing (IPPS’97), pp. 166–192. Springer, London (1997)
8. Ruscio, J.F., Heffner, M.A., Varadarajan, S.: DejaVu: transparent user-level checkpointing migration and recovery for distributed systems. In: IEEE Int. Parallel and Distributed Processing Symp. (2007)
9. Baker, K.R., Trietsch, D.: Principles of Sequencing and Scheduling. Wiley, New York (2009)
10. Moon, Y.-H., Youn, C.-H.: Integrated approach towards adaptive state-tracking job migration for maximising performance benefit. IEE Electron. Lett. **46**(25), 1659–1661 (2010)
11. Song, S., Hwang, K., Kwok, Y.: Risk-resilient heuristics and genetic algorithms for security-assured grid job scheduling. IEEE Trans. Comput. **55**(6), 703–719 (2006)
12. Rodríguez, G., Pardo, X.C., Martín, M.J., González, P.: Performance evaluation of an application-level checkpointing solution on grids. Future Gener. Comput. Syst. **26**(7), 1012–1023 (2010)
13. Mu’alem, A.W., Feitelson, D.G.: Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. IEEE Trans. Parallel Distrib. Syst. **12**(6), 529–543 (2001)
14. Gonzalez Pico, C.A., Wainwright, R.L.: Dynamic scheduling of computer tasks using genetic algorithms. In: Evolutionary computation, 1994. IEEE World Congress on Computational Intelligence, Proc. of the First IEEE Conference on, vol. 2, pp. 829–833 (1994)

15. Logs of Real Parallel Workloads: http://www.cs.huji.ac.il/labs/parallel/workload/logs.html (2011)
16. Albert, E., Arenas, P., Gnaim, S., Herraiz, I., Puebla, G.: Comparing cost functions in resource analysis. In: Proc. of the Int. Conference on Foundational and Practical Aspects of Resource Analysis (FOPARA) (2009)

Yong-Hyuk Moon received the B.S. degree in Computer Engineering from Dankook University, Seoul, Korea in 2003 and the M.S. degree in Information and Communication Engineering from Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea in 2006. Currently, he is a Ph.D. candidate in Department of Information and Communications Engineering at KAIST. Since 2006, he is also with the Software Research Laboratory in Electronics and Telecommunications Research Institute (ETRI), Daejeon, Korea. His research interests are in areas of distributed computing systems, networks and algorithms.

Chan-Hyun Youn is a Professor with the Department of Electrical Engineering and is a Director of the Grid Middleware Center, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea. He is also a Vice President of Grid Forum Korea (OGF-KR). He received his B.S. and M.S. degrees in Electronics Engineering from Kyungpook National University, Daegu, Korea, in 1981 and 1985, respectively. He also received a Ph.D. in Electrical and Communications Engineering from Tohoku University, Japan, in 1994. Before joining the University, he worked for Korea Telecommunications (KT) as a Leader of High-Speed Networking Team. He also was a Visiting Scholar at MIT, Cambridge, USA in 2004.