A Hyper-Heuristic Scheduling Algorithm for Cloud

Chun-Wei Tsai, Wei-Cheng Huang, Meng-Hsiu Chiang, Ming-Chao Chiang, and Chu-Sing Yang

Abstract—Rule-based scheduling algorithms have been widely used on many cloud computing systems because they are simple and easy to implement. However, there is plenty of room to improve the performance of these algorithms, especially by using heuristic scheduling. As such, this paper presents a novel heuristic scheduling algorithm, called hyper-heuristic scheduling algorithm (HHSA), to find better scheduling solutions for cloud computing systems. The diversity detection and improvement detection operators are employed by the proposed algorithm to dynamically determine which low-level heuristic is to be used in finding better candidate solutions. To evaluate the performance of the proposed method, this study compares the proposed method with several state-of-the-art scheduling algorithms, by having all of them implemented on CloudSim (a simulator) and Hadoop (a real system). The results show that HHSA can significantly reduce the makespan of task scheduling compared with the other scheduling algorithms evaluated in this paper, on both CloudSim and Hadoop.

Index Terms—Cloud computing, evolutionary algorithm, scheduling, and Hadoop

1 Introduction

Until now we are still actively looking for possible solutions to enhance the performance of information systems for computation, analysis, and storage. Since distributed and parallel computing was widely used to enhance the performance of a variety of computer systems, several strategies have been presented for different approaches and congenital restrictions in different eras. No matter which consideration it is for, how to efficiently manage computer resources is definitively a vital research issue. Among them, scheduling [1], [2], [3] is essential in the success of enhancing the performance of a computer system.

With the advance of computer and internet technologies, paradigms of cloud computing [4], [5], [6], [7] have been successfully used on several information systems in recent years. Although cloud computing systems nowadays provide a better way to carry out the submitted tasks in terms of responsiveness, scalability, and flexibility, most job and task scheduling problems on cloud computing systems are still either NP-hard or NP-complete [8]. According to our observations, most rule-based scheduling algorithms (e.g., exhaustive and deterministic algorithms) are widely used on today's cloud computing systems because they are simple and easy to implement. Unfortunately, these rule-based scheduling algorithms are inappropriate for large-scale or complex scheduling problems because the results of these scheduling strategies are usually far from optimal. This implies that there is plenty of room to improve the scheduling algorithms for cloud computing systems.

A promising research direction, which applies modern heuristics [9] to scheduling on cloud computing systems, has attracted several researchers from different research domains. The extended version of heuristics combines two or more heuristics into a single heuristic algorithm to leverage their strengths for scheduling, called hybrid heuristic-based scheduling algorithm. Different from most hybrid heuristic-based scheduling algorithms that employed only one or two heuristics, the proposed algorithm is a hyper-heuristic-based algorithm that integrates several heuristic algorithms. The basic idea of the proposed algorithm is to leverage the strengths of heuristic algorithms, such as simulated annealing [10], genetic algorithm [11], particle swarm optimization [12], and ant colony optimization [13], by integrating them into a single algorithm. The main contributions of the paper can be summarized as follows:

1) A novel high-performance hyper-heuristic algorithm is proposed for scheduling on cloud computing systems to reduce the makespan.
2) Two detection operators—one for diversity detection and one for improvement detection—are proposed for the proposed algorithm to determine the timing to employ the low-level heuristic (LLH) algorithm.
3) The proposed algorithm can be applied to both sequence-dependent (see also Section 5.1.1) and sequence-independent (see also Section 5.2.1) scheduling problems.
4) To evaluate its performance on both a cloud simulator and a real cloud system, the proposed algorithm and several other scheduling algorithms (for the purpose of comparison) are implemented on both the CloudSim [14] and Hadoop.

• *C.W. Tsai is with the Department of Applied Informatics and Multimedia, Chia Nan University of Pharmacy & Science, Tainan 71710, Taiwan. E-mail: cwtsai0807@gmail.com.*
• *W.C. Huang, M.H. Chiang, and M.C. Chiang are with the Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung 80424, Taiwan. E-mail: demon2506@yahoo.com.tw, m993040011@student.nsysu.edu.tw, mcchiang@cse.nsysu.edu.tw.*
• *C.S. Yang is with the Department of Electrical Engineering, National Cheng Kung University, Tainan 70101, Taiwan. E-mail: csyang.ee.ncku.edu.tw.*

Manuscript received 15 Sept. 2013; revised; 16 Mar. 2014; accepted 18 Mar. 2014. Date of publication 6 Apr. 2014; date of current version 30 July 2014.
Recommended for acceptance by I. Bojanova, R.C.H. Hua, O. Rana, and M. Parashar.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TCC.2014.2315797

5) A detailed analysis of the parameter settings is also given to show the impact they may have on the performance of the proposed algorithm.

The remainder of the paper is organized as follows. Section 2 begins with a brief introduction to the traditional scheduling, followed by the scheduling on cloud computing. In addition to the scheduling problem and its relevant technologies, Section 3 provides a brief discussion on heuristic algorithms and compares the differences between hybrid-heuristic and hyper-heuristic algorithms. Section 4 describes in detail the proposed algorithm and gives a simple example. The simulation results, on both CloudSim and Hadoop, are discussed in Section 5. Conclusions and future trends are drawn in Section 6.

2 Scheduling

2.1 Scheduling Problems

2.1.1 Traditional Scheduling

The scheduling problem [15], [16] can be defined as follows: Find an optimal solution to schedule a given set of tasks $T = \{T_1, T_2, \ldots, T_n\}$ to a given set of machines $M = \{M_1, M_2, \ldots, M_m\}$ subject to a predefined set of constraints and measurements. For instance, one of the widely used measurements for the scheduling problem is the so-called makespan $C_{\max}(s)$, which is defined as the completion time of the last task. Mathematically, the scheduling problem is to

$$\text{minimize} \quad f(s) = C_{\max}(s), \tag{1}$$

where s is a candidate solution, and, letting C_j denote the completion time of job j, $C_{\max}(s) = \max_j C_j$ is the completion time of the last job. Moreover, with advance of computer systems, the scheduling problem has evolved from single processor, single tasking, to multiprocessor, multitasking, then to large scale distributed system nowadays. To measure the solutions for different kinds of scheduling problems, makespan is no longer the only choice. Instead, a number of measurements have been developed. Examples are maximum lateness, maximum tardiness, and maximum flowtime [1].

Several traditional scheduling algorithms have been presented, such as earliest due date first (EDD) [17], critical path method (CPM) [18], project evaluation and review technique (PRET) [19], dynamic programming [20], and branch-and-bound [21]. These methods are generally much easier to implement than metaheuristic and hybrid scheduling algorithms because they are all designed based on one or a few particular rules (ways) to manage and arrange the tasks, but metaheuristic and hybrid scheduling algorithms are not. Some of them even guarantee to find the optimal solution for the scheduling problem in question, but they are all suffered from the fact of not being able to find the optimal solution in a reasonable time, especially when the problem becomes either too complex or too large.

In fact, it is the other two important disadvantages that take the researchers into consideration of modifying the search strategy of these traditional scheduling algorithms. First, most full search algorithms, such as the well-known branch-and-bound and dynamic programming, are normally time consuming because the number of checks they have to perform is very large. Second, in the case of the deterministic algorithms (such as EDD), although they are very fast and easy to implement, they are easily falling into local optima. The classical exhaustive algorithms are generally good enough to provide the optimal solution quickly for small scheduling problems [22], but not large scheduling problems [23]. In this paper, by the small scheduling problems, we mean problems for which all the solutions can be checked in a reasonable time by using classical exhaustive algorithms running on modern computer systems. In contrast, by the large scheduling problems, we mean problems for which not all the solutions can be checked in a reasonable time by using the same algorithms running on the same computer systems. These observations make it easy to understand that exhaustive algorithms will take a prohibitive amount of time to check all the candidate solutions for large scheduling problems because the number of candidate solutions is simply way too large to be checked in a reasonable time. As a result, researchers have paid their attention to the development of scheduling algorithms that are efficient and effective, such as heuristics.

2.1.2 Scheduling on Cloud

With computing systems being shifted to cloud-based systems progressively, one of the main characteristics is that it works on a pay-as-you-use basis [24]. Several studies attempted to define the scheduling problem on cloud systems as the workflow problem, which can be further classified into two levels: service-level (platform layer and static scheduling) and task-level (unified resource layer and dynamic scheduling) [25]. Different from grid computing, the user can install their programs on the virtual machines (VMs) and determine how to execute their programs on the cloud computing system. For these reasons, although both grid computing and cloud computing are heterogeneous, the key issues they face are very different. A good example is the cost and latency of data transfer on these environments. That is why some studies added more considerations to their definitions of scheduling on cloud. For instance, a couple of studies [26], [27] used directed acyclic graph (DAG) to define the scheduling problem on cloud. The basic idea is to use the vertices of a DAG to represent a set of tasks and the edges between the vertices to represent the dependencies between the tasks. Then, the scheduling problem on cloud can be formulated for the solution s as follows [27]:

$$\begin{aligned} \text{minimize} \quad & f(s) = C_{\max}(s) + \sum_{i=1}^{n} \sum_{j=1}^{m} C_{ij}, \\ \text{subject to} \quad & C_{\max}(s) \le U(s), \\ & C(s) \le B(s), \end{aligned} \tag{2}$$

where $f(s)$ is the objective function; $C_{\max}(s)$ the completion time of the last job (also called makespan); n the number of tasks; m the number of machines; C_{ij} the cost of processing the ith task on the jth machine; $U(s)$ the number of overdue tasks; $C(s) = \sum_{j=1}^{m} C_{ij}$ the total cost of s; and $B(s)$ the restriction on the budget for the tasks of s. Moreover, since the study described in [27] also takes into account the maximum expenditure for the provisioning of services or applications on cloud, such as QoS, the budget is also considered as part of the scheduling problem for cloud when necessary.

Recently, Hadoop [28] was widely used in deploying cloud computing system. In addition to being used by several international companies (e.g., IBM, Google, AOL, eBay, Yahoo, and Amazon) [29], the features provided by the open-source environment and MapReduce framework have made it a very popular cloud computing platform [30]. Its scalability for large-scale data and systems has also been found in recent researches, such as big data approach [31] and smart grid [32]. The basic idea is so that the users of such a system can submit jobs to the Hadoop cluster, which will then split the jobs into tasks of a given size and provide a number of slots to run tasks in parallel. The processing speed of these slots depends on the types of jobs submitted to the cluster. The definitions of parallel machines for scheduling can be divided into three kinds: (1) identical machines in parallel, (2) machines in parallel with different speeds, and (3) unrelated machines in parallel [1]. Since a Hadoop cluster generally consist of machines with different speeds, its task scheduling problem can be considered as the second kind of parallel machines for scheduling.

2.2 Scheduling Methods for Cloud

To develop more efficient scheduling algorithms for cloud, some recent reviews were given in [33], [34], which encompass non-metaheuristic and metaheuristic scheduling algorithms, such as earliest-finish-time-based algorithm and genetic algorithm. Also given in [33] was a comparison sheet to classify the scheduling algorithms discussed there by methods, parameters, and factors of scheduling. In addition, the method presented in [35] uses feedback information to estimate the earliest finish time to dynamically adjust the resource allocation.

Among researches on non-metaheuristic scheduling algorithms, the earliest start time, latest start time, and makespan [36], [37] are usually used to evaluate the scheduling results; however, to precisely describe cloud environment, additional measurements (objective functions) are also used in describing and measuring the scheduling on cloud. For instance, the cost of workflow and the deadline defined by a user were used in [38] while the price-based model was used in [39]. Besides the additional measurements, another study [36] combined the static and dynamic scheduling algorithms into a cloud system to adjust its resource allocation strategies on the fly. The main advantages of this kind of algorithms are: (1) they are simple and easy to implement; (2) some rule-based deterministic algorithms can find acceptable solutions quickly; and (3) most of them are compatible to each other; thus, some studies have tried to integrate two or more non-metaheuristic algorithms to solve the scheduling problem. Theoretically, one of the major disadvantages of these algorithms is in that the results obtained by these algorithms may be far from optimal or even acceptable. In practice, our observation shows that because they are simple and easy to implement on a cloud system, several cloud systems use traditional scheduling algorithms to manage the computing resources, such as Hadoop, which uses first in first out (FIFO) [40] as the default scheduling algorithm.

Although the FIFO scheduler of Hadoop can achieve a high utilization as far as a cluster is concerned, it provides no fairness at all. Even worse, it sometimes leads to a long response time, especially when the number of jobs to be run is beyond the available computing resources. For a concrete example, consider the following scenario: A very "long" job arrives the job queue followed by a "short" job with the same priority. The FIFO scheduler of Hadoop will schedule the first job and allocate all the resources it needs. This implies that the second job must wait until the first job is completed. The result is obviously a very long response time for the second job. For a cluster that has enough capacity to run all the jobs at the same time, most schedulers can satisfy the demand of the input jobs and tasks. It means that each job can be run and completed before the deadline. Assume a high map-capacity Hadoop cluster has 200 maps. Most of the schedulers can finish the jobs when there are 10 input jobs each of which use 20 maps, i.e., when there are total $10 \times 20 = 200$ tasks to be assigned to the 200 map slots. When the input jobs and tasks are too large to be run by the cluster (system) at the same time, the overall computation time will differ because different scheduling strategies may decide to schedule the jobs to be run and the resources to be allocated differently. Moreover, most, if not all, of the schedulers shipped with Hadoop today do not take into consideration the so-called makespan—the objective hyper-heuristic scheduling algorithm (HHSA) described herein is trying to improve.

To solve this problem, Facebook developed Hadoop Fair Scheduler (HFS) and improved it with Delay Scheduling [41] to get better data locality and throughput. HFS calculates each task's "fair share," sorts them in increasing order, and then picks the one with least fair share when there are empty slots. In the scenario above, HFS will schedule the shorter job first when it receives low fair share, thus finishing the shorter job before the longer one completes, by sacrificing only a little delay for a dramatically faster response time of the second job. On the other hand, Yahoo developed Capacity Scheduler [42] to deal with the same scheduling problem. By guaranteeing the capacity of the queues for different jobs, Capacity Scheduler can also achieve fairness, but it requires a more sophisticated control over the cluster. In summary, these two schedulers are not focused on pursuing the optimal solution; rather, they try to enhance the response time of each job. As such, this paper is aimed at demonstrating that by estimating and predicting the processing speed and completion time of each machine and each job, HHSA can generate feasible solutions that optimize the makespan not taken into consideration by the other schedulers. In addition, our experiments show that HHSA can eventually provide better solutions in terms of makespan.

In addition to non-metaheuristic scheduling algorithms, an alternative is the heuristic scheduling algorithms [9] that can be used to provide better scheduling plans than do the rule-based scheduling algorithms for cloud computing. As shown in Fig. 1a, the basic idea of heuristics is to use three key operators—*transition*, *evaluation*, and *determination*—to "search" for the possible solutions on the convergence process. Note that in Fig. 1a, t denotes the iteration number; $t_{\max}$ the maximum number of iterations or the stop criteria. More precisely, the transition operator creates the solution s, by using methods which could be either perturbative or constructive or both [43]; the evaluation operator measures

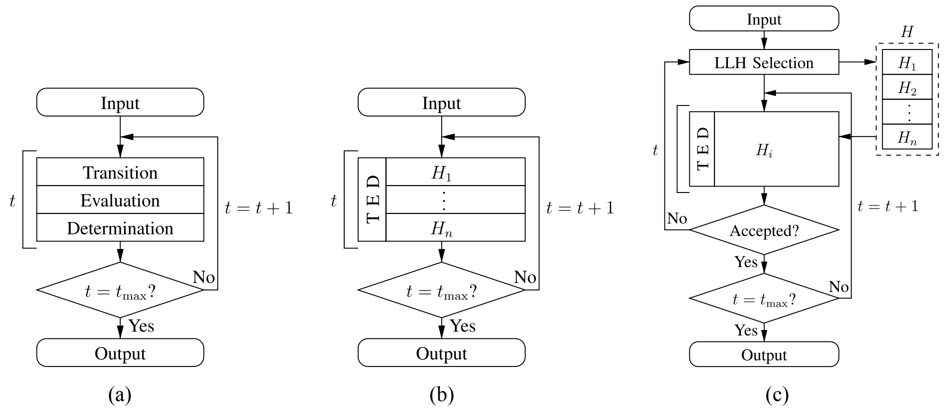


Fig. 1. Outline of (a) heuristics, (b) hybrid-heuristics, and (c) hyper-heuristics.

the fitness of s by using a predefined measurement; and then the determination operator determines the next search directions based on the s from the transition operator and the evaluation operator. Because heuristics use "tactical guess" to find the possible solutions, it has a much better chance to find a better result than do most rule-based deterministic algorithms.

3 Integration of Heuristic Algorithms

As we mentioned in Section 2.2, heuristic algorithms are able to provide better results than rule-based algorithms for complex scheduling problems on cloud computing systems. Moreover, as far as this paper is concerned, by the complex scheduling problems, we mean scheduling problems that are NP-complete [8], thus having a huge solution space for which all the solutions cannot be checked by any known exhaustive algorithm in a reasonable time, not only in a traditional information system (non-cloud virtualized cluster) but also in a cloud system. Each heuristic algorithm, however, has its pros and cons as far as scheduling is concerned. For example, ant colony optimization may provide a better scheduling solution than other scheduling algorithms, but it takes much more computation time than the others. An alternative to create a scheduling algorithm is by integrating two or more heuristic algorithms or combining heuristic algorithms with rule-based algorithms, Generally speaking, an integration that invokes two or more heuristic algorithms at each iteration reflects the complementary advantages of these algorithms to find a better result than a single heuristic algorithm does. As shown in Fig. 1b, the basic idea of hybrid-heuristic algorithm is to combine heuristic algorithms to perform the transition (T), evaluation (E), and determination (D) at each iteration, where H_i denotes one of the heuristic algorithms. This kind of integration may compensate for the intrinsic weak points of specific heuristic algorithms. For instance, the search diversity of tabu search may be increased by integrating it with genetic algorithm or other heuristics. However, a critical problem is that although the hybrid-heuristic may have a higher chance to find a better result than a single heuristic does, it generally takes a longer computation time than heuristics at each iteration of the convergence process.

Recently, another way to combine two or more heuristic algorithms, called hyper-heuristic algorithm [44], has attracted many researchers from different domains. As shown in Fig. 1c, one of the heuristic algorithms in the pool of candidate heuristics $\{H_1, H_2, \ldots, H_n\}$ will be selected by the low-level heuristic selection operator as the heuristic algorithm to be performed, which is referred to as H_i in Fig. 1c. For this kind of heuristics, low-level heuristic selection and acceptance are the two essential operators to determine which heuristic algorithm is to be selected and whether the selected heuristic algorithm will be continuously used or not. For this reason, several LLH selection operators have been presented in different studies [45] to choose the H_i. Different from the LLH selection operator that plays the role of determining the heuristic algorithm, the acceptance operator plays the role of determining the timing to select a new heuristic algorithm, such as when the H_i gets stuck with a solution for a certain number of iterations or when the H_i falls into local optimum.

In spite of all these extensions, there still lacks a simple and efficient design for scheduling on cloud today. For example, although most rule-based scheduling algorithms are very simple, they are unable to find the optimal or approximate solution in a reasonable time because most scheduling problems are NP-complete. Now, let us look back at the heuristic algorithms. Like the hybrid-heuristic, hyper-heuristic also attempts to use two or more heuristics on the convergence process. Unlike the hybrid-heuristic, the basic idea of hyper-heuristic is to use "one and only one" heuristic algorithm at each iteration. With these characteristics, hyper-heuristic algorithms can then maintain a high search diversity to increase the chance of finding better solutions at later iterations while not increasing the computation time.

The time complexities of heuristic, hybrid-heuristic, and hyper-heuristic are definitely an important issue, especially a comparison between the time complexities of these approaches. As described in [46], the expected running time of heuristic is $O(M \log M)$ where M indicates the number of subsolutions of each solution of the problem. Tseng and Yang in a later study [47] pointed out that the time complexity of heuristic is $O(\ell N M^2)$ where ℓ indicates the number of iterations the algorithm performs; N the population size; and M the number of subsolutions. In [48], the time complexity of hybrid-heuristic algorithm for scheduling was given. For each iteration, the computation of makespan takes $O(mn)$, simulated annealing takes $O(mn^2)$, and

```
1  Set up the parameters.
2  **Input** the scheduling problem.
3  Initialize the population of solutions $Z = \{z_1, z_2, ..., z_N\}$.
4  Randomly select a heuristic algorithm $H_i$ from the candidate pool $H$.
5  While the termination criterion is not met
6      Update the population of solutions $Z$ by using the selected algorithm $H_i$.
7      $F_1$ = Improvement_Detection($Z$).
8      $F_2$ = Diversity_Detection($Z$).
9      If $\psi(H_i, F_1, F_2)$
10         Randomly select a new $H_i$.
11         $Z$ = Perturb($Z$).
12     End.
13 End.
14 **Output** the best so far solution as the final solution.
```

Fig. 2. Hyper-heuristic scheduling algorithm.

Nawaz, Enscore, and Ham (NEH) takes $O(mn^3)$, where m denotes the number of machines; and n the number of jobs. Thus, the time complexity of hybrid-heuristic algorithm is $O(\ell mn^3)$. This is due to the fact that the NEH operator takes $O(mn^3)$. Putting these observations together, the time complexity of hybrid-heuristic algorithm can be obtained as $O(\ell M N^2)$, assuming that (1) $m \times n$ of the hybrid-heuristic algorithm equals M of the heuristic algorithm, which is usually true for the scheduling problems,[1] and (2) the time complexity of all the other operators of heuristic algorithms are bounded from above by NM^2. Similar to the hybrid-heuristic algorithm, the hyper-heuristic algorithm also uses two or more heuristic algorithms, but the time complexity of each operator is usually bounded from above by $O(MN^2)$. As a result, the time complexity of the hyper-heuristic algorithm is also $O(\ell MN^2)$ if all the operators (or heuristic algorithms) used are bounded from above by MN^2.

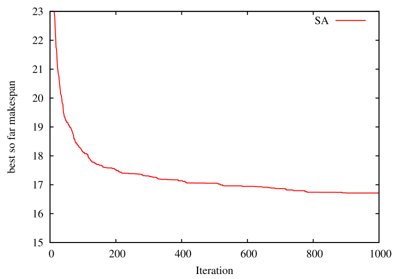
4 The Proposed Algorithm

4.1 Hyper-Heuristic Scheduling Algorithm

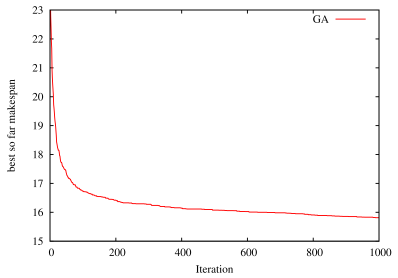
The basic idea of the proposed algorithm is to use the diversity detection and improvement detection operators to balance the intensification and diversification in the search of the solutions during the convergence process. The proposed algorithm, called Hyper-Heuristic Scheduling Algorithm, is as shown in Fig. 2. Line 1 sets up the parameters $\phi_{\max}$ and ϕ_{ni}, where $\phi_{\max}$ denotes the maximum number of iterations the selected low-level heuristic algorithm is to be run; ϕ_{ni} the number of iterations the solutions of the selected low-level heuristic algorithm are not improved. Line 2 reads in the tasks and jobs to be scheduled, i.e., the problem in question. Line 3 initializes the population of solutions $Z = \{z_1, z_2, \dots, z_N\}$, where N is the population size. On line 4, a heuristic algorithm H_i is randomly selected from the candidate pool $H = \{H_1, H_2, \dots, H_n\}$.

As far as the proposed algorithm described herein is concerned, the low-level heuristic candidate pool consists of simulated annealing [10], genetic algorithm [11], particle swarm optimization [12], and ant colony optimization [13]. The selected hyper-heuristic algorithm (LLH) H_i will then be performed repeatedly until the termination criterion is met, as shown in lines 5-13. More precisely, the selected LLH H_i will evolve the solution Z for $\phi_{\max}$ iterations by using the determine function $\psi(H_i, F_1, F_2)$, as defined in Eq. (3), to balance the intensification and diversification of the search directions, which in turn rely on the information provided by the improvement detection operator denoted F_1 (as shown in line 7) and by the diversity detection operator denoted F_2 (as shown in line 8) to decide whether to select a new LLH or not. For single-solution-based heuristic algorithms (SSBHA), only F_1 is used whereas for population-based heuristic algorithms (PBHA), both F_1 and F_2 are used.

1. This means that the solution is encoded as a matrix of size m by n, i.e., a matrix with m rows and n columns, where each row corresponds to a machine; each column corresponds to a job. Therefore, the number of subsolutions of each solution is $m \times n = M$.



(a) Best so far makespan of SA.



(b) Best so far makespan of GA.

Fig. 3. Results of applying heuristic algorithms to the j30 data set for the workflow problem on CloudSim.

$$\psi(H, F_1, F_2) = \begin{cases} \text{false}, & \text{if } H \in \mathbf{S} \text{ and } F_2 = \text{true}, \\ \text{false}, & \text{if } H \in \mathbf{P} \text{ and } F_1 = \text{true and } F_2 = \text{true}, \\ \text{true}, & \text{otherwise}, \end{cases} \tag{3}$$

where $\mathbf{S}$ denotes the set of SSBHAs; $\mathbf{P}$ the set of PBHAs. When the determine function $\psi(H_i, F_1, F_2)$ returns a true, the proposed algorithm will randomly select a new heuristic algorithm H_i and then return the solution Z to the perturbation operator to fine-tune the result, as shown in lines 9-11. Note that the single-solution-based heuristic algorithms employ one and only one search direction at each iteration on the convergence process while the population-based heuristic algorithms employ multiple search directions at each iteration.

4.1.1 *The Improvement Detection Operator*

A simple random method is used to select the low-level heuristic H_i from the candidate pool H. According to our observation, the best so far makespan (BSFMK) for both SA and GA could continue to improve the results at early iterations (e.g., less than 200 iterations), but it is difficult to improve the results at later iterations (e.g., after 800 iterations), especially when the search directions converge to a small number of directions, as shown in Fig. 3.

From these observations, a simple approach to detecting when to select a new LLH is as given below:

$$F_1 = \begin{cases} \text{false}, & \text{BSFMK is not improved after } t_{ni} \text{ iterations}, \\ \text{true}, & \text{otherwise}. \end{cases} \tag{4}$$

This approach only checks to see whether the solutions found by H_i, i.e., BSFMK, are being improved or not. If the selected H_i cannot improve the BSFMK after a row of ϕ_{ni} iterations, the improvement detection operator will return a false value to the high level hyper center to indicate that it should pick up a new LLH. Note that the improvement detection operator will return a false value in three cases: the maximum number of iterations $\phi_{\max}$ is reached, the number of iterations ϕ_{ni} the solutions are not improved is reached, and when the stop condition is reached.

4.1.2 The Diversity Detection Operator

In addition to the improvement detection operator, the diversity detection operator is used by HHSA to decide "when" to change the low-level heuristic algorithm H_i. Here is how it works. The diversity of the initial solution $D(Z_0)$ will be used as a threshold ω, i.e., $\omega = D(Z_0)$. The diversity of the current solution $D(Z)$ is computed as the average of the task distances between individual solutions which are defined as follows: If a task in two different individuals is assigned to the same VM, the task distance is 0; otherwise, the task distance is 1.

If the diversity of the current solution $D(Z)$ is less than the threshold ω (the diversity of the initial solution), this operator will return false, and HHSA will then randomly select a new LLH, as given below:

$$F_2 = \begin{cases} \text{true}, & D(Z) > \omega, \\ \text{false}, & \text{otherwise}. \end{cases} \tag{5}$$

Note that ω is defined as $\mu^1 - 3 \times \sigma^1$ where μ^1 and σ^1 denotes, respectively, the average distance and the standard deviation of the distances of the initial solution.

4.1.3 The Perturbation Operator

In addition to the perturbation operators of H_i itself, the proposed algorithm will perturb the solutions obtained by H_i before they are passed on to the newly selected LLH. This means that the candidate solutions created by the low-level heuristic H_i can be perturbed by the simulated annealing mechanism we presented herein, by assigning a different temperature to each individual to balance the intensification and diversification of the search. More precisely, the perturbation temperature T_k is used to determine the probability of randomly changing the subsolutions, such as changing the first subsolution of the solution 0001 from 0 to 1, thus obtaining the solution $1{,}001$. This mechanism implies that the higher the temperature, the higher the opportunity to escape from the local search space to find better solutions. More precisely, the perturbation temperature T_k is defined as

$$T_k = \begin{cases} T_{\max} \cdot \dfrac{\phi_{\max} - \phi}{\phi_{\max}} \cdot \dfrac{f(z_{\text{best}}) - f(z_k)}{f(z_{\text{best}}) - f(z_{\text{worst}})}, & \text{if } H_i \in \mathbf{P}, \\ T_{\max} \cdot \dfrac{\phi_{\max} - \phi}{\phi_{\max}} \cdot \dfrac{1}{N}, & \text{otherwise}, \end{cases} \tag{6}$$

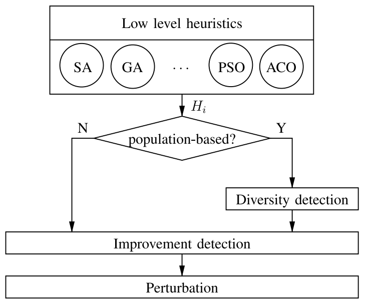


Fig. 4. Example showing how HHSA works.

where T_k $(1 \leq k \leq N)$ denotes the temperature assigned to the kth individual, $T_{\max}$ the maximum temperature, $\phi_{\max}$ the maximum number of iterations LLH will be performed, and ϕ the number of iterations LLH is actually performed. Also, $f(\cdot)$ is the fitness function; z_k, z_{best}, and z_{worst} are, respectively, the finesses of the kth solution, the best fitness of the population, and the worst fitness of the population. If the low-level heuristic is an SSBHA, it implies that no population information will be available. In this case, the perturbation operator will only use the number of iterations to decide the temperature of the current best individual from the population of LLH which will be passed to the newly selected LLH.

4.2 An Example

A simple example is given in Fig. 4 to depict how HHSA works. As this example shows, there are two cases to deal with: one is SSBHAs while the other is PBHAs. If the randomly selected algorithm is an SSBHA, such as simulation annealing (SA), then it will be evolved by using the improvement detection operator to check if there is an improvement in the fitness of the current solution. On the contrary, if a PBHA is chosen, it is the diversity of the current solutions (i.e., the solutions in the population) that will be checked by using the diversity detection operator before the improvement detection operator is applied. After that, the perturbation operator will be applied to either the current solution or the population before they are passed on to the newly selected LLH. This process will be repeated until the stop condition is reached. As shown in Fig. 4, the hyper-heuristic algorithm contains many low-level heuristic algorithms. After the candidate low-level heuristic algorithms to be in the pool are decided, the proposed algorithm will randomly select a low-level heuristic algorithm H_i from the pool. Then, if H_i is a PBHA, the proposed algorithm will first perform the diversity detection operator. Finally, for both the PBHA and SSBHA, the improvement detection and perturbation operators will be performed. The process will be repeated until a new low-level heuristic algorithm is selected.

5 RESULTS

In this section, two kinds of scheduling problems are used to evaluate the performance of the proposed algorithm. One is the workflow scheduling problem; the other is the Hadoop map-task scheduling problem.

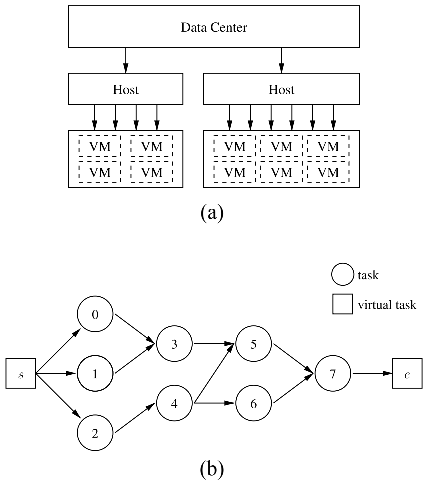


Fig. 5. Example showing how CloudSim works by assigning tasks to VMs. (a) virtual machines. (b) workflow tasks.

5.1 Workflow Scheduling on CloudSim

5.1.1 Data Sets and Parameter Settings

CloudSim [14] is a tool for emulating a cloud computing environment. In this study, CloudSim is used as the emulator in solving the workflow scheduling problem. As shown in Fig. 5a, CloudSim can be used to construct a data center with a set of virtual machines as the resource. Each task of a workflow can be started by assigning it to a VM once its parent tasks are completed. For example, as shown in Fig. 5b, task 3 will be started once its parent tasks 0 and 1 are completed. Besides, the starting and ending tasks, s and e, are virtual task and are not to be assigned to any VM. For more details and other workflow examples, readers are referred to [49] and [50].

The empirical analysis was conducted on an ASUS PC with 2.67 GHz Intel i7-920 CPU and 4 GB of memory running Fedora 12 with Linux 2.6.31 and using CloudSim to construct four virtual machines in a data center. The workflow data sets from the studies described in [51], [52] and PSPLIB [53] are employed to compare the performance of the proposed algorithm and other scheduling algorithms evaluated in this paper. Among these data sets, the e-Economic, fMRI, and protein annotation have, respectively, 10, 15, and 15 tasks [51], [52] while the j30, j60, and j90 have, respectively, 30, 60, and 90 tasks [53]. More precisely, fMRI is a data set for evaluating workflow that can be processed in parallel while e-Economic is a data set for evaluating workflow that can be processed sequentially. For a scheduling problem, the data set of [53] contains serial and parallel tasks. Each simulation was carried out 30 runs. The maximum number of iterations each run is set equal to 1,000. The population size is set equal to 50 for PBHAs. In HHSA, the maximum number of iterations of low-level algorithm $\phi_{\max}$ is set equal to 50, and the non-improved iteration threshold ϕ_{ni} is set equal to 5. The other parameter settings of the scheduling algorithms are as shown in Table 1.

TABLE 1
Parameter Settings of the Five Cloud Scheduling Algorithms for Workflow Scheduling on Cloudsim

Algorithm	Parameters
SA	temperature $t = 10$ temperature reduce rate $t_r = 0.9$
GA	mutation probability $m = 0.95$
ACO	pheromone updating fact $\rho = 0.1$ choosing probability $q_0 = 0.85$ related influence weights $\alpha = \beta = 1$
PSO	inertia weight $\omega = 0.8$ acceleration coefficient $c_1 = c_2 = 1.0$
HHSA	max iteration of low-level algorithm $\phi_{\max} = 50$ non-improved iteration threshold $\phi_{\text{ni}} = 5$

5.1.2 Simulation Results of Workflow Scheduling

To evaluate the performance of HHSA for the workflow scheduling problem, we compare it with two traditional rule-based algorithms and four heuristic algorithms, namely, min-min [54], max-min [54], simulated annealing [55], genetic algorithm [56], particle swarm optimization [57], and ant colony optimization [58]. For the data sets e-Economic, e-Protein, and fMRI, the results in Table 2 show that heuristics usually can find better results than traditional scheduling algorithms (i.e., min-min, max-min, and FIFO) in terms of makespan. In addition, the results of ACO are similar to those of the proposed algorithm for the first three data sets shown in Table 2. For the large data sets, j30, j60, and j90, the results of four heuristics (SA, GA, PSO, and ACO) are better than min-min, max-min, and FIFO. Moreover, the results also show that HHSA outperforms all the other scheduling algorithms, namely, min-min, max-min, FIFO, SA, GA, PSO, and ACO for the last three data sets shown in Table 2.

The results of best-so-far makespan given in Fig. 6 show that the min-min and max-min get almost exactly the same makespan in all iterations. For the SA, the higher the temperature which will get a larger fine-tune rate for the current solution, the poorer the best-so-far makespan compared to the other non-traditional algorithms at the early iterations. On the other hand, with a lower temperature, SA will converge quickly. For the population-based heuristic algorithms GA, PSO, and ACO, the search directions will converge in about 500 iterations. Fig. 6 also shows that HHSA converges faster than the other heuristic algorithms, and it usually takes no more than 200 iterations. In addition, because HHSA is able to automatically choose the suitable low-level heuristic algorithm and use the perturbation method to improve the results, the end results of HHSA are usually better than the other scheduling algorithms, especially for complex and large-scale problems. A closer look at these simulation results shows that the max-min algorithm does not scale well as the number of tasks increases. For example, for small data sets, the results depicted in Figs. 6a, 6b, and 6c show that max-min can provide a result that is close to the other heuristic scheduling algorithms in terms of the makespan (more precisely, the differences are no more than 3). However, as the number of tasks increases, the results described in Figs. 6d, 6e, and 6f show that the differences between max-min and other heuristic scheduling algorithms are also increased. For instance, Fig. 6d shows that the differences between max-min and other heuristic scheduling algorithms are more than five; Figs. 6e and 6f show that the differences between them are more than 10. On the other hand, for the min-min algorithm, the results given in Figs. 6d, 6e and 6f show that min-min is closer to

TABLE 2
Simulation Results of the Workflow Scheduling Algorithms Evaluated in this Study

dataset		min_min	max_min	FIFO	SA	GA	PSO	ACO	HHSA
e-Economic	Average	5.91	5.50	7.00	5.17	5.21	5.26	**5.16**	**5.16**
	Best	5.91	5.50	7.00	**5.16**	**5.16**	**5.16**	**5.16**	**5.16**
	Worst	5.91	5.50	7.00	5.25	5.41	5.75	**5.16**	**5.16**
	S.D.	0.00	0.00	0.00	0.02	0.09	0.13	**0.00**	**0.00**
e-Protein	Average	8.75	8.75	9.38	7.87	7.76	7.83	**7.50**	**7.50**
	Best	8.75	8.75	9.38	7.62	**7.50**	**7.50**	**7.50**	**7.50**
	Worst	8.75	8.75	9.38	8.25	8.25	8.5	**7.50**	**7.50**
	S.D.	0.00	0.00	0.00	0.17	0.27	0.28	**0.00**	**0.00**
fMRI	Average	10.0	10.0	10.7	9.74	9.64	9.66	**9.50**	**9.50**
	Best	10.0	10.0	10.7	**9.50**	**9.50**	**9.50**	**9.50**	**9.50**
	Worst	10.0	10.0	10.7	10.0	10.0	10.0	**9.50**	**9.50**
	S.D.	0.00	0.00	0.00	0.16	0.14	0.15	**0.00**	**0.00**
j30	Average	18.7	20.3	20.5	16.7	15.8	16.2	15.8	**15.4**
	Best	18.7	20.3	20.5	16.0	**15.0**	15.6	15.5	**15.0**
	Worst	18.7	20.3	20.5	17.5	16.5	17.0	16.2	**15.8**
	S.D.	0.00	0.00	0.00	0.35	0.35	0.39	0.18	**0.15**
j60	Average	37.5	41.1	41.5	35.9	33.2	34.2	32.7	**31.9**
	Best	37.5	41.1	41.5	34.0	31.5	32.3	31.6	**31.0**
	Worst	37.5	41.1	41.5	38.8	34.6	36.5	33.2	**32.5**
	S.D.	0.00	0.00	0.00	0.97	0.78	1.08	**0.34**	0.46
j90	Average	56.0	61.9	54.0	55.9	50.8	51.2	48.9	**48.0**
	Best	56.0	61.9	54.0	53.1	48.5	49.0	48.2	**47.0**
	Worst	56.0	61.9	54.0	59.8	53.7	54.6	49.8	**48.7**
	S.D.	0.00	0.00	0.00	1.71	1.20	1.44	**0.33**	0.46

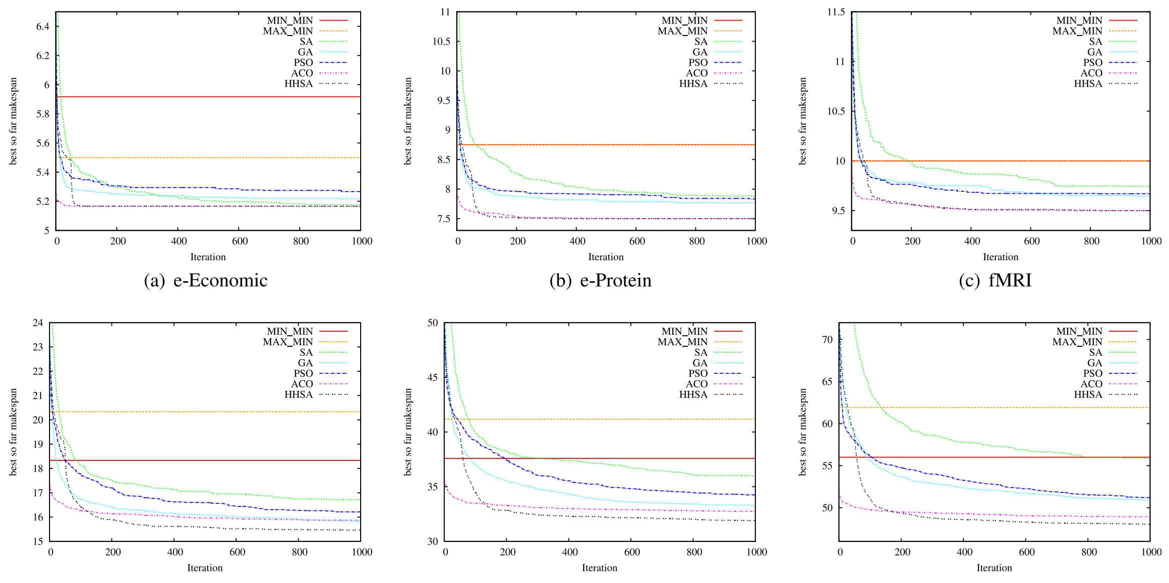
N.B.: The units of measurement for each column is in seconds. S.D. represents the standard deviation.

the other heuristic scheduling algorithm than max-min for large-scale data sets. Fig. 6 also provides the convergence information of the scheduling algorithms compared in this study. The results given in Fig. 6a show that the search process of these scheduling algorithms will converge to a stable state very quickly when the data set is small. A good example is the results of ACO and HHSA, which show that the result at iteration 100 is very close to the result at iteration 1,000. But the results depicted in Fig. 6b show that the search process of ACO and HHSA can still find a better result after iteration 100, meaning that the heuristic scheduling algorithms still have a chance to find a better result at later iterations when the data set becomes large. In Figs. 6c to 6f, this kind of results becomes quite clear. In addition, based on the observations of the convergence of these scheduling algorithms, we can easily know their performance. For example, for small data sets, the convergence speeds of ACO and HHSA described in Figs. 6a, 6b, and 6c are quite close to each other. However, as the number of tasks increases, the convergence speeds of ACO and HHSA are as shown in Figs. 6d, 6e, and 6f. They show that HHSA has a higher chance to find a better result than ACO does because it keeps finding a better result after the ACO has reached a stable state.

5.2 Hadoop Map-Task Scheduling

5.2.1 Data Sets and Parameter Settings

Another distributed system, Hadoop, provides slots as the resources to be allocated to the independent jobs, as shown in Fig. 7. Each job will be divided into multiple independent



(a) e-Economic (b) e-Protein (c) fMRI

(d) j30 (e) j60 (f) j90

Fig. 6. Convergence analysis by best-so-far makespan.

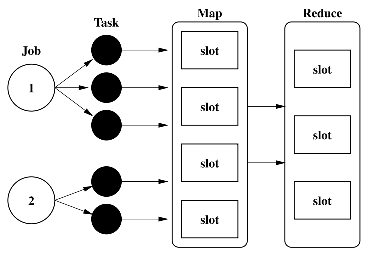


Fig. 7. Job executing on Hadoop.

map tasks according to a predefined map size, and the number of reduce tasks is given when submitting the jobs to the Hadoop cluster. Each scheduler is carried out for 30 runs, with Hadoop restarted at the end of each run. The maximum number of iterations each run is set equal to 50; the population size is set equal to 10 for PBHAs. The parameter settings of the scheduling algorithms of Hadoop are slightly different from those used in the simulations on CloudSim and are as shown in Table 3. By the nature of most evolutionary algorithms, the computation of fitness (makespan in this case) is required. In order to compute the makespan, the algorithm needs to know the job size and the processing speed. For the simulations, both attributes are given in the data set. But on Hadoop users can only acquire the size of each map task. The size of each reduce task remains unknown. Since the makespan of each reduce task cannot be predicted, only map task scheduling is applicable. Therefore, for the proposed algorithm, our focus is on scheduling tasks to map slots.

As mentioned previously, in addition to the size of each map task, we also need the processing speed of each task. To obtain this critical attribute, the algorithm first checks if the job exists in the record of processing speed. If not, one of the tasks in the job will be prioritized and scheduled immediately. After the map task is finished, we can divide the size of the map task by the processing time to get the speed which then can be used to calculate the processing time of the rest of the map tasks and to predict the total makespan.

In order to simulate the production environment, the job inter-arrival times in the experiment are roughly exponentially distributed with a mean of 14 seconds, as randomly sampled from a Facebook trace [41]. The data sets for the grep jobs on Hadoop are strings each of size 100 bytes and each on a line by itself randomly generated using a Python script. The sizes and submission periods are also randomly generated based on the statistics described in [41]. Fifty jobs are submitted in the experiment, and the total submission schedule is 13 minutes long. The sizes and the distributions of the jobs are as listed in Table 4. In this study, we also tried submitting larger jobs in a shorter period. More precisely, in the test, jobs to sort two 1 GB, two 3 GB, two 5 GB, and one 10 GB files are submitted in 1 minute.

TABLE 3
Parameters of the Five Cloud Scheduling Algorithms Evaluated

Algorithm	Parameters
SA	temperature $t = 10$ temperature reduce rate $t_r = 0.9$
GA	mutation probability $m = 0.95$
ACO	trail persistence $\rho = 0.8$ choosing probability $q_0 = 0.85$ relative importance $\alpha = \beta = 1$
PSO	inertia weight $\omega = 0.8$ acceleration coefficient $c_1 = c_2 = 1.0$
HHSA	the max number of iterations of LLH $\phi_{\max} = 50$ the max number of non-improved iterations $\phi_{\text{ni}} = 10$

TABLE 4
Data Set Generated for Benchmark

# Maps	% Jobs in Benchmark	# Jobs in Benchmark
1	56%	28
2	14%	7
3-20	16%	8
21-60	14%	7

In this study, Hadoop was running on both single-node (machine) and multi-node setups in a virtualized environment. The single-node setup is an Intel i7-3820 with 64 G RAM, which runs Gentoo Linux with kernel version 3.7.10. Hadoop was compiled from source version 1.1.3 using sun-jdk-1.6 toolchain. By properly configuring the map and task capacities of Hadoop, a single machine can still provide multiple working slots. For example, a machine with eight map task capacities is equivalent to a cluster consisting of four machines each of which provide two map task capacities. Meanwhile, each slot can be treated as an independent virtual machine in the Cloud environment, which can also be mapped to the virtual machines of CloudSim in the simulation.

For the multi-node setup, the Hadoop cluster was running on four virtual machines hosted on a cluster with AMD Opteron 6128 processors running XenServer. Each virtual machine has eight cores and 16 GB RAM and runs CentOS with kernel version 2.6.32. Hadoop was compiled from source version 1.1.3 using sun-jdk-1.6 toolchain. One of the four virtual machines is configured as the worker and job tracker node for scheduling jobs. The other three are configured as worker nodes. Each node is configured with four map slots, implying that there are total 16 map slots.

This study uses the map/reduce grep program distributed with Hadoop for the benchmark. The data sets are randomly generated so that each line contains 100 bytes of data. To evaluate the performance of HHSA, the same data sets are fed to FIFO and Fair Scheduler [41]. For larger data sets, TeraSort [59], which is also provided by Hadoop, is used to generate and sort the data. Moreover, another data set, retrieved from the log of 1998 World Cup website [60] provided by Internet Traffic Archive, for analyzing the web user access behavior is used to evaluate the performance of the proposed algorithm when it is applied to a real complex scheduling problem. In this experiment, the data are used by a map/reduce web log analyzer to calculate the reference counts of URLs.

5.2.2 *Simulation Results of Hadoop Map-Task Scheduling*

The experimental results of Hadoop are as shown in Table 5 and Table 6. For the grep jobs, the three schedulers give the similar results in both the single-node and multi-node setup. The results show that for the grep jobs, the Hadoop cluster was under-loaded, meaning that when new jobs arrived, previous jobs and tasks had already finished, thus making the scheduling results, i.e., the performance, of three schedulers similar to each other. This is justified by

TABLE 5
Performance of HHSA on Hadoop with Single-Node (in Seconds)

Task Type		FIFO	Fair Scheduler	HHSA
grep	Average	**858**	861	859
	Best	857	859	**850**
	Worst	**862**	864	866
	S.D.	**1.00**	1.41	5.19
Log Analyzer	Average	1065	1040	**1037**
	Best	1042	1037	**1018**
	Worst	1094	**1048**	**1048**
	S.D.	14.93	**2.65**	6.78
TeraSort	Average	904	756	**626**
	Best	838	710	**531**
	Worst	1024	833	**829**
	S.D.	43.54	**29.42**	66.98

the average task waiting time depicted in Fig. 10, to be discussed in later section.

When facing a burst, HHSA can make a better scheduling decision. For the TeraSort jobs, HHSA outperforms the other two schedulers. In the single-node setup, HHSA beats Fair Scheduler and FIFO by 17.19 and 30.75 percent, respectively, on average. In the multi-node setup, HHSA again outperforms the other two schedulers. On average, HHSA is 11.36 percent faster than FIFO and 4.56 percent faster than Fair Scheduler. As the results show, by optimizing the makespan, HHSA can provide a better scheduling decision in terms of the best-so-far makespan. The results of web log analyzer show that HHSA performs marginally better than FIFO, by 2.62 and 0.25 percent, respectively, in the single-node and multi-node setup. This is due to the fact that the web log analyzing job takes $O(n)$ time; thus, the time costs for all the tasks are very close to each other. Under this circumstance, FIFO can easily produce a good schedule that is close to that produced by the proposed algorithm. On the other hand, the time cost of each TeraSort task varies greatly, thus largely increasing the difficulty of scheduling. In this case, HHSA outperforms the other two schedulers significantly.

The performance of FIFO and HHSA on CloudSim and Hadoop is as shown in Table 7. For both CloudSim and Hadoop, a set of jobs the size of which are close to those of TeraSort benchmarks are generated and used. Then, the size of each task is randomly generated so that it falls in the range of $64\ \mathrm{MB} \pm 30\%$; each job will be divided into a set of smaller tasks of the same size though the processing time of each task is still not necessarily the same. When most tasks are of the same size, FIFO, which uses greedy strategy to scheduling tasks, can achieve a high utilization and good makespan. However, when the tasks are of different size, FIFO will become the worst method. Moreover, the simulation results also show that HHSA outperforms FIFO, especially when the problem is complex. The results are consistent with those of Table 2, where the sizes of jobs in the data sets for CloudSim are more diverse than those in this experiment.

TABLE 6
Performance of HHSA on Hadoop with Multi-Node (in Seconds)

Task Type		FIFO	Fair Scheduler	HHSA
grep	Average	**851**	852	853
	Best	848	**843**	848
	Worst	861	861	861
	S.D.	**2.23**	3.87	3.16
Log Analyzer	Average	795	802	**793**
	Best	752	749	**741**
	Worst	832	876	**829**
	S.D.	**21.07**	26.67	22.56
TeraSort	Average	1180	1096	**1046**
	Best	1077	1027	**932**
	Worst	1326	1249	**1178**
	S.D.	65.12	**46.97**	66.02

TABLE 7
Performance of HHSA on CloudSim and Hadoop (in Seconds)

Platform		FIFO	HHSA
CloudSim	Average	989	**952**
	Best	989	**937**
	Worst	989	**980**
	S.D.	0.00	12.29
Hadoop (Multi-Node)	Average	1180	**1046**
	Best	1077	**932**
	Worst	1326	**1178**
	S.D.	65.12	66.02

5.3 Analysis

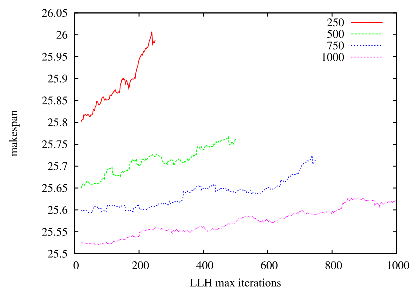
The maximum number of iterations of LLH $\phi_{\max}$ and the maximum number of non-improved iterations ϕ_{ni} set up for the workflow scheduling problem may not be suitable for the Hadoop map-task scheduling problem. As such, to evaluate the performance of HHSA for the Hadoop map-task scheduling problem, four data sets with a varying number of independent tasks, namely, 25 tasks, 50 tasks, 100 tasks, and 200 tasks, are used. Here, our focus is on the average makespan of 30 runs with $\phi_{\max}$ in the range of 20 to Ψ and ϕ_{ni} in the range of 1 to Ψ where Ψ is the maximum number of iterations of HHSA. This experiment is aimed at measuring the performance of HHSA when applying to the Hadoop map-task scheduling problem with the population size fixed at 50 for the PBHAs. This eventually takes into consideration all the combinations of the parameter settings shown in Table 8.

5.3.1 *The Maximum Number of Iterations of LLH*

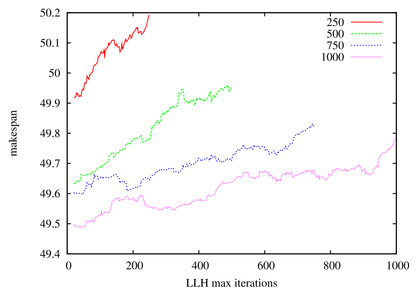
Fig. 8 shows the makespans for a varying maximum number of iterations of LLH $\phi_{\max}$ in the range $[20, \Psi]$ and a varying maximum number of iterations $\Psi \in \{250, 500, 750, 1{,}000\}$ for each data set. It is worth noting that the maximum number of non-improved iterations ϕ_{ni} is set equal to $\phi_{\max}$. This test is aimed to help us understand whether the value of $\phi_{\max}$ will influence the scheduling result. As the results in Fig. 8 show, the larger the number of iterations, the better the makespan. For instance, 250 iterations give a worse result than the other three cases. The results further show that the lower the maximum number of iterations of low-level algorithm $\phi_{\max}$, the better the makespan. Apparently, as the value of $\phi_{\max}$ approaches the maximum number of iterations, it implies that the probability of using LLH is getting smaller. In the worse case, only one of the LLH's will be used. That is, HHSA will degenerate to one of the low-level heuristics, which in turn will reduce the chance of balancing the intensification and diversification in the

TABLE 8
Parameter Settings for the Analysis

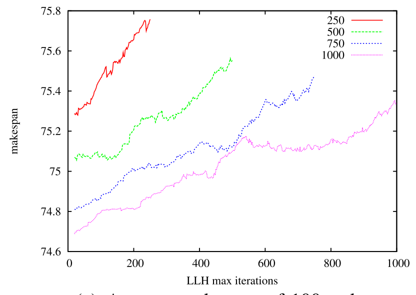
Parameter	Settings
max number of iterations (Ψ)	250, 500, 750, 1000
max number of iterations of LLH ($\phi_{\max}$)	$[20, \Psi]$
max number of non-improved iterations (ϕ_{ni})	$[1, \Psi]$



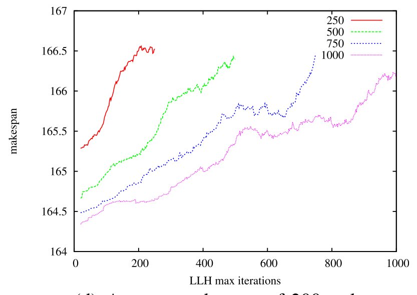
(a) Average makespan of 25 tasks



(b) Average makespan of 50 tasks

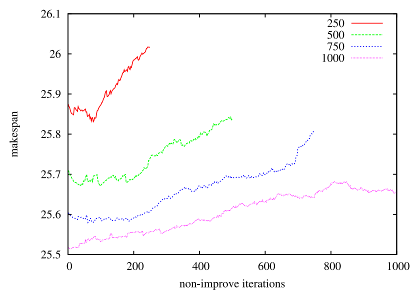


(c) Average makespan of 100 tasks

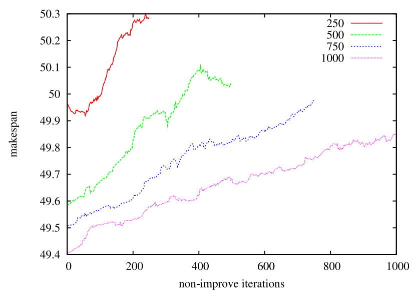


(d) Average makespan of 200 tasks

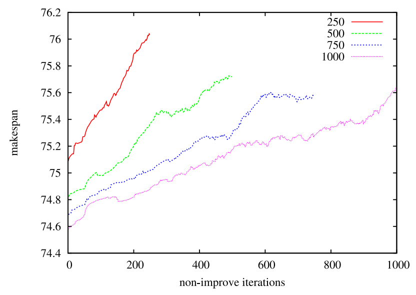
Fig. 8. Average makespan for a varying number of tasks and $\phi_{\max}$.



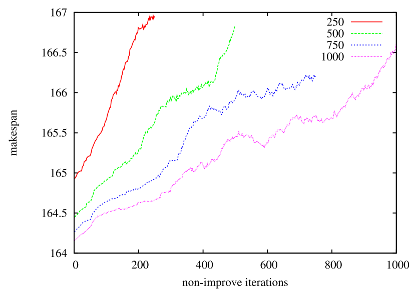
(a) Average makespan of 25 tasks



(b) Average makespan of 50 tasks



(c) Average makespan of 100 tasks



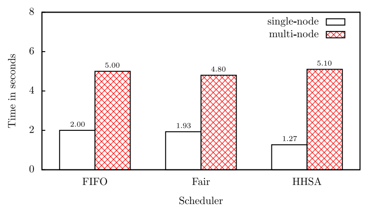
(d) Average makespan of 200 tasks

Fig. 9. Average makespans for a varying number of tasks and ϕ_{ni}.

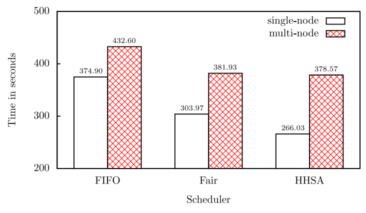
search of solutions. In addition, Figs. 8b, 8c and 8d show that for problems larger than that given in Fig. 8a, a similar result is obtained; that is, decreasing the value of $\phi_{\max}$ for the proposed algorithm can provide better results in most cases in terms of makespan. Consequently, as Fig. 8 shows, the smaller the value of $\phi_{\max}$, the better the makespan.

5.3.2 The Maximum Number of Non-Improved Iterations

Furthermore, Fig. 9 shows the makespans for a varying maximum number of non-improved iterations $\phi_{\text{ni}} \in \{1, 2, \dots, \Psi\}$ and a varying maximum number of iterations $\Psi \in \{250, 500, 750, 1{,}000\}$ for each data set. In order to eliminate the influence the maximum number of iterations of the selected LLH may have, the value of $\phi_{\max}$ is always set equal to the maximum number of iterations Ψ. When the maximum number of non-improved iterations is set equal to a small value, it means that if the best-so-far makespan of the selected LLH is not improved within the given number of iterations, the selected LLH will be replaced by another LLH. This implies that a smaller value of ϕ_{ni} will increase the diversity of the search directions of the selected LLH, thus improving the quality for not only the small problem shown in Fig. 9a but also the larger and complex problems depicted in Figs. 9b, 9c,



(a) grep



(b) TeraSort

Fig. 10. Average task waiting times for grep and TeraSort.

and 9d. In contrast, a larger value of ϕ_{ni} will increase the speed of convergence. Since the diversity and the quality of makespan may not be improved at the same time, our observation shows that a suitable value of ϕ_{ni} can not only get a better mean makespan, it also uses LLH more efficiently.

5.3.3 *The Average Task Waiting Time*

The average task waiting times for grep and TeraSort in the single-node and multi-node setup on Hadoop are as shown in Fig. 10. The small waiting times of grep given in Fig. 10a imply that the cluster is under-loaded and thus most tasks can be scheduled immediately. These results also explain that for small data sets in a cluster that is under-loaded, the performance of these scheduling algorithms is pretty much the same. On the other hand, the results of TeraSort described in Fig. 10b indicate that for large data sets in a cluster that is fully loaded, compared to FIFO, HHSA can reduce the average task waiting time of TeraSort by 29.04[2] and 12.49 percent,[3] respectively, in the single-node and multi-node setup. This means that by optimizing the makespan, tasks in the queue can be completed earlier; thus, incoming tasks can be scheduled sooner and the average task waiting time can be further reduced. Moreover, the results of HHSA for TeraSort on Hadoop show that HHSA takes only 0.0144 milliseconds on average, meaning that the computation time of HHSA is negligible in most cases.

5.3.4 *The Cloud Rental Payment*

In terms of makespan, on Hadoop, HHSA is about 11.36 percent faster than FIFO on average. The multi-node setup used in this paper is based on the "Standard On-Demand Extra Large Instances" of Amazon Elastic MapReduce service [61], which costs 0.12 US dollars per hour. Though the proposed algorithm described herein is aimed at optimizing the makespan instead of the budget. The reality is that by reducing the makespan, it also reduces the total cost of cloud service. According to the pricing and taking into account the improvement of the makespan, the proposed method can reduce $0.12 \times 11.36\% = 0.014$ US dollars per hour for each standard on-demand extra large instance. Although HHSA takes a longer computation time than the other schedulers, the difference is negligible because it is compensated by a better makespan, which will save much more time than HHSA takes.

2. $[(374.90 - 266.03)/374.90] \times 100 = 29.04\%$.
3. $[(432.60 - 378.56)/432.60] \times 100 = 12.49\%$.

5.4 Summary

As mentioned in Section 2.1.2, the characteristics of a cloud system are quite different from those of the information systems available today. A typical characteristic of a cloud system is in that the number of tasks it has to handle is much larger than that an information system available today has to handle, thus making it hard to schedule all the tasks. Another typical characteristic that affects the design of scheduling algorithms for a cloud system is in that a cloud system usually contains a number of virtual machines (or nodes) that add more uncertainties for task scheduling. Still another characteristic is in that a cloud system usually has many heterogeneous tasks to be scheduled, thus making it much more difficult to schedule than tasks on the information systems available today.

To solve these problems, a high-performance scheduling algorithm which can provide a better scheduling result in a reasonable time is needed. This explains why a hyper-heuristic-based scheduling algorithm is presented in this paper, which uses various search strategies to solve the scheduling problem of a cloud for providing a better result than traditional scheduling algorithms and other heuristic scheduling algorithms do. Because the design of the proposed algorithm is not for a particular cloud system, it can be applied to many applications that require task scheduling, such as science computing on Cloud. But the high latency inherent in the heuristic-based algorithms will make them unsuitable for the low-level scheduling of IaaS. Moreover, to the best of our knowledge, there exists no general and efficient way to automatically determine the number of resources that is used for each application for a very simple reason. The number of resources required may differ for different input to the application.

Compared to most traditional rule-based deterministic scheduling algorithms (e.g., min-min, max-min, and FIFO) that use only one search direction, the proposed algorithm uses multiple search directions at each iteration during the convergence process, thus having a higher chance to find a better result than the rule-based deterministic algorithms do. The results depicted in Sections 5.1.2 and 5.2.2 and Table 6 show that in terms of makespan, the proposed algorithm outperforms the traditional scheduling algorithms in most cases, for not only the single-node but also the multi-node data sets. Compared to the other heuristic scheduling algorithms, the proposed algorithm, again in terms of makespan, can still provide a better result than the other heuristic scheduling algorithms, in terms of not only the results of CloudSim but also the results of Hadoop, as shown in Sections 5.1.2 and 5.2.2. One of the reasons that can be easily justified is that HHSA leverages the strengths of many search strategies provided by different heuristic scheduling algorithms (low-level algorithms), thus making it more diverse in terms of the directions for which it searches during the convergence process, especially when compared to the other heuristic scheduling algorithms.

For the stability issue, HHSA is not designed for interconnectivity and locality because they will degrade the performance of Hadoop. As the authors of [62] pointed out, if the nodes of a virtualized Hadoop cluster actually reside in the same physical node, the overhead incurred for transferring data between these nodes is generally smaller than transferring the data between physical nodes. Hence, if the overhead of connection is too large but the scheduling problem is too small, the performance of the proposed algorithm will be degraded; otherwise, the performance of the proposed algorithm will be better than the other scheduling algorithms, especially when the scheduling problem becomes too complex or too large.

6 Conclusions

This study presents a high-performance hyper-heuristic algorithm to find better scheduling solutions for cloud computing systems. The proposed algorithm uses two detection operators to automatically determine when to change the low-level heuristic algorithm and a perturbation operator to fine-tune the solutions obtained by each low-level algorithm to further improve the scheduling results in terms of makespan. As the simulation results show, the proposed algorithm can not only provide better results than the traditional rule-based scheduling algorithms, it also outperforms the other heuristic scheduling algorithms, in solving the workflow scheduling and Hadoop map-task scheduling problems on cloud computing environments. In addition, the simulation results show further that the proposed algorithm converges faster than the other heuristic algorithms evaluated in this study for most of the data sets. In brief, the basic idea of the proposed "hyper-heuristic" algorithm is to leverage the strengths of all the low-level algorithms while not increasing the computation time, by running one and only one low-level algorithm at each iteration. This is fundamentally different from the so-called hybrid heuristic algorithm, which runs more than one low-level algorithm at each iteration, thus requiring a much longer computation time. Our experiences show that the proposed algorithm can be applied to different cloud computing systems to enhance the performance of scheduling problems. In the future, our focus will be on finding more effective detection operators and perturbation method to enhance the performance of the proposed algorithm. We will also attempt to apply HHSA to different research domains.

Acknowledgments

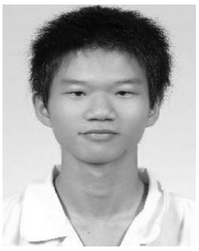
The authors would like to thank the editors and anonymous reviewers for their valuable comments and suggestions on the paper that greatly improve the quality of the paper. This work was supported in part by the National Science Council of Taiwan, R.O.C., under Contracts NSC102-2221-E-041-006, NSC102-2221-E-110-054, and NSC102-2219-E-006-001.

References

[1] M. Pinedo, *Scheduling: Theory, Algorithms and Systems*. Englewood Cliffs, NJ, USA: Prentice Hall, 1995.
[2] *Scheduling Theory and Its Applications*, P. Chrétienne, E. G. Coffman, J. K. Lenstra, and Z. Liu, eds., Hoboken, NJ, USA: Wiley, 1995.
[3] A. Allahverdi, C. T. Ng, T. C. E. Cheng, and M. Y. Kovalyov, "A survey of scheduling problems with setup times or costs," *Eur. J. Oper. Res.*, vol. 187, no. 3, pp. 985–1032, 2008.
[4] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, 2010.
[5] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *Proc. Grid Comput. Environ. Workshop*, 2008, pp. 1–10.
[6] K. Bilal, M. Manzano, S. Khan, E. Calle, K. Li, and A. Zomaya, "On the characterization of the structural robustness of data center networks," *IEEE Trans. Cloud Comput.*, vol. 1, no. 1, pp. 64–77, Jan.–Jun. 2013.
[7] X. Lei, X. Liao, T. Huang, H. Li, and C. Hu, "Outsourcing large matrix inversion computation to a public cloud," *IEEE Trans. Cloud Comput.*, vol. 1, no. 1, pp. 78–87, Jan. 2013.
[8] M. R. Garey and D. S. Johnson, *Computer and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA, USA: Freeman, 1979.
[9] C. W. Tsai and J. Rodrigues, "Metaheuristic scheduling for cloud: A survey," *IEEE Syst. J.*, vol. 8, no. 1, pp. 279–297, Mar. 2014.
[10] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.
[11] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. Ann Arbor, MI, USA: Univ. of Michigan Press, 1975.
[12] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," in *Proc. IEEE Int. Conf. Neural Netw.*, 1995, pp. 1942–1948.
[13] M. Dorigo and L. M. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 53–66, Apr. 1997.
[14] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw.: Practice Experience*, vol. 41, no. 1, pp. 23–50, 2011.
[15] J. Błażewicz, K. H. Ecker, E. Pesch, G. Schmidt, and J. Węglarz, *Scheduling Computer and Manufacturing Processes*. New York, NY, USA: Springer, 2001.
[16] Y. T. J. Leung, *Handbook of Scheduling: Algorithms, Models and Performance Analysis*. London, U.K.: Chapman & Hall, 2004.
[17] K. M. Elsayed and A. K. Khattab, "Channel-aware earliest deadline due fair scheduling for wireless multimedia networks," *Wireless Personal Commun.*, vol. 38, no. 2, pp. 233–252, 2006.
[18] W. H. Kohler, "A preliminary evaluation of the critical path method for scheduling tasks on multiprocessor systems," *IEEE Trans. Comput.*, vol. C-24, no. 12, pp. 1235–1238, Dec. 1975.
[19] R. Ingalls and D. Morrice, "PERT scheduling with resources using qualitative simulation graphs," in *Proc. Simul. Conf.*, 2000, vol. 1, pp. 362–370.
[20] H.-J. Schütz and R. Kolisch, "Approximate dynamic programming for capacity allocation in the service industry," *Eur. J. Oper. Res.*, vol. 218, no. 1, pp. 239–250, 2012.
[21] D. Shi and T. Chen, "Optimal periodic scheduling of sensor networks: A branch and bound approach," *Syst. Control Lett.*, vol. 62, no. 9, pp. 732–738, 2013.
[22] S. M. Johnson, "Optimal two- and three-stage production schedules with setup times included," *Naval Res. Logistics Quart.*, vol. 1, no. 1, pp. 61–68, 1954.
[23] M. R. Garey, D. S. Johnson, and R. Sethi, "The complexity of flowshop and jobshop scheduling," *Math. Oper. Res.*, vol. 1, no. 2, pp. 117–129, 1976.
[24] D. Daniel and S. J. Lovesum, "A novel approach for scheduling service request in cloud with trust monitor," in *Proc. Int. Conf. Signal Process., Commun., Comput. Netw. Technol.*, 2011, pp. 509–513.
[25] Z. Wu, X. Liu, Z. Ni, D. Yuan, and Y. Yang, "A market-oriented hierarchical scheduling strategy in cloud workflow systems," *The J. Supercomput.*, vol. 63, no. 1, pp. 256–293, 2013.
[26] B. Saovapakhiran, G. Michailidis, and M. Devetsikiotis, "Aggregated-DAG scheduling for job flow maximization in heterogeneous cloud computing," in *Proc. IEEE Global Telecommun. Conf.*, 2011, pp. 1–6.
[27] M. Rahman, X. Li, and H. Palit, "Hybrid heuristic for scheduling data analytics workflow applications in hybrid cloud environment," in *Proc. IEEE Int. Symp. Parallel Distrib. Process. Workshops*, 2011, pp. 966–974.

[28] Apache Hadoop. [Online]. Available: http://hadoop.apache.org, 2014.
[29] Y. M. Huang and Z. H. Nie. Cloud computing with Linux and Apache Hadoop. [Online]. Available: http://www.ibm.com/developerworks/aix/library/au-cloud_apache/, 2014.
[30] B. David. Open source cloud computing with Hadoop. [Online]. Available: http://www.linuxjournal.com/content/open-source-cloud-computing-hadoop, 2014.
[31] A. O'Driscoll, J. Daugelaite, and R. D. Sleator, "'Big data', Hadoop and cloud computing in genomics," *J. Biomed. Inf.*, vol. 46, no. 5, pp. 774–781, 2013.
[32] Y. Song, M. Wu, and L. Yang, "Study of smart grid marketing system architecture based on Hadoop platform of cloud computing," *J. Commun. Comput.*, vol. 9, pp. 741–743, 2012.
[33] A. Bala and I. Chana, "A survey of various workflow scheduling algorithms in cloud environment," in *Proc. Nat. Conf. Inf. Commun. Technol.*, 2011, pp. 26–30.
[34] N. Kaur, T. S. Aulakh, and R. S. Cheema, "Comparison of workflow scheduling algorithms in cloud computing," *Int. J. Adv. Comput. Sci. Appl.*, vol. 2, no. 10, pp. 81–86, 2011.
[35] J. Li, M. Qiu, Z. Ming, G. Quan, X. Qin, and Z. Gu, "Online optimization for scheduling preemptable tasks on IaaS cloud systems," *J. Parallel Distrib. Comput.*, vol. 72, no. 5, pp. 666–677, 2012.
[36] J. Li, M. Qiu, J. Niu, Y. Chen, and Z. Ming, "Adaptive resource allocation for preemptable jobs in cloud systems," in *Proc. Int. Conf. Intel. Syst. Design Appl.*, 2010, pp. 31–36.
[37] C. Lin and S. Lu, "Scheduling scientific workflows elastically for cloud computing," in *Proc. IEEE Int. Conf. Cloud Comput.*, 2011, pp. 746–747.
[38] S. Abrishami and M. Naghibzadeh, "Deadline-constrained workflow scheduling in software as a service cloud," *Scientia Iranica*, vol. 19, no. 3, pp. 680–689, 2012.
[39] Y. C. Lee, C. Wang, A. Y. Zomaya, and B. B. Zhou, "Profit-driven scheduling for cloud services with data access awareness," *J. Parallel Distrib. Comput.*, vol. 72, no. 4, pp. 591–602, 2012.
[40] K. Kousalya and P. Balasubramanie, "Ant algorithm for grid scheduling powered by local search," *Int. J. Open Problem Comput. Sci. Math.*, vol. 1, no. 3, pp. 222–240, 2008.
[41] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling," in *Proc. Eur. Conf. Comput. Syst.*, 2010, pp. 265–278.
[42] Yahoo!. Hadoop capacity scheduler. [Online]. Available: http://hadoop.apache.org/docs/r2.3.0/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html, 2014.
[43] H. H. Hoos and T. Stützle, *Stochastic Local Search: Foundations & Applications*. San Mateo, CA, USA: Morgan Kaufmann, 2004.
[44] P. Cowling and G. Kendall, "A hyperheuristic approach to scheduling a sales summit," in *Proc. Practice Theory Autom. Timetabling III*, 2001, pp. 176–190.
[45] P. I. Cowling, G. Kendall, and E. Soubeiga, "A hyperheuristic approach to scheduling a sales summit," in *Proc. Practice Theory Autom. Timetabling*, 2000, pp. 176–190.
[46] B. K. Ambati, J. Ambati, and M. M. Mokhtar, "Heuristic combinatorial optimization by simulated Darwinian evolution: A polynomial time algorithm for the traveling salesman problem," *Biol. Cybern.*, vol. 65, no. 1, pp. 31–35, 1991.
[47] L. Y. Tseng and S. B. Yang, "A genetic approach to the automatic clustering problem," *Pattern Recognit.*, vol. 34, no. 2, pp. 415–424, 2001.
[48] D. Laha and U. Chakraborty, "An efficient hybrid heuristic for makespan minimization in permutation flow shop scheduling," *The Int. J. Adv. Manuf. Technol.*, vol. 44, no. 5/6, pp. 559–569, 2009.
[49] L. Ramakrishnan, C. Koelbel, Y.-S. Kee, R. Wolski, D. Nurmi, D. Gannon, G. Obertelli, A. YarKhan, A. Mandal, T. M. Huang, K. Thyagaraja, and D. Zagorodnov, "VGrADS: Enabling e-science workflows on grids and clouds with fault tolerance," in *Proc. Conf. High Performance Comput. Netw., Storage Anal.*, 2009, pp. 47:1–47:12.
[50] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *Proc. Int. Conf. High Performance Comput., Netw., Storage Anal.*, 2011, pp. 1–12.
[51] Y. Zhao, M. Wilde, I. Foster, J. Voeckler, T. Jordan, E. Quigg, and J. Dobson, "Grid middleware services for virtual data discovery, composition, and integration," in *Proc. Workshop Middleware Grid Comput.*, 2004, pp. 57–62.
[52] A. O'Brien, S. Newhouse, and J. Darlington, "Mapping of scientific workflow within the e-protein project to distributed resources," in *Proc. UK e-Science All Hands Meeting*, Nottingham, U.K., 2004, pp. 404–409.
[53] R. Kolisch and A. Sprecher, "PSPLIB—A project scheduling problem library: OR software—ORSEP operations research software exchange program," *Eur. J. Oper. Res.*, vol. 96, no. 1, pp. 205–216, 1997.
[54] J. Blythe, S. Jain, E. Deelman, Y. Gil, and K. Vahi, "Task scheduling strategies for workflow-based applications in grids," in *Proc. IEEE Int. Symp. Cluster Comput. Grid*, 2005, pp. 759–767.
[55] S. Benedict and V. Vasudevan, "Scheduling of scientific workflows using simulated annealing algorithm for computational grids," *Int. J. Soft. Comput.*, vol. 2, no. 5, pp. 606–611, 2007.
[56] J. Yu and R. Buyya, "A budget constrained scheduling of workflow applications on utility grids using genetic algorithms," in *Proc. Workflows Support Large-Scale Sci.*, 2006, pp. 1–10.
[57] T. Chen, B. Zhang, X. Hao, and Y. Dai, "Task scheduling in grid based on particle swarm optimization," in *Proc. Int. Symp. Parallel Distrib. Comput.*, 2006, pp. 238–245.
[58] W. N. Chen and J. Zhang, "An ant colony optimization approach to a grid workflow scheduling problem with various QoS requirements," *IEEE Trans. Syst., Man, Cybern., Part C: Appl. Rev.*, vol. 39, no. 1, pp. 29–43, Jan. 2009.
[59] O. O'Malley. TeraByte Sort on Apache Hadoop. [Online]. Available: http://sortbenchmark.org/YahooHadoop.pdf, 2014.
[60] I. T. Archive. World Cup 98. [Online]. Available: http://ita.ee.lbl.gov/html/contrib/WorldCup.html, 2014.
[61] Amazon, Amazon elastic mapreduce. [Online]. Available: http://aws.amazon.com/elasticmapreduce, 2014.
[62] H. S. Shin, K. H. Kim, C. Y. Kim, and S. I. Jung, "The new approach for inter-communication between guest domains on virtual machine monitor," in *Proc. Int. Symp. Comput. Inf. Sci.*, 2007, pp. 1–6.

Chun-Wei Tsai received the MS degree in management information system from National Pingtung University of Science and Technology, Pingtung, Taiwan, in 2002, and the PhD degree in computer science from National Sun Yat-sen University, Kaohsiung, Taiwan, in 2009. He was also a postdoctoral fellow with the Department of Electrical Engineering, National Cheng Kung University, Tainan, Taiwan before joining the faculty of Applied Geoinformatics, and then the faculty of Information Technology, Chia Nan University of Pharmacy & Science, Tainan, Taiwan, in 2010 and 2012, respectively, where he is currently an assistant professor. His research interests include information retrieval, evolutionary computation, internet technology, and combinatorial optimization.



Wei-Cheng Huang received the BS degree in computer science and information engineering from the National Pingtung Institute of Commerce, Pingtung, Taiwan, in 2010, and the MS degree in computer science and engineering from National Sun Yat-sen University, Kaohsiung, Taiwan, in 2012. He is currently working toward the PhD degree in computer science and engineering from National Sun Yat-sen University. His current research interests include evolutionary computation, clustering, scheduling, and cloud computing.

Meng-Hsiu Chiang received the BS degree in computer science and engineering from National Sun Yat-sen University, Kaohsiung, Taiwan, in 2010. He is currently working toward the MS degree in computer science and engineering from National Sun Yat-sen University. His current research interests include scheduling, distributed systems, and cloud computing.

Ming-Chao Chiang received the BS degree in management science from National Chiao Tung University, Hsinchu, Taiwan, ROC, in 1978, and the MS, MPhil, and PhD degrees in computer science from Columbia University, New York, NY, in 1991, 1998, and 1998, respectively. He has more than 12 years of experience in the software industry encompassing a wide variety of roles and responsibilities in both large and start-up companies before joining the faculty of the Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung, Taiwan, ROC, in 2003, where he is currently an associate professor. His current research interests include image processing, evolutionary computation, and system software.

Chu-Sing Yang received the BS degree in engineering science, and the MS and PhD degrees in electrical engineering from National Cheng Kung University, Tainan, Taiwan, in 1976, 1984 and 1987, respectively. He joined the faculty of the Department of Electrical Engineering, National Sun Yat-sen University, Kaohsiung, Taiwan, as an associate professor, in 1988. Since 1993, he has been a professor of the Department of Computer Science and Engineering, National Sun Yat-sen University. He was a chair of Department of Computer Science and Engineering, National Sun Yat-sen University from August 1995 to July 1999, a director of Computer Center, National Sun Yat-sen University from August 1998 to October 2002, and a program chair of ICS-96 and Program Co-Chair of ICPP-2003. He joined the faculty of the Department of Electrical Engineering, National Cheng Kung University, Tainan, Taiwan, as a professor in 2006. His research interests include web mining, embedded system, network management, and wireless communication.