J. Parallel Distrib. Comput. 73 (2013) 1106–1115

Contents lists available at SciVerse ScienceDirect

J. Parallel Distrib. Comput.

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

An effective iterated greedy algorithm for reliability-oriented task allocation in distributed computing systems

Qinma Kang [a,b,*], Hong He [a], Jun Wei [b]

[a] School of Information Engineering, Shandong University, Weihai 264209, China

[b] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China

HIGHLIGHTS

- Reliability-oriented task assignment in distributed systems is investigated.
- A simple and effective IG algorithm is proposed for the problem.
- Comparison studies show the superiority of our algorithm over others.
- Convergence analysis is also conducted.

ARTICLE INFO

Article history:
Received 11 April 2012
Received in revised form
21 February 2013
Accepted 18 March 2013
Available online 26 March 2013

Keywords:
Distributed computing
Task allocation
System reliability
Iterated greedy algorithm
Meta-heuristics

ABSTRACT

This paper investigates the problem of allocating parallel application tasks to processors in heterogeneous distributed computing systems with the goal of maximizing the system reliability. The problem of finding an optimal task allocation for more than three processors is known to be NP-hard in the strong sense. To deal with this challenging problem, we propose a simple and effective iterative greedy algorithm to find the best possible solution within a reasonable amount of computation time. The algorithm first uses a constructive heuristic to obtain an initial assignment and iteratively improves it in a greedy way. We study the performance of the proposed algorithm over a wide range of parameters including problem size, the ratio of average communication time to average computation time, and task interaction density. The viability and effectiveness of our algorithm is demonstrated by comparing it with recently proposed task allocation algorithms for maximizing system reliability available in the literature.

1. Introduction

Distributed systems have become increasingly popular as cost-effective alternatives to traditional high performance computing systems. These systems often consist of many geographically distributed heterogeneous processors interconnected via communication networks. To effectively exploit such systems, an important challenge is to develop task allocation algorithms that assign each task partitioned from an application to its best suitable processor for parallel execution. In the literature, the task allocation problem has been extensively studied with the main concern on various performance criteria, such as minimizing the total sum of execution and communication time [8,27,31,34,37,41] or minimizing the application turnaround time [1,4,10,13,28,30]. Inherently, distributed systems are more complex than centralized ones. In such a large system, machine and network failures are inevitable and can have an adverse effect on applications executing on the system. Hence, ensuring the reliability of distributed systems is of critical importance along with task allocation, especially for some mission critical applications, such as aircraft control, industrial process control, and banking systems.

System redundancy is the traditional technique to achieve fault tolerance and thus reliability [7,9,11,12,16,19,33]. A distributed computing system is redundant if it possesses software redundancy (e.g. task replication among processing nodes) and/or hardware redundancy (e.g. multiple processors at a processing node, and multiple communication channels connecting each pair of processing nodes) [11]. However, this is an expensive approach. Moreover, in many situations, the system configuration is fixed and we have no freedom to introduce system redundancy. Hence, we have to turn to a task allocation strategy to achieve high system reliability. This paper investigates the task allocation problem with the aim of maximizing distributed system reliability without extra hardware and/or software costs.

Several approaches have been proposed in the literature for maximizing reliability of distributed systems without redun-

* Correspondence to: School of Information Engineering, Shandong University, 180 Wenhua Road, Weihai 264209, China.
E-mail addresses: qmkang@sina.com, qmkang@sdu.edu.cn (Q. Kang).

0743-7315/$ – see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jpdc.2013.03.008

dancy [5,15,17,21,29,38]. These allocation schemes may be broadly classified into two main categories. First, there are the exact methods that try to find the optimal allocation for the given objective. Currently, the most competitive exact algorithms are based on state space search techniques to determine the optimal solution [17,21,29]. However, the computational complexity of the reliability problem on distributed systems has been proved to be NP-hard in the strong sense [20]. These methods are exponential in nature and may demand heavy computation time. The other approach is to develop good heuristics to solve the problem. These techniques may give suboptimal results, but are much less expensive. They are useful in applications where an optimal solution is not obtainable within a critical time limit. The algorithms based on this approach can be further divided into the following two subcategories.

The greedy heuristics construct a feasible solution from scratch and the mapping of tasks to processors is done without backtracking. In these algorithms the tasks are first sorted on a given criterion and are then mapped in that order on the processors. The allocation of one task is dependent on the mapping of previously scanned tasks. During the solution construction these algorithms maintain feasibility, that is, they return allocations that respect resource constraints. Though the process of greedy heuristics is fast, the solution quality is not satisfactory [29].

The meta-heuristics start from an initial solution or a set of solutions and try to improve the solution(s) by utilizing some strategies. In recent years, a growing body of literature suggests the use of meta-heuristic search methods for the task allocation problem. For example, Vidyarthi and Tripathi applied a simple genetic algorithm (GA) to quickly find a near optimal allocation [35]. Attiya and Hamam developed a simulated annealing (SA) algorithm for the problem and evaluated its performance in comparison with branch-and-bound technique [5]. Yin et al. presented a hybrid algorithm that combines particle swarm optimization (PSO) with a hill climbing heuristic [38]. They claimed that their approach outperform a GA in terms of effectiveness and efficiency for the test-cases studied. Kang et al. proposed a honey bee mating optimization (HBMO) algorithm for tackling the problem [15]. Through simulations over a wide range of parameters, they compared their algorithm experimentally to other popular algorithms with favorable results.

Because of the intractable nature of the task assignment problem, new effective techniques are always desirable to obtain the best-possible solution in an efficient manner. The iterated greedy (IG) heuristic is an effective stochastic local search algorithm recently developed for combinatorial optimization problems. Though simple and easy to implement, the IG algorithm has exhibited state-of-the-art performances for several problems in computer science and engineering fields, such as set covering problems [22], flow shop scheduling problems [23,25,26], sequencing single-machine tardiness problems [40], dynamic parallel machine scheduling [39], just to name a few. In our previous work, the IG algorithm has also been successfully applied to task assignment problems where the goal is to minimize the total sum of execution and communication time [14]. Thus, we intend to further extend the application of IG to the task allocation problem for maximizing reliability of heterogeneous distributed computing systems.

It is noteworthy to mention that in their standard forms, each of these meta-heuristics has a unique search mechanism that allows them to find high quality solutions. As we will detail, IG works by iterating over greedy constructive heuristics. The proposed method does not belong to the larger class of population based evolutionary algorithms, which generate solutions to optimization problems using techniques inspired by natural evolution, such as fitness evaluation, selection, crossover and mutation.

The rest of this paper is organized as follows: In the next section, the problem that will be addressed in this paper is formulated.

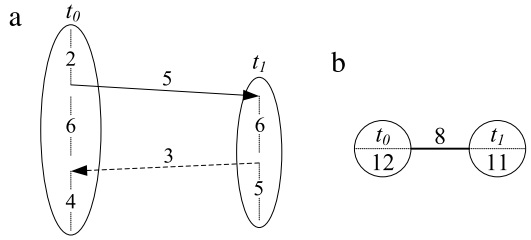

Fig. 1. (a) Examples of a program with two tasks, (b) TIG model for this program.

The proposed IG heuristic is elaborated in Section 3. The computational results of applying the proposed algorithm to the problem instances are provided in Section 4, as well as comparisons of the performance against three state-of-the-art meta-heuristics from the relevant literature. Finally, some concluding remarks are made in Section 5.

2. Problem formulation

Different task allocation models for reliability have been used in the literature. In this work, we follow [5,15,17,29,38] to formulate the task allocation problem for maximizing system reliability.

2.1. Problem statement

The distributed system is assumed to consist of a set of heterogeneous processors interconnected via high-speed networks in an arbitrary fashion. The processors may have different resource capacities and the communication links may have different bandwidths. A failure rate is associated with each component (processor or communication link) in the system. Failures of components are assumed to be statistically independent and follow a Poisson distribution. We also assume the target system contains dedicated communication hardware so that computation can be overlapped with communication.

In the distributed system, an application to be assigned is modeled as an undirected graph called task interaction graph (TIG), where the vertices represent the tasks partitioned from the application and the undirected edges represent the inter-task communication demands. In this model, temporal dependencies in the execution of tasks are not explicitly addressed: all the tasks are considered simultaneously executable and communications can take place either at any time or intermittently during the program execution. The TIG model is well suited to modeling coarse-grained applications that alternate computation phases and communications inside tasks [24]. Fig. 1(a) shows an example of a program with two tasks, t_0 and t_1, which are composed of different computation phases and communications, as represented inside each task. The corresponding TIG model is shown in Fig. 1(b).

Due to the heterogeneity of processors, a task may take different execution time if it is executed on different processors. A vector is associated with each task representing the expected execution time of the task on different processors in the system, where these values depend on the work to be performed by the task as well as on the attributes of processors and can be obtained based on task profiling and analytical benchmarking [3,18,36]. The execution of a task will consume a specific amount of resource from its assigned processor. Similarly, if two communicating tasks are executed on different processors, the inter-task communication may take different time if transmitted through different communication paths. If they are assigned to the same processor, the communication time is assumed to be negligible since the communication via shared-memory is much faster than that through message passing over the network. The goal is concerned with finding a task allocation that

maximizes the system reliability and satisfies all of the resource constraints.

The notations that will be used in our problem formulation are listed below:

- N The number of tasks forming an application
- K The number of processors available in the distributed system
- t_i ith task of an application
- p_k kth processor in the distributed system
- X An $N \times K$ binary matrix corresponding to a task allocation
- x_{ik} An element of allocation matrix X; $x_{ik} = 1$ if task t_i is assigned to processor p_k, and $x_{ik} = 0$ otherwise
- e_{ik} The expected execution time of task t_i on processor p_k
- d_{ij} The data quantity to be transmitted between tasks t_i and t_j measured in some units
- w_{kl} The transmission rate of communication path between terminal processors p_k and p_l
- λ_k The failure rate of processor p_k
- μ_{kl} The failure rate of communication path between terminal processors p_k and p_l
- m_i The resource requirement of task t_i from its execution processor
- M_k The available resource capacity of processor p_k

2.2. Reliability model

The reliability of a distributed system is defined as the probability that the system can execute the entire application successfully [5,29,38]. That is, the system reliability is the product of the probability that each processor is operational during the time of processing the tasks assigned to it, and the probability that each communication path is operational during the active period of data communication between the terminal processors of the path. Hence, successful execution of an application is mainly dependent on reliability of the system components and on the distribution of the application tasks to the available processors in the system. In the following we first present reliability expressions to the system components and then provide an explicit objective function to the system reliability.

The reliability of processor p_k during a time interval t is $e^{-\lambda_k t}$ [5,29]. Under a task allocation X, the time required for the execution of the tasks assigned to processor p_k is $\sum_{i=1}^{N} x_{ik}e_{ik}$, and then the corresponding processor reliability can be formulated as

$$R_k(X) = e^{-\lambda_k \sum_{i=1}^{N} x_{ik}e_{ik}}. \tag{1}$$

Similarly, the reliability of the communication path connecting processors p_k and p_l during a time interval t is $e^{-\mu_{kl}t}$ [5,29]. Under a task allocation X, the time required for data communication between the terminal processors p_k and p_l is $\sum_{i=1}^{N}\sum_{j\neq i} x_{ik}x_{jl}d_{ij}/w_{kl}$, then the corresponding path reliability is given by

$$R_{kl}(X) = e^{-\mu_{kl}\sum_{i=1}^{N}\sum_{j\neq i} x_{ik}x_{jl}d_{ij}/w_{kl}}. \tag{2}$$

As the system reliability requires that all involved components are operational during the elapsed time for executing an application, the system reliability with the task allocation X is computed as follows:

$$R(X) = \prod_{k=1}^{K} R_k(X) \prod_{k=1}^{K}\prod_{k\neq l} R_{kl}(X) = e^{-Z(X)} \tag{3}$$

where

$$Z(X) = \sum_{i=1}^{N}\sum_{k=1}^{K} \lambda_k x_{ik}e_{ik} + \sum_{i=1}^{N}\sum_{j\neq i}\sum_{k=1}^{K}\sum_{k\neq l} \mu_{kl}x_{ik}x_{jl}d_{ij}/w_{kl}. \tag{4}$$

The first term of the function $Z(X)$ reflects the unreliability caused by the execution of tasks on processors of various reliabilities and the second term reflects the unreliability caused by the inter-processor communication through different paths of various reliabilities.

Maximizing the system reliability is equivalent to minimizing $Z(X)$. With system resource constraints taken into account, the task allocation model for system reliability is formulated as follows:

$$\text{Minimize } Z(X) \tag{5}$$

s.t.

$$\sum_{k=1}^{K} x_{ik} = 1 \quad \forall i = 1, 2, \ldots, N \tag{6}$$

$$\sum_{i=1}^{N} m_i x_{ik} \leq M_k \quad \forall k = 1, 2, \ldots, K \tag{7}$$

$$x_{ik} \in \{0, 1\} \quad \forall i, k. \tag{8}$$

Constraint (6) states that each task should be assigned to exactly one processor. Constraint (7) ensures that the total resource requirements of the tasks assigned to each processor must not exceed its resource capacity. Constraint (8) guarantees that x_{ik} is a binary variable.

The model above defines a 0–1 integer programming problem with a quadratic objective and it is known to be NP-hard [20]. Although exact algorithms such as A^* algorithms and branch and bound have been proposed [17,21,29], they are computationally very expensive. An alternative is to find high quality solutions efficiently using meta-heuristics. In this paper, we propose a new IG algorithm for this purpose.

3. Iterated greedy algorithm

The IG algorithm starts from an initial solution and then tries to improve the incumbent solution by iterating over three main phases: destruction, construction and acceptance criterion. During the first phase, some components are randomly extracted from the incumbent solution. In the second phase, a new complete candidate solution is reconstructed using a greedy constructive heuristic; hopefully, the new solution is better than the one obtained from the previous iteration. Afterwards, an acceptance criterion is used to decide whether the newly constructed solution will substitute the current one or not. By applying these phases in an iterative way, one seeks for high quality solutions. Besides, an optional local search procedure can be easily added to improve the constructed solutions. A generic IG algorithm is outlined in Fig. 2 [25]. In the following subsections, we will detail all the components tailored for the task allocation problem with the goal of maximizing system reliability.

3.1. Initial solution

As with all iterative-improvement-based heuristics, the IG algorithm starts from an initial solution. For the task allocation problem under investigation, we use the greedy constructive heuristic (Alg3) proposed by Shatz et al. [29] to rapidly obtain an initial feasible solution. The basic idea of Alg3 is that to reduce the system unreliability, tasks with longer execution time should be allocated to more reliable processors (with smaller λ values), and tightly coupled tasks (tasks heavily communicating with each other) should be allocated to the same processor or to neighboring processors

```
procedure IteratedGreedy
{
    X_0 = GenerateInitialSolution;
    X = LocalSearch(X_0);                % optional
    repeat
        X_p = Destruction (X);
        X_c = Construction (X_p);
        X* = LocalSearch (X_c);          % optional
        X = AcceptanceCriterion (X, X*);
    until termination condition met
}
```

Fig. 2. An outline of a generic iterated greedy algorithm.

with most reliable links. Before entering into details, the definition of task communication cost is given by [29]

$$\text{The communication cost for task } t_i\text{: } C_i = \sum_{j=1}^{N} d_{ij}. \tag{9}$$

The task communication cost, C_i, is a measure of the inter-task communications involving task t_i. Thus, $C_i > C_j$ means that task t_i communicates more with other tasks than task t_j does.

Based on the objective function (4), the algorithm below first sorts the tasks in non-increasing order of task communication cost (Step 1) and then assigns all tasks one by one such that each time a task is assigned, the system unreliability incurred is minimized (Steps 2 and 3). The time complexity of this algorithm is $O(NK)$.

Step 1: Order all tasks in $T = (t_1, t_2, \ldots, t_N)$ so that the task communication costs are in non-increasing order.

Step 2: Assign t_1 to the processor p_k for which $e_{1k}\lambda_k$ is minimum over all processors and no system constraint is violated.

Step 3: For $i = 2$ to N do
Assign t_i to the processor p_k such that

$$f(t_i) = e_{ik}\lambda_k + \sum_{j=1}^{i-1}\sum_{l\neq k} x_{jl}\mu_{kl}d_{ij}/w_{kl} \tag{10}$$

is minimum over all processors and the allocation of t_i to processor p_k does not violate any of the system constraints. Apparently, $f(t_i)$ is the system unreliability incurred by task t_i under the current partial assignment.

3.2. Destruction and construction phases

The central destruction–reconstruction idea is employed for the solution modification in the proposed method, i.e., the two procedures are responsible for moving towards new regions in the solution space. The destruction procedure is applied to the incumbent solution X. In this step, a given number d of tasks, chosen randomly and without repetition, are removed from their current processors and inserted into a sequence denoted as π_d in the order they are chosen. This procedure results in a partial assignment X_p and a sequence of d tasks which have to be reassigned to yield a complete candidate solution. The parameter d, called destruction level, determines a degree the incumbent solution should be destructed. Apparently, the goal of such destruction is not to destroy the current solution absolutely; on the contrary, it is desirable that the resulting partial solution inherits some characteristics of the result from previous iterations. Therefore, it is important that a proper level of destruction is kept up. The destruction level will be determined through experimental studies (see Section 4.2). The computational complexity of the destruction phase is $O(dN)$.

The construction phase starts with the partial assignment X_p and sequentially reassigns the tasks in π_d, one at a time, using Step 3 of the Alg3 given in Section 3.1. In more detail, each task in π_d is reassigned onto processor p_q such that

$$q = \text{argmin}\left\{e_{\pi_d[i]k}\lambda_k + \sum_{j=1}^{N-d+i-1}\sum_{l\neq k} x_{jl}\mu_{kl}d_{\pi_d[i]j}/w_{kl}\right\},$$
$$1 \le k \le K \tag{11}$$

where $\pi_d[i]$ denotes the task occupying position i in π_d, $i = \{1, \ldots, d\}$; the argmin gives an index of the machine at which $f(\pi_d[i])$ is minimized according to Eq. (10). Thus, at each construction step, the current partial assignment is extended by allocating one task in π_d to the processor which yields the smallest system unreliability. This process is repeated until a complete solution is constructed. The construction phase has a computational complexity of $O(dNK)$.

3.3. Local search

The essence of local search is to search for a better solution (local optimum) in the "surroundings", i.e. neighborhood of the current solution. To further improve the performance of the basic IG algorithm, a local search procedure may be applied to the solution just constructed. There are many different alternatives for the local search algorithm that can be considered. Following the idea of having a simple and easily implementable algorithm, we choose a rather straightforward local search algorithm that is based on the transfer neighborhood. The transfer neighborhood of an assignment is defined as the set of assignments that can be obtained by considering the movement of one task from its current processor to another one.

We implement a local search heuristic that searches this neighborhood in a particular way: In each local search step, a task is removed from its current processor (at random and without repetition) and then reassigned to all possible processors that are capable of processing the task. The current assignment X is replaced by the best ones, only if an improvement of X can be obtained. The procedure ends when no further improvements are found, i.e., when the assignment is a local optimum with respect to the transfer neighborhood. Since this local search is applied until a local optimum is found, we can only derive the computational complexity of one single pass. This complexity is $O(N^2K)$. The outline of this procedure is shown in Fig. 3.

3.4. Acceptance criterion

Acceptance criterion determines whether the new assignment is accepted or not as the incumbent solution for the next iteration. One of the simplest acceptance criteria is to accept solutions with better objective values. However, an IG algorithm using this acceptance criterion is prone to stagnation due to insufficient diversification [25]. Hence, we considered a simple SA type acceptance criterion which accepts worse solutions only with a certain probability. In particular, X^* is accepted with a probability p given by

$$p(T, X, X^*) = \begin{cases} 1 & \text{if } Z(X^*) \le Z(X) \\ \exp\left(\dfrac{Z(X) - Z(X^*)}{T}\right) & \text{otherwise} \end{cases} \tag{12}$$

where T is a control parameter called temperature and it is computed following the suggestions of Stützle as follows [32]: After the initial solution X_0 is locally optimized to yield solution X, we

```
procedure LocalSearch(X)
{  improved = true;
   while (improved) do
      improved = false;
      generate a permutation π of tasks randomly;
      for i = 1 to N do
         X* = best assignment obtained by reassigning task π [i] to any possible processors;
         if Z(X*) < Z(X)
            X = X*;
            improved = true;
         end if
      end for
   end while
   return X
}
```

Fig. 3. The local search procedure based on the transfer neighborhood.

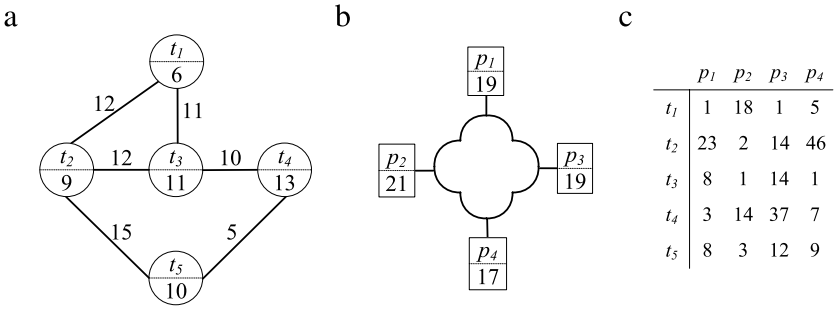

Fig. 4. Examples of (a) a task interaction graph, (b) a processor network, and (c) execution time matrix.

set the initial temperature $T_{\text{init}} = 0.025 \times Z(X)$, that is, a solution which is 2.5% worse than the current one is accepted with probability $1/e$. The temperature is lowered every iteration during the run of the algorithm according to a geometric cooling scheme, by setting $T_{i+1} = 0.9 \times T_i$.

3.5. Termination condition

The termination condition usually involves a fixed number of iterations or execution time, or the detection of a stagnation situation. Since our algorithm will be compared to other heuristics, the stopping criterion is the computational time here. The computation time is usually proportional to the problem scale, and it can be determined experimentally based on the observation that all algorithms can converge before the cutoff time (see Section 4.3).

3.6. An illustrative example

To illustrate how the IG algorithm works, consider a simple allocation problem consisting of 5 tasks and 4 processors as shown in Fig. 4. The weights associated with nodes in Fig. 4(a) indicate the resource requirements of tasks; while those in Fig. 4(b) show the available resource capacities of processors. For simplicity, we assume that all communication paths have the same bandwidth and the inter-task communication time is directly labeled on the edges in Fig. 4(a). The execution time of each task on every processor is shown in Fig. 4(c). Table 1 gives the failure rates of all communication paths; while the failure rates of processors are shown in Table 2.

Table 1
Failure rates of communication paths for the example.

	p_1	p_2	p_3	p_4
p_1	–	0.00029	0.00023	0.00021
p_2	0.00029	–	0.00025	0.00020
p_3	0.00023	0.00025	–	0.00027
P_4	0.00021	0.00020	0.00027	–

Table 2
Failure rates of processors for the example.

p_1	p_2	p_3	p_4
0.00009	0.00008	0.00009	0.00007

For illustrative simplicity, the solution is encoded as an integer vector S of size N while the ith integer of the vector represents the index k ($1 \leq k \leq K$) of the processor to which task t_i is assigned. Fig. 4 shows the encoding of a solution to the problem instance in Fig. 3. For example, $S[1] = 4$ means that the first task is assigned to the fourth processor. Formally, $S[i] = k$ implies that $x_{ik} = 1$, and $x_{il} = 0, \forall l \neq k$. A partial assignment means that some tasks are unassigned; the value of the ith element equal to x indicates that the ith task has not been assigned yet.

Applying Alg3 to this example results in the following algorithm trace:

Calculate task communication costs:

$C_1 = 23, C_2 = 39, C_3 = 33, C_4 = 15, C_5 = 20.$

Order tasks as $T = (t_2, t_3, t_1, t_5, t_4)$.

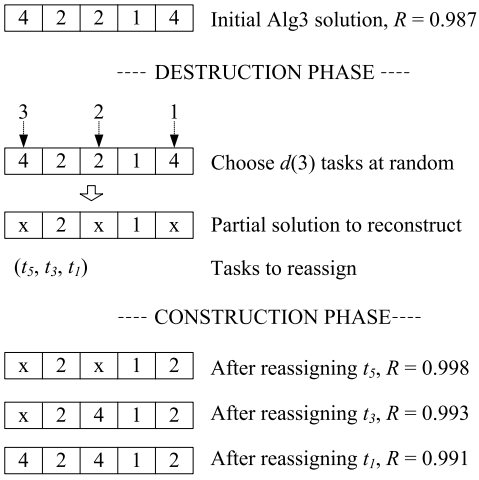

Fig. 5. Example for the application of one iteration of the IG algorithm without local search to an instance.

Calculate the values of $e_{2k}\lambda_k$ $(1 \leq k \leq 4)$ to obtain (0.00207, 0.00016, 0.00126, 0.00322).

The second value is the minimum so assign t_2 to p_2.

Calculate the values of $f(t_3)$ for all possible assignments to obtain (0.00420, 0.00008, 0.00426, 0.00247).

The second value is the minimum so assign t_3 to p_2.

The algorithm allocates the remaining tasks in a similar way, resulting in the initial solution as shown in Fig. 5. For this assignment, the system reliability is

$$\exp(-0.01304) = 0.987.$$

Fig. 5 illustrates the application of one IG iteration to the problem instance without considering the local search phase and using $d = 3$. The example shows that starting from the solution given by the Alg3 ($R = 0.987$) a new solution of $R = 0.991$ is reached by removing t_5, t_3 and t_1 and reassigning them in the way described above. This new solution is accepted by the acceptance criterion since it is better than the starting solution. Furthermore, this new assignment is known to be an optimal solution for the instance with an exhaustive enumeration. Of course, this will not always be the case.

4. Experimental evaluation

The task allocation problem investigated in this paper is concerned with the various execution time of the same task on heterogeneous processors, the various inter-task communication overheads which occur when tasks communicate each other through different channels, and the different failure rates of processors and communication links. Resource constraints imposed by the processors in the target system further complicate the problem. Hence, it is hard to derive some performance bounds theoretically. For this reason we choose to test the proposed IG algorithm by a comprehensive experimental evaluation and comparison with the other recently proposed algorithms for the problem at hand. We have implemented the SA of Attiya and Hamam [5], the hybrid PSO (HPSO) of Yin et al. [38], and the HBMO of Kang et al. [15] in that they are recently proposed meta-heuristics for the problem considered. Also, the constructive heuristic algorithm Alg4 proposed by Shatz et al. [29] has been implemented for comparison purpose. Additionally, we have compared our proposed method against a commercial solver IBM-ILOG CPLEX 12.4, which we will simply refer to as CPLEX. All the algorithms are coded in MATLAB 7.10 and executed on a 1.86 GHz Pentium Dual processor with 1 GB main memory running under a Windows XP environment.

4.1. Experiment settings

Because there are no generally accepted benchmark problems for the task allocation with the goal of maximizing reliability in heterogeneous distributed computing systems, we have chosen to evaluate the comparative algorithms by simulating a wide range of scenarios similar to those used by other researchers [5,15,17,29,38]. The scenarios are generated using different parameters, as described below:

- Number of tasks in the application TIG (N).
- Number of processors in the distributed computing system (K).
- Task interaction density (D). It is the probability that there is an interaction between two tasks, the task interaction density quantifies the ratio of the inter-task communication demands for a TIG and can serve as one of the key factors that affect the problem complexity.
- Communication to computation time ratio (*CCR*). It is the ratio of the average communication time to the average computation time. Thus the average communication time between tasks is set to the average execution time of tasks multiplied by the *CCR*. By using a range of *CCR* values, different types of applications can be accommodated. If a TIG has a low *CCR*, it is considered as a computation intensive application; otherwise, it is a communication intensive application.

The expected execution time of all tasks taking heterogeneity into consideration are generated using the coefficient of variation (*COV*) based method described in [2] because the method provides great control over the dispersion of the execution time values than the common range based method [6]. The *COV* is defined as the ratio of the standard deviation of task execution time to the mean, and its values are used to quantify task heterogeneity and machine heterogeneity [2]. Let the expected time to compute (ETC) matrix be $\{x_{ik}\}_{N\times K}$ where x_{ik} denotes the execution time of task i on processor k. First, a task vector of the expected execution time which size equals the number of tasks is generated following a gamma distribution with a mean of 20 and a *COV* of 0.9 (task heterogeneity). Then the ETC values for each task on all the machines are produced following a gamma distribution using each element of the task vector as the mean and a *COV* of 0.9 (machine heterogeneity). The ETC matrix is of "inconsistent" type, i.e., there is no special structure in the execution time matrix.

The values of (N, K) are set to (20, 6), (30, 8), and (40, 10), respectively, to testify the algorithm with different problem sizes. Three levels of task interaction density (0.3, 0.5, and 0.7) are used for modeling the communication requirements among tasks. The values of the other parameters are generated randomly following a uniform distribution: the transmission rates of links are between 1 and 10 Mb/s; the volumes of data to be transmitted among tasks are generated such that the *CCR* is 0.2, 0.5, 1.0, 2.0, or 5.0; the failure rates of processors and communication links are yielded in the ranges (0.00005, 0.00010) and (0.00015, 0.00030), respectively; the task resource requirements vary form 5 to 15, while the processor capacities are generated as $r \times \phi/M$ where r is a uniform random number in the range (1.2, 2) and $\phi = \sum_{i=1}^{N} r_i$ is the total resource requirement across all the tasks. As such, we obtain a testing data set of 45 problem instances for evaluating the comparative performances of the competing methods.

4.2. Choice of the destruction level

A crucial issue concerns the destruction level (number of tasks removed from a solution in the destruction phase) which may have an impact on running time and the solution quality of the proposed algorithm. To determine the appropriate value of parameter d, a preliminary optimization stem was performed on several instances

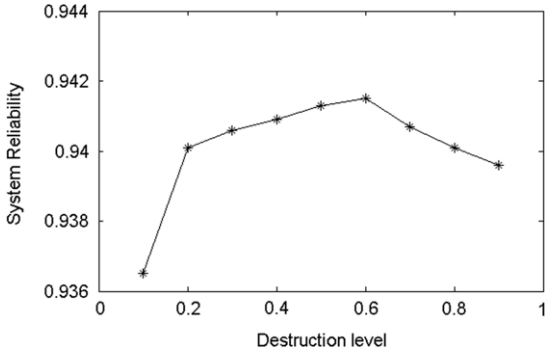

Fig. 6. Effect of different destruction level on solution quality.

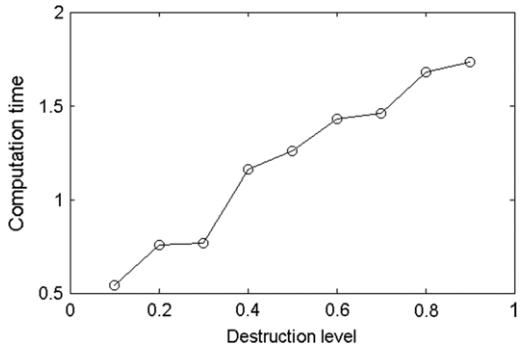

Fig. 7. Effect of different destruction level on computation time.

from the above benchmark problem set. The destruction level is varied from $0.1N$ to $0.9N$ with an increment of $0.1N$. The computational results for most instances are rather similar and, hence, we illustrate the main findings giving a typical example result. The computational results on an instance of (N, K, CCR, D) being equal to $(30, 8, 0.5, 1)$ is illustrated in Figs. 6 and 7.

From Fig. 6, it can be intuitively seen that the best performance is obtained by the setting of $d = 0.6N$, so this value was adopted for all further experiments in this study. Obviously, the bigger the destruction level is, the more running time is needed, which is demonstrated by the experimental results shown in Fig. 7. The figures illustrate that the destruction level should be neither too strong nor too weak. If the destruction level is too strong, the IG algorithm will have excessive randomness resulting in a very low probability of finding better solutions. On the other hand, if it is not strong enough, the construction and the local search procedures will rapidly go back to a previous local optimum.

4.3. Convergence analysis

To analyze the convergence behaviors and determine a computation time limit for all the meta-heuristic algorithms, we have recorded the variations of the system reliability obtained by every algorithm at each iteration step. For the population based algorithms, such as HPSO and HBMO, their best objective function values at each iteration step are recorded. Figs. 8–10 display the typical running behaviors of the algorithms for solving three problem instances with a computation time limit fixed to $N \times K \times 20$ ms. Setting the time limit in this way allows more computation time as the number of tasks or the number of processors increases. We observe that the system reliability obtained by the IG algorithm increases on the whole as the number of iterations increases. At the early stage of iteration, the solution quality improves quickly, and the inferior solution could be accepted due to the SA type acceptance criterion. At the later stage, the curves flatten off. This phenomenon reflects the fact that the best performance of IG is

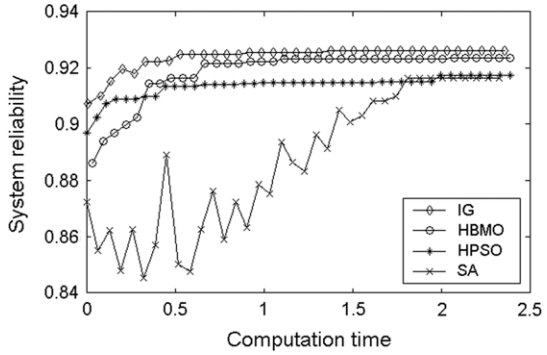

Fig. 8. The typical convergence behaviors of the IG, HBMO, HPSO, and SA for an instance $(N, K, D, CCR) = (20, 6, 0.3, 2)$.

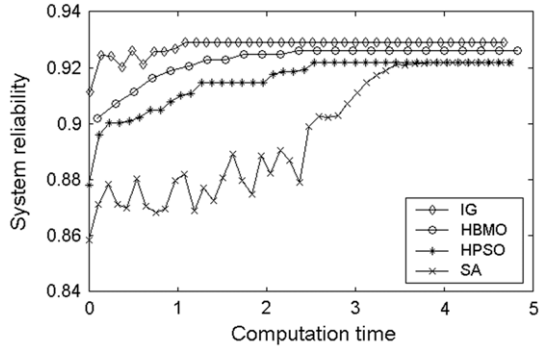

Fig. 9. The typical convergence behaviors of the IG, HBMO, HPSO, and SA for an instance $(N, K, D, CCR) = (30, 8, 0.5, 1)$.

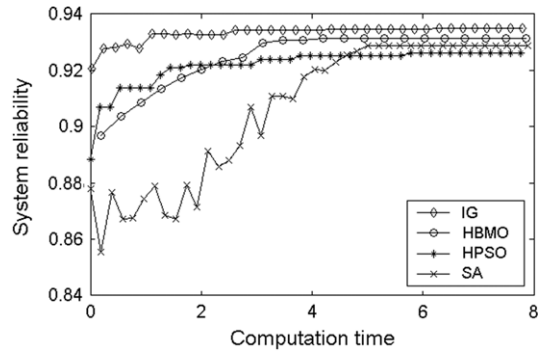

Fig. 10. The typical convergence behaviors of the IG, HBMO, HPSO, and SA for an instance $(N, K, D, CCR) = (40, 10, 0.7, 0.5)$.

achieved. Therefore, it is demonstrated that IG is of effective and efficient ability of global search. It is also observed that all the algorithms can converge before termination. Based on the observations, we set the computation time limit to $N \times K \times 20$ ms for the following experiments.

Our experiments show that CPLEX with default settings is successful on problem instances of small size but fails to solve any of the problem instances described in Section 4.1 due to running out of memory. To obtain feasible solutions for comparison purpose, we set the solution of Alg4 as the starting point of CPLEX and its maximum allowed running time is set to $N \times K \times 100$ ms which is five times CPU time of the competing meta-heuristics. CPLEX works on the mixed integer programming model (5)–(8) and gives the optimal solution if it stops before the time limit or a feasible solution otherwise.

Table 3
Comparison of IG with SA, HPSO, HBMO, Alg4, and CPLEX.

Instances			IG			SA			HPSO			HBMO			Alg4		CPLEX	
(N, K)	D	CCR	R_{ard}	R_{avg}	R_{std}	R_{ard}	R_{avg}	R_{std}	R_{ard}	R_{avg}	R_{std}	R_{ard}	R_{avg}	R_{std}	R_{ard}	R_{avg}	R_{ard}	R_{avg}
(20, 6)	0.3	0.2	0.015	0.976	0.0002	0.183	0.974	0.0004	0.129	0.975	0.0003	0.120	0.975	0.0002	0.954	0.967	0.044	0.976
		0.5	0.023	0.983	0.0002	0.471	0.979	0.0007	0.190	0.981	0.0010	0.243	0.981	0.0004	0.564	0.978	0.013	0.983
		1.0	0.156	0.961	0.0015	1.287	0.948	0.0004	0.642	0.954	0.0006	0.301	0.958	0.0025	1.762	0.944	0.834	0.953
		2.0	0.084	0.938	0.0007	1.790	0.921	0.0012	0.670	0.932	0.0017	0.466	0.933	0.0018	2.341	0.916	1.280	0.926
		5.0	0.530	0.857	0.0031	2.433	0.836	0.0073	2.424	0.836	0.0060	0.599	0.851	0.0057	10.53	0.766	0.045	0.856
	0.7	0.2	0.001	0.985	0.0001	0.164	0.983	0.0004	0.168	0.983	0.0007	0.145	0.984	0.0012	0.298	0.982	0.002	0.985
		0.5	0.001	0.972	0.0001	0.121	0.971	0.0011	0.082	0.971	0.0005	0.246	0.970	0.0004	0.838	0.964	0.003	0.972
		1.0	0.023	0.941	0.0003	0.571	0.936	0.0017	0.432	0.937	0.0009	0.387	0.938	0.0004	1.429	0.928	0.170	0.940
		2.0	0.144	0.923	0.0012	1.881	0.906	0.0006	0.724	0.917	0.0016	0.859	0.915	0.0015	2.839	0.897	0.668	0.917
		5.0	0.788	0.809	0.0041	2.735	0.787	0.0034	2.175	0.792	0.0073	1.624	0.796	0.0017	4.050	0.777	1.199	0.800
(40, 10)	0.3	0.2	0.016	0.963	0.0002	0.101	0.962	0.0002	0.227	0.961	0.0016	0.485	0.959	0.0002	0.584	0.958	0.187	0.961
		0.5	0.156	0.943	0.0013	0.599	0.937	0.0021	0.608	0.937	0.0020	0.515	0.938	0.0013	2.120	0.923	1.865	0.925
		1.0	0.073	0.897	0.0007	0.952	0.888	0.0030	0.867	0.889	0.0049	0.554	0.892	0.0033	3.619	0.864	1.479	0.883
		2.0	0.341	0.846	0.0026	1.716	0.831	0.0046	2.276	0.827	0.0136	0.831	0.839	0.0030	4.259	0.810	2.693	0.823
		5.0	0.558	0.701	0.0003	3.414	0.677	0.0043	4.692	0.668	0.0123	1.064	0.694	0.0032	13.28	0.608	3.018	0.680
	0.7	0.2	0.027	0.960	0.0003	0.111	0.960	0.0003	0.391	0.958	0.0010	0.644	0.955	0.0005	1.167	0.950	0.000	0.961
		0.5	0.177	0.926	0.0010	0.476	0.922	0.0026	0.411	0.923	0.0019	0.727	0.920	0.0009	1.428	0.913	0.002	0.927
		1.0	0.132	0.894	0.0016	0.716	0.888	0.0040	0.936	0.886	0.0009	0.977	0.885	0.0013	1.096	0.884	0.663	0.888
		2.0	0.154	0.828	0.0011	2.706	0.806	0.0028	1.373	0.817	0.0032	1.039	0.820	0.0026	0.976	0.820	2.161	0.810
		5.0	0.286	0.553	0.0020	3.590	0.533	0.0026	4.036	0.530	0.0016	2.233	0.540	0.0034	7.041	0.514	3.239	0.535
Average			0.184	0.893	0.0011	1.301	0.882	0.0022	1.173	0.884	0.0032	0.703	0.887	0.0018	3.059	0.868	0.978	0.885

4.4. Experimental results

Alg4 and CPLEX are deterministic and only one run is necessary. For the stochastic approaches such as IG, SA, HPSO and HBMO, each independent run of the same algorithm on a particular testing instance may yield a different result; we thus run each algorithm 20 times for every problem instance and report the statistical results. Table 3 shows the experimental results obtained from the computations. Each cell belonging to the column with the caption 'R_{ard}' reports the average relative deviation in percentage obtained by the corresponding algorithm for each problem instance. The R_{ard} is computed as follows:

$$R_{ard} = (R_{best} - R_{avg})/R_{best} \times 100 \quad (13)$$

where R_{avg} is the average system reliability on a given instance over 20 independent runs of the corresponding algorithm, and R_{best} is the best system reliability obtained by all the algorithms for the same instance. In the resulting table, the caption 'R_{std}' denotes the standard deviation of system reliability values obtained by the corresponding algorithm.

The comparative results for all algorithms present in Table 3 show that the IG algorithm performs slightly better than HBMO, the superiority of IG over SA, HPSO and Alg4 is more pronounced in terms of the mean performance for all the experiments, and there is no problem instance for which SA, HPSO, HBMO and Alg4 could report a better performance than the IG algorithm. This is remarkable, given the simplicity of the IG algorithm, when compared to the much more complex algorithms (HPSO and HBMO), each of which require to maintain a population of solutions and also more complex recombination operators among solutions. Even though using the solution of Alg4 as the starting point and taking more CPU time, as commented in Section 4.3, CPLEX still performs worse than our proposed method on average, and among the 20 problem instances, CPLEX can only report the optimal solution for one problem. What is more, the experimental results indicate the solution quality of CPLEX is affected by starting points, manifesting the need for heuristic or meta-heuristic algorithms which can derive high quality solutions to large problems within reasonable amounts of time.

It is also observed from the results that the performance differences become larger as the *CCR* increases. An intuitive explanation may be that as the inter-task communication time gets larger compared to the task execution time, the proper allocation of a task becomes increasingly dependent on where other tasks with which it communicates are assigned. Hence, the guiding mechanism for the search used in the IG algorithm appears to be the main reason for the difference in performance. Furthermore, tracking the trends in the mean performance, as the problem scale and the task interaction density increase, IG performs much better than the other competing algorithms. In addition, the IG algorithm performs best in terms of standard deviation of the system reliability, since the mean standard deviation generated by IG is much smaller than those by other stochastic algorithms. The results indicate that the IG algorithm is more suited to large-scaled task allocation problems.

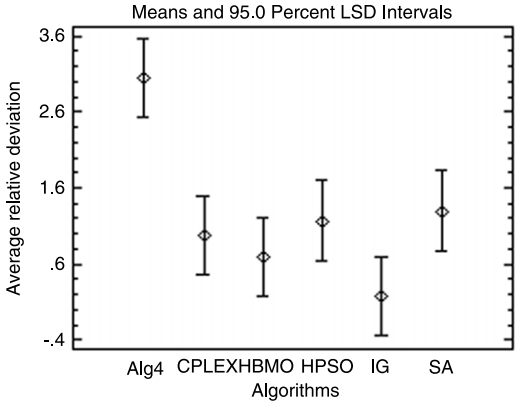

Fig. 11. Means plot and LSD intervals with 95% confidence level for all tested algorithms.

To validate the statistical significance of the observed differences in solution quality, the experimental results are analyzed by means of an analysis of variance technique. This analysis has a single factor, i.e., the type of algorithm with 6 levels and the average relative percentage deviation is the response variable. Fig. 11 shows the means plot along with Least Significant Difference (LSD) intervals at 95% confidence level for the tested algorithms.

As can be seen, IG is indeed statistically better than all the other algorithms. HBMO statistically outperforms SA, HPSO, Alg4, and CPLEX, and there is no clear statistically significant difference between SA and HPSO at a 95% confidence level. So, it can be

concluded that the proposed IG algorithm is more effective than the other competing algorithms for the task assignment with the goal of maximizing system reliability in the heterogeneous distributed computing environments.

5. Conclusions and future work

To our knowledge the first application of the IG algorithm to the task allocation problem in heterogeneous distributed computing systems with the criterion of maximizing reliability is reported in this paper. The proposed method is mainly composed of a solution initialization, a solution modification procedure, and a very straightforward local search heuristic. The performance of the proposed algorithm is evaluated in comparison with state-of-the-art methods on a number of randomly generated mapping problem instances. The experimental results show that the solution quality of the IG algorithm is better than those of the competing methods in most situations. This fact is especially remarkable if we consider the high level of sophistication reached by the algorithmic techniques that have been applied to the problem and the inherent simplicity of the proposed methods. Therefore, these results indicate that the IG technique is an attractive alternative for solving the reliability-oriented task allocation problem in heterogeneous computing systems. For future research, extending our approach for solving other performance criterion or even multi-objective task assignment problems is interesting.

Acknowledgments

The authors thank the anonymous referees and the editor for their valuable comments and suggestions. This work is supported by the National Grand Fundamental Research 973 Program of China under Grant No. 2009CB320704, and the National High Technology Research and Development Program of China under Grant No. 2012AA011204.

References

[1] I. Ahmad, M.K. Dhodhi, Task assignment using problem-space genetic algorithm, Concurrency, Pract. Exp. 7 (1995) 411–428.
[2] S. Ali, H.J. Siegel, et al., Representing task and machine heterogeneities for heterogeneous computing systems, Tamkang J. Sci. Eng. 3 (3) (2000) 195–207.
[3] A.M. Al-Qawasmeh, A.A. Maciejewski, Characterizing task-machine affinity in heterogeneous computing environments, in: Proceedings of 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Ph.D. Forum, IPDPSW, 2011, pp. 34–44.
[4] B. Arafeh, K. Day, A. Touzene, A multilevel partitioning approach for efficient tasks allocation in heterogeneous distributed systems, J. Syst. Archit. 54 (2008) 530–548.
[5] G. Attiya, Y. Hamam, Task allocation for maximizing reliability of distributed systems: a simulated annealing approach, J. Parallel Distrib. Comput. 66 (2006) 1259–1266.
[6] T.D. Braun, H.J. Siegel, et al., A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing system, J. Parallel Distrib. Comput. 6 (2001) 810–837.
[7] A.O. Charles Elegbede, C. Chu, K.H. Adjallah, F. Yalaoui, Reliability allocation through cost minimization, IEEE Trans. Reliab. 52 (2003) 106–111.
[8] W.H. Chen, C.S. Lin, A hybrid heuristic to solve a task allocation problem, Comput. Oper. Res. 27 (3) (2000) 287–303.
[9] C.C. Chiu, Y.S. Yeh, J.S. Chou, A fast algorithm for reliability-oriented task assignment in a distributed system, Comput. Commun. 25 (17) (2002) 1622–1630.
[10] Y. Hamam, K.S. Hindi, Assignment of program modules to processors: a simulated annealing approach, European J. Oper. Res. 122 (2000) 509–513.
[11] C.C. Hsieh, Optimal task allocation and hardware redundancy policies in distributed computing systems, European J. Oper. Res. 147 (2003) 430–447.
[12] C.C. Hsieh, Y.C. Hsieh, Reliability and cost optimization in distributed computing systems, Comput. Oper. Res. 30 (2003) 1103–1119.
[13] M. Kafil, I. Ahmad, Optimal task assignment in heterogeneous distributed computing systems, IEEE Concurr. 6 (1998) 42–51.
[14] Q.M. Kang, H. He, H.M. Song, Task assignment in heterogeneous computing systems using an effective iterated greedy algorithm, J. Syst. Softw. 84 (2011) 985–992.
[15] Q.M. Kang, H. He, H.M. Song, R. Deng, Task allocation for maximizing reliability of distributed computing systems using honeybee mating optimization, J. Syst. Softw. 83 (2010) 2165–2174.
[16] S. Kartik, C.S.R. Murthy, Improved task-allocation algorithms to maximize reliability of redundant distributed computing systems, IEEE Trans. Reliab. 44 (1995) 575–586.
[17] S. Kartik, C.S.R. Murthy, Task allocation algorithms for maximizing reliability of distributed computing systems, IEEE Trans. Comput. 46 (1997) 719–724.
[18] M. Kiran, A.H.A. Hashim, Execution time prediction of imperative paradigm tasks for grid scheduling optimization, Int. J. Comput. Sci. Netw. Secur. 9 (2) (2009) 155–163.
[19] A. Kumar, D.P. Agrawal, A generalized algorithm for evaluating distributed-program reliability, IEEE Trans. Reliab. 42 (1993) 416–426.
[20] M.S. Lin, D.J. Chen, The computational complexity of the reliability problem on distributed systems, Inform. Process. Lett. 64 (1997) 143–147.
[21] A. Mahmood, Task allocation algorithms for maximizing reliability of heterogeneous distributed computing systems, Control Cybernet. 30 (2001) 115–130.
[22] E. Marchiori, A. Steenbeek, An evolutionary algorithm for large scale set covering problems with application to airline crew scheduling, Lecture Notes in Comput. Sci. 1803 (2000) 367–381.
[23] Q.K. Pan, L. Wang, B.H. Zhao, An improved iterated greedy algorithm for the no-wait flow shop scheduling problem with makespan criterion, Int. J. Adv. Manuf. Technol. 38 (2008) 778–786.
[24] C. Roig, A. Ripoll, F. Guirado, A new task graph model for mapping message passing applications, IEEE Trans. Parallel Distrib. 18 (2007) 1740–1753.
[25] R. Ruiz, T. Stützle, A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem, European J. Oper. Res. 177 (3) (2007) 2033–2049.
[26] R. Ruiz, T. Stützle, An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives, European J. Oper. Res. 187 (3) (2008) 1143–1159.
[27] S. Salcedo-Sanz, Y. Xu, X. Yao, Hybrid meta-heuristics algorithms for task assignment in heterogeneous computing systems, Comput. Oper. Res. 33 (3) (2006) 820–835.
[28] A. Salman, I. Ahmad, S. Al-Madani, Particle swarm optimization for task assignment problem, Microprocess. Microsyst. 26 (2002) 363–371.
[29] S.M. Shatz, J.P. Wang, M. Goto, Task allocation for maximizing reliability of distributed computer systems, IEEE Trans. Comput. 41 (1992) 1156–1168.
[30] C. Shen, W. Tsai, A graph matching approach to optimal task assignment in distributed computing systems using minimax criterion, IEEE Trans. Comput. 34 (1985) 197–203.
[31] H.S. Stone, Multiprocessor scheduling with the aid of network flow algorithms, IEEE Trans. Softw. Eng. SE-3 (1) (1977) 85–93.
[32] T. Stützle, Iterated local search for the quadratic assignment problem, European J. Oper. Res. 174 (2006) 1519–1539.
[33] P.A. Tom, C. Murthy, Algorithms for reliability-oriented module allocation in distributed computing systems, J. Syst. Softw. 40 (1998) 125–138.
[34] B. Ucar, C. Aykanat, K. Kaya, M. Ikinci, Task assignment in heterogeneous computing systems, J. Parallel Distrib. Comput. 66 (2006) 32–46.
[35] D.P. Vidyarthi, A.K. Tripathi, Maximizing reliability of distributed computing systems with task allocation using simple genetic algorithm, J. Syst. Archit. 47 (2001) 549–554.
[36] J. Yang, A. Khokhar, A. Ghafoor, Estimating execution time for parallel tasks in heterogeneous processing (HP) environment, in: Proceedings of Heterogeneous Computing Workshop, 1994, pp. 23–28.
[37] P.Y. Yin, S.S. Yu, P.P. Wang, Y.T. Wang, A hybrid particle swarm optimization algorithm for optimal task assignment in distributed systems, Comput. Stand. Interfaces 28 (2006) 441–450.
[38] P.Y. Yin, S.S. Yu, P.P. Wang, Y.T. Wang, Task allocation for maximizing reliability of a distributed system using hybrid particle swarm optimization, J. Syst. Softw. 80 (2007) 724–735.
[39] K.C. Ying, H.M. Cheng, Dynamic parallel machine scheduling with sequence-dependent setup times using an iterated greedy heuristic, Expert Syst. Appl. 37 (4) (2010) 2848–2852.
[40] K.C. Ying, S.W. Lin, C.Y. Huang, Sequencing single-machine tardiness problems with sequence dependent setup times using an iterated greedy heuristic, Expert Syst. Appl. 36 (3) (2009) 7087–7092.
[41] D.X. Zou, L.Q. Gao, S. Li, J.H. Wu, X. Wang, A novel global harmony search algorithm for task assignment problem, J. Syst. Softw. 83 (10) (2010) 1678–1688.

Qinma Kang received his Ph.D. in computer software and theory from Tongji University, Shanghai, China, in 2011. He is currently an associate professor with the School of Information Engineering, Shandong University, Weihai, China. His research interests include scheduling and resource allocation for parallel and distributed computing systems, optimization algorithms and application, etc.

Hong He received her Ph.D. in computer software and theory from Shandong University, Jinan, China, in 2002. She is currently an associate professor with the School of Information Engineering, Shandong University, Weihai, China. Her research interests include algorithm analysis and design, network optimization, Grid computing etc.

Jun Wei received his B.Sc. and Ph.D. in computer science from Wuhan University, Hubei, China in 1992 and 1997, respectively. He was a Postdoctoral Researcher at the Hong Kong University of Science and Technology, Hong Kong, China. He is currently a Professor in the Institute of Software, Chinese Academy of Sciences, Beijing, China. His research interests include service oriented computing, middleware, mobile computing, and software engineering. He has published over 50 papers in International journals and conferences.