Available online at www.sciencedirect.com
ScienceDirect
Applied Mathematics and Computation 184 (2007) 407–420
APPLIED MATHEMATICS AND COMPUTATION
www.elsevier.com/locate/amc

Multi-objective task allocation in distributed computing systems by hybrid particle swarm optimization

Peng-Yeng Yin *, Shiuh-Sheng Yu, Pei-Pei Wang, Yi-Te Wang

Department of Information Management, National Chi Nan University, 303, University Road, Nantou 545, Taiwan

Abstract

In a distributed computing system (DCS), we need to allocate a number of modules to different processors for execution. It is desired to maximize the processor synergism in order to achieve various objectives, such as throughput maximization, reliability maximization, and cost minimization. There may also exist a set of system constraints related to memory and communication link capacity. The considered problem has been shown to be NP-hard. Most existing approaches for task allocation deal with a single objective only. This paper presents a multi-objective task allocation algorithm with presence of system constraints. The algorithm is based on the particle swarm optimization which is a new metaheuristic and has delivered many successful applications. We further devise a hybrid strategy for expediting the convergence process. We assess our algorithm by comparing to a genetic algorithm and a mathematical programming approach. The experimental results manifest that the proposed algorithm performs the best under different problem scales, task interaction densities, and network topologies.

Keywords: Multi-objective task allocation problem; Distributed computing systems; Distributed system reliability; Hybrid strategy; Particle swarm optimization; Genetic algorithm

1. Introduction

The configuration of a distributed computing system (DCS) involves a set of cooperating processors communicating over the communication links. To increase the system throughput, it is desired to allocate the modules of a distributed program to the processors according to some objectives, ranging from the minimization of execution and communication cost [1–3], to the maximization of system reliability and safety [4,5], to the increasing of fault tolerance using software and hardware redundancy [6,7]. Moreover, the system components (processors and communication links) may be capacitated with limited amount of resource which constrains the demand of the allocated modules. This paper deals with the task allocation problem in which multiple objectives are considered and a set of resource constraints are imposed.

* Corresponding author.
E-mail address: pyyin@ncnu.edu.tw (P.-Y. Yin).

0096-3003/$ - see front matter © 2006 Elsevier Inc. All rights reserved.
doi:10.1016/j.amc.2006.06.071

Among others, system cost and reliability are two of the most concerned objectives to leverage the performance of a DCS. The execution of a module will incur an *execution cost* from its allocated processor, and some *communication cost* will be paid if two communicating modules are executed on different processors. The task allocation which incurs the minimum system cost (the summation of execution and communication costs) is considered to be the optimal. On the other hand, the *distributed system reliability* (DSR) is the probability for the successful completion of distributed programs which require that all the allocated processors and involved communication links are operational during the mission. It is desired to increase the DSR when the applications are reliability-critical such as the electricity power and military weapon systems. Most of the existing approaches solve the problem by optimizing a single objective (cost minimization or reliability maximization), few of them have addressed the multi-objective issue [8].

Unfortunately, both of the minimization of system cost and the maximization of system reliability are NP-hard problems [9,10]. As such, there exists a numerous number of literature for providing alternative solutions. They can be broadly classified into three categories. (1) *Mathematical programming approaches* [3,4,6,11–13] such as linear programming, branch-and-bound, and state-space algorithms, are developed to solve small problems. (2) *Customized algorithms* [1,5,14,15] taking into account the specific system configurations can provide exact or approximate solutions under some scenarios. (3) *Metaheuristic algorithms* such as genetic algorithms (GA) [16–19] and simulated annealing (SA) [20,21], have also been applied to derive sub-optimal solutions with reasonable time.

Since mathematical programming approaches are seeking the exact solutions, they are computationally prohibitive if the problem size is large. Further, the customized algorithms rely on specific network configurations and are constrained in limited applications. Recently, the development of metaheuristic optimization theory has been flourishing, several new metaheuristic algorithms such as Tabu search (TS) [22], Ant Colony Optimization (ACO) [23], and Particle Swarm Optimization (PSO) [24] have demonstrated successful applications in diverse fields. They are able to provide quality solutions with reasonable time.

Inspired by the success of metaheuristic algorithms, this paper presents a hybrid PSO (HPSO) algorithm to solve the multi-objective task allocation problem (MOTAP) for system cost and reliability optimization. Efficient particle coding scheme has been adopted and a hill-climbing heuristic is embedded in the PSO iteration to expedite the convergence. The experimental results manifest that the HPSO reports quality solutions on a large set of simulated instances involving different problem scales, task interaction densities, and network topologies.

The remainder of this paper is organized as follows. Section 2 formulates the MOTAP for system cost and reliability optimization. Section 3 describes the proposed HPSO algorithm. Section 4 reports the comparative performances and convergence analysis. Finally, Section 5 concludes this work.

2. Problem formulation

2.1. Nomenclature

x_{ik}	decision variable: $x_{ik} = 1$ if module i is allocated to processor k, and $x_{ik} = 0$ otherwise
n	number of processors
r	number of modules
p_k	processor k
l_{kb}	communication link connecting p_k and p_b
u_k	execution cost of processor p_k per unit time
u_{kb}	communication cost of link l_{kb} per unit time
λ_k	failure rate of processor p_k
μ_{kb}	failure rate of communication link l_{kb}
e_{ik}	incurred accumulative execution time (AET) if module i is executed on processor k
c_{ij}	incurred intermodule communication (IMC) load (in some unit of data quantity) between modules i and j
w_{kb}	transmission rate of communication link l_{kb}
m_i	memory resource requirements of module i from its execution processor

M_k amount of memory resource capacitated with processor p_k
s_i computation resource requirements of module i from its execution processor
S_k amount of computation resource capacitated with processor p_k

2.2. Problem statement

We consider the task allocation problem with the following scenarios:

- The processors involved in the DCS are heterogeneous. Hence, the processors may be capacitated with various units of memory and computation resources and they may have different processing speeds and failure rates. Also, the communication links may have different bandwidths and failure rates.
- A module may take different execution time if it is executed on different processors. An amount of data may take different communication time if transmitted through different communication links. The system cost and reliability are both dependent upon the execution and communication times.
- The execution of a module will also consume a specific amount of memory and computation resource from its assigned processor.
- The state of processors and communication links is either operational or failed. Failure events are statistically independent.

Our goal is to search a task allocation that minimizes the system cost and maximizes the system reliability simultaneously while satisfying all of the resource constraints. We render the network topology by the *processor interaction graph* (PIG). The PIG, denoted by $G_1(P, L)$ where $P = \{p_i\}_{i=1,2,\ldots,n}$ and $L = \{l_{kb}\}_{1 \leqslant k < b \leqslant n}$ are the sets of processors and communication links respectively, illustrates how the processors are connected in the DCS. Fig. 1 gives some typical examples of PIG, including linear, ladder, cross, tree, and star, where the nodes indicate the processors and the edges correspond to the communication links.

The intermodule communication (IMC) among the distributed modules of a task to be executed in a DCS can be described by the *task interaction graph* (TIG). We denote the TIG by $G_2(V, E)$ where $V = \{v_i\}_{i=1,2,\ldots,r}$ is a set of r nodes indicating the r modules and $E = \{c_{ij}\}$ is a set of edges specifying the IMC among these modules. Fig. 2 gives an example of a TIG where the IMC among five modules are specified.

The complexity of the TIG can be measured by the task interaction density d as follows:

$$d = \frac{|E|}{r(r-1)/2}, \tag{1}$$

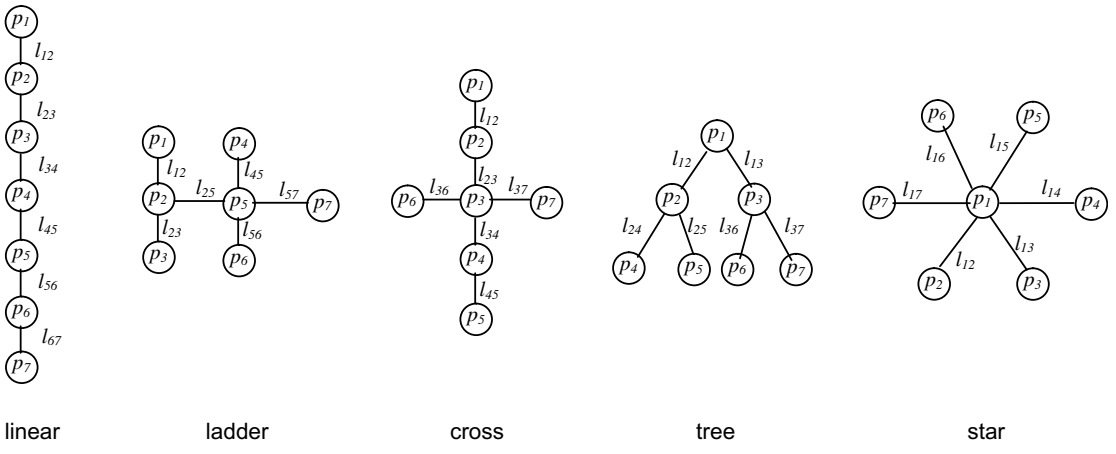

Fig. 1. Typical examples of PIG.

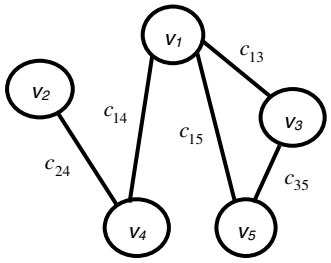

Fig. 2. An example of TIG.

where $|E|$ calculates the number of channels of requested IMC demands in the TIG, and $r(r-1)/2$ indicates the maximal number of possible IMC channels among r modules. Therefore, the task interaction density quantifies the ratio of the IMC for a TIG and can serve as one of the key factors that affect the problem complexity.

2.2.1. System cost

We assume that the execution and communication costs are time-dependent which fits the scenario that longer execution or communication times will incur heavier cost on the involved processors or communication links. Given a task allocation $\boldsymbol{X} = \{x_{ik}\}_{1\leqslant i\leqslant r, 1\leqslant k\leqslant n}$, the execution cost of processor p_k during the accumulative execution time (AET) interval t is $u_k t$. Since the total elapse time for executing the modules assigned to p_k is $\sum_{i=1}^{r} x_{ik} e_{ik}$, the execution cost of all processors can be computed by $\sum_{k=1}^{n}\sum_{i=1}^{r} u_k x_{ik} e_{ik}$.

Similarly, since the total elapse time for handling the IMC via l_{kb} is $\sum_{i=1}^{r}\sum_{i\neq j} x_{ik}x_{jb}(c_{ij}/w_{kb})$, the communication cost incurred over the communication link l_{kb} is $\sum_{i=1}^{r}\sum_{i\neq j} u_{kb}x_{ik}x_{jb}(c_{ij}/w_{kb})$, and the total system communication cost is thus given by $\sum_{k=1}^{n-1}\sum_{b>k}\sum_{i=1}^{r}\sum_{i\neq j} u_{kb}x_{ik}x_{jb}(c_{ij}/w_{kb})$.

The system cost which is defined as the sum of the execution and communication costs is computed as follows:

$$C(\boldsymbol{X}) = \sum_{k=1}^{n}\sum_{i=1}^{r} u_k x_{ik} e_{ik} + \sum_{k=1}^{n-1}\sum_{b>k}\sum_{i=1}^{r}\sum_{i\neq j} u_{kb}x_{ik}x_{jb}(c_{ij}/w_{kb}). \tag{2}$$

2.2.2. System reliability

To evaluate the distributed system reliability (DSR) for the successful completion of a mission with a task allocation, we follow Shatz's formulation [11] to compute the probability that all involved components (processors and communication links) are operational during the mission. Analogous to the system cost, the system reliability is also time-dependent. Let the task allocation be $\boldsymbol{X} = \{x_{ik}\}_{1\leqslant i\leqslant r, 1\leqslant k\leqslant n}$, the reliability of processor p_k during the AET for executing the modules assigned to it follows the Poisson distribution $e^{-\lambda_k \sum_{i=1}^{r} x_{ik} e_{ik}}$. Similarly, the reliability for the communication link l_{kb} during the total IMC transmission time is $e^{-\mu_{kb}\sum_{i=1}^{r}\sum_{i\neq j} x_{ik}x_{jb}(c_{ij}/w_{kb})}$. As such, the system reliability that all involved processors and communication links are operational during the elapse time is computed as follows:

$$R(\boldsymbol{X}) = \prod_{k=1}^{n} e^{-\lambda_k \sum_{i=1}^{r} x_{ik} e_{ik}} \prod_{k=1}^{n-1}\prod_{b>k} e^{-\mu_{kb}\sum_{i=1}^{r}\sum_{i\neq j} x_{ik}x_{jb}(c_{ij}/w_{kb})}. \tag{3}$$

2.2.3. Multi-objective formulation

Both the system cost and the system reliability described above (see Eqs. (2) and (3)) have quadratic objective functions, we propose a weighting summation to optimize the two criteria simultaneously, while satisfying all the system resource constraints. The mathematical formulation of the MOTAP is as follows:

$$\text{Min} \quad Z(\boldsymbol{X}) = C(\boldsymbol{X}) + \alpha/R(\boldsymbol{X}), \tag{4}$$

$$\text{subject to} \quad \sum_{k=1}^{n} x_{ik} = 1 \quad \forall\, i = 1, 2, \ldots, r, \tag{5}$$

$$\sum_{i=1}^{r} m_i x_{ik} \leqslant M_k \quad \forall\, k = 1, 2, \ldots, n, \tag{6}$$

$$\sum_{i=1}^{r} s_i x_{ik} \leqslant S_k \quad \forall\, k = 1, 2, \ldots, n, \tag{7}$$

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

The objective function (4) combines the two optimization objectives (cost minimization and reliability maximization) into a single function by a scaling factor α. The scaling factor plays two roles: *First*, it normalizes the values of $C(\boldsymbol{X})$ and $R(\boldsymbol{X})$ to comparative ranges such that $Z(\boldsymbol{X})$ will not be dominated by a single objective. *Second*, α can be used as a weighting parameter which controls the relative significance of each objective. Constraint (5) enforces that each module should be assigned to exactly one processor. Constraints (6) and (7) correspond to the resource constraints where the memory and computation resource capacity of each processor should be no less than the total amount of resource requirements of all of its assigned modules. Constraint (8) guarantees that x_{ik} are binary variables.

3. Proposed algorithm

The problem formulation of MOTAP is a 0–1 programming with a quadratic objective function which has been known to be NP-hard. Seeking the optimal solution to the problem is computationally prohibitive due to the enormous computations. An alternative is to find approximate solutions efficiently based on metaheuristics. In the following, we propose a hybrid metaheuristic algorithm for tackling the problem.

3.1. Review of particle swarm optimization

In 1995, Kennedy and Eberhart [24] have proposed the particle swarm optimization (PSO) algorithm which is closely related to the field of evolutionary computation. The PSO is inspired by the behavior of bird flocking and fish schooling. A swarm consisting of a population of birds/fishes flocks synchronously, changes direction suddenly, scatters and regroups iteratively, and finally perches on a target, in order to escape from enemies or search for food. The PSO mimics the interesting behavior and serves as a function optimizer. It keeps a population of individuals which are potential solutions to the optimization problem. By taking advantage of individual cognition and social interaction, the swarm improves the solutions iteratively and eventually converges to the optimal solution. The convergence and parameterization aspects of the PSO have been discussed thoroughly [25,26]. The attractiveness of the PSO includes the features: natural metaphor, stochastic move, adaptivity, and positive feedback.

The basic PSO algorithm (see Fig. 3) is outlined as follows. Initially, a swarm of particles is generated at random. Each particle is a candidate solution to the optimization problem. Let the problem solution be described by r variables, we denote the ith particle by $P_i = (p_{i1}, p_{i2}, \ldots, p_{ir})^{\mathrm{T}} \in R^r$, and it changes position

1. Create an initial swarm at random.
2. Repeat
   - 2.1 Update the personal and global best experiences.
   - 2.2 Update particles' positions based on best experiences.

   Until the stopping criterion is satisfied.

Fig. 3. Summary of the basic PSO algorithm.

iteratively according to two types of experiences: personal best and global best experiences. This is evidenced from the socio-psychological view that the personal cognition and social interaction play important roles in learning. This is a positive feedback process which increases the success rate in foraging for the social system, or in the optimization language, it increases the probability for targeting the optimal solution. Analogous to other evolutionary computation algorithms, the PSO algorithm is terminated when the stopping criterion is satisfied which can be set as a maximal number of experienced iterations or a maximal number of iterations between two consecutive improvements on the global experience.

3.2. Hybrid PSO for the multi-objective task allocation problem

To cope with the multi-objective task allocation problem (MOTAP), we propose the PSO features relevant to particle representation, fitness function, particle movement, and hybrid strategy. The details are presented in the following.

3.2.1. Particle representation

To solve an optimization problem with the PSO, one needs to *encode* the candidate solution to the underlying problem into a particle vector form for conducting the evolutionary computation, then the evolved vector is *decoded* to the solution form to evaluate its merit of fitness. The representation of the particle vector should be compact (for efficient storage use) and simple (for fast encoding/decoding process). For the considered MOTAP, one may let a particle correspond to all decision variables, $\boldsymbol{X} = \{x_{ik}\}_{1\leqslant i\leqslant r, 1\leqslant k\leqslant n}$, however, this representation is not efficient since $\boldsymbol{X}$ is a sparse matrix. Instead, we represent the ith particle as $P_i = (p_{i1}, p_{i2}, \ldots, p_{ir})^{\mathrm{T}}$ where $p_{i,j}$ indicates the index of the allocated processor for the jth module. The proposed particle representation is both compact (only r cells are needed) and simple ($p_{i,j} = k$ implies that $x_{jk} = 1$ and $x_{jl} = 0$ for $\forall\ l \neq k$).

The PSO swarm consists of N particles and the initial swarm is generated at random. The swarm particles iteratively improve their solution quality (evaluated by the fitness function) based on personal cognition and social interaction by the particle movement formula.

3.2.2. Fitness function

The fitness function *fitness* ($\boldsymbol{X}$) measures to what extent the particle solution $\boldsymbol{X}$ satisfies the objective of the constrained optimization problem. Two criteria should be concerned. *First*, the solution resulting in a better objective value from the objective function is considered to satisfy more the objective of the problem than the solution with a worse objective value. *Second*, the solutions are divided into several categories according to the feasibility with which they meet the problem constraints. Solutions which satisfy all of the constraints are grouped in a category that deserves the highest merit. While the solutions which violate at least one of the constraints are further classified in accordance with the quantified amount they mismatch the constraints. The solutions with lower mismatch amount are preferential than the solutions with higher mismatch amount.

With the above considerations, we define the fitness function as follows:

$$\text{fitness}(\boldsymbol{X}) = Z(\boldsymbol{X})^{-1} + J(\boldsymbol{X})^{-1}, \tag{9}$$

where $Z(\boldsymbol{X})$ is the minimization objective function of the MOTAP (see Eq. (4)) and $J(\boldsymbol{X})$ is the quantified amount of mismatch if $\boldsymbol{X}$ is infeasible; otherwise, $J(\boldsymbol{X})$ is set as 0. Therefore, the higher the fitness value of $\boldsymbol{X}$, the more satisfactory the solution $\boldsymbol{X}$.

Since the constraints (5) and (8) are always satisfied if our particle representation is adopted, $J(\boldsymbol{X})$ is only related to the constraints (6) and (7) and it can be defined as follows:

$$J(\boldsymbol{X}) = \delta_1 \times \sum_{k=1}^{n} \max\left(0, \sum_{i=1}^{r} m_i x_{ik} - M_k\right) + \delta_2 \times \sum_{k=1}^{n} \max\left(0, \sum_{i=1}^{r} s_i x_{ik} - S_k\right), \tag{10}$$

where δ_1 and δ_2 are the weights controlling the relative significance of respective constraints.

3.2.3. Particle movement

According to the socio-psychological theory the learning is fulfilled through personal cognition and social interaction. The PSO enforces the socio-learning process by tallying the best experiences observed by individual particle and the entire swarm during the evolution. The personal best experience, denoted by $pbest_i$, is the experienced position by particle P_i which receives the highest fitness value during flying. The swarm's best experience has been defined in two versions. In the local best version, each particle remembers the best position $lbest_i$ among the competing particles within its local topological neighborhood. For the global best version, the best position *gbest* is determined by any particles in the entire swarm. It has been empirically shown that the local best version outperforms the global best version in a number of applications.

The convergence issue of the PSO has been studied thoroughly. Several alternative solutions have been proposed ranging from the maximum velocity limit [24], to the inertia weight [27], to the constriction factor model [28]. Among them, the constriction factor model recently proposed by Clerc [28] has received the attention from many PSO practitioners. It proceeds as follows. At each evolutionary iteration, the particle P_i modifies its velocity v_{ij} and position p_{ij} through each dimension j by referring to the personal best experience ($pbest_i$) and the swarm's best experience ($lbest_i$, if the local best version is used) using Eqs. (11)–(13) as follows:

$$v_{ij} \leftarrow K[v_{ij} + c_1 rand_1(pbest_i - p_{ij}) + c_2 rand_2(lbest_i - p_{ij})], \tag{11}$$

$$p_{ij} \leftarrow p_{ij} + v_{ij}, \tag{12}$$

where c_1 and c_2 are the cognitive and interaction coefficients, $rand_1$ and $rand_2$ are random real numbers drawn from $U(0,1)$, and K is the constriction coefficient satisfying

$$K = \frac{2}{\left|2 - (c_1 + c_2) - \sqrt{(c_1 + c_2)^2 - 4(c_1 + c_2)}\right|} \quad \text{s.t.} \quad c_1 + c_2 > 4. \tag{13}$$

In many applications, setting $c_1 = c_2 = 2.05$ has received satisfactory results.

At each iteration, the PSO flies each particle through the solution space using Eqs. (11)–(13) such that the particles learn through the personal cognition ($pbest_i$) and the social interaction ($lbest_i$) while still exploring new areas by the random multipliers ($rand_1$ and $rand_2$) to escape from the barrier of the local optimality. When the algorithm is terminated with a given maximum number of iterations, the best experienced position by the entire swarm is reported as the final solution.

3.2.4. Hybrid strategy

Many empirical findings have manifested that the hybrid strategy interweaving the PSO with other kinds of searching methods outperforms a simple plan using the PSO alone. For example, Wang and Li [29] used each particle as a seed to conduct a simulated annealing searching process and verified the strength of the hybrid algorithm on a testbed of optimization functions. Meng et al. [30] embedded the chaotic search within the PSO to avoid the stagnation and increase the convergence speed. Another hybrid strategy proposed by Liu et al. [31] employed the line search to determine the maximum step size of the particle movement rationally. In light of this, we devise a parameter-wise hill-climbing search heuristic and embed it into the PSO algorithm.

At the end of each PSO iteration, we improve the solution quality of each particle by scanning along its r cells sequentially for possible updating. We examine each cell and look for possible replacement with other values if a higher fitness is obtained, such that the particle is improved from both views of objective function and constriction satisfaction. When a cell is examined, the values of the remaining $r - 1$ cells remain unchanged. To save the computational time, the scan-along process is conducted only once for each particle.

4. Experimental results and discussion

We have implemented the proposed hybrid PSO algorithm (referred to as HPSO hereafter), a genetic algorithm (GA), and a Lingo program (Exact) to analyze their comparative performances. The experimental environment is a 2.4 GHz PC with 256MB RAM. The settings of the competing algorithms are detailed in the following:

- HPSO: The particle representation, fitness function, particle movement, and the hybrid scheme employed by the algorithm have been described in Section 3.2. The scaling factor α (see Eq. (4)) is determined by the minimum of $C(\boldsymbol{X})$ from 10 random solutions. Further, we let $\delta_1 = \delta_2 = 1$ in Eq. (10) for setting equal importance on each constraint. The swarm size is optimally tuned as 80 particles according to preliminary experiments.
- GA: To make a fair comparison, the algorithm uses the same representation scheme as the HPSO does, i.e., the chromosome is represented as $P_i = (p_{i1}, p_{i2}, \ldots, p_{ir})^{\mathrm{T}}$ where $p_{i,j} = k$ implies that $x_{jk} = 1$ and $x_{jl} = 0$ for $\forall\, l \neq k$. The fitness function given in Eq. (9) is also used for evaluating the merit of fitness of each chromosome. The algorithm ducks a population of 80 chromosomes and evolves with the roulette-wheel selection, single-point crossover, and uniform mutation operations. The crossover and mutation rates are optimally tuned as 0.7 and 0.1, respectively.
- Exact: The Lingo program solves the 0–1 quadratic programming problem (Eqs. (4)–(8)) and reports the exact optimal solution. However, when the Lingo program fails to solve the problem within 24 h, it is terminated with an infeasible solution or an unknown status and we discard such cases for further comparison.

To fairly compare the algorithms, the HPSO and GA are terminated when they have experienced a maximum number of fitness evaluations (here, we set this number to 80,000) since the evaluation of fitness function is the most time-consuming component for the HPSO and GA, and the number of fitness evaluations indicates the amount of the computation resource used by the algorithms. Further, both HPSO and GA are stochastic-based algorithms and each independent run of the same algorithm on a particular problem instance may yield a different result, we thus run each algorithm 10 times and report the average results.

Table 1
The objective values and computational times obtained using different algorithms tested on the linear network topology

Linear			HPSO		GA		Exact	
n	r	d	$Z(\boldsymbol{X})$	CPU time	$Z(\boldsymbol{X})$	CPU time	$Z(\boldsymbol{X})$	CPU time
6	8	0.3	535.282	6.600	535.282	13.800	535.282	121
		0.5	844.520	6.800	844.613	14.200	835.819	110
		0.8	855.011	7.000	856.556	14.200	847.368	113
	12	0.3	1437.252	11.000	1616.327	18.200	NA	NA
		0.5	1654.923	11.400	1898.597	19.200	NA	NA
		0.8	2712.131	11.800	2785.471	20.400	NA	NA
7	9	0.3	176.638	8.000	176.638	14.400	176.638	3627
		0.5	461.402	8.400	466.185	15.200	438.202	3625
		0.8	812.952	8.600	812.952	15.400	812.952	3612
	13	0.3	1190.142	17.800	1204.166	20.400	NA	NA
		0.5	2127.170	12.800	2151.343	20.800	NA	NA
		0.8	3451.466	13.600	3524.238	22.000	NA	NA
8	10	0.3	640.656	9.000	673.318	17.600	NA	NA
		0.5	631.792	9.400	750.047	17.000	NA	NA
		0.8	1229.001	9.800	1282.784	17.400	NA	NA
	16	0.3	2009.679	17.200	2293.542	22.800	NA	NA
		0.5	3353.591	17.000	3709.509	25.000	NA	NA
		0.8	7760.850	18.600	8214.603	26.400	NA	NA
9	11	0.3	1033.969	10.800	1036.263	17.400	NA	NA
		0.5	1295.372	11.000	1302.985	17.800	NA	NA
		0.8	2391.183	11.600	2443.743	18.600	NA	NA
	17	0.3	2444.573	17.400	2602.165	25.800	NA	NA
		0.5	4379.877	18.000	4453.623	27.200	NA	NA
		0.8	7013.613	19.400	7253.068	29.000	NA	NA
Average			2101.794	12.208	2203.667	19.592		

NA: not available.

4.1. Dataset description

To assess the comparative performances of the competing algorithms, a large simulated MOTAP dataset is generated. Five typical types of PIG as shown in Fig. 1, namely linear, ladder, cross, tree, and star are considered. For each PIG, we set the numbers of processors (n) and modules (r) equal to $(6,8)$, $(6,12)$, $(7,9)$, $(7,13)$, $(8,10)$, $(8,16)$, $(9,11)$, and $(9,17)$, respectively, in order to testify the algorithms with different problem scales. Also, for each setting of (n,r), we consider three different TIGs with various task interaction density d equivalent to 0.3, 0.5, and 0.8. Therefore, there are totally $5 \times 8 \times 3 = 120$ problem instances in our dataset which covers a broad range of real-world DCS applications with different network topology, problem scale, and task interaction density.

The values of the other system parameters are generated at random with the uniform distributions of the following ranges: the module accumulative execution time (AET) is between 15 and 25, the intermodule communication (IMC) load is between 15 and 25, the failure rate of processors and communication links is yielded in the ranges (0.00005, 0.00010) and (0.00015, 0.00030), the memory and computation capacity of each processor varies from 100 to 200, and the memory and computation requirement of each module ranges from 1 to 60.

4.2. Comparative performances

We apply each algorithm to solve all of the 120 problem instances in our dataset and list their comparative results in Tables 1–5. The obtained MOTAP objective values $Z(\boldsymbol{X})$ from each competing algorithm and the needed CPU times (in seconds) are reported. We have the following observations.

Table 2
The objective values and computational times obtained using different algorithms tested on the ladder network topology

Ladder			HPSO		GA		Exact	
n	r	d	$Z(\boldsymbol{X})$	CPU time	$Z(\boldsymbol{X})$	CPU time	$Z(\boldsymbol{X})$	CPU time
6	8	0.3	542.684	6.200	542.684	11.800	542.684	119
		0.5	838.685	6.000	841.279	12.000	837.607	107
		0.8	869.650	6.600	882.522	12.000	863.513	116
	12	0.3	1128.308	10.000	1221.942	15.600	NA	NA
		0.5	2055.484	10.200	2070.345	16.600	NA	NA
		0.8	2505.686	10.600	2686.133	16.600	NA	NA
7	9	0.3	562.014	6.600	580.880	12.600	542.154	3129
		0.5	443.425	6.800	561.197	12.600	442.815	3512
		0.8	1012.775	7.000	1179.836	12.800	1011.704	3647
	13	0.3	745.058	11.200	746.513	18.600	NA	NA
		0.5	1546.016	11.800	1803.112	18.800	NA	NA
		0.8	3302.137	12.400	3773.913	20.400	NA	NA
8	10	0.3	548.692	7.600	611.945	15.800	NA	NA
		0.5	991.385	8.000	1099.826	16.600	NA	NA
		0.8	2351.041	8.400	2367.652	16.400	NA	NA
	16	0.3	1832.447	21.200	2042.980	22.200	NA	NA
		0.5	3158.701	22.200	3297.412	23.800	NA	NA
		0.8	6731.436	23.600	6852.184	25.000	NA	NA
9	11	0.3	465.900	11.000	512.726	16.000	NA	NA
		0.5	1554.876	11.400	1657.546	16.400	NA	NA
		0.8	2512.232	11.600	2608.806	18.400	NA	NA
	17	0.3	3023.153	15.400	3308.757	25.200	NA	NA
		0.5	4562.435	16.200	5067.531	25.800	NA	NA
		0.8	9271.078	17.600	9467.498	28.000	NA	NA
Average			2189.804	11.650	2324.384	17.917		

Table 3
The objective values and computational times obtained using different algorithms tested on the cross-network topology

Cross			HPSO		GA		Exact	
n	r	d	$Z(\boldsymbol{X})$	CPU time	$Z(\boldsymbol{X})$	CPU time	$Z(\boldsymbol{X})$	CPU time
6	8	0.3	544.654	6.000	549.915	12.400	539.658	116
		0.5	836.685	6.400	838.148	12.800	834.601	111
		0.8	875.365	6.200	879.749	13.000	871.513	121
	12	0.3	1108.366	10.000	1225.599	16.000	NA	NA
		0.5	2018.958	10.400	2020.270	17.000	NA	NA
		0.8	2455.991	10.400	2573.036	17.800	NA	NA
7	9	0.3	552.112	7.400	562.007	13.800	527.110	3454
		0.5	413.873	7.400	471.464	13.800	413.873	3763
		0.8	874.867	7.600	1050.565	13.800	874.867	3377
	13	0.3	759.089	10.800	822.866	17.600	NA	NA
		0.5	1514.997	11.000	1707.580	18.800	NA	NA
		0.8	3046.973	11.800	3254.543	20.400	NA	NA
8	10	0.3	537.947	8.600	570.072	15.200	NA	NA
		0.5	920.917	10.000	950.204	15.000	NA	NA
		0.8	2089.825	10.400	2342.643	15.600	NA	NA
	16	0.3	1678.774	18.200	1933.921	23.400	NA	NA
		0.5	3052.037	15.400	3128.797	24.200	NA	NA
		0.8	6260.065	16.600	6951.114	25.400	NA	NA
9	11	0.3	1190.203	9.800	1229.421	17.200	NA	NA
		0.5	1000.945	9.600	1111.596	17.000	NA	NA
		0.8	1990.747	10.200	2119.287	17.800	NA	NA
	17	0.3	2293.922	15.000	3052.894	24.800	NA	NA
		0.5	6308.418	17.000	6560.250	26.800	NA	NA
		0.8	6450.499	17.600	6545.803	27.400	NA	NA
Average			2032.343	10.992	2185.489	18.208		

First, although the Exact algorithm can find the global optimal solution for the MOTAP, it is limited to the use of small problems. In particular, in our simulations, the Exact algorithm successfully solves the problem instances with $(n, r) = (6, 8)$ and $(7, 9)$ for all tested network topologies and task interaction density d; however, it fails to deliver solutions for problems of larger size due to the exponentially increasing computational time. *Secondly*, among the 30 problem instances which have been solved by the Exact algorithm, the HPSO algorithm can also obtain the global optimal solution for 11 of them while the GA can only report exact solution for five problems. For those problems which the Exact algorithm cannot deal with, the HPSO algorithm always produces better objective values than GA. *Thirdly*, as for the computational times consumed by the comparative algorithms, the Exact method spends nearly two minutes for solving the problems in the $(n = 6,\ r = 8)$ category, and about 2 h for tackling the problems with $(n = 7,\ r = 9)$, the other problems are too hard for the Exact algorithm to solve within 24 h. On the other hand, the HPSO and the GA algorithms can derive exact or sub-optimal solutions for all of the test problem instances, manifesting the need for using metaheuristic algorithms in solving complex MOTAP problems. The HPSO algorithm is more computationally efficient than the GA algorithm in our simulations, although the latter can also obtain exact or approximate solutions for all test problems within 30 s.

4.3. Information gain

The HPSO is a metaheuristic algorithm which is a master strategy guiding other heuristics to search in a strategic way instead of simply moving to local optima. The aim of this strategy is to look beyond neighborhood such that the searching will not get stuck by the barrier of local optimality. This feature also makes HPSO being able to manage the particles to interact with each other through an indirect media using swarm's local or global experiences. The individual particles learn more about the solution space and receive informa-

Table 4
The objective values and computational times obtained using different algorithms tested on the tree network topology

Tree			HPSO		GA		Exact	
n	r	d	$Z(\boldsymbol{X})$	CPU time	$Z(\boldsymbol{X})$	CPU time	$Z(\boldsymbol{X})$	CPU time
6	8	0.3	545.654	6.400	557.245	13.400	545.654	121
		0.5	838.223	6.600	839.090	13.600	833.609	105
		0.8	866.613	6.600	866.613	13.800	866.613	114
	12	0.3	1112.210	10.400	1202.863	17.600	NA	NA
		0.5	2055.032	10.600	2080.255	18.400	NA	NA
		0.8	2508.617	11.000	2713.774	19.800	NA	NA
7	9	0.3	582.695	8.000	592.047	14.400	547.157	2978
		0.5	441.818	10.000	443.425	14.600	441.818	3508
		0.8	1054.109	8.800	1064.124	15.000	1001.703	3255
	13	0.3	802.547	11.800	840.213	19.200	NA	NA
		0.5	1709.632	11.800	1831.987	19.800	NA	NA
		0.8	3447.616	12.800	3668.092	21.200	NA	NA
8	10	0.3	509.089	8.600	542.089	15.800	NA	NA
		0.5	996.833	9.000	1006.679	16.600	NA	NA
		0.8	2348.909	10.000	2390.924	17.400	NA	NA
	16	0.3	1850.211	17.400	2037.972	23.800	NA	NA
		0.5	3082.347	18.200	3253.075	25.000	NA	NA
		0.8	6501.722	19.200	6720.778	26.400	NA	NA
9	11	0.3	492.454	11.200	514.290	18.000	NA	NA
		0.5	1454.682	11.200	1646.251	18.400	NA	NA
		0.8	2550.584	12.000	2752.928	19.200	NA	NA
	17	0.3	3149.375	17.800	3298.810	26.800	NA	NA
		0.5	4526.521	18.200	4966.499	27.400	NA	NA
		0.8	9499.565	19.400	9656.569	28.800	NA	NA
Average			2205.294	11.958	2311.941	19.350		

tion gain as the evolution proceeds. Since the particles learn by interactions, all the *pbest* can continuingly improve during evolution and will resort to near-optimal solutions. We thus can define the information gain as how pure the collection of *pbest* is, and we would like to use a measure to quantify the information gain obtained from the social interactions conducted by the HPSO.

Here, we employ a statistic measure widely used in information theory, named *entropy*, for measuring the purity of the set of all *pbest* as the evolution proceeds. Let $P_j(s)$ be the probability that a *pbest* allocates the jth module to the sth processor, viz.

$$P_j(s) = \Pr\{pbest_{ij} = s|_i = 1, 2, \ldots, N\} \quad \forall s = 1, 2, \ldots, n. \tag{14}$$

The entropy over the probability density function $P_j(s)$, $s = 1, 2, \ldots, n$, is given by

$$E_j = -\sum_{s=1}^{n} P_j(s)\log_2 P_j(s). \tag{15}$$

E_j describes the purity over the collection of all *pbest* for the decision about the allocation of the jth module. To provide an overall measure, we compute the expected entropy as $\bar{E} = \sum_{j=1}^{r} E_j/r$. The smaller the value of $\bar{E}$ is, the higher the purity of the set of all *pbest*, and the higher the information gain.

Fig. 4 corresponds to the curve showing the variations of the expected entropy ($\bar{E}$) as the number of fitness evaluations increases. The expected entropy decreases rapidly from 2.23 during the stage before 20,000 fitness evaluations because the particles widely explore the solution space and gain rich information through interactions, resulting in a fast decreasing rate on $\bar{E}$. After the 20,000th fitness evaluation, the decreasing rate becomes gentle, and the value of $\bar{E}$ finally decreases to 1.05. This phenomenon reflects the fact that the particles are more similar to each other at this stage, only few cells of the particles can be further tuned through interactions, and the information gain has been maximized.

Table 5
The objective values and computational times obtained using different algorithms tested on the star network topology

Star			HPSO		GA		Exact	
n	*r*	*d*	$Z(\boldsymbol{X})$	CPU time	$Z(\boldsymbol{X})$	CPU time	$Z(\boldsymbol{X})$	CPU time
6	8	0.3	540.654	6.400	547.495	14.200	540.654	130
		0.5	842.495	6.600	843.586	14.600	839.006	106
		0.8	894.584	6.800	896.374	14.400	861.512	115
	12	0.3	1414.331	10.000	1493.162	19.000	NA	NA
		0.5	1850.526	10.000	1926.071	19.400	NA	NA
		0.8	3108.935	10.400	3241.080	20.600	NA	NA
7	9	0.3	392.946	7.400	401.897	15.400	392.946	3711
		0.5	582.825	7.400	649.157	15.800	575.422	3654
		0.8	1249.849	8.800	1251.402	15.400	1085.624	3085
	13	0.3	1588.639	12.600	1644.372	21.200	NA	NA
		0.5	1847.337	12.600	1890.389	21.200	NA	NA
		0.8	2920.360	14.200	2931.999	22.000	NA	NA
8	10	0.3	700.897	10.000	753.302	16.800	NA	NA
		0.5	1054.911	10.000	1275.549	17.800	NA	NA
		0.8	1235.156	10.400	1307.135	17.800	NA	NA
	16	0.3	2399.263	19.200	2490.227	27.400	NA	NA
		0.5	4400.804	22.800	4452.371	27.200	NA	NA
		0.8	6787.760	19.400	6923.338	27.200	NA	NA
9	11	0.3	1419.397	11.400	1437.542	18.600	NA	NA
		0.5	981.723	11.800	1272.548	19.000	NA	NA
		0.8	2529.373	12.200	2568.038	19.400	NA	NA
	17	0.3	2852.755	18.000	2912.041	27.600	NA	NA
		0.5	5060.010	18.400	5346.050	29.200	NA	NA
		0.8	7718.745	19.400	7784.645	30.200	NA	NA
Average			2265.595	12.342	2343.324	20.475		

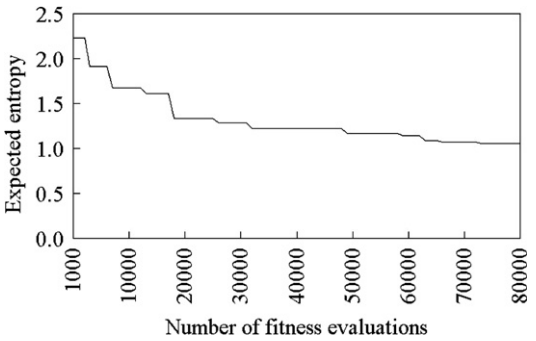

Fig. 4. The expected entropy ($\bar{E}$) over the collection of all the *pbest* versus the number of fitness evaluations.

4.4. Worst-case analysis

To ensure the Quality of Service (QoS) of an application running in a DCS, it is crucial to analyze the worst optimal system reliability and cost we could get after a specific number of repetitive runs of our algorithm. Since the proposed HPSO is a stochastic algorithm and each separate run of the program could yield a different result, we need to analyze it's worst case after repetitive testing. As we have formulated the multiple objectives of the MOTAP as a weighted summation ($Z(\boldsymbol{X})$), we run the HPSO 1000 times on a problem instance and plot the variations of the worst optimal $Z(\boldsymbol{X})$ we could get after different numbers of repetitive runs. The worst-case analysis shown in Fig. 5 may be important when the application needs to guarantee the

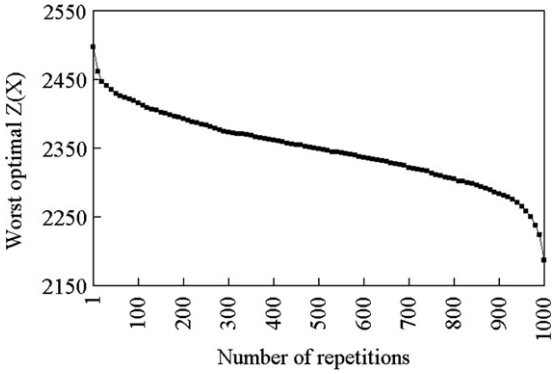

Fig. 5. The worst optimal $Z(\boldsymbol{X})$ versus the number of repetitive runs of HPSO.

quality of the solution. For example, the HPSO can derive a task allocation with $Z(\boldsymbol{X})$ value lower than 2400 after 120 repetitive runs in the worst case. But in the general case the HPSO is likely to derive such a quality-guaranteed solution in a single run with a very high probability equivalent to $1 - 120/1000 = 88\%$.

5. Conclusions

In this paper, we have proposed a hybrid particle swarm optimization (HPSO) algorithm which intends to minimize the cost and maximize the reliability simultaneously for executing programs in a distributed computing system. The HPSO initializes a swarm of particles each of which corresponds to a candidate solution to the underlying problem. These particles iteratively improve their quality through collective experiences of personal cognition and social interactions. This is a positive feedback process such that the intelligence of the entire swarm is enriched. Penalty functions tailored to the system constraints are devised in order to deal with infeasible solutions. The HPSO embeds a local search heuristic into the evolutionary iterations for expediting the convergence. The performance of the proposed method is compared to a genetic algorithm and an exact algorithm. The experimental results manifest that the HPSO reports quality solutions on a large set of simulated instances involving different problem scales, task interaction densities, and network topologies. The information gain and the worst-case analyses of the HPSO have been theoretically and empirically conducted.

References

[1] C.H. Lee, K.G. Shin, Optimal task assignment in homogeneous networks, IEEE Transactions on Parallel and Distributed Systems 8 (1997) 119–129.
[2] A.T. P, C.S.R. Murthy, Optimal task allocation in distributed systems by graph matching and state space search, The Journal of Systems and Software 46 (1999) 59–75.
[3] A. Ernst, H. Hiang, M. Krishnamoorthy, Mathematical programming approaches for solving task allocation problems, in: Proceedings of the 16th National Conference of Australian Society of Operations Research, 2001.
[4] S. Kartik, S.R. Murthy, Task allocation algorithms for maximizing reliability of distributed computing systems, IEEE Transactions on Computers 46 (1997) 719–724.
[5] S. Srinivasan, N.K. Jha, Safety and reliability driven task allocation in distributed systems, IEEE Transactions on Parallel and Distributed Systems 10 (1999) 238–251.
[6] S. Kartik, S.R. Murthy, Improved task-allocation algorithms to maximize reliability of redundant distributed computing systems, IEEE Transactions on Reliability 44 (1995) 575–586.
[7] C.C. Hsieh, Optimal task allocation and hardware redundancy policies in distributed computing systems, European Journal of Operational Research 147 (2003) 430–447.
[8] C.C. Hsieh, Y.C. Hsieh, Reliability and cost optimization in distributed computing systems, Computers and Operations Research 30 (2003) 1103–1109.
[9] V.M. Lo, Task assignment in distributed systems, Ph.D. dissertation, Department of Computer Science, University of Illinois, October, 1983.
[10] M.S. Lin, D.J. Chen, The computational complexity of the reliability problem on distributed systems, Information Processing Letters 64 (1997) 143–147.

[11] S.M. Shatz, J.P. Wang, M. Goto, Task allocation for maximizing reliability of distributed computer systems, IEEE Transactions on Computers 41 (1992) 1156–1168.
[12] A. Billionnet, M.C. Costa, A. Sutter, An efficient algorithm for a task allocation problem, Journal of ACM 39 (1992) 502–518.
[13] G.H. Chen, J.S. Yur, A branch-and-bound-with-underestimates algorithm for the task assignment problem with precedence constraint, in: Proceedings of the 10th International Conference on Distributed Computing Systems, 1990, 494–501.
[14] D.M. Nicol, D.R. O'Hallaron, Improved algorithm for TAP pipelined and parallel computations, IEEE Transactions on Computers 40 (1991) 295–306.
[15] D. Fernandez-Baca, A. Medepalli, Parametric task allocation on partial k-trees, IEEE Transactions on Computers 42 (1993) 738–742.
[16] E.S.H. Hou, N. Ansari, H. Ren, A genetic algorithm for multiprocessor scheduling, IEEE Transactions on Parallel and Distributed Systems 5 (1994) 113–120.
[17] A.B. Hadj-Alouane, J.C. Bean, K.G. Murty, A hybrid genetic/optimization algorithm for a task allocation problem, Journal of Scheduling 2 (1999) 189–201.
[18] A.K. Tripathi, B.K. Sarker, N. Kumar, A GA based multiple task allocation considering load, International Journal of High Speed Computing (2000) 203–214.
[19] D.P. Vidyarthi, A.K. Tripathi, Maximizing reliability of distributed computing system with task allocation using simple genetic algorithm, Journal of Systems Architecture 47 (2001) 549–554.
[20] F.T. Lin, C.C. Hsu, Task assignment scheduling by simulated annealing, in: Proceeding of Conference on Computer and Communication Systems, 1990, pp. 279–283.
[21] Y. Hamam, K.S. Hindi, Assignment of program tasks to processors: a simulated annealing approach, European Journal of Operational Research 122 (2000) 509–513.
[22] F. Glover, Tabu search—Part I, ORSA Journal of Computing 1 (1989) 190–206.
[23] M. Dorigo, L. Gambardella, Ant colony system: a cooperative learning approach to the traveling salesman problem, IEEE Transactions on Evolutionary Computation 1 (1997) 53–66.
[24] J. Kennedy, R.C. Eberhart, Particle swarm optimization, in: Proceedings of IEEE International Conference on Neural Networks, vol. 4, 1995, pp. 1942–1948.
[25] M. Clerc, J. Kennedy, The particle swarm explosion, stability, and convergence in a multidimensional complex space, IEEE Transaction on Evolutionary Computation 6 (2002) 58–73.
[26] I.C. Trelea, The particle swarm optimization algorithm: convergence analysis and parameter selection, Information Processing Letters 85 (2003) 317–325.
[27] Y. Shi, R.C. Eberhart, A modified particle swarm optimizer, in: Proceedings of the IEEE International Conference on Evolutionary Computation, 1998, pp. 69–73.
[28] M. Clerc, The swarm and the queen: towards a deterministic and adaptive particle swarm optimization, in: Proceedings of the 1999 Congress on Evolutionary Computation, 1999, pp. 1951–1957.
[29] X.H. Wang, J.J. Li, Hybrid particle swarm optimization with simulated annealing, in: Proceedings of the 3rd International Conference on Machine Learning and Cybernetics, Shanghai, vol. 4, 2004, pp. 2402–2405.
[30] H.J. Meng, P. Zheng, R.Y. Wu, X.J. Hao, Z. Xie, A hybrid particle swarm algorithm with embedded chaotic search, in: Proceedings of the IEEE International Conference on Cybernetics and Intelligent Systems, Singapore, vol. 1, 2004, pp. 367–371.
[31] Y. Liu, Z. Qin, Z. Shi, Hybrid particle swarm optimizer with line search, in: Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, vol. 4, 2004, pp. 3751–3755.