Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2014, Article ID 151394, 13 pages
http://dx.doi.org/10.1155/2014/151394

Research Article

Simulated Annealing Algorithm Combined with Chaos for Task Allocation in Real-Time Distributed Systems

Wenbo Wu, Jiahong Liang, Xinyu Yao, and Baohong Liu

College of Information System and Management, National University of Defense Technology, Changsha, Hunan 410073, China

Correspondence should be addressed to Wenbo Wu; just4thin@gmail.com

Received 17 May 2014; Revised 24 July 2014; Accepted 3 August 2014; Published 14 August 2014

Academic Editor: Jun-Juh Yan

This paper addresses the problem of task allocation in real-time distributed systems with the goal of maximizing the system reliability, which has been shown to be NP-hard. We take account of the deadline constraint to formulate this problem and then propose an algorithm called chaotic adaptive simulated annealing (XASA) to solve the problem. Firstly, XASA begins with chaotic optimization which takes a chaotic walk in the solution space and generates several local minima; secondly XASA improves SA algorithm via several adaptive schemes and continues to search the optimal based on the results of chaotic optimization. The effectiveness of XASA is evaluated by comparing with traditional SA algorithm and improved SA algorithm. The results show that XASA can achieve a satisfactory performance of speedup without loss of solution quality.

1. Introduction

In many application domains, (e.g., astronomy, genetic engineering, and military systems), increased complexity and scale has led to the need for more powerful computation resources; distributed systems (DS) have emerged as a powerful platform for addressing this issue, alternating to traditional high performance computing systems. DS consists of a set of cooperating nodes (either homogeneous or heterogeneous) communicating over the communication links. An application running in a DS could be divided into a number of tasks and executed concurrently on different nodes in the system, referred to as the task allocation problem (TAP). To improve the performance of DS, several studies have been devoted to the TAP with the main concern on the performance measures such as minimizing the execution and communication cost [1–3], minimizing the application turnaround time [4, 5], and achieving better fault tolerance [6, 7].

On the other hand, the real-time property is required in many DS (e.g., military systems). In such system, the application should complete its work before deadline, not only promising the logical correctness. While the complexity of DS could increase the potential of system failure, because in such a large and complex system, the nodes and communication links failures are inevitable. Hence, the reliability is a crucial requirement for DS, especially for the real-time DS (RTDS).

Distributed system reliability (DSR) has been defined by Kumar et al. [8] as the probability for the successful completion of distributed programs which requires that all the allocated processors and involved communication links are operational during the execution lifetime. Redundancy and diversity is the traditional technique to attain better reliability [6, 7, 9–14]. They process hardware and/or software redundancy, hence impose extra cost. Moreover, in many situations, the system configuration is fixed and we have no freedom to introduce system redundancy. Task allocation is the alternative way to improve DS reliability, and this method does not require additional resources, neither hardware nor software.

The TAP with the goal of maximizing the DSR is a typical combinatorial optimization problem; unfortunately, this has been shown to be NP-hard in strong sense, and the computational complexity of optimal algorithms (e.g., branch and bound technique) is exponential in nature. We cannot obtain the optimal results in reasonable time for large scale problems. Hence, several heuristic and metaheuristic algorithms have been implemented, such as genetic algorithm

(GA) [7, 14, 15], simulated annealing algorithm (SA) [16], particle swarm optimization (PSO) [17], honeybee mating optimization (HMO) [18], cat swarm optimization (CSO) [19], and iterated greedy algorithm (IG) [20]. These algorithms may obtain suboptimal results, but they can sharply reduce the calculation time.

A common thread among these algorithms is that they all start from a randomly chosen initial solution or a set of solutions in the solution space and then repeat the exploration-decision procedure until convergence and obtaining good enough solutions (maybe suboptimal results) [21]. Here, exploration means obtaining new solutions based on the current solution, in SA algorithm; for instance, new solutions are chosen from the neighbors of the current solution. Decision is made after exploration: a new solution is either accepted or rejected according to some rules, if the new solution is accepted, then it becomes new current solution, and move on, otherwise drop it, start new exploration. According to a series of the exploration-decision procedures, the quality of the obtained solutions becomes better and better until meeting the termination condition which is often defined as some convergent situations. Hence, the convergence speed of algorithms is affected by the choice of initial solutions and rules that are applied in the exploration-decision procedures.

Simulated annealing algorithm is one of the earliest and most wildly used optimization approaches; it introduces random factor in searching process and models the annealing of solids as Metropolis process [22]. SA algorithm accepts "worse" solution with some probabilities related to the current temperature, so it can escape from the local optima and find the global optimal solution. The convergence speed of SA algorithm is depending on its initial solution and cooling schedule.

Chaos is a bounded unstable dynamic behavior that exhibits sensitive dependence on initial conditions and includes infinite unstable periodic motions [23]. Although it appears to be stochastic, it occurs in a deterministic nonlinear system under deterministic conditions. In recent years, chaotic optimization algorithm (COA) has aroused intense interests due to its ergodicity, easy implementation, and ability to escape local optima [24]. However, COA is lack of heuristic, mostly needs a large number of iterations to reach the global optimum, which means its convergence speed is slow.

In this paper, we propose a combinational algorithm called XASA (Chaotic Adaptive Simulated Annealing), where X alludes to the Greek spelling of chaos ($\chi\alpha o\varsigma$) and which is proposed to solve the TAP in RTDS with the goal of maximizing the DSR. We take into account several kinds of constraints including deadline. XASA starts from COA and obtains several optima via its ergodicity, then SA algorithm will operate based on these optima in relative smaller ranges to find the best solution. This method can overcome the slow convergence of SA algorithm and COA without loss of solution quality.

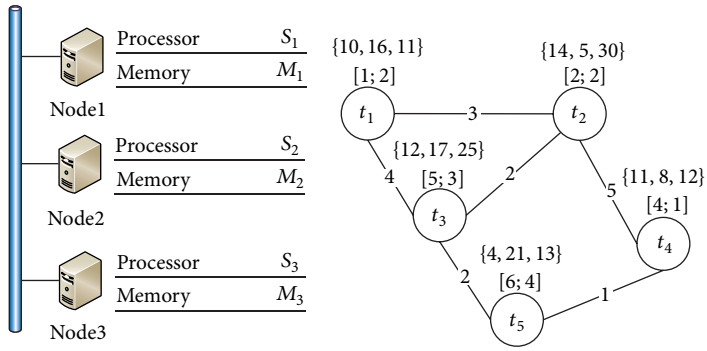
The rest of this paper is organized as follows. Section 2 presents the related work in the application of SA algorithm and Chaotic Optimization to the TAP, and the contributions of this paper are stated. Section 3 describes the formulation of the TAP with the goal of maximizing reliability, and the solution approach is presented in Section 4 with some details of implementation. Section 5 discusses the performance evaluation of proposed algorithm according to several experiments and analyses. And Section 6 concludes this work.

2. Related Works

The idea of simulated annealing algorithm was proposed by Metropolis et al. [22] in 1953, and was applied to optimization problems by Kirkpatrick et al. [25] in 1983. To our knowledge, the first application of SA algorithm to the TAP was made by van Laarhoven et al. [26] in 1992, which applied SA algorithm to a job shop scheduling problem. From then on, several works [27–29] have been done that compare SA algorithm to other optimization algorithms for problems related to the TAP. Attiya and Hamam applied SA algorithm to solve the TAP and compared it with branch-and-bound technique in 2006 in terms of maximizing the reliability of DS [16]; extending this work, Faragardi et al. proposed improved SA algorithms to solve this problem [30, 31], using the hybrid of SA and tabu search with a nonmonotonic cooling schedule in 2012 and adding systematic search of neighborhood and memory to SA algorithm in 2013.

COA often combines with other optimization algorithm to overcome its drawbacks and take advantage of its beneficial property such as ergodicity, for example, chaotic simulated annealing (CSA) [32], chaotic particle swarm optimization (CPSO) [23], and chaotic improved imperialist competitive algorithm (CICA) [33] et al. CSA was proposed by Chen and Aihara [32] to solve combinatorial optimization problems in 1995, which used Hopfield neural networks. Then, several other papers have expanded this work [34–37]. Mingjun and Huanwen have proposed another version of CSA to solve the optimization problem of continuous functions [38]. Most of these works focus on the application of continuous functions optimization, while the method proposed by Chen and Aihara is actually based on artificial neural network, not SA algorithm. Ferens and Cook [39] adapted CSA developed by Mingjun and Huanwen into the TAP in 2013, where chaos was infused into a solution by setting the number of perturbations made by the value of a chaotic variable. However, this method does not make full use of the beneficial property of COA and does not improve the convergence speed either.

The present paper differs from the above mentioned researches because it can combine the advantages of both two algorithms. Firstly, with the ergodicity property, COA can get the skeleton of solution space by chaotic walking in it, thereby, preventing the result from falling into local optima. Secondly, based on the results of COA, we can easily determine the cooling schedule of SA algorithm which is very important to the performance of SA algorithm but hard to deal with. Lastly, several adaptive schemes are used in SA algorithm; all these schemes including COA preliminary results can increase the convergence speed significantly.



(a) A distributed system (b) A task graph

Figure 1: Results of the evaluation.

3. Problem Statement

We consider a heterogeneous DS that runs a real-time application. Each node of DS may have different processing speed, memory size, and failure rate. Moreover, the communication links may also have different bandwidth and failure rate. The state of nodes and communication links is either operational or failed, and the failure events are statistically independent. We also assume that the failure rates of nodes and communication links are constant.

There are M tasks to be executed on a DS with N nodes, and $M > N$ for most cases. Tasks executed on the node require resources, including processing load resources and memory space. Additionally, two tasks executed on different nodes require communication bandwidth to communicate with each other. Figure 1 illustrates a simple case consisting of 5 tasks and 3 nodes. An application running on DS can be represented by a task interaction graph $G(V, E)$ as shown in Figure 1(b), where V represents a set of tasks and E represents the interactions between two tasks. Each task $i \in V$ is associated with two properties: $\{q_{i1}, q_{i2}, \ldots, q_{iN}\}$ represents the execution time of the task at different node and $[s_i; m_i]$ represents the processing load and memory requirements of the task. And the label of each edge $(i, j) \in E$ represents the communication requirements among tasks.

The purpose of this work is to find a task assignment that all M tasks are assigned to N nodes (note that one task should be assigned to one and only one node, while one node can execute multitasks or none), so that the overall system reliability is maximized, the deadline and other requirements of tasks are satisfied, and the capacities of the system resources are not violated.

3.1. Notations. The notations that are used to formulate the problem are listed in Abbreviation Section.

3.2. System Reliability. Reliability of a distributed system may be defined as the probability that the system can run the entire application successfully [7, 14, 40]. Due to the independence of the failures of the node and path, the system reliability is the product of the components reliabilities. Which is the product of the reliabilities of all nodes and communication links.

We assume that all components of the node except processor are perfect, which means the reliability of the node equals the reliability of its processor. The reliability of the processor at time t is $e^{-\lambda_k t}$ [16]; under a task assignment X, the total execution time of tasks that are assigned to p_k is $\sum_{i=1}^{M} x_{ik} e_{ik}$, so the reliability of the node p_k is

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

Similarly, the reliability of the communication link l_{kb} at time t is $e^{-\varphi_{kb} t}$, under a task assignment X; the total communication time via l_{kb} is $\sum_{i=1}^{M} \sum_{i \neq j} x_{ik} x_{jb} (c_{ij}/\omega_{kb})$, so the reliability of the communication link l_{kb} is

$$R_{kb}(X) = e^{-\varphi_{kb} \sum_{i=1}^{M} \sum_{i \neq j} x_{ik} x_{jb} (c_{ij}/\omega_{kb})}. \tag{2}$$

Hence, we obtain the reliability of the system:

$$R(X) = \prod_{k=1}^{N} R_k(X) \prod_{k=1}^{N-1} \prod_{b=k+1}^{N} R_{kb}(X) = e^{-Y(X)}, \tag{3}$$

where

$$Y(X) = \sum_{k=1}^{N} \sum_{i=1}^{M} \lambda_k x_{ik} e_{ik} + \sum_{k=1}^{N} \sum_{b=k+1}^{N} \sum_{i=1}^{M} \sum_{i \neq j} \varphi_{kb} x_{ik} x_{jb} \left(\frac{c_{ij}}{\omega_{kb}} \right). \tag{4}$$

3.3. Constraints. In order to achieve a satisfactory allocation, there are several basic constraints of the TAP in RTDS that should be met. Traditionally, the aim of allocation constraints is devoted to not violate the availability of the system resources, like memory capacity. While the deadline requirement should be met for real-time property as well.

(i) Memory Constraint. The total amount of memory requirements of tasks assigned to a node should not exceed the capacity of the node. That is,

$$\sum_{i=1}^{M} m_i x_{ik} \leq M_k, \quad \forall k \in [1, N]. \tag{5}$$

(ii) Computation Resource Constraints. The total amount of computation resource requirements of tasks assigned to a node should not exceed the capacity of the node. That is,

$$\sum_{i=1}^{M} s_i x_{ik} \le S_k, \quad \forall k \in [1, N]. \tag{6}$$

(iii) Communication Resource Constraints. The total amount of communication resource requirements of tasks via a communication link should not exceed the capacity of the link. That is,

$$\sum_{i=1}^{M} \sum_{j \ne i} c_{ij} x_{ik} x_{jb} \le C_{kb}, \quad 1 \le k < b \le N. \tag{7}$$

(iv) Deadline Constraints. All tasks should complete execution before their deadline. Since there is no priority of tasks, all tasks that are assigned to a node can execute in any order. We should take account of the worst case, that is, considering task always executed at last. Hence, the deadline constraints is

$$\sum_{k=1}^{N} x_{ik} \sum_{j=1}^{M} e_{jk} x_{jk} \le d_i, \quad \forall i \in [1, M]. \tag{8}$$

3.4. Problem Formulation. According to the above discussion, we can tell that maximizing the reliability of RTDS is equivalent to minimizing the object function $Y(X)$, with all constraints mentioned before. Hence, we can formulate the TAP by the following combinatorial optimization problem:

$$\begin{aligned} &\min && Y(X) \\ &\text{subject to} && (5) \sim (8). \end{aligned} \tag{9}$$

4. Task Allocation Solution

This section describes basic SA algorithm briefly at first, with the discussion of the cooling schedule which has a significant effect on its convergence speed then presents the XASA and explains the details of how it can be applied into the TAP in terms of the statement proposed in this paper.

4.1. Basic Simulated Annealing Algorithm. The SA algorithm starts from a randomly chosen initial solution and generates a series of Markov chains according to the descent of the control parameter (i.e., temperature). In these Markov chains, a new solution is chosen by making a small random perturbation of the solution, and, if the new solution is better, then it is kept, but if it is worse it is kept with some probability related to the current temperature and the difference between the new solution and the previous solution. According to a series of iteration of solutions, an optimal one was found. The SA algorithm applied in the TAP is listed as follows.

Step 1. Choose an initial task arrangement (X_s) at random.

Step 2. Calculate the cost (f_s) of X_s.

Step 3. Set the initial solution as the optimal $f^* \leftarrow f_s$, $X^* \leftarrow X_s$.

Step 4. Initialize the temperature ($T = T_0$).

Step 5. Select a neighbor (X_n) of X_s.

Step 6. Calculate the cost (f_n) of X_n.

Step 7. If $f_n \le f_s$, then $X_s \leftarrow X_n$ and $f_s \leftarrow f_n$, otherwise go to Step 9.

Step 8. If $f_n < f^*$, then $X^* \leftarrow X_n$ and $f^* \leftarrow f_n$, go to Step 10.

Step 9. If random$(0, 1) < e^{-(f_n - f_s)/T}$, then $X_s \leftarrow X_n$ and $f_s \leftarrow f_n$.

Step 10. Repeat Step 5 to Step 9 for a given number of iterations.

Step 11. Reduce the temperature via some cooling function $T = f(T)$.

Step 12. If the termination condition is satisfied (e.g. $T \le T_f$), then go to Step 13, otherwise go to Step 5.

Step 13. Output the solution.

Note that the neighborhood defines the procedure to move from a solution point to another solution point [16]. In this paper, a neighbor is obtained by randomly choosing a task i among M tasks and replacing its current assigned node with another randomly selected one.

4.2. Cooling Schedule. It has been shown that the SA algorithm converges to the global optimal with probability 1 [41], which needs a sufficiently slow cooling schedule (i.e., sufficient hot initial temperature, sufficient low final temperature, and sufficient slow cooling speed). However, the required slow cooling schedule may lead to an unacceptably long solution convergence time which can be exponential.

The cooling schedule is a set of parameters, which controls the procedure of SA algorithm so that it can be asymptotic converge to a suboptimal in reasonable time. The cooling schedule is made up of these parameters.

(i) The Initial Value of the Control Parameter (i.e., Temperature) T_0. The initial temperature represents one of the most important parameters in SA algorithm. If the initial temperature is very high, it will take very long time to be convergent. On the other hand, poor solutions are obtained if the initial temperature is low. A basic principle of choosing initial temperature is that the acceptance probability of worse solutions is close to 1. Which means the exchanging of neighboring solutions should be almost freely at first. Hence, we can determine the initial temperature T_0 via initial acceptance probability of worse solution P_0. In this paper, P_0 is set to be 0.9. $P_0 = e^{-\Delta/T_0}$, where Δ is the difference between the neighboring solutions, so $T_0 = -\Delta / \ln P_0$. We can use the

$\overline{\Delta} = f_{\max} - f_{\min}$ as the estimation of Δ, where $f_{\max}$ and $f_{\min}$ is the maximum and minimum of energy function among randomly chosen K solutions. So the initial temperature T_0 is determined by the following formula:

$$T_0 = \frac{f_{\min} - f_{\max}}{\ln P_0}. \tag{10}$$

(ii) The Cooling Function $F(T)$. The cooling function defines the cooling method of temperature T. A commonly used type of cooling function is exponential descent function: $F(T) = \alpha T$, where α is constant and $\alpha < 1$. So that the cooling speed depends on the parameter α, and we call it the cooling factor. A large value of α represents slow cooling, which yields good solution, but expensive in time. α is set to be slightly less than 1 in the most reported literatures, and it is chosen to be 0.95 in this paper.

(iii) The Final Value of Control Parameter T_f, Termination Condition in Another Word. The criterion for termination can be either final temperature or steady state of the system. The former one can control the total calculation time, but not the solution quality: a given final temperature may introduce calculation redundancy for small scale problem but obtain poor solution for large scale problem. On the other hand, the latter one can take into account both time and quality. In this paper, the SA algorithm will terminate if the solution remains unchanged (neither upgrading nor downgrading) for a given number of iterations. The number of iterations that solution remains unchanged is chosen to be $M \times N$. Furthermore, the validation of the final solution should be satisfied as well, which will be discussed later.

(iv) The Length of Markov Chain L_k. It is the number of inner loop repetitions. This parameter is chosen to be $M \times (N - 1)$, which is the size of solution neighborhood, because each task can be assigned to another $N - 1$ nodes.

The cooling schedule has a significant effect on the results of the algorithm, especially on the convergence speed. Besides, the initial solution of SA algorithm can affect the convergence speed as well.

4.3. Chaotic Optimization Algorithm. The chaotic variables are produced by the following well-known one-dimensional logistic map:

$$z_{k+1} = \mu z_k (1 - z_k), \quad k = 0, 1, \ldots, \tag{11}$$

where $\mu = 4$, $z_k \in [0, 1]$. The logistic map has special characters such as the ergodicity, stochastic property and sensitivity dependence on initial conditions. The chaotic optimization algorithm applied in this paper is listed as follows.

Step 1. Initialize the chaotic vector $\mathbf{Z}$ at random, note that the value of chaotic variables cannot be 0, 0.25, 0.5, 0.75, and 1.0, which are the fixed points of logistic map, and all chaotic variables are different from each other.

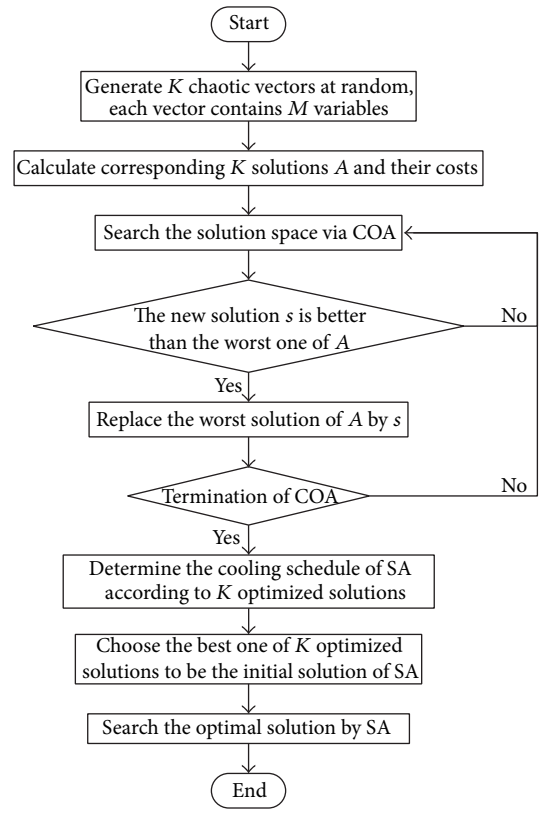


Figure 2: The flowchart of XASA.

Step 2. Generate the solution vector $\mathbf{A}$ via $\mathbf{Z}$, then generate the task assignment X via $\mathbf{A}$ and calculate the cost function f.

Step 3. Set the initial solution as the optimal $f^* = f$, $\mathbf{Z}^* = \mathbf{Z}$.

Step 4. Calculate new chaotic vector $\mathbf{Z}$ via formula [9].

Step 5. Generate X as Step 2, calculate the cost function f.

Step 6. If $f < f^*$, then $f^* = f$, $\mathbf{Z}^* = \mathbf{Z}$.

Step 7. Repeat Steps 4, 5, and 6, until f^* remains unchanged for a given number of iterations.

Note that the iteration number in Step 7 is chosen to be as same as SA algorithm, which is discussed before, while the validation of the solution is not required in COA, since it is not heuristic and it will take very long time to get convergence if there are few valid solutions in large scale cases.

4.4. Simulated Annealing Algorithm Combined with Chaos. The basic idea of our proposed algorithm is simulated annealing combined with chaotic search and adding some adaptive schemes to the cooling schedule so that we can improve the convergence speed without loss of solution quality. The flowchart of XASA is shown in Figure 2.

There are 4 schemes to speed up the convergence of SA algorithm.

First, we apply COA to find K optimized solutions, which is a preliminary search in the solution space and the solution distribution can be found according to the ergodicity of chaos system. Hence, we can search optimal solution via SA algorithm in a relative smaller range with optimized initial solution.

Second, the initial temperature T_0 can also be smaller based on the results of COA because we can replace the $f_{\max}$ and $f_{\min}$ with that of K optimized solutions.

Third, the length of Markov chain L_k is constant in SA algorithm, while it is adaptive in XASA. The algorithm will jump out of the inner loop if the rejections of new solution exceed a given threshold θ. The threshold is given by the formula: $\theta = \min(L_k \times \delta_1, \theta \times \delta_2)$ at each temperature. Because P_0 is set to be 0.9, we can set the initial value of θ to be $\lceil L_k \times 0.05 \rceil$. δ_1 represents the maximum threshold, and $\delta_1 < 1$; in this paper, it is set to be 0.6; δ_2 represents the increasing speed of θ, this is similar to the cooling factor α, so it is set to be 1.05.

Fourth, the cooling factor α is adaptive in XASA as well: the more solutions are accepted (both better and worse solutions) at the current temperature, the smaller the cooling factor is, and vice versa. The rationale is that high temperature at the beginning of the algorithm generates numerous solution acceptances; thus, a rapid reduction of temperature can be made, while fewer solutions will be accepted as the temperature is cooling down; thus, a slow cooling speed should be applied since we need carefully a solution search. Hence, the cooling factor of XASA is $\alpha' = \alpha \times e^{-\kappa/(\kappa+4\times n)}$, where κ is the acceptance number of last inner loop, and n is the actual length of Markov chains. Note that $\kappa \in [0, n]$, so $e^{-\kappa/(\kappa+4\times n)} \in [0.8187, 1]$ and $\alpha' \in [0.7778, 0.95]$.

To implement the algorithm, some details should be presented as follows.

(1) Solution Representation. In this paper, solutions are presented with a vector $\mathbf{A}(\mathbf{M},\mathbf{1})$; each element represents a task, its value is between 1 and N, denoting which node this task is assigned to. In order to apply COA, we use a chaotic vector $\mathbf{Z}$ related to $\mathbf{A}(\mathbf{M},\mathbf{1})$, where $\mathbf{A} = [\mathbf{Z} \times (N-1) + 1]$. A task assignment X of the TAP is generated by A as follows:

$$X(i,k) = \begin{cases} 1, & k = A(i), \quad i = 1, 2, \ldots, M, \\ 0, & k \neq A(i), \quad k = 1, 2, \ldots, N. \end{cases} \tag{12}$$

(2) Energy Function. We integrate the object function $Y(X)$ and all constraints into a cost function to fit the SA algorithm framework. And the cost function is used as the energy function.

All constraints are formulated as penalty functions as follows:

$$\begin{aligned} E_M &= \sum_{k=1}^{N} \max\left(0, \sum_{i=1}^{M} m_i x_{ik} - M_k\right), \\ E_S &= \sum_{k=1}^{N} \max\left(0, \sum_{i=1}^{M} s_i x_{ik} - S_k\right), \\ E_C &= \sum_{k=1}^{N} \sum_{b=k+1}^{N} \max\left(0, \sum_{i=1}^{M} \sum_{i \neq j} c_{ij} x_{ik} x_{jb} - C_{kb}\right), \\ E_D &= \sum_{i=1}^{M} \max\left(0, \sum_{k=1}^{N} x_{ik} \sum_{j=1}^{M} e_{jk} x_{jk} - d_i\right). \end{aligned} \tag{13}$$

As all constraints are of the same importance, we use a common coefficient γ for all penalty function. Hence, the energy function is

$$f(X) = Y(X) + \gamma\left(E_M + E_S + E_C + E_D\right). \tag{14}$$

The criterion of choosing penalty function coefficient γ is that it should scale the values of the penalty functions to the comparable values as that of object function $Y(X)$ such that the procedures of the algorithm will be toward the direction of penalty avoiding; hence, the valid solution can be found with high probability. Besides, the validation of a solution can be represented as $f(X) = Y(X)$.

5. Performance Evaluation

To evaluate the performance of the proposed algorithm, both SA and XASA are coded in Matlab and tested for numerous randomly generated task sets that are allocated onto a RTDS. There are two variations of SA algorithm implemented in this paper, the traditional one (SA1) and the improved one (SA2). SA2 applies the last two adaptive schemes of XASA, that is, adaptive length of Markov chains and cooling factor. All other components of SA2 are as same as SA1, including initial solution, initial temperature, and termination condition. The used computation system is Matlab 7.11.0, with Intel Core i7-2600 @ 3.40 GHz and 16 Gb main memory under a Windows 7 environment.

5.1. Experiment Parameters Settings. All DS parameters are followed by the former researches [16, 17, 40]. The failure rates of processors and communication links are given in the ranges [0.00005–0.00010] and [0.00015–0.00030], respectively. The time of processing a task at different processors is given in the range [15–25]. The memory requirement of each task and node memory capacity is given in the range [1–10] and [100–200], respectively. The task processing load versus node processing capacity is given in the ranges [1–50] and [100–300]. The value of data to be communicated between tasks is given in the range [5–10]. The bandwidth and load capacity of communication links are given in the ranges [1–4] and [100–200]. The range of task deadline value is [10–200].

Table 1: Simulation results for the case of 12 nodes.

Cases		XASA		SA1		SA2		Δt_1	Δt_2	Δt_3	ΔR_1	ΔR_2
N	M	R_{avg}	t_{avg}	R_{avg}	t_{avg}	R_{avg}	t_{avg}					
12	16	0.9349	3.336	0.9361	39.266	0.9353	10.409	91.50%	67.95%	73.49%	0.13%	0.04%
	18	0.9135	3.914	0.9152	50.188	0.9151	13.237	92.20%	70.43%	73.62%	0.18%	0.17%
	20	0.8948	4.014	0.8992	57.456	0.8968	14.454	93.01%	72.23%	74.84%	0.49%	0.22%
	22	0.8733	4.918	0.8781	55.815	0.8777	14.474	91.19%	66.02%	74.07%	0.55%	0.50%
	25	0.8299	5.325	0.8373	84.469	0.8367	22.463	93.70%	76.30%	73.41%	0.89%	0.81%
25	40	0.6000	23.187	0.6048	377.104	0.6028	96.961	93.85%	76.09%	74.29%	0.79%	0.47%
	45	0.5320	36.523	0.5334	484.005	0.5329	123.667	92.45%	70.47%	74.45%	0.25%	0.16%
	50	0.4537	36.300	0.4553	606.927	0.4565	161.550	94.02%	77.53%	73.38%	0.35%	0.61%
	55	0.3732	57.156	0.3753	720.068	0.3726	195.631	92.06%	70.78%	72.83%	0.58%	−0.14%
	60	0.2866	157.777	0.2886	795.438	0.2860	217.155	80.16%	27.34%	72.70%	0.71%	−0.19%
Average								91.42%	67.51%	73.71%	0.49%	0.27%

Table 2: Simulation results for the case of 12 nodes.

Cases		XASA				SA1			SA2		
N	M	R_{std}	t_{std}	V_1	V_2	R_{std}	t_{std}	V	R_{std}	t_{std}	V
12	16	0.0016	0.3035	91.10%	100.00%	0.0019	0.8582	99.99%	0.0013	0.6208	100.00%
	18	0.0008	0.6041	85.57%	100.00%	0.0020	0.9552	98.76%	0.0018	0.8322	99.21%
	20	0.0062	0.4114	72.39%	100.00%	0.0029	1.3870	98.12%	0.0036	0.3914	97.63%
	22	0.0016	0.6186	54.30%	100.00%	0.0015	1.3489	93.23%	0.0017	0.5886	94.07%
	25	0.0049	0.7613	29.27%	99.82%	0.0038	2.9824	85.99%	0.0031	1.2888	86.88%
25	40	0.0042	1.5797	21.67%	99.87%	0.0026	8.7429	89.62%	0.0027	4.6770	91.39%
	45	0.0014	5.4829	9.85%	99.67%	0.0024	12.3919	76.51%	0.0027	3.4313	81.95%
	50	0.0039	5.6073	2.06%	99.30%	0.0030	11.4135	71.62%	0.0025	11.0605	79.76%
	55	0.0028	8.1346	0.16%	96.58%	0.0019	8.9093	58.05%	0.0023	7.7128	67.33%
	60	0.0022	9.3194	0.03%	89.76%	0.0016	15.5811	55.63%	0.0022	8.4473	67.03%
Average		0.0029	3.2823	36.64%	98.50%	0.0024	6.4570	82.75%	0.0024	3.9051	86.53%

The network topology is star, N is set to be 12 and 25, and M is set to be $[16, 18, 20, 22, 25]$ and $[40, 45, 50, 55, 60]$ for two cases, respectively. The coefficient of penalty function γ is set to be 1. The number of randomly chosen solutions at the beginning of the algorithm is set to be $K = 10$. And the initial solution of the two SA algorithms is one of these K solutions which is chosen at random. Because SA is a stochastic algorithm, each independent run of the algorithm on a same application may yield different result, we, thus, run all of the three algorithms on an application 10 times and obtain the average values.

5.2. Experiment Results. Table 1 summarizes the reliability and calculation time of all TAPs by deploying the XASA and other two SA algorithms. The title R with suffix *avg* represents *Reliability* with the average value of 10 independent runs, and T represents *Time* where the unit is second. Δt_1 is the acceleration ratio of XASA versus SA1, where $\Delta t_1 = (t_{SA1} - t_{XASA})/t_{SA1} \times 100\%$. Δt_2 and Δt_3 is the acceleration ratio of XASA versus SA2 and SA2 versus SA1, respectively. ΔR represents the average deviation in percentage between XASA and SA algorithms in terms of reliability, where $\Delta R_i = (R_{SAi} - R_{XASA})/R_{SAi} \times 100\%, i = 1, 2$.

The comparative results from Table 1 show that XASA can sharply reduce the convergence time against the other two SA algorithms, while the solution quality (i.e., reliability) is slightly worse (less than 0.01 in terms of value and 1% in percentage). Note that the value of Δt_3 is steady in the range of 72% ~ 75% with small variation, which means the third and fourth adaptive schemes take a fixed effect on the SA algorithm. Furthermore, Δt_1 is steady in all cases except the last one ($N = 25, M = 60$) as well, which has an average value of 92.66% with standard deviation 0.1036. In our preliminary experiments, the value of function $Y(X)$ is always below 2 during the whole algorithm, while the value of the penalty function can be hundreds. Besides, according to the experiment parameters set in this paper, there is no significant difference between nodes, nor do the tasks. Hence, there are lots of local minima in the solution space without considering the validation of solutions. Constraints are easy to be satisfied when the problem scale is small, so COA can obtain several valid solutions, and the initial temperature for the SA algorithm in the next step of XASA can be relatively small; therefore, it is fast to get convergence. On the other hand, COA can hardly obtain valid solutions in the case of large scale (e.g., $N = 25, M = 60$ in this paper), if there are insufficient valid solutions (less than K), a large initial

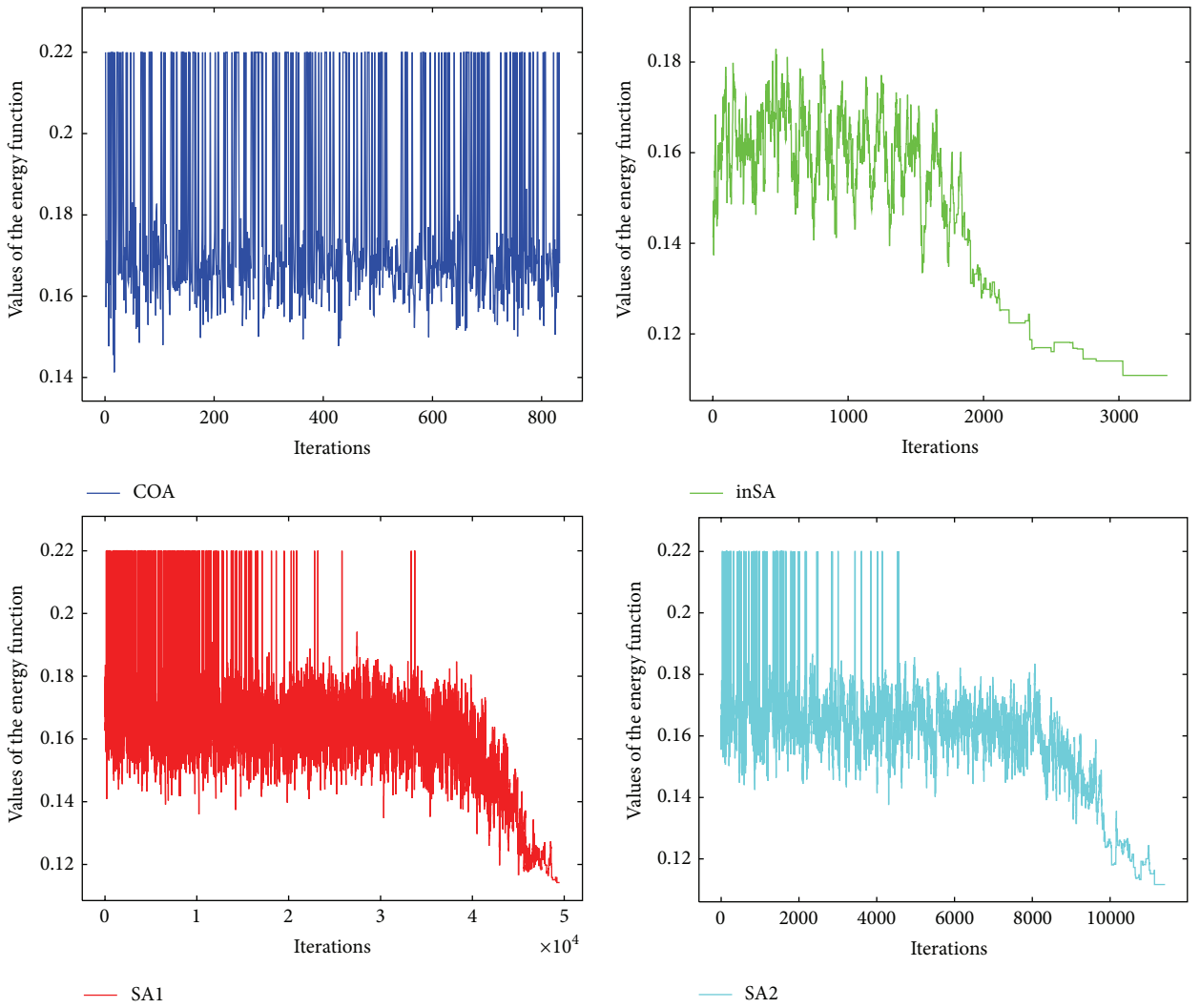


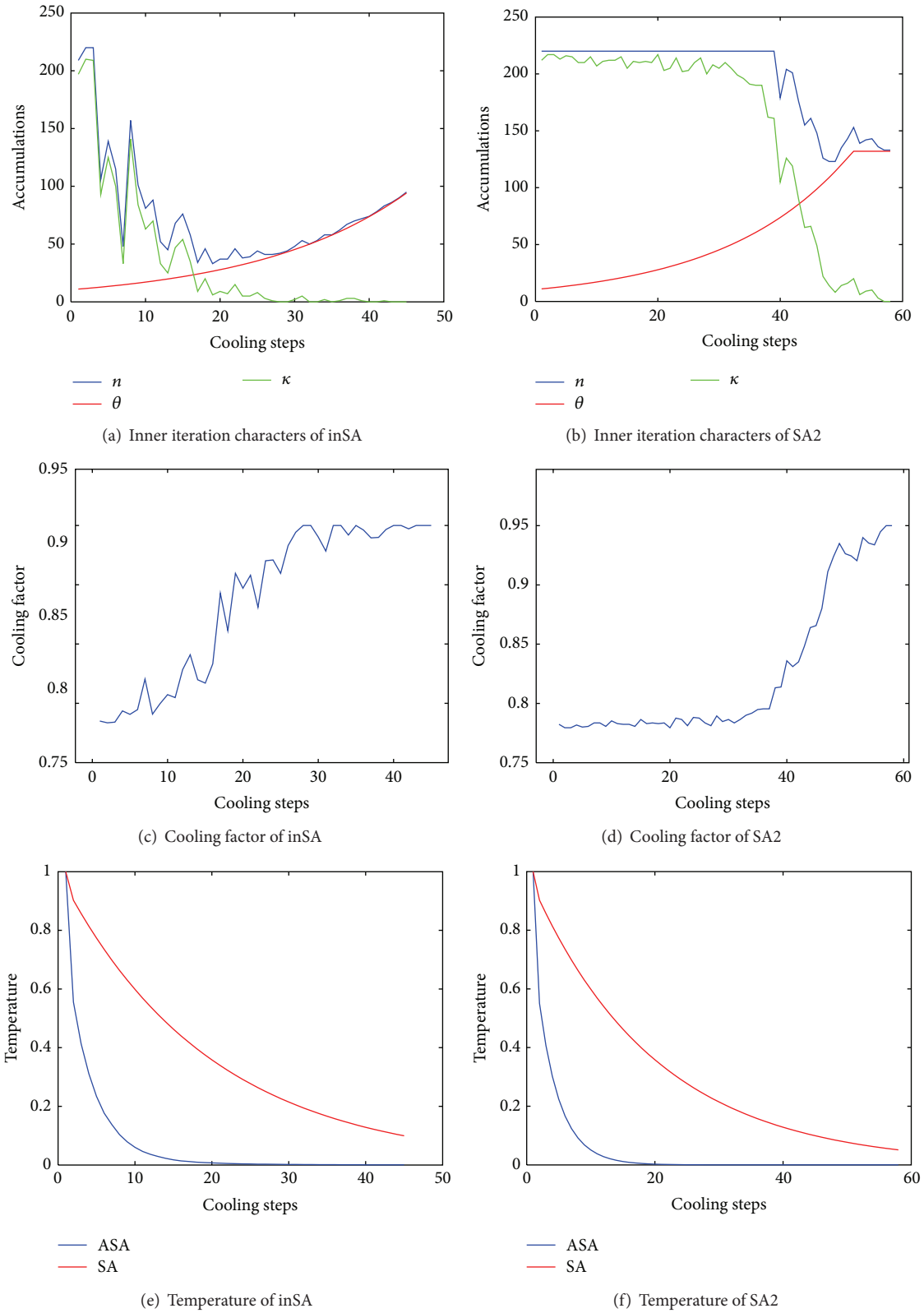
Figure 3: Energy function at each iteration for $N = 12$, $M = 20$.

temperature will be set because of the giant value of penalty function, this will affect the convergence speed significantly. Additionally, constraints are not so easy to be satisfied when problem scale is large, so there are much less local minima in the solution space; this will slow down the convergence speed as well. All these factors weaken the speedup effect based on COA, so the performance in the last case is not so good.

Table 2 shows some overall statics characters of all three algorithms. Where the R_{std} and t_{std} represent the standard deviation of reliability and calculation time, V_1 represents the valid solutions in percentage of COA in XASA and V_2 represents that of inSA (inSA represents the SA algorithm in XASA), the other two V columns have the same meaning. As we can see, XASA is the best in terms of mean value of time standard deviation, and there is no case that SA1 excels XASA in this criterion. V_2 gets the best result compared to other two algorithms as well. Note that V_1 shows poor results in the large cases, this is an evidence for our analysis of the large scale issue presented before.

5.3. Time Series Analysis. Figure 3 shows the values of the energy function, which are calculated at each iteration of the algorithms for the case $N = 12$, $M = 20$. Note that the values of the invalid solutions are set to be 0.22, because the real values of the invalid solutions are too large, and they may conceal the details of the valid ones.

As we can see, COA cannot guarantee the validation of the solutions, it is completely stochastic without heuristic. However, it can generate a good start for the next step of XASA, which is shown in the results of inSA, where all solutions are valid and the convergence speed is quite fast. The inSA begins to reach to the good enough solutions before 2000 iterations. The other two SA algorithms start with the worse condition, and spend much more iterations to reach to the good enough solutions. They both need thousands of iterations to explore the solution space which generates lots of invalid solutions as COA but the cost is much more expensive. Because of the adaptive schemes, SA2 can quickly pass through invalid solutions as can be seen in Figure 3. Hence, these schemes are effective.



(a) Inner iteration characters of inSA

(b) Inner iteration characters of SA2

(c) Cooling factor of inSA

(d) Cooling factor of SA2

(e) Temperature of inSA

(f) Temperature of SA2

Figure 4: Cooling schedule characters for $N = 12$, $M = 20$.

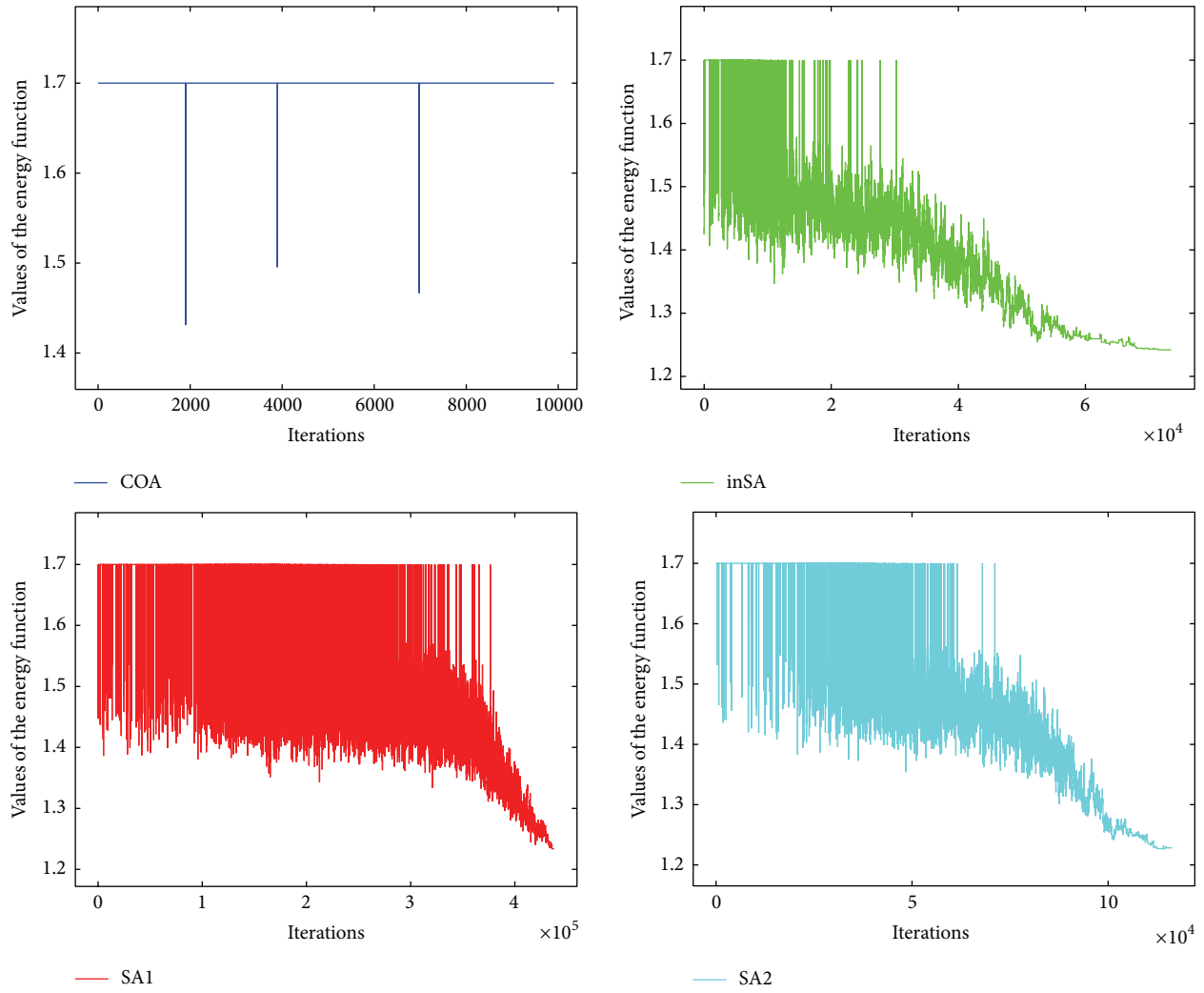


Figure 5: Energy function at each iteration for $N = 25$, $M = 60$.

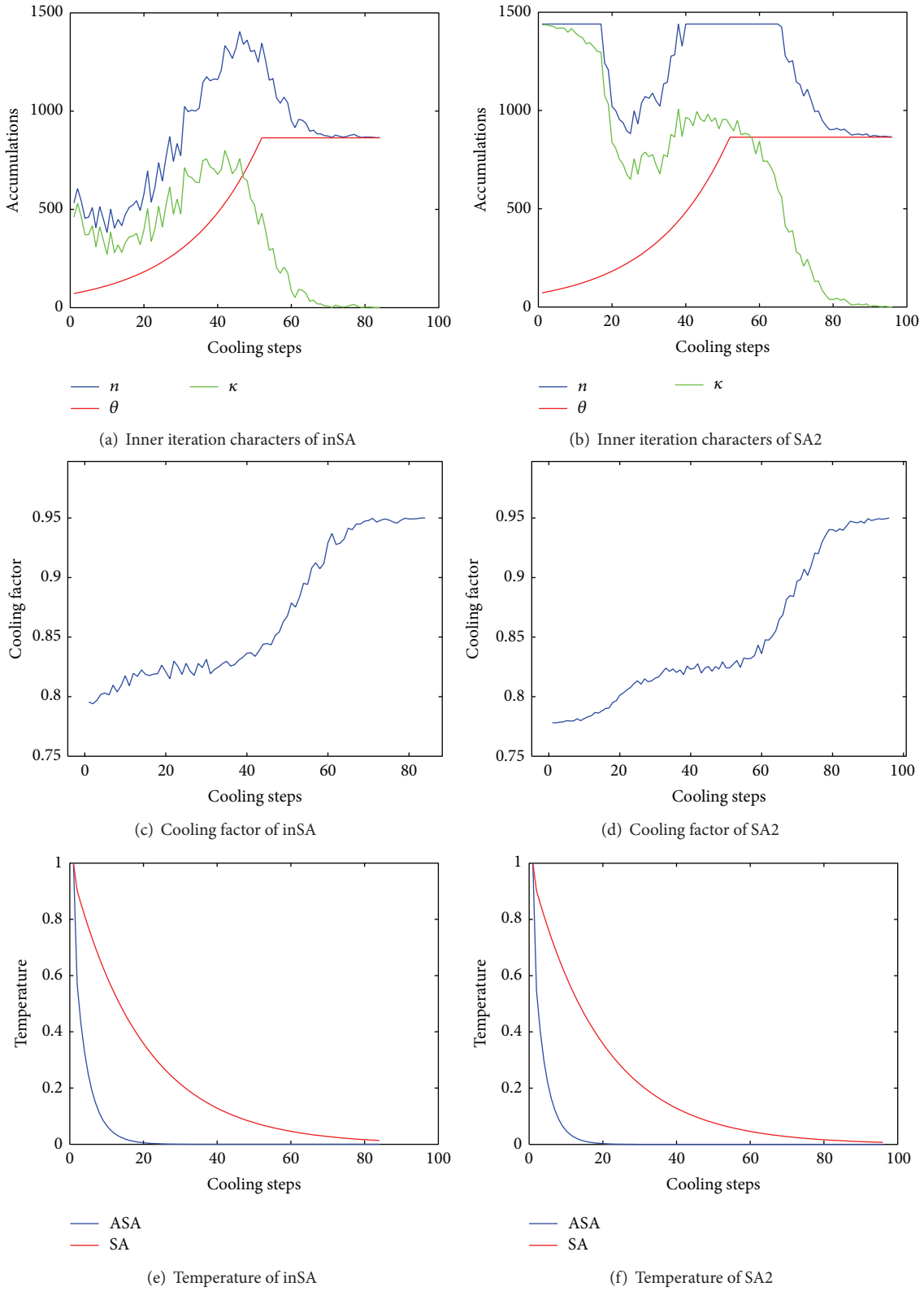
Figure 4 shows the details of the adaptive cooling schedule schemes, which are calculated at each cooling step of the algorithms for the case $N = 12$, $M = 20$. The left three figures are the results of inSA, and right ones are the results of SA2. Note that the temperature in Figures 4(c) and 4(f) is set to be unitary. At the beginning of the cooling steps, the acceptance rates of the new solutions are both high in two algorithms; hence, the cooling factor is small, and the temperature reduces rapidly, while the actual length of Markov chains n is much smaller in inSA than SA2, and the cooling factor increases faster as well. It is caused by the differences of the initial temperature and initial solution, since inSA and SA2 are actually the same. Hence, COA can truly improve the convergence speed.

Figure 5 shows the details of the case $N = 25$, $M = 60$ where the invalid solution values are set to be 1.7. The result of COA is quite bad, only three valid solutions are found; therefore, inSA cannot get a good start and it has to explore wide solution space at the beginning which generates lots of invalid solutions.

Figure 6 shows the results of the case $N = 25$, $M = 60$ as Figure 4. The cooling factor and temperature curve are not so different between inSA and SA2, and the character n cannot bring as much benefit as before. These cause the inefficient situation of the largest case.

6. Conclusions

In this paper, we consider a heterogeneous DS that runs a real-time application, to achieve maximization of system reliability with task allocation technique. By formulating the reliability and constraints, we model this problem as a combinatorial problem. To solve this problem with fast convergence speed, we improve the well-known simulated annealing algorithm based on the analysis of the cooling schedule of SA algorithm which has a significant effect on its convergence speed. Then, we propose the algorithm XASA, which combines SA algorithm with chaotic optimization algorithm with several adaptive schemes. The experimental results show that the



(a) Inner iteration characters of inSA

(b) Inner iteration characters of SA2

(c) Cooling factor of inSA

(d) Cooling factor of SA2

(e) Temperature of inSA

(f) Temperature of SA2

Figure 6: Cooling schedule characters for $N = 25$, $M = 60$.

proposed algorithm can achieve a satisfactory performance of speedup, while the solution quality is just slightly worse.

Notations

M: Number of tasks
N: Number of nodes
p_k: Node k
t_i: Task i
l_{kb}: Communication link between p_k and p_b
x_{ik}: Whether or not t_i is on p_k
e_{ik}: Execution time of t_i on p_k
c_{ij}: Communication cost between t_i and t_j
C_{kb}: Communication capacity of l_{kb}
ω_{kb}: Bandwidth of l_{kb}
λ_k: Failure rate of p_k
φ_{kb}: Failure rate of l_{kb}
m_i: Memory required by t_i
M_k: Memory capacity of p_k
s_i: Processing load of t_i
S_k: Processing capacity of p_k
d_i: Deadline of t_i.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

The authors thank the anonymous referees and the editor for their valuable comments and suggestions. This work is supported by the National Natural Science Foundation of China (Grant no. 61374185).

References

[1] C. H. Lee and K. G. Shin, "Optimal task assignment in homogeneous networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, no. 2, pp. 119–129, 1997.

[2] T. P. Ajith and C. S. R. Murthy, "Optimal task allocation in distributed systems by graph matching and state space search," *Journal of Systems and Software*, vol. 46, no. 1, pp. 59–75, 1999.

[3] G. Attiya and Y. Hamam, "Static task assignment in distributed computing systems," in *Proceedings of the 21st IFIP TC7 Conference on System Modeling and Optimization*, 2003.

[4] M. Kafil and I. Ahmad, "Optimal task assignment in heterogeneous distributed computing systems," *IEEE Concurrency*, vol. 6, no. 3, pp. 42–51, 1998.

[5] G. Attiya and Y. Hamam, "Optimal allocation of tasks onto networked heterogeneous computers using minimax criterion," in *Proceedings of the International Network Optimization Conference (INOC '03)*, Paris, France, 2003.

[6] S. Kartik and C. S. R. Murthy, "Improved task-allocation algorithms to maximize reliability of redundant distributed computing systems," *IEEE Transactions on Reliability*, vol. 44, no. 4, pp. 575–586, 1995.

[7] C. Hsieh, "Optimal task allocation and hardware redundancy policies in distributed computing systems," *European Journal of Operational Research*, vol. 147, no. 2, pp. 430–447, 2003.

[8] V. K. P. Kumar, S. Hariri, and C. S. Raghavendra, "Distributed program reliability analysis," *IEEE Transactions on Software Engineering*, vol. 12, no. 1, pp. 42–50, 1986.

[9] S. M. Shatz and J. Wang, "Models and algorithms for reliability-oriented task-allocation in redundant distributed-computer systems," *IEEE Transactions on Reliability*, vol. 38, no. 1, pp. 16–27, 1989.

[10] A. Kumar and D. P. Agrawal, "Generalized algorithm for evaluating distributed-program reliability," *IEEE Transactions on Reliability*, vol. 42, no. 4, pp. 416–426, 1993.

[11] P. A. Tom and C. S. R. Murthy, "Algorithms for reliability-oriented module allocation in distributed computing systems," *Journal of Systems and Software*, vol. 40, no. 2, pp. 125–138, 1998.

[12] C. C. Chiu, Y. S. Yeh, and J. S. Chou, "A fast algorithm for reliability-oriented task assignment in a distributed system," *Computer Communications*, vol. 25, no. 17, pp. 1622–1630, 2002.

[13] A. O. Charles Elegbede, C. Chu, K. H. Adjallah, and F. Yalaoui, "Reliability allocation through cost minimization," *IEEE Transactions on Reliability*, vol. 52, no. 1, pp. 106–111, 2003.

[14] C. Hsieh and Y. Hsieh, "Reliability and cost optimization in distributed computing systems," *Computers & Operations Research*, vol. 30, no. 8, pp. 1103–1119, 2003.

[15] D. P. Vidyarthi and A. K. Tripathi, "Maximizing reliability of distributed computing system with task allocation using simple genetic algorithm," *Journal of Systems Architecture*, vol. 47, no. 7, pp. 549–554, 2001.

[16] G. Attiya and Y. Hamam, "Task allocation for maximizing reliability of distributed systems: a simulated annealing approach," *Journal of Parallel and Distributed Computing*, vol. 66, no. 10, pp. 1259–1266, 2006.

[17] P. Yin, S. S. Yu, P. P. Wang, and Y. T. Wang, "Task allocation for maximizing reliability of a distributed system using hybrid particle swarm optimization," *Journal of Systems and Software*, vol. 80, no. 5, pp. 724–735, 2007.

[18] Q. M. Kang, H. He, H. M. Song, and R. Deng, "Task allocation for maximizing reliability of distributed computing systems using honeybee mating optimization," *Journal of Systems and Software*, vol. 83, no. 11, pp. 2165–2174, 2010.

[19] R. Shojaee, H. R. Faragardi, S. Alaee, and N. Yazdani, "A new Cat Swarm Optimization based algorithm for reliability-oriented task allocation in distributed systems," in *Proceedings of the 6th International Symposium on Telecommunications (IST '12)*, pp. 861–866, IEEE, Tehran, Iran, November 2012.

[20] Q. M. Kang, H. He, and J. Wei, "An effective iterated greedy algorithm for reliability-oriented task allocation in distributed computing systems," *Journal of Parallel and Distributed Computing*, vol. 73, no. 8, pp. 1106–1115, 2013.

[21] C. Blum and A. Roli, "Metaheuristics incombinatorial optimization: overview and conceptual comparision," *IEEE Transactions on Evolutionary Computation*, vol. 32, no. 1, pp. 48–77, 2003.

[22] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *The Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087–1092, 1953.

[23] B. Liu, L. Wang, Y. Jin, F. Tang, and D. Huang, "Improved particle swarm optimization combined with chaos," *Chaos, Solitons and Fractals*, vol. 25, no. 5, pp. 1261–1271, 2005.

[24] L. Wnag, D. Z. Zheng, and Q. S. Lin, "Survey on chaotic optimization methods," *Computing Technology and Automation*, vol. 20, no. 1, pp. 1–5, 2001 (Chinese).

[25] S. Kirkpatrick, J. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[26] P. J. M. van Laarhoven, E. H. L. Aarts, and J. K. Lenstra, "Job shop scheduling by simulated annealing," *Operations Research*, vol. 40, no. 1, pp. 113–125, 1992.

[27] P. Eles, Z. Peng, K. Kuchcinski, and A. Doboli, "System level hardware/software partitioning based on simulated annealing and tabu search," *Design Automation for Embedded Systems*, vol. 2, no. 1, pp. 5–32, 1997.

[28] T. Wiangtong, P. K. Cheung, and W. Luk, "Comparing three heuristic search methods for functional partitioning in hardware-software codesign," *Design Automation for Embedded Systems*, vol. 6, no. 4, pp. 425–449, 2002.

[29] F. Ferrandi, P. L. Lanzi, C. Pilato, D. Sciuto, and A. Tumeo, "Ant colony heuristic for mapping and scheduling tasks and communications on heterogeneous embedded systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 6, pp. 911–924, 2010.

[30] H. R. Faragardi, R. Shojaee, and N. Yazdani, "Reliability-aware task allocation in distributed computing systems using hybrid simulated annealing and tabu search," in *Proceedings of the IEEE 9th International Conference on High Performance Computing and Communication*, pp. 1088–1095, IEEE, Liverpool, UK, 2012.

[31] H. R. Faragardi, R. Shojaee, M. A. Keshtkar, and H. Tabani, "Optimal task allocation for maximizing reliability in distributed real-time systems," in *Proceedings of the IEEE/ACIS 12th International Conference on Computer and Information Science (ICIS '13)*, pp. 513–519, IEEE, Niigata, Japan, 2013.

[32] L. Chen and K. Aihara, "Chaotic simulated annealing by a neural network model with transient chaos," *Neural Networks*, vol. 8, no. 6, pp. 915–930, 1995.

[33] S. Talatahari, B. Farahmand Azar, R. Sheikholeslami, and A. H. Gandomi, "Imperialist competitive algorithm combined with chaos for global optimization," *Communications in Nonlinear Science and Numerical Simulation*, vol. 17, no. 3, pp. 1312–1319, 2012.

[34] L. Chen and K. Aihara, "Combinatorial optimization by chaotic dynamics," in *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, pp. 2921–2926, IEEE, Orlando, Fla, USA, October 1997.

[35] L. Wang and K. Smith, "On chaotic simulated annealing," *IEEE Transactions on Neural Networks*, vol. 9, no. 4, pp. 716–718, 1998.

[36] L. Wang and F. Tian, "Noisy chaotic neural networks for solving combinatorial optimization problems," in *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN '00)*, vol. 4, pp. 37–40, IEEE, Como, Italy, 2000.

[37] K. Masuda and E. Aiyoshi, "Solution to combinatorial problems by using chaotic global optimization method on a simplex," in *Proceedings of the 41st SICE Annual Conference*, pp. 1313–1318, IEEE, 2002.

[38] J. Mingjun and T. Huanwen, "Application of chaos in simulated annealing," *Chaos, Solitons and Fractals*, vol. 21, no. 4, pp. 933–941, 2004.

[39] K. Ferens and D. Cook, "Chaotic walk in simulated annealing search space for task allocation in a multiprocessing system," *International Journal of Cognitive Informatics and Natural Intelligence*, vol. 7, no. 3, pp. 58–79, 2013.

[40] S. Kartik and C. S. R. Murthy, "Task allocation algorithms for maximizing reliability of distributed computing systems," *IEEE Transactions on Computers*, vol. 46, no. 6, pp. 719–724, 1997.

[41] B. Hajek, "Cooling schedules for optimal annealing," *Mathematics of Operations Research*, vol. 13, no. 2, pp. 311–329, 1988.