Task allocation for maximizing reliability of distributed systems: A simulated annealing approach

ARTICLE *in* JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING · OCTOBER 2006

Impact Factor: 1.18 · DOI: 10.1016/j.jpdc.2006.06.006

2 AUTHORS:

Gamal Attiya
Faculty of Electronic Engineering
45 PUBLICATIONS **133** CITATIONS
SEE PROFILE

Yskandar Hamam
Tshwane University of Technology
234 PUBLICATIONS **853** CITATIONS
SEE PROFILE

Available online at www.sciencedirect.com

ScienceDirect

J. Parallel Distrib. Comput. 66 (2006) 1259–1266

Journal of Parallel and Distributed Computing

www.elsevier.com/locate/jpdc

Task allocation for maximizing reliability of distributed systems: A simulated annealing approach

Gamal Attiya*, Yskandar Hamam

ESIEE Paris, Laboratory A^2SI, Cité Descartes-BP 99, 93162 Noisy-Le-Grand, France

Received 29 June 2004; received in revised form 19 June 2006; accepted 20 June 2006

Abstract

This paper addresses the problem of task allocation in heterogeneous distributed systems with the goal of maximizing the system reliability. It first develops an allocation model for reliability based on a cost function representing the unreliability caused by the execution of tasks on the system processors and the unreliability caused by the interprocessor communication time subject to constraints imposed by both the application and the system resources. It then presents a heuristic algorithm derived from the well-known simulated annealing (SA) technique to quickly solve the mentioned problem. The performance of the proposed algorithm is evaluated through experimental studies on a large number of randomly generated instances. Indeed, the quality of solutions are compared with those derived by using the branch-and-bound (BB) technique.
© 2006 Elsevier Inc. All rights reserved.

Keywords: Task allocation; Distributed systems; Reliability; Optimization; Simulated annealing

1. Introduction

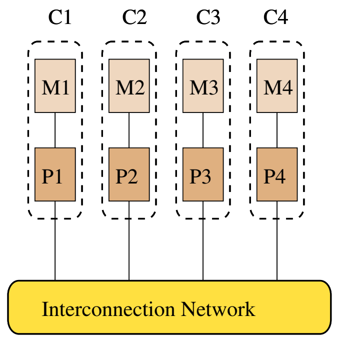
Distributed systems have emerged as a powerful platform for executing high performance parallel applications, alternative to the very expensive massively parallel machines. A parallel application could be divided into a number of tasks and executed concurrently on different processors in the system. In reality, however, the performance of a parallel application on a distributed system is mainly dependent on the allocation of the tasks comprising the application onto the available processors in the system, referred to as the *task allocation* problem. If the allocation is not carefully implemented, processors in the system may spend most of their time waiting for each other instead of performing useful computations.

Several studies have been devoted to this problem with the main concern on the performance measures such as minimizing the total sum of execution and communication cost(time) [8,2] or minimizing the application turnaround time [9,3]. Inherently, distributed systems are more complex than centralized systems. The added complexity could increase the potential for system failures. Hence, ensuring reliability of distributed systems is of critical importance along with the task allocation.

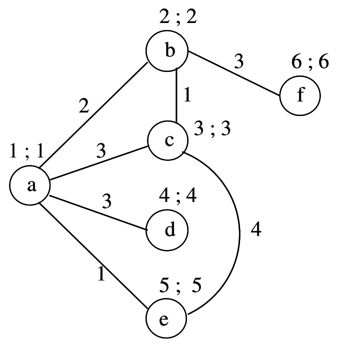
Redundancy is the traditional technique to improve reliability of distributed systems [15,13,14,18,5,21,22,16,10,19,4,6,7]. A distributed system is redundant if it possesses software and/or hardware redundancy. In [15,13,14,18,5], software redundancy (e.g., data/file replication among processors) is considered and algorithms are proposed to seek the minimal data/file replication while retaining the system reliability. In [21,22,16,10,19,4,6,7], hardware redundancy (e.g., multiple processors and communication links) is considered and some models are developed under different levels of redundancy. Inherently, redundancy is an expensive approach. Moreover, many times, the system redundancy is not available or infeasible. In this situation, a distributed system may execute a parallel application with high reliability if the tasks of the application are assigned carefully onto appropriate processors in the system taking into account the failure rates of both the processors and the communication links. The obvious idea is that, tasks with longer execution time should be allocated to more reliable processors and edges of higher communication times should be allocated onto the most reliable links (paths). Hence, the main motivation of this paper is to introduce

* Corresponding author. Fax: +33 1 45 92 66 99.
E-mail address: attiyag@esiee.fr (G. Attiya).

0743-7315/$ - see front matter © 2006 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2006.06.006



(a)



(b)

	P1	P2	P3	P4
a	10	16	11	90
b	12	60	30	8
c	15	40	8	50
d	4	15	25	10
e	14	10	99	12
f	12	70	6	45

(c)

Fig. 1. An example of a distributed system and a task interaction graph. (a) A distributed system, (b) a task graph, and (c) execution costs.

reliability to distributed systems with no extra hardware and/or software costs.

Some work has been done in the past for maximizing reliability of distributed systems without redundancy [17,11,20]. In [17,11], optimal approaches were developed based on the well known A^* algorithm. But, these approaches are exponential in nature and may demand heavy computation time. In [20], a heuristic approach, based on the well-known genetic algorithm (GA), was developed to quickly find a near optimal allocation. In this paper, an allocation algorithm based on the simulated annealing (SA) technique is developed to allocate tasks onto processors of a distributed system with the goal of maximizing the system reliability. Unlike the other approaches, this paper takes into account several kinds of constraints and characteristics that are essential to both the application and the distributed system such as different requirements of the application tasks and heterogeneity of the system resources. The performance of the proposed algorithm is evaluated through experimental studies on a large number of randomly generated instances. Furthermore, the quality of solutions are compared with those derived by using the branch-and-bound (BB) technique.

The rest of this paper is organized as follows. Section 2 describes the task allocation problem. Section 3 provides an explicit cost function to the distributed system reliability while Section 4 presents an allocation model for reliability. Section 5 presents an allocation algorithm derived from the well known simulated annealing technique. Section 6 discusses the simulation results while Section 7 summarizes the paper conclusions.

2. Problem statement

The problem being addressed in this paper is concerned with allocating tasks of a parallel application onto processors of a distributed system with the goal of maximizing the system reliability. The distributed system consists of a set of heterogeneous computers interconnected via a communication network as shown in Fig. 1(a). Each computer has computation facility and its own memory while the communication network has a limited communication capacity. Indeed, a failure rate is associated with each component (processor and link) in the system. On the other hand, a parallel application is represented by a task interaction graph $G(V, E)$, as shown in Fig. 1(b). Where, V represents a set of tasks and E represents a set of edges. Each task $i \in V$ is labeled by memory and processing load requirements while each arc $(i, j) \in E$ is labeled by communication requirements among tasks. Indeed, a vector is associated with each task representing the execution time of the task at different processors in the system, as shown in Fig. 1(c). Note that, if a task cannot be executed at a particular processor, the corresponding element in thevectoris set to ∞,i.e.,very large value.

Briefly, we are given a set of M tasks representing a parallel application to be executed on a distributed system of N processors. Tasks of the given application require certain computer resources such as computational load and memory capacity. They also communicate at a given rate. On the other hand, the system resources are capacitated and a failure rate is associated with each component. The purpose is to allocate each of the M tasks to one of the N processors such that the overall system reliability is maximized, the requirements of tasks and edges are met, and the capacities of the system resources are not violated.

3. Preliminaries

In a distributed system, each component exists in one of two states: operational or faulty. In order for a parallel application to run successfully on a distributed system, each processor must be operational during the time of processing its tasks and each path must be operational during the active period of data communication between the terminal processors of the path. Hence, successful execution of the application is mainly dependent on reliability of the system components (processors and links) and on the distribution of the application tasks to the available processors in the system. The following subsections first present reliability expressions to the system components and then provide an explicit cost function to the system reliability. To do so, let X be an $M \times N$ binary matrix corresponding to an assignment of M tasks onto N processors such that an element $X_{ip} = 1$ if a task i is assigned to a processor p and $X_{ip} = 0$ otherwise.

3.1. Processor reliability

A processor reliability R_p is the probability that the processor p is operational for execution of tasks that are assigned to it.

Define λ_p as the failure rate of a processor p during a time interval t, then the reliability of the processor p in the time t is $e^{-\lambda_p t}$ [6,7,17,11,20]. Under a task assignment X, the time t represents the time required to execute all the tasks assigned to p. Let C_{ip} be the time of processing a task i on a processor p, then the processor reliability may be formulated as

$$R_p = e^{-\lambda_p \sum_i C_{ip} X_{ip}}.$$

The summation gives the total time elapsed in executing all the tasks that are assigned to the processor p.

3.2. Path/link reliability

A path reliability R_{pq} is the probability that the path pq is operational for communicating data between the terminal processors of the path. A path is sequence of communication links from a sender to a receiver. Define μ_{pq} as the failure rate of a path pq during a time interval t, then the reliability of the path pq in the time t is $e^{-\mu_{pq} t}$ [6,7,17,11,20]. Under a task assignment X, the time t represents the interval required for data communication between the terminal processors of the path pq. Let C_{ijpq} be the time of transferring data between tasks i and j if they are assigned to different processors p and q, then the path reliability may be formulated as

$$R_{pq} = e^{-\mu_{pq} \sum_i \sum_{j \neq i} C_{ijpq} X_{ip} X_{jq}}.$$

The summation gives the total time required for communicating data between the terminal processors (p and q) of the path pq.

From the components reliabilities, it is clear that, allocating tasks of large execution times to more reliable processors is a good approach to increase the execution reliability in the system. Also, allocating edges of higher communication times to more reliable links is a good approach to increase the communication reliability in the system.

3.3. System reliability

Reliability of a distributed system R_{sys} may be defined as the probability that the system can run the entire application successfully [6,17,11]. In other words, the system reliability is the product of the components reliabilities. That is, the product of the probability that each processor be operational during the period of tasks execution and the probability that each path/link be operational during the period of interprocessor communication. Hence, the system reliability may be formulated as

$$R_{\text{sys}} = \left[\prod_p R_p\right] \left[\prod_p \prod_{q \neq p} R_{pq}\right] = e^{-Z},$$

where

$$Z = \sum_p \sum_i \lambda_p C_{ip} X_{ip} + \sum_p \sum_{q \neq p} \sum_i \sum_{j \neq i} \mu_{pq} C_{ijpq} X_{ip} X_{jq}.$$

The first term of the function Z represents the unreliability caused by the execution of tasks on the system processors of various reliabilities and the second term reflects the unreliability caused by the interprocessor communication through different links/paths of various reliabilities.

4. Allocation model for reliability

It is clear from the above discussion that, maximizing reliability of a distributed system is equivalent to minimizing the unreliability cost function Z. However, in order to achieve a satisfactory allocation, additional constraints should be considered with the cost function to meet the application requirements and not violate the availability of the system resources. In the following, the allocation constraints are first described and then an allocation model for reliability is presented.

4.1. Allocation constraints

The allocation constraints depend on the characteristics of both the application involved (such as tasks requirements and inter-task communication requirements) and on the system resources including the computation speed of processors, the availability of memory and communication network capacity. To describe the allocation constraints, let TC_p denote the set of tasks that are mapped to a processor p under a task assignment X.

- *No redundancy*: This paper considers a model where a task must be allocated to exactly one processor, i.e., no software redundancy. That is, the following equality must hold with each task i:

$$\sum_p X_{ip} = 1.$$

- *Processing load constraints*: For a task assignment X, the total processing load required by all tasks assigned to a processor p must be less than or equal to the available computational load of p. Let p_i denote the processing load requirements of a task i and let P_p denote the available processing load of a processor p, then the following inequality must hold at each processor p.

$$\sum_{i \in TC_p} p_i \leqslant P_p.$$

- *Memory constraints*: For a task assignment X, the total memory required by all tasks assigned to a processor p must be less than or equal to the available memory capacity of p. Define m_i as the amount of memory required by a task i and M_p as the available memory at a processor p, then the following inequality must hold at each processor p.

$$\sum_{i \in TC_p} m_i \leqslant M_p.$$

- *Communication capacity constraints*: For a task assignment X, the total communication capacity required by all edges mapped to a communication link/path pq must be less than or equal to the available communication capacity of the path. Let b_{ij} denote the amount of communication capacity required to communicate data between tasks i and j if they are assigned to different processors p and q, and let R_{pq} denote the available communication capacity of the link/path pq. Then the following inequality must hold at each communication link/path pq.

$$\sum_{i \in TC_p} \sum_{j \neq i, j \in TC_q} b_{ij} \leqslant R_{pq}.$$

4.2. Reliability model

From the unreliability cost function Z and the above constraints, the allocation model for reliability may be formulated as follows:

$$\begin{aligned} \min \quad Z = & \sum_p \sum_i \lambda_p C_{ip} X_{ip} \\ & + \sum_p \sum_{q \neq p} \sum_i \sum_{j \neq i} \mu_{pq} C_{ijpq} X_{ip} X_{jq} \end{aligned}$$

s.t.

$$\begin{aligned} & \sum_p X_{ip} = 1 && \forall\, tasks\ i, \\ & \sum_{i \in TC_p} p_i \leqslant P_p && \forall\, processors\ p, \\ & \sum_{i \in TC_p} m_i \leqslant M_p && \forall\, processors\ p, \\ & \sum_{i \in TC_p} \sum_{j \neq i, j \in TC_q} b_{ij} \leqslant R_{pq} && \forall\, paths\ pq. \end{aligned}$$

The above model defines an integer programming problem. An optimal solution to this problem may be found by enumerating all possible allocations. But, for N processors and M tasks, this requires $O(N^M)$ computation time. Hence, this paper presents a heuristic algorithm to quickly solve the mentioned problem.

5. Allocation algorithm

This section presents a heuristic algorithm derived from the well known SA technique. It first defines the basic concepts of the SA and then explains how it may be employed for solving the allocation problem in terms of reliability.

5.1. Basic concepts

SA is a global optimization technique which attempts to find the lowest point in an energy landscape [12,1]. The technique was derived from the observations of how slowly cooled molten metal can result in a regular crystalline structure. The distinctive feature of the algorithm is that it incorporates random jumps to potential new solutions. This ability is controlled and reduced as the algorithm progresses.

Clearly, the SA emulates the physical concepts of *temperature* and *energy* to represent and solve the optimization problems. The objective function of the optimization problem is treated as the energy of a dynamic system while the temperature is introduced to randomize the search for a solution. The state of the dynamic system being simulated is related to the state of the system being optimized. The procedure is the following: the system is submitted to a high temperature and is then slowly cooled through a series of temperature levels. At each level, the algorithm searches for the system equilibrium state through elementary transformations which will be accepted if they reduce the system energy. However, as the temperature decreases, smaller energy increments may be accepted and the system eventually settles into a low energy state. This property makes the algorithm to escape from a local optimal configuration and close, if not identical, to the global minimum. The probability of acceptance an uphill move is a function of the temperature and the magnitude of the increase Δ. The algorithm presented in this paper uses $\exp(-\Delta/T)$, where T is the temperature.

5.2. Simulated annealing algorithm

The algorithm starts by randomly selecting an initial solution s and computes the energy/cost E_s at the current solution s. After setting an initial temperature T, a neighbor finding strategy is invoked to generate a neighbor solution n to the current solution s and compute the corresponding energy/cost E_n. If the energy E_n at the neighbor solution n is lower than the current energy E_s, then the neighbor solution is accepted as a current solution. Otherwise, a probability function $\exp(-\Delta/T)$ is evaluated to determine whether the neighbor solution may be accepted as a current solution, where $\Delta = E_n - E_s$. After a thermal equilibrium is reached at the current temperature T, the value of T is decreased by a cooling factor α and the number of inner repetitions is increased by an increasing factor β. The algorithm continues from the current solution point searching for a thermal equilibrium at the new temperature level. The whole process terminates when either the lowest energy point is found or no upward/downward jumps have been taken for a number of successive thermal equilibrium. The structure of the algorithm may be sketched as follows:

```
Randomly select an initial solution s;
Compute the cost at this solution $E_s$;
Select an initial temperature T;
Select a cooling factor $\alpha < 1$;
Select an initial chain $n_{rep}$;
Select a chain increasing factor $\beta > 1$;
Repeat
  Repeat
    Select a neighbor solution n to s;
    Compute the cost at n, $E_n$;
    $\Delta = E_n - E_s$;
    If $\Delta < 0$,
      $s = n$; $E_s = E_n$;
    Else
      Generate a random value x in the range (0,1);
      If $x < exp(-\Delta/T)$,
        $s = n$; $E_s = E_n$;
      End
    End
  Until iteration = $n_{rep}$ (equilibrium state at T)
  Set $T = \alpha * T$;
  Set $n_{rep} = \beta * n_{rep}$;
Until stopping condition = true
$E_s$ is the cost and s is the solution.
```

5.3. Applying the algorithm

To implement the SA algorithm, a number of decisions must be made. These decisions are concerned with the choice of an energy function, a cooling schedule, a neighborhood structure and the choice of annealing parameters such as an initial temperature, a cooling factor, an increasing factor of the inner loop repetitions and a stopping condition. Each decision need to be made with care as they effect the speed of the algorithm and the quality of solutions.

5.3.1. Energy function

The energy function is the heart of the SA algorithm. It shapes the energy landscape and affects how the algorithm reaches a solution. For the allocation problem, the energy function is complex. It represents the objective function to be optimized and it has to penalize the following characteristics:

(i) a task redundancy,
(ii) a processor with a load utilization $\succ$ 100%,
(iii) a processor with a memory utilization $\succ$ 100%,
(iv) a link/channel with a communication capacity utilization $\succ$ 100%.

These characteristics should be penalized to achieve the application requirements and validate the availability of the system resources. In our case, the first property is penalized by constructing an allocation vector $A(M, 1)$ whose element $A(i)$ represents the processor p where the task i is allocated. At each movement of neighboring solutions, one of the tasks is moved from one processor to another. Therefore, each task cannot be allocated to more than one processor. The second property is penalized by comparing the processing load requirements of all the tasks mapped to a processor p and the available processing load of p. An energy component E_p is determined such that $E_p = 1$ if the processing load requirements exceed the available load of p and $E_p = 0$ otherwise. Similarly, the third and the fourth properties are penalized. The third property is penalized by returning an energy component E_m such that $E_m = 1$ if the memory requirements of all the tasks mapped to a processor p exceed the available memory at p and $E_m = 0$ otherwise. Also, the fourth property is penalized by returning an energy component E_c such that $E_c = 1$ if the communication capacity requirements of all edges mapped to a link/path pq exceed the available capacity of the path and $E_c = 0$ otherwise. Let k be a penalty factor, then the energy function E may be formulated as follows:

$$E = Z + k(E_p + E_m + E_c).$$

5.3.2. Neighborhood structure

The neighborhood defines the procedure to move from a solution point to another solution point. In this paper, the neighbor function is simple. A neighboring solution is obtained by choosing at random a task i from the current allocation vector A and assign it to another randomly selected processor p.

5.3.3. Cooling schedule

The cooling schedule defines the procedure to reduce the temperature as an equilibrium state is reached. This process is governed by the number of inner loop repetitions n_{rep} and the cooling rate α. In this paper, a geometric cooling schedule is used. The temperature T is reduced so that $T = \alpha * T$, where α is a constant less than 1. At each temperature, the chain (the number of repetitions of the inner loop) is updated in a similar manner: $n_{\text{rep}} = \beta * n_{\text{rep}}$, where β is a constant greater than 1.

5.3.4. Annealing parameters

The annealing parameters are concerned with the choice of an initial temperature T, a cooling factor α, an increasing factor β, and a stopping condition. These factors have a significant effect on the success of the annealing algorithm. Note that, the best values of these parameters may differ from application to application and possibly from instance to instance. The annealing parameters may be described as in the following:

- *Initial temperature T*: The initial temperature represents one of the very important parameters in the SA algorithm. If the initial temperature is very high, the execution time of the algorithm becomes very long. On the other hand, if the initial temperature is low, poor results are obtained. The initial temperature must be hot enough to allow an almost free exchange of neighboring solutions. One approach is that the system can be rapidly heated fairly until the proportion of accepted moves to rejected moves reaches a required value. The proportion of accepted moves, which represents a suitability volatile system, to rejected moves can be decided beforehand. At this point, the cooling schedule can start. This method corresponds to the physical analogy in which a substance is heated quickly to its liquid state before being cooled slowly according to the annealing schedule. In our case, the initial temperature is set after executing a sufficiently large number of random moves, such that the worst move would be allowed. Let T be the initial temperature, c_r and c_i be numbers corresponding to cost reduction and cost increase, respectively, c_a be the average cost increase value of the c_i trials, and a_0 be the desired initial acceptance value. Then, the following relation may be written

$$a_0 = \frac{c_r + c_i e^{-c_a/T}}{c_r + c_i}$$

which gives the initial temperature as

$$T = -\frac{c_a}{\log(\frac{c_r}{c_i}(a_0 - 1) + a_0)}.$$

- *Cooling factor α*: The cooling factor α represents the rate at which the temperature T is reduced. This factor is vital to the success of any annealing process. A rapid reduction yields a bad local optimum and slow cooling yields expensive in time. Most reported successes in the literature use values between 0.8 and 0.95 with bias to the higher end of the range. In this paper, the cooling rate is chosen to be $\alpha = 0.90$.
- *Increasing factor β*: The increasing factor β represents the rate at which the inner number of repetitions is increased

as the temperature is reduced. It is important to spend long time at lower temperature to ensure that a local optimum has been fully explored. In this paper, the increasing factor β is chosen to be $\beta = 1.05$. The lower the temperature, the bigger the number of inner loop repetitions.

- *Stopping condition*: The criterion for stopping can be expressed either in terms of the temperature parameter or in terms of the steady state of the system at the current solution. Other rules attempt to identify a stopping condition by specifying that a number of iterations or temperature levels must have passed without an acceptance. The simplest rule is to specify the total number of iterations and stop when this number has been completed. In our case, the final temperature is chosen to give a low acceptance value.

6. Performance evaluation

The proposed algorithm is coded in Matlab and tested for a large number of randomly generated task graphs that are allocated onto a distributed system. The simulation program contains two major parts. The first part reads as input the number of tasks M and the number of processors N. It then generates a task graph and equivalent parameters: tasks execution times, memory and processing load requirements, and communication capacity requirements. Considering a particular topology of a distributed system, the program also generates the system parameters: available memory, processing load and communication capacity. For generating the parameters, the program uses the following test data which is similar to the one used in [17,11]. The failure rates of processors and communication links are given in the ranges [0.0005–0.00010] and [0.00015–0.00030], respectively. The time of processing a task at different processors is given in the range [15–25]. The memory requirements of each task is given in the range [1–10]. The value of data to be communicated between tasks is given in the range [5–10]. The average task connectivity (i.e., the average number of neighbors to a task) is 3. The second part of the simulation program formulates the problem and applies the SA algorithm to derive a task allocation and the associated system reliability.

For evaluation, two system configurations are considered: a distributed system of four computers with bus topology and a distributed system of six computers with fully connected topology. Furthermore, five sets of randomly generated problems with sizes $M = 4, 8, 12, 16$ and 20 are used. For each problem set, five task graphs of the same size are generated randomly. For each graph, 10 simulation runs are conducted by the SA algorithm and the average values of both the unreliability cost function Z and the algorithm computation time are computed over these simulation runs. The generated instances are also solved by using the BB technique [2,3] to test the quality of solutions obtained by the SA algorithm.

Tables 1 and 2 summarize the simulation results by deploying the SA and the BB algorithms to solve the generated problems. Table 1 presents the results for the case of four computers with bus topology while Table 2 shows the results for the case of six computers with fully connected topology. In the tables, the first column indicates the problem sets, where M is the number of tasks and N is the number of processors. The second and the fourth columns represent the values of the unreliability cost function Z obtained by the SA and the BB algorithms, respectively. The computing times for the two algorithms are listed in the third and the fifth columns, respectively. The last column represents the average deviation ΔZ in percentage between the suboptimal and the optimal solutions; $\Delta Z = \frac{Z_{SA}-Z_{BB}}{Z_{BB}} \times 100$, where Z_{SA} and Z_{BB} are the near optimal and optimal values of the unreliability cost function Z. The results show that, unlike the exponential nature of the BB technique, the computation time of the SA algorithm slowly increases with the problem size. Furthermore, the SA algorithm quickly finds a near optimal allocation with average deviation not exceeding 4% from the global optimum solutions that are obtained by applying the BB technique.

Table 1
Simulation results for the case of four computers of bus topology

Case (M, N)	Z_{SA}	$time_{SA}$ (s)	Z_{BB}	$time_{BB}$ (s)	$\Delta Z\%$
1 (4,4)	0.0109	3.9798	0.0109	0.5000	0.00
2 (4,4)	0.0188	5.3467	0.0188	0.8600	0.00
3 (4,4)	0.0223	5.5733	0.0223	0.9220	0.00
4 (4,4)	0.0110	8.2388	0.0110	0.5780	0.00
5 (4,4)	0.0362	5.0735	0.0362	1.5780	0.00
1 (8,4)	0.0531	13.1015	0.0531	12.0470	0.00
2 (8,4)	0.0160	22.4501	0.0160	10.8600	0.00
3 (8,4)	0.0313	7.7907	0.0313	13.1090	0.00
4 (8,4)	0.0242	26.2031	0.0242	14.2650	0.00
5 (8,4)	0.0140	34.0092	0.0140	22.9690	0.00
1 (12,4)	0.0230	26.1799	0.0228	39.6400	0.88
2 (12,4)	0.0257	16.5968	0.0256	49.7190	0.39
3 (12,4)	0.0367	14.3721	0.0363	30.2810	1.10
4 (12,4)	0.0277	30.8140	0.0274	40.3280	1.09
5 (12,4)	0.0365	22.7845	0.0365	46.7030	0.00
1 (16,4)	0.0481	16.8673	0.0470	120.4210	2.34
2 (16,4)	0.0390	19.5375	0.0378	64.1090	3.17
3 (16,4)	0.0325	44.8157	0.0317	102.8120	2.52
4 (16,4)	0.0565	17.9219	0.0544	99.8590	3.86
5 (16,4)	0.0383	27.4424	0.0372	331.9850	2.96
1 (20,4)	0.0388	25.1154	0.0374	255.6090	3.74
2 (20,4)	0.0494	18.3970	0.0481	163.5780	2.70
3 (20,4)	0.0296	29.2769	0.0285	193.7030	3.86
4 (20,4)	0.0481	21.3857	0.0473	291.3430	1.69
5 (20,4)	0.0449	26.8031	0.0432	618.3280	3.94

Table 3 shows the effect of task allocation on the computing time and on the system reliability. Considering a parallel application of eight tasks and a distributed system of four computers with bus topology, the table shows the unreliability cost function Z, the computing time (i.e., the total sum of execution and communication times) and the system reliability at 10 valid allocations. As shown in the table, at constant failure rates of the system components, the system reliability is changed with the task distribution. This means that, reliability of a distributed system depends not only on the reliability of its components

Table 2
Simulation results for the case of six computers of fully connected topology

Case (M, N)	Z_{SA}	$time_{SA}$ (s)	Z_{BB}	$time_{BB}$ (s)	$\Delta Z\%$
1 (4,6)	0.0077	4.9753	0.0077	3.4380	0.00
2 (4,6)	0.0132	3.7875	0.0131	6.8120	0.76
3 (4,6)	0.0099	4.4562	0.0099	3.7030	0.00
4 (4,6)	0.0074	5.3529	0.0074	2.8440	0.00
5 (4,6)	0.0098	7.1657	0.0098	3.8130	0.00
1 (8,6)	0.0148	12.1626	0.0144	27.2040	2.78
2 (8,6)	0.0151	17.7281	0.0150	35.0940	0.67
3 (8,6)	0.0172	8.5517	0.0171	54.6090	0.58
4 (8,6)	0.0130	14.4468	0.0130	33.8440	0.00
5 (8,6)	0.0144	25.9892	0.0143	90.2340	0.70
1 (12,6)	0.0186	29.0999	0.0186	116.0160	0.00
2 (12,6)	0.0228	23.8343	0.0225	285.2650	1.33
3 (12,6)	0.0175	19.6750	0.0173	115.4530	1.16
4 (12,6)	0.0229	30.1034	0.0225	307.6250	1.78
5 (12,6)	0.0183	29.7455	0.0178	254.5780	2.23
1 (16,6)	0.0250	34.9531	0.0246	1016.30	1.63
2 (16,6)	0.0247	27.9514	0.0241	1553.70	2.49
3 (16,6)	0.0242	31.9986	0.0233	1189.40	3.86
4 (16,6)	0.0226	30.1076	0.0222	460.1710	1.80
5 (16,6)	0.0318	32.5439	0.0306	1028.10	3.92

Table 3
Effect of task allocation on the computing time and the system reliability

Iteration	Z	Computing time	System reliability
1	0.0211	165	0.9791
2	0.0205	158	0.9797
3	0.0195	160	0.9806
4	0.0190	157	0.9812
5	0.0186	145	0.9815
6	0.0179	151	0.9822
7	0.0162	159	0.9839
8	0.0152	162	0.9849
9	0.0150	154	0.9851
10	0.0140	149	0.9861

(hardware and software) but also on the distribution of the tasks on the available processors in the system. Hence, a parallel application can be executed with high reliability if the various tasks of the application are assigned carefully to the appropriate processors in the system considering the failure probabilities of both the processors and the communication links. In the table, iteration number 5 achieves the minimum computing time, i.e., the minimum execution and communication times, which is 145 but the system reliability equals 0.9815. On the other hand, iteration number 10 achieves the optimal system reliability which is 0.9861 but the computing time equals 149. Hence, a trade off exists between minimizing the total computing time and maximizing the system reliability and a compromise should be made between these conflicting objectives with the task allocation problem.

7. Conclusions

The task allocation problem in terms of reliability is investigated in this paper. The problem is first formulated as an optimization problem composed of a cost function representing the unreliability caused by the execution of tasks on the system processors and the unreliability caused by the interprocessor communication times subject to constraints imposed by both the application and the system resources. An allocation algorithm based on the simulated annealing (SA) technique is then developed to quickly find a near optimal solution to this problem. The performance of the algorithm is evaluated through experimental studies on a large number of randomly generated instances and the quality of solutions are compared with those derived by using the BB technique. The simulation results show that, in most of the tested cases, the SA algorithm finds a near optimal allocation efficiently with average deviation not exceeding 4% from the global optimum solutions; therefore, it is a desirable approach to solve the allocation problem. The next step of this work is to develop an exact algorithm to find an optimal allocation that maximizes the system reliability in acceptable computation time.

References

[1] E. Aarts, J. Korst, Simulated Annealing and Boltzmann Machines, Wiley, New York, 1989.

[2] G. Attiya, Y. Hamam, Static task assignment in distributed computing systems, 21st IFIP TC7 Conference on System Modeling and Optimization, Sophia Antipolis, Nice, France, 2003.

[3] G. Attiya, Y. Hamam, Optimal allocation of tasks onto networked heterogeneous computers using minimax criterion, in: Proceedings of International Network Optimization Conference (INOC'03), Evry/Paris, France, 2003, pp. 25–30.

[4] A.O. Charles Elegbede, C. Chu, K.H. Adjallah, F. Yalaoui, Reliability allocation through cost minimization, IEEE Trans. Reliab. 52 (2003) 106–111.

[5] C.C. Chiu, Y.-S. Yeh, J.-S. Chou, A fast algorithm for reliability-oriented task assignment in a distributed system, Comput. Commun. 25 (2002) 1622–1630.

[6] C.-C. Hsieh, Optimal task allocation and hardware redundancy policies in distributed computing systems, European J. Oper. Res. 147 (2003) 430–447.

[7] C.-C. Hsieh, Y.-C. Hsieh, Reliability and cost optimization in distributed computing systems, Comput. Oper. Res. 30 (2003) 1103–1119.

[8] C.-C. Hui, S.T. Chanson, Allocating task interaction graphs to processors in heterogeneous networks, IEEE Trans. Parallel Distrib. Systems 8 (1997).

[9] M. Kafil, I. Ahmed, Optimal task assignment in heterogeneous distributed computing systems, IEEE Concurrency 6 (1998) 42–51.

[10] S. Kartik, C.S.R. Murthy, Improved task-allocation algorithms to maximize reliability of redundant distributed computing systems, IEEE Trans. Reliab. 44 (1995) 575–586.

[11] S. Kartik, C.S.R. Murthy, Task allocation algorithms for maximizing reliability of distributed computing systems, IEEE Trans. Comput. 46 (1997).

[12] S. Kirkpatrick, C.D. Gelatt, J.M.P. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671–680.

[13] A. Kumar, D.P. Agrawal, A generalized algorithm for evaluating distributed-program reliability, IEEE Trans. Reliab. 42 (1993) 416–426.

[14] A. Kumar, R.M. Pathak, Y.P. Gupta, Genetic algorithm based approach for file allocation on distributed systems, Comput. Oper. Res. 22 (1995) 41–54.

[15] R.M. Pathak, A. Kumar, Y.P. Gupta, Reliability oriented allocation of files on distributed systems, in: Proceedings of the Third IEEE Symposium on Parallel and Distributed Processing, 1991, pp. 886–893.
[16] S.M. Shatz, J.-P. Wang, Models and algorithms for reliability-oriented task-allocation in redundant distributed-computer systems, IEEE Trans. Reliab. 38 (1989) 16–27.
[17] S.M. Shatz, J.P. Wang, M. Goto, Task allocation for maximizing reliability of distributed computer systems, IEEE Trans. Comput. 41 (1992) 1156–1168.
[18] P.A. Tom, C. Murthy, Algorithms for reliability-oriented module allocation in distributed computing systems, J. Systems Software 40 (1998) 125–138.
[19] A.K. Verma, M.T. Tamhankar, Reliability-based optimal task-allocation in distributed-database management systems, IEEE Trans. Reliab. 46 (1997) 452–459.
[20] D.P. Vidyarthi, A.K. Tripathi, Maximizing reliability of distributed computing systems with task allocation using simple genetic algorithm, J. Systems Architecture 47 (2001) 549–554.
[21] J.-P. Wang, S.M. Shatz, Reliability-oriented task allocation in redundant distributed systems, in: Proceedings of 12th International Conference on Computer Software and Applications (COMPSAC 88), 1988, pp. 276–283.
[22] J.-P. Wang, S.M. Shatz, Task allocation for optimized system reliability, in: Proceedings of Seventh Symposium on Reliable Distributed Systems, 1988, pp. 82–90.