Task Allocation for Maximizing Reliability of Distributed Computing Systems Using Genetic Algorithms

A. Y. Hamed
Department of Computer Science, Faculty of Science, Sohag University
Sohag, , Egypt.

Abstract The problem of the task allocation in distributed computing system is to need to allocate a number of tasks to different processors for execution. The paper deals with the problem of task allocation in heterogeneous distributed computing systems with the goal of maximizing the system reliability. We present a genetic algorithm to obtain the optimal solution for this problem. In the performance of the algorithm we consider mor one parameter such as the number of tasks, the number of processors, and task interaction denisty of applications. The experimental results illustrate the effectiveness of this algorithm over conventional algorithms.

Keywords: ***Distributed computing systems, Genetic Algorithms, Task Allocations, and Maximizing Reliability*****.**

I. INTRODUCTION

A distributed computing system (DCS) consists of a set of multiple processors interconnected by communication links. A very common interesting problem in DCS is the task allocation. This problem deals with finding an optimal allocation of tasks to the processors so that the system reliability is maximized without violating any of the system constraints. **A** task to be run on the distributed system consists of a set of modules. Each of the modules comprising a task will execute on one of the processors and communicate with some other modules of the task.

Many researchers have been presented to improve the performance of a DCS in several issues arise such as the maximization of system reliability and safety, [1, 2, 19] and the achievement of better fault tolerance using software and hardware redundancy, [3, 4]. Meanwhile, resource constraints may be imposed by memory size of processors and capacity of communication links. This paper investigates the task allocation problem that aims to maximize the system reliability subject to resource constraints. Distributed system reliability (DSR) has been defined by [5] 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. There are two major DSR evaluation approaches in the literature. Kumar, [5] evaluated the distributed program reliability (DPR) by searching the entire minimal file spanning trees (MFST's), which provide accessibility to the required data files for the program. Then the DSR can be computed by multiplying the DPR's of all distributed programs. However, some system parameters such as the execution times of programs and communication loads on the links are not considered in this model. They assume that all processors and communication links have constant reliability. Shatz, [6] proposed another DSR evaluation model where failures from processors or communication links are time-dependent, which fits the scenario that modules with longer execution or communication times will increase the failure probability of involved processors or communication links. Unfortunately, the computational complexity for evaluating the DSR has been shown to be NP-hard, [7]. Researchers, however, have developed alternative algorithms for tackling this problem. These methods can be divided into two categories: exact algorithms and approximation algorithms. The exact algorithms strive to find an optimal task allocation for small-sized instances. Kartik, and Murthy, [1, 3] used the idea of branch and bound with underestimates and reorder the modules according to module independence for reducing the computations required. Verma, [8] employed the branch and bound technique for solving the reliability-based multiple join problems in distributed database management systems.

The approximation algorithms, on the other hand, derive sub-optimal task allocations within reasonable times. Kartik and Murthy, [1, 3] also developed a heuristic approach from their exact algorithm by assuming that the best solution is more likely to be found in the least cost path thereby reducing the worst-case time complexity of the algorithm. Srinivasan, [2] proposed a clustering- based heuristic which groups heavily communicating modules into clusters in order to reduce the inter module communication (IMC) as much as possible.

The development of the met heuristic optimization theory has been flourishing during the last decade [9- 11]. Applying met heuristic algorithms for conquering the task allocation problem has several benefits. (1) Exact algorithms search for optimal solutions and are thus computationally intensive, while meta heuristic algorithms deriving near-optimal solutions within reasonable times are more suitable for real-time applications. (2) Many successful applications, [12] have shown the superiority of meta heuristic algorithms over heuristic algorithms in terms of quality of the final solutions obtained, so that careful design and implementation of the meta heuristic algorithms can improve the results substantially.

Genetic algorithms (GAs) have also been adopted for solving the problem and obtained promising results. Vidyarthi, [13] used a simple GA to maximize the reliability of DCS with task allocation. Hsieh, [4] proposed a hybrid GA that combines the GA with a local search procedure. The experimental results show that the hybrid GA produces better task allocation than the simple GA. Gas, [14] belong to a branch of computational intelligence called met heuristic.

In this paper, we present a genetic algorithm for solving the task allocation problem with the goal of maximizing the system reliability. The experimental results reveal that the proposed algorithm produces better task allocation than other algorithms on a large set of simulated problem instances, and the difference is larger for large problems.

The remainder of this paper is organized as follows. Section 2 describes the task allocation problem for maximizing reliability. Section 3 shows how to compute the system reliability. Section 4 presents the proposed genetic algorithm in detail. Experimental results are presented in section 5. Finally, Section 6 concludes this work.

Notations

x_{ik} Decision variable: $x_{ik} = 1$ if module i is allocated to processor k, and $x_{ik} = 0$ otherwise
p Number of processors
n Number of tasks
l_{kb} Communication link connecting two processors k and b
λ_k Failure rate of processor k
μ_{kb} Failure rate of communication link l_{kb}
e_{ik} Incurred accumulative execution time (AET) if task i is executed on processor k
c_{ij} Incurred intermodule communication (IMC) cost between task i and j if they are executed on different processors
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 k
l_i Computation resource requirements of module i from its execution processor
L_k Amount of computation resource capacitated with processor k.
gen The generation counter.
Pm The GA mutation rate.
Pc The GA crossover rate.
Maxgen The required number of generations
Pop_size The population size

II. The PROBLEM DESCRIPTION

The problem is concerned with an optimal allocation of the tasks of a parallel application on to the processors in DCS. An optimal allocation is one that maximizes the system reliability function subject to the system constraints. The distributed system consists of a set of heterogeneous processors interconnected via a communication network as shown in Fig 1(a). A distributed application is represented by a task interaction graph (TIG) as shown in Fig. 1(b). We consider the following assumptions with the task allocation problem.

The assumptions

- The processors involved in the DCS are heterogeneous. Hence, the processors may be constrained with various units of memory and computation resources and they may have different processing speeds and failure rates. Moreover, the communication links may have different bandwidths and failure rates. A communication subsystem is assumed to handle the interprocessor communication, and the communication can be performed concurrently.
- The execution of a task will consume a specific amount of memory and computation resource from its assigned processor. Two modules, if executed on different processors, may communicate with each other and incur a specific amount of intermodule communication (IMC) cost measured in some unit of data quantity.
- A module may take different accumulative execution time (AET) if it is executed on different processors. An amount of IMC cost may take

different durations of transmission time if transmitted through different communication links.

- The state of processors and communication links is either operational or down. Failure events are statistically independent

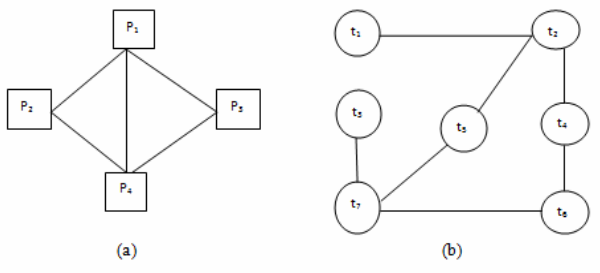

Figure 1 A distributed system and a task interaction graph. (a) A distributed system. (b) A task graph

The above assumptions are basically similar to those presented by [6] from which several task allocation techniques with reliability maximization have been developed [1-4], [18]

Briefly, the purpose of this paper is to find a task allocation that maximize the system reliability and satisfies all of the source constraint. A task execution process in a DCS can be described by the processor interaction graph (PIG) and the task interaction graph (TIG). The PIG illustrates how the processors are connected in the network topology of the computation environment. The TIG renders the intermodule communication cost incurred by the mission. An important characteristic of TIG is the task interaction density, denoted by d, which measures how communication intensive a task is. We define d as the ratio of the number of intermodule communication requests to the number of pairs of different tasks. As d increases, the intermodule communication becomes more intensive and the reliability derived could be lower due to involvement of more communication links. Moreover, the CPU time required will slightly increase with large d because of the extra computations for the reliability related to those involved communication links, [15].

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

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

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, [16] .

III. THE SYSTEM RELIABILITY

The reliability of a distributed computing system for a given application is the reliability that the application assigned to the processors in a system by some task assignment *x*, can run successfully during the execution lifetime [4], [15]. That is the system reliability is the product of the probability that each processor is operational during the time of processing the tasks assigned to it, and the probability that each communication path is operational during the active period of data communication between the terminal processors of the path, [17].

The reliability of processor *k* during a time interval *t* is $e^{-\lambda_k t}$, [19]. Under a task allocation *x*, the time required to execute all the tasks assigned to processor *k* is $\sum_{i=1}^{n} x_{ik} e_{ik}$, and then the corresponding processor reliability can be formulated as follows:

$$R_k(x) = e^{-\lambda_k \sum_{i=1}^{n} x_{ik} e_{ik}} \quad (1)$$

Similarly, the reliability of the path *kb* during a time interval *t* is $e^{-\mu_{kb} t}$, [19]. Under a task allocation *x*, the time required for data communication between the terminal processors *k* and *b* is $\sum_{i=1}^{n} \sum_{j \neq i} x_{ik} x_{jb} (c_{ij} / w_{kb})$, then the corresponding path reliability can be given by the following equation:

$$R_{kb}(x) = e^{-\mu_{kb}\sum_{i=1}^{n}\sum_{j\neq i} x_{ik}x_{jb}(c_{ij}/w_{kb})} \quad (2)$$

As the system reliability requires that all involved components are operational during the elapsed time for the execution, the DSR with the task allocation *x* is computed as follows:

$$R(x) = \prod_{k=1}^{p} R_k(x) \prod_{k=1}^{p}\prod_{k\neq b} R_{kb}(x) \quad (3)$$

Maximizing the system reliability is equivalent to minimizing the following cost.

$$Cost(x) = \sum_{k=1}^{p}\sum_{i=1}^{n} \lambda_k x_{ik} e_{ik} + \sum_{k=1}^{p-1}\sum_{b>k}\sum_{i=1}^{n}\sum_{i\neq j} \mu_{kb} x_{ik} x_{jb} (c_{ij}/w_{kb}) \quad (4)$$

That is, we can write R(x) in another form as follows:

$$R(x) = e^{-Cost(x)}$$

With the system resource constraints taken into account the task allocation model for system reliability is formulated as follows:

$$MinCost(x) \quad (5)$$

Such that

$$\sum_{k=1}^{p} x_{ik} = 1 \qquad \forall i = 1, 2, \ldots\ldots\ldots\ldots n \quad (6)$$

$$\sum_{i=1}^{n} m_i x_{ik} \leq M_k \qquad \forall k = 1, 2, \ldots\ldots\ldots\ldots p \quad (7)$$

$$\sum_{i=1}^{n} l_i x_{ik} \leq L_k \qquad \forall k = 1, 2, \ldots\ldots\ldots\ldots p \quad (8)$$

$$x_{ik} \in [0, 1] \qquad \forall i, k \quad (9)$$

IV. THE PROPOSED GENETIC ALGORITHM

To solve the problem of task allocation in DCS via GAs, it is necessary to find a mapping of a potential candidate for a solution onto a sequence of binary digits, the so called chromosome. In the proposed genetic algorithm, we consider the four components: (1) an encoding method that is a genetic representation (genotype) of solutions to the program. (2) A way to create an initial population of chromosomes, (3) the objective function (4) the genetic operators (crossover and mutation) that alter the genetic composition of offspring during reproduction.

A. Encoding Method

In our case, however, it is more efficient to represent chromosomes as strings of integers. The length of the chromosomes is given by the number of tasks that should be allocated. Every gene in the chromosome represents the processor where the task is running on. Fig. 2 gives an exemplary mapping of *n* tasks on *m* processors.

t_1	t_2	t_3	t_4	t_5	t_6	…	…	…	t_n
P_1	P_2	P_1	P_m	P_1	P_2	…	…	…	P_m

Figure 2: The task allocation in the form of chromosome

B. Initial Population

The initial population is generated according to the following steps:

- A chromosome x in the initial population can be generated as shown in Fig 1.
- The chromosome must be contain only m none zero element
- The chromosome must be containing all numbers of the processors as shown in Fig. 1.
- Repeat steps 1 to 4 to generate pop_size number of chromosomes.

C. The Objective Function

That is find a task allocation *x* such that the overall system reliability is maximized.

$$Max \left\{ R(x) = \prod_{k=1}^{p} R_k(x) \prod_{k=1}^{p} \prod_{k \neq b} R_{kb}(x) \right\}$$

D. The Genetic Operations

- Crossover operations
- Mutation Operations

a) The Crossover Operation

The crossover operation is used to breed a child from two parents by one cut point. The crossover operation will perform if the crossover ratio (Pc>=0.95) is verified. The cut point is selected randomly. The crossover operation is performed as follows:

- Select two chromosomes randomly from the current population.
- Randomly select the cut point
- Fill the components of the chromosome
  1. By taking the components of the first chromosome (from the first gene to the cut point) and fill up to the child.
  2. Also, tacking the components of the second chromosome (from the cut point+1 to the last gene) and fill up to the child.

b) The Mutation Operation

The mutation operation is performed on bit-by-bit basis. In the proposed approach, the mutation operation will perform if the mutation ratio (Pm) is verified. The mutation ratio, Pm in this approach will be 0.2 and is estimated randomly. The point to be mutated is selected randomly.

V. The PROPOSED GENETIC ALGORITHM

The following algorithm and flowchart explain how we can use the above assumptions and proposed functions to find a task allocation *x* such that the overall system reliability is maximized.

The Proposed Algorithm

1. Input: : Set the parameters: pop_size, maxgen, Pm, Pc .
2. Steps:
3. Generate the initial population as in section 4.2.
4. Rs = 0 // Initial value for system reliability
5. gen=1
6. While (gen <= maxgen) do
7. P =1
8. While (p <= pop_size) do
9. Genetic operations
   - Select two chromosomes from the parent population randomly
   - Apply crossover according to Pc (Pc >=0.9).
   - Mutate the new child according to Pm (Pm<=0.2).
10. Compute the reliability of the new child R(x) according to Eq. 3.
11. If (R(x) > Rs) Rs=R(x) and save this child as a candidate solution (x)
12. P ← p+1.
13. End do
14. Set gen =gen + 1
15. End do
16. Output Rs and x.

VI. EXPERIMENTAL RESULTS

In this section we show the effectiveness of the above algorithm by applying it on the following example:

The number of processors *p* in heterogeneous distributed computing systems is varied as 6 and 8; the number of tasks n varies through the values 10, 20 and 30, to verify the proposed algorithm with different problem scales. For each pair of (n, p), we consider three different TIGs with three different task interaction density values 0.2, 0.5, and 0.8. The values of other system parameters are generated randomly with the ranges listed in Table 1, [17].

The experimental environment is a 2.93 GHz PC with 4 GB RAM. The parameters setting in this algorithm are: pop_size = 20, P_m<= 0.1, $P_{c>}$=0.9, maxgen =100. The proposed genetic algorithm is compared with HBMO algorithm, [17].

TABLE 1. System parameters and the corresponding testing ranges

System parameters	Testing ranges
Failure rate of processor	0.00005 - 0.00010
Failure rate of communication link	0.00015 - 0.00030
Accumulative execution time (AET)	15 - 25
Intermodule communication (IMC) cost	15 - 25
Memory resource requirement	5 - 15
Computation resource requirement	5 - 15
Memory resource capacity	100 - 200
Computation resource capacity	100 - 200

Table 2 shows the distributed system reliability (DSR), computational time, and allocation of tasks

obtained using the proposed algorithm and HBMO algorithm, [17].

TABLE 2. The Reliability, Computational time, and Task Allocation Obtained by the Proposed and HBMO Algorithm

			Proposed Algorithm			**HBMO(CCR=2.0)**	
n	p	D	**R(x)**	**t**	**x**	**R(x)**	**t**
10	6	0.2	0.998	0.062	1 1 3 1 3 3 4 4 3 3	0.995	0.45
		0.5	0.995	0.078	5 3 6 1 1 6 1 1 2 6	0.993	0.55
		0.8	0.991	0.078	3 3 1 6 6 2 3 6 2 6	0.993	0.67
	8	0.2	0.998	0.109	5 3 5 8 8 4 5 5 8 2	0.990	0.67
		0.5	0.996	0.141	2 2 2 2 4 4 1 3 3 1	0.990	0.48
		0.8	0.992	0.14	8 4 8 1 7 6 8 8 8 6	0.991	0.51
20	6	0.2	0.989	0.28	3 1 2 6 6 2 1 1 4 1 1 3 1 1 1 1 1 4 2 1	0.976	1.69
		0.5	0.981	0.265	2 1 1 3 1 1 2 1 1 2 2 6 1 3 2 2 2 2 6 1	0.982	2.06
		0.8	0.955	0.281	4 2 3 4 2 6 5 4 4 5 2 2 4 4 1 5 3 6 3 4	0.982	1.96
	8	0.2	0.989	0.421	8 5 3 5 8 3 1 2 3 5 4 1 7 4 2 8 4 1 4 1	0.981	2.48
		0.5	0.971	0.468	3 5 4 7 7 2 4 4 3 3 4 5 4 3 5 3 6 3 5 7	0.977	3.19
		0.8	0.942	0.421	7 7 5 7 6 4 5 6 4 4 2 2 4 4 4 5 2 3 4 2	0.982	3.01
30	6	0.2	0.964	0.577	6 6 5 4 4 2 5 4 6 5 6 6 5 5 4 6 3 4 2 4 2 6 3 5 3 5 5 3 6 4	0.973	4.20
		0.5	0.917	0.577	4 4 4 4 4 6 3 4 3 4 4 6 4 1 3 6 3 4 1 1 3 6 3 6 1 6 1 2 5 3	0.975	3.53
		0.8	0.915	0.561	3 2 1 5 2 5 2 5 5 5 2 1 1 1 6 4 3 5 5 3 1 2 2 5 5 3 4 5 1 1	0.970	4.13
	8	0.2	0.971	0.967	1 5 5 5 1 4 5 5 4 5 2 2 4 5 1 1 8 5 3 5 3 3 5 7 2 6 5 8 3 8	0.974	5.04
		0.5	0.930	0.983	7 7 3 8 3 5 5 8 5 7 8 8 3 4 8 8 1 5 4 8 5 8 4 8 8 3 7 2 5 1	0.975	6.74
		0.8	0.906	1.014	6 7 7 8 7 5 5 7 2 6 7 6 5 7 6 3 3 3 5 4 5 5 7 4 2 7 5 5 7 6	0.978	7.58

Fig. 3. Shows the system reliability for the tasks (10, 20, 30) with task interaction density (D=0.2) allocation on 6 processors.

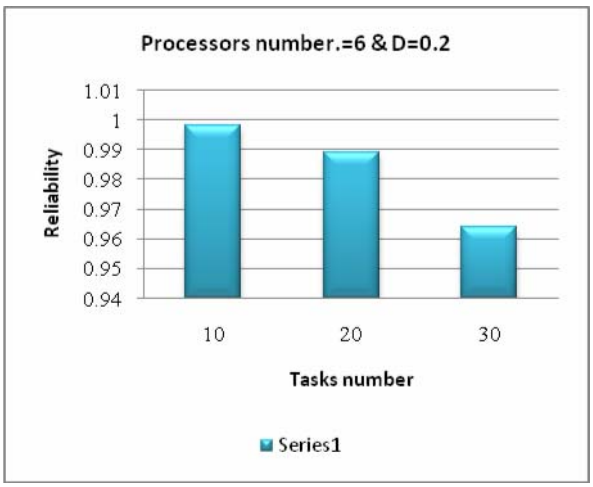

Figure 3 the system reliability for the tasks (10, 20, 30) with D=0.2 allocation on 6 processors.

Fig. 4, Shows the system reliability for the 20 tasks with task interaction density (D=0.2) allocation on 6 and 8 processors.

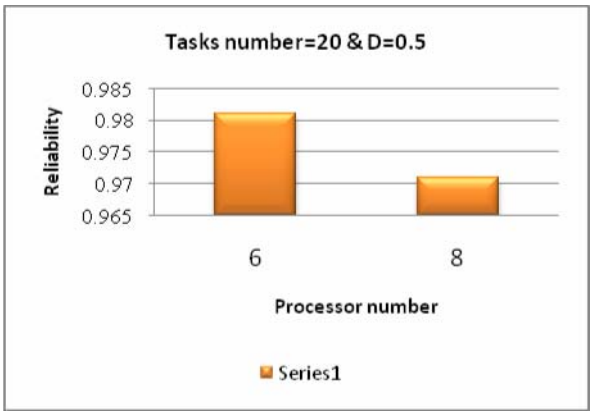

Figure 4 the system reliability for the 20 tasks with D=0.2 allocation on 6 and 8 processors.

Fig. 5, Shows the system reliability for the 30 tasks with task interaction density D (0.2, 0.5, 0.8) allocation on 6 processors.

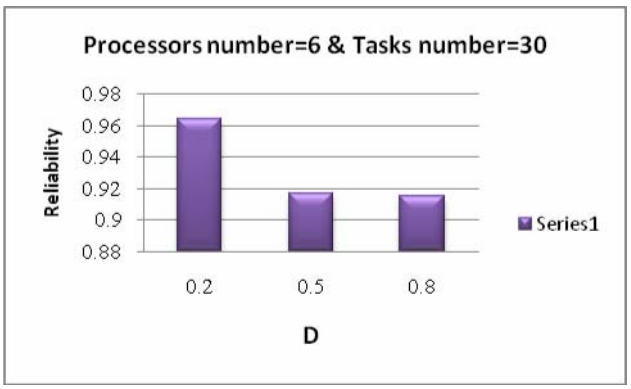

Figure 5 the system reliability for the 30 tasks with D (0.2, 0.5, and 0.8) allocation on 6 processors.

Fig 6. Shows the system reliability obtained by the proposed and HBMO algorithm for the tasks (10, 20, 30) with task interaction density (D=0.5) allocation on 6 processors.

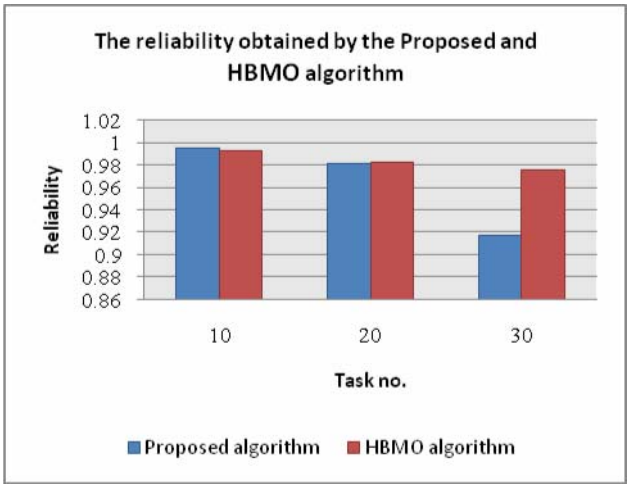

Figure 6 **the** system reliability obtained by the proposed and HBMO algorithm for the tasks (10, 20, 30)

Fig. 7 shows the execution time obtained by the proposed and HBMO algorithm for the tasks (10, 20, 30) with task interaction density (D=0.5) allocation on 6 processors

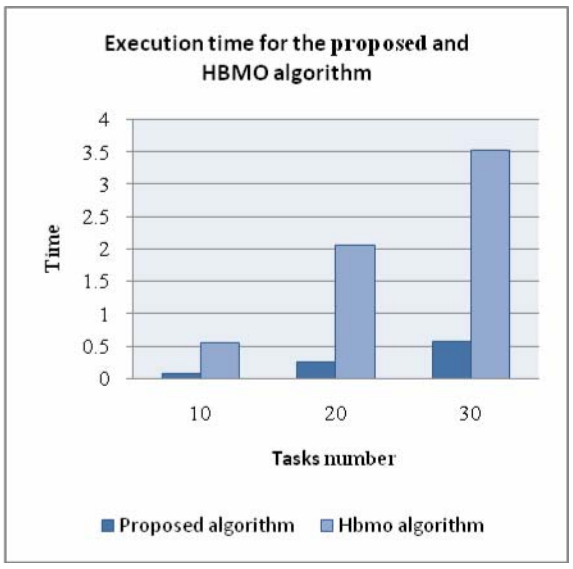

Figure 7 the execution time obtained by the proposed and HBMO algorithm for the tasks (10, 20, 30)

Fig. 8, 9, and 10 show the allocation of the tasks (10, 20, 30) with task interaction density (D=0.5) on 8 processors.

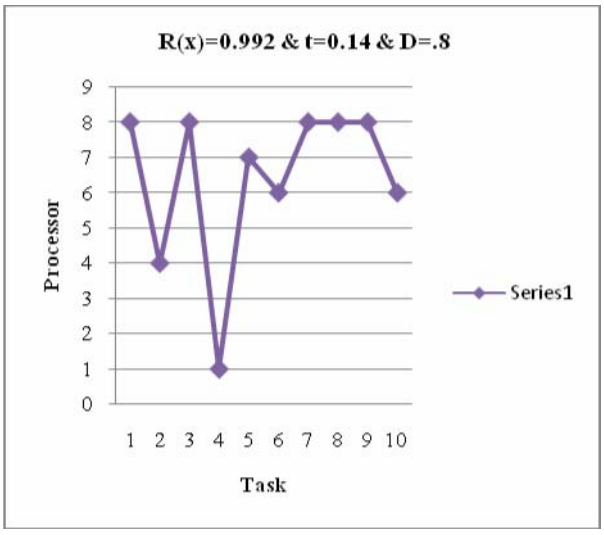

Figure 8 task allocations of 10 tasks on 8 processors

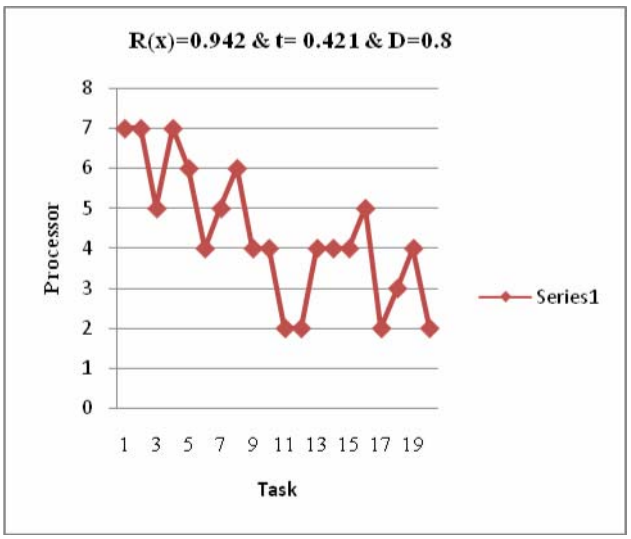

Figure 9 task allocation of 20 tasks on 8 processors

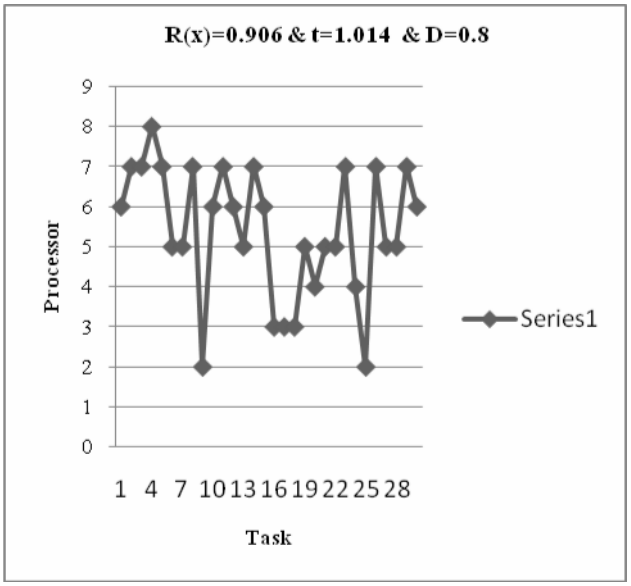

Figure 10 task allocations of 30 tasks on 8 processors

VII. CONCLUSION

In this paper, we have proposed a genetic algorithm which maximizes the distributed system reliability (DSR) of executing successfully a task consisting of several modules. The performance of the proposed algorithm is evaluated in comparison with HBMO algorithm, [17] for a number of randomly generated mapping problem instances. The results showed that

the solution quality of the proposed algorithm is better than HBMO for all the test cases.

REFERENCES

[1] Kartik, S., Murthy, S.R., Task allocation algorithms for maximizing reliability of distributed computing systems. IEEE Transactions on Computers 46, 719–724, 1997.

[2] Srinivasan, S., Jha, N.K., Safety and reliability driven task allocation in distributed systems. IEEE Transactions on Parallel and Distributed Systems 10, 238–251, 1999.

[3] Kartik, S., Murthy, S.R., Improved task-allocation algorithms to maximize reliability of redundant distributed computing systems. IEEE Transactions on Reliability 44, 575–586, 1995.

[4] Hsieh, C.C., Hsieh, Y.C., Reliability and cost optimization in distributed computing systems. Computers and Operations Research 30, 1103–1109, 2003.

[5] Kumar, V.K.P., Raghavendra, C.S., Hariri, S., Distributed program reliability analysis. IEEE Transactions on Software Engineering 12, 42– 50, 1986.

[6] Shatz, S.M., Wang, J.P., Goto, M., Task allocation for maximizing reliability of distributed computer systems. IEEE Transactions on Computers 41, 1156–116, 1992.

[7] Lin, M.S., Chen, D.J., The computational complexity of the reliability problem on distributed systems. Information Processing Letters 64, 143–147. , 1997.

[8] Verma, A.K., Tamhankar, M.T., Reliability-based optimal task allocation in distributed-database management systems. IEEE Transactions on Reliability 46, 452–459, 1997.

[9] Glover, F., 1989. Tabu search – Part I. ORSA Journal of Computing 1, 190–206.

[10] Dorigo, M., Gambardella, L., 1997. Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Transaction on Evolutionary Computation 1, 53–66.

[11] Kennedy, J., Eberhart, R.C., Particle swarm optimization. Proceedings of the IEEE International Conference on Neural Networks IV, 1942– 1948, 1995.

[12] Shigenori, N., Takamu, G., Toshiku, Y., Yoshikazu, F., A hybrid particle swarm optimization for distribution state estimation. IEEE Transaction on Power Systems 18, 60–68, 2003.

[13] Vidyarthi, D.P., Tripathi, A.K., Maximizing reliability of distributed computing system with task allocation using simple genetic algorithm. Journal of Systems Architecture 47, 549–554, 2001.

[14] Goldberg, D.E., 1989. Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Reading, MA.

[15] Peng-Yeng Yin *, Shiuh-Sheng Yu, Pei-Pei Wang, Yi-Te Wang, Task allocation for maximizing reliability of a distributed system using hybrid particle swarm optimization, The Journal of Systems and Software 80 , 724–735, 2007.

[16] Peng-Yeng Yin *, Shiuh-Sheng Yu, Pei-Pei Wang, Yi-Te Wang, Multi-objective task allocation in distributed computing systems by hybrid particle swarm optimization, Applied Mathematics and Computation 184, 407– 420, 2007.

[17] Qin- Ma Kang, Hong He,..etc, Task allocation for maximizing reliability of distributed computing systems using honeybee mating optimization, The journal of Systems and Software 83, 2165-2174, 2010.

[18] Hsieh, C.C., Optimal task allocation and hardware redundancy policies in distributed computing systems. European Journal of Operational Research 147, 430–447, 2003.

[19] Gamal Attiya, Yskandar Haam, Task allocation for maximizing reliability of distributed systems: A simulated annealing approach, J. Parallel Distrib. Comput. 66, 1259-1266, 2006.