Static Workload Distribution of Parallel Applications in Heterogeneous Distributed Computing Systems with Memory and Communication Capacity Constraints

Marwa Shouman
Department of Computer Science and Engineering,
Faculty of Electronic Eng.,
Minufiya University, Egypt.

Gamal Attiya
Department of Computer Science and Engineering,
Faculty of Electronic Eng.,
Minufiya University, Egypt.

Ibrahim Z. Morsi
Department of Electrical Engineering,
Faculty of Engineering,
Minufiya University, Egypt.

ABSTRACT
This paper addresses the problem of static load balancing in heterogeneous distributed computing systems taking into account both memory and communication capacity constraints. The load balancing problem is first modeled as an optimization problem. Then, a heuristic approach, called Adaptive Genetic Algorithm (AGA), is proposed to solve the problem. The performance of the proposed algorithm is evaluated by simulation studies on randomly generated instances and the results are compared with that obtained by applying both the Genetic Algorithm (GA) and the Simulated Annealing (SA). Also, the qualities of the results are compared with the optimal solutions that obtained by applying the Brach-and-Bound (BB) algorithm.

General Terms
Distributed Computing, Allocation and Scheduling, Heuristics.

Keywords
Load Balancing, Mapping, Simulated Annealing, Genetic Algorithm, Heuristics.

1. INTRODUCTION
Distributed computing systems have become competitive in providing the power of super machines with only a small initial cost. Such system has further benefit of providing the industry with an easy and modular upgrade path to increase the power of the system by increasing the number of networked computers. A major problem arises with such system is how to balance the workload of a parallel application over the available computers of the system so as to minimize the application turnaround time. If this step is not done properly, an increase in the number of computers may actually result in a decrease of total throughput. This degradation is caused by what is commonly called the *'saturation effect'* which occurs due to heavy communication traffic incurred by data transfer between tasks that reside on separate computers.

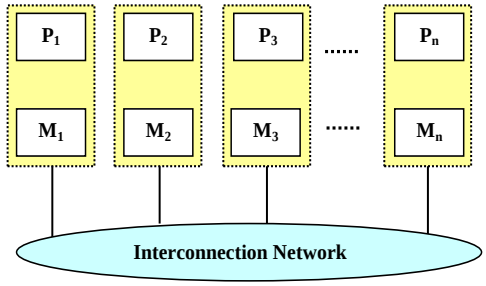
Several approaches have been suggested to solve the load balancing problem. They may be roughly classified into two main categories, namely, *exact algorithms* and *heuristic methods*. The exact algorithms may be developed using different strategies such as graph theory [1], state space search [2-4] and mathematical programming [5-7]. However, the exact algorithms are limited by the time required to obtain an optimal solution, where the time grows exponentially with the problem size. Thus, they are properly used for small problems. On the other hand, heuristic methods provide fast and effective means for obtaining suboptimal solutions. Different approaches may be used for building heuristics such as greedy heuristics [8], tabu search [9], genetic algorithm [10-15], simulated annealing [16-18] and clustering [19]. Most of the existing approaches however deal with homogenous systems or heterogeneous systems without considering different types of constraints that may be imposed by the system resources or the application tasks. The load balancing problem becomes more complicated when the system contains heterogeneous components such as different processors with different speeds and different resources such as memory and communication capacities.

This paper tackles the load balancing problem in heterogeneous distributed computing systems taking into account both memory and communication capacity constraints. It first models the load balancing problem as an optimization problem. It then presents a modified genetic algorithm, called Adaptive Genetic Algorithm (AGA) that adapts the mutation rate of the well known genetic algorithm, to solve the problem in less computation time. The performance of the proposed algorithm is evaluated by simulation studies on randomly generated instances and the results are compared with that obtained by applying two heuristic approaches; Genetic Algorithm (GA) and Simulated Annealing (SA). Also, the qualities of the results are compared with the optimal solutions that obtained by applying the Brach-and-Bound (BB) algorithm.

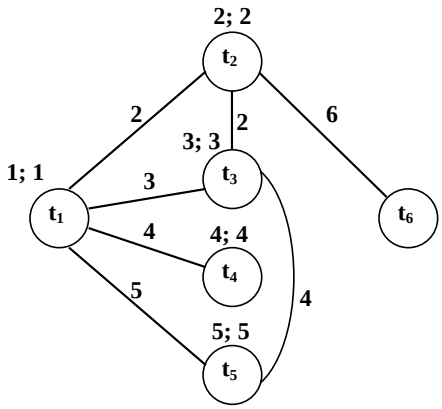
The remainder of this paper is organized as follows; Section 2 illustrates the load balancing problem while Section 3 presents a mathematical model for the problem. The proposed algorithm is described in Section 4 and the performance evaluation is presented and discussed in Section 5. Finally, the paper conclusions are given in Section 6.

2. LOAD BALANCING PROBLEM
A distributed computing system consists of a set of *N* heterogeneous computers interconnected by an interconnection network, as shown in Figure 1(a). Each computer has some computational facilities and a local memory. Furthermore, the interconnection network has communication capacity and cost of transferring a

data unit from one computer to another. A parallel application, on the other hand, consists of a set of *M* communicating tasks as shown in Figure 1(b). Each task requires some computational facilities and communicating tasks require some communication capacity. The load balancing problem is concerned with balancing the workload distribution of the parallel application tasks onto processors of the distributed computing system to minimize the application turnaround time. The workload distribution must be done such that the requirements of tasks are met and the availability of the system resources is not violated.



(a) Distributed Computing System



(b) Parallel Application

Figure 1: Distributed System and Parallel Application

3. PROBLEM FORMULATION

Formulating a mathematical model to the load balancing problem involves two steps:

1. Formulate a cost function to represent the main objective.
2. Formulate some constraints or inequalities in terms of both the tasks requirements and the availability of the system resources.

To do so, let X be an M x N binary matrix corresponding to an assignment of M tasks onto N processors such that

$$X_{ip} = \begin{cases} 1 & \text{if task } i \text{ is assigned to processor } p \\ 0 & \text{Otherwise} \end{cases}$$

3.1 Cost Function

The objective is to balance the workload over all processors in the system so as to minimize the completion time of the entire application. For an assignment X, a processor load comprises of the execution time and the communication time associated with tasks that are assigned to it. The required time by the heaviest loaded processor determines the entire application turnaround time. Therefore, balancing the workload may be achieved by minimizing the load at the maximum loaded processor.

The total workload (L_p) at processor p is defined as the sum of the total execution time ($EXEC_p$) and communication time ($COMM_p$) corresponding to the tasks that are assigned to the processor p. Hence, the workload at processor p may be formulated as:

$$L_p = EXEC_P + COMM_p$$

Where,

$$EXECp = \sum_i C_{ip} X_{ip} \text{ and } COMMp = \sum_{q \neq p} \sum_i \sum_{j \neq i} C_{ijpq} X_{ip} X_{jq}$$

C_{ip} is the cost of processing task i at processor p and C_{ijpq} is the cost of communication between task i and task j if i is assigned to processor p and j is assigned to processor q.

The maximum load (C_{max}) at the heaviest loaded processor may be formulated as

$$C_{max} = \max \{L_p \mid 1 \leq p \leq N\}$$

To minimize completion time, the cost at the heavy loaded processor should be minimized, i.e,

$$\min C_{max}$$

In other words

$$\min \max_p \left\{ \sum_i C_{ip} X_{ip} + \sum_{q \neq p} \sum_i \sum_{j \neq i} C_{ijpq} X_{ip} X_{jq} \right\}$$

This represents the main cost function to be optimized by the algorithms.

3.2 Constraints

During workload distribution, additional constraints should be considered to achieve the application requirements and validate the availability of the system resources**.**

Location constraints: Each task should be assigned to one and only one processor on which it is entirely executed without preemption. The following equality must hold at each task

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

Memory Constraints: The total memory required by all tasks assigned to a processor p must be less than or equal to the available memory capacity of the processor p. Let m_i denotes the amount of memory required for processing a task i and M_p defines the available memory at processor p, then the following inequality must hold at each processor p in the system:

$$\sum_i m_i \le M_p$$

Processing Constraints: The total processing time required by all tasks assigned to a processor p must be less than or equal to the available computational time of processor p. Let p_i denotes the processing time requirements of a task i and p_p denotes the available processing time of processor p, then the following inequality must hold at each processor p in the system

$$\sum_i p_i \le P_p$$

Communication Capacity Constraints: The total communication capacity required by all edges mapped to a communication path/link *pq* must be less than or equal to the available communication capacity of the path *pq*. Let b_{ij} denotes the amount of communication capacity required to communicate data between tasks i and j residing at different processors p and q respectively, and A_{pq} denotes the avai1able communication capacity of the path/link pq. Then, the following inequality must hold at each communication path/link pq.

$$\sum_i \sum_{j \ne i} b_{ij} \le A_{pq}$$

3.3 Mathematical Model

From sections 3.1 and 3.2, the load balancing problem may be formulated as:

$$minmax_p \left\{ \sum_i C_{ip} X_{ip} + \sum_{q \ne p} \sum_i \sum_{j \ne i} C_{ijpq} X_{ip} X_{jq} \right\}$$

Subject to

$$\sum_p X_{ip} = 1 \qquad \forall \text{ tasksi}$$

$$\sum_i m_i \le M_p \qquad \forall \text{ processorsp}$$

$$\sum_i p_i \le P_p \qquad \forall \text{ processorsp}$$

$$\sum_i \sum_{j \ne i} b_{ij} \le A_{pq} \qquad \forall \text{ pathspq}$$

In this model, the cost function is formulated to minimize the maximum cost at a bottleneck processor p, where, the cost is the total time required for the processor p to process all the tasks assigned to it and to communicate with other processor q. Also, several constraints are incorporated into the model so as to achieve the application requirements and validate the availability of the system resources.

4. PROPOSED ALGORITHM

The proposed algorithm is basically based on modifying the simple genetic algorithm so as to obtain good quality solutions in less computational time. In the following, we first present the simple genetic algorithm and then present our modifications to it.

4.1 Simple Genetic Algorithm

The general flowchart of the Simple Genetic Algorithm (SGA) is shown in Figure 2. The algorithm starts by generating an initial population of random candidate solutions. For load balancing problem, each individual in the population represents a random assignment of the application tasks onto the processors of the distributed system. Each individual is then awarded a score based on its performance. The individuals with the best scores are chosen to be parents. The parents are cut and spliced together using crossover to make children. The generated children are mutated based on a mutation rate σ, then scored, and the best individuals are chosen to be parents of the next generation. At some point the process is terminated and the best scored individual in the population is taken as the final result.

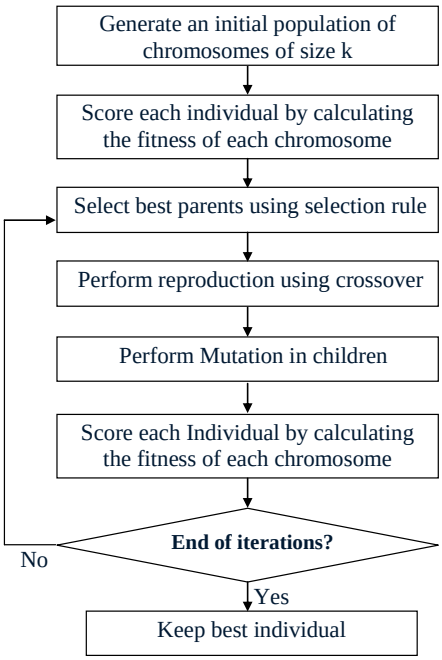


Figure 2: Simple Genetic Algorithm.

4.2 Adaptive Genetic Algorithm

In the simple genetic algorithm the mutation rate σ is considered to be constant over all generations. The main idea of the proposed modifications is to adapt the mutation rate parameter σ dynamically based on the search process to maximize relative improvement. A well-known example of this type of parameter adaptation is the "***1/5 success rule***" in (1+1) evolution strategies [20]. This rule states that the ratio of successful mutations (*a mutation is called successful if it produces an offspring that is better than the parents*) to all mutations should be 1/5. Hence, if the ratio is greater than 1/5 then the step size (i.e., mutation rate σ) should be increased, and if the ratio is less than 1/5, the step size should be decreased.

To describe the mutation rate parameter adaptation, let p_s be the relative frequency of successful mutations measured over some n number of generations, t is the current generation number varying from 0 to total number of generations, and c is

constant value $0.817 \leq c \leq 1$ [20]. Thus, the σ parameter adaptation is:

if (t mod n = 0) then

$$\sigma(t) = \begin{cases} \sigma(t-n)/c & if \quad p_s > 1/5 \\ \sigma(t-n).c & if \quad p_s < 1/5 \\ \sigma(t-n) & if \quad p_s = 1/5 \end{cases}$$

Else

σ **(t) =** σ **(t-1);**

By using this mechanism, changes in the parameter value are now based on the feedback from the search process, and the σ -adaptation happens every n generations. If n =1, this means that the σ -adaptation happens with each generation.

Figure 3 shows the interaction of the σ -adaptation mechanism with the simple genetic algorithm. The modified algorithm is called Adaptive genetic Algorithm (AGA).

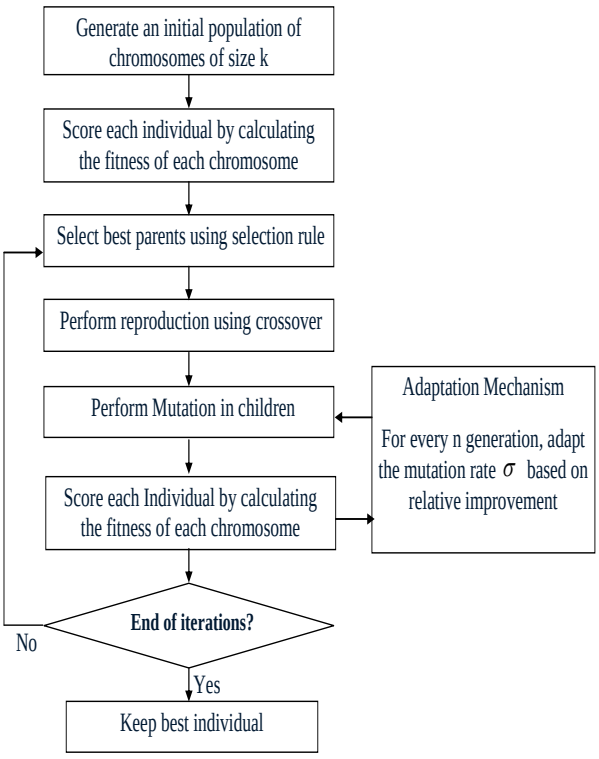


Figure 3: Adaptive Genetic Algorithm

4.2.1 Population Encoding

An initial population of size K chromosomes is randomly generated. Each chromosome in the population consists of M genes (equivalent to the same M number of tasks) and each gene is represented by a number r such that $1 \leq r \leq N$. Where, N is the number of processors. The generated chromosomes are then scored and evolved for a specified number of generations.

4.2.2 Selection

At each iteration, S pairs of parents are chosen. Each pairs produces two children using crossover, so, 2S children are created. Mutations are then applied and the children are scored using fitness function. From the (K + 2S) individuals (original population plus the created children), the best scored K chromosomes are selected for the next generation, and the process repeated. The final distribution is taken as the fit individual (exhibiting the lowest cost) after specified number of iterations.

4.2.3 Crossover

Crossover refers to the mixing of information from both parents to create children. In load balancing, the crossover process implies a child will receive the left side from the first parent and the right side from the second parent. The second child receives the complement parts that not taken by the first child, i.e., it receives the right side from the first parent and the left side from the second parent, with the same crossover location. Where, the crossover location is randomly selected.

4.2.4 Mutations

Following the creation of children by the crossovers, mutations are applied to the children. In load balancing, a mutation means replacing a distributed task from one processor to another processor by a randomly chosen task and a randomly chosen processor.

5. PERFORMANCE EVALUATION

The proposed algorithm is coded in *Matlab* and evaluated for a large number of randomly generated instances that being mapped into a distributed system of N computers with bus topology. The qualities of the results are compared with those obtained by using the Branch-and-Bound (BB) algorithm [7] which applied on the same instances. In the following subsections, the test conditions for applying SA [15], SGA and AGA to solve the load balancing problem are first given and then some simulation results are discussed.

5.1 Test Conditions

In applying the SA algorithm [15], a neighboring solution is obtained by choosing a task randomly and assigns it to another randomly selected processor p. For the cooling process, a geometric cooling schedule is used. An initial temperature T is set after executing a sufficiently large number of random moves such that the worst move would be allowed. The temperature is reduced so that $T = \alpha \times T$, where α=0.90. At each temperature, the chain n_{rep} is updated in a similar manner: $n_{rep} = \beta \times n_{rep}$, where β= 1.05. The stopping criterion is to reach the last iteration.

In applying SGA, each chromosome consists of M genes. Each gene is represented by a number r such that $1 \leq r \leq N$. Where, M is the number of tasks and N is the number of processors. An initial population of size K=100 chromosomes is created. Consequently, the population is ranked and the best 50% of chromosomes are chosen to go forward to the next generation. Crossover is performed on all the selected chromosomes by selecting random pairs using Roulette Wheel method without replacement. The mutation probability is chosen to be 0.1 and C_{max} is used as the evaluation (fitness) function.

In applying AGA, an initial population of size K=100 chromosomes is randomly generated. For each iteration, a set of 15 pairs of parents are chosen by tournament selection, whereby each parent is selected as the best of 5 randomly chosen chromosomes from the best 10 candidates. Crossover is performed on all the selected parents by selecting random pairs using Roulette Wheel method. So, the 15 pairs of parents create 30 children. Mutation is done based on the adapted mutation ratio and the individuals are scored using C_{max} as the evaluation (fitness) function. Over these 130 chromosomes (original population of 100 plus the 30 children), the 100 best scoring chromosomes are retained for the next generation. The mutation rate is adapted with each generation (i.e., n=1).

5.2 Simulation Results

Figures 4 and 5 show the simulation results of assigning randomly generated instances onto a distributed computing system of 4 computers with LAN topology.

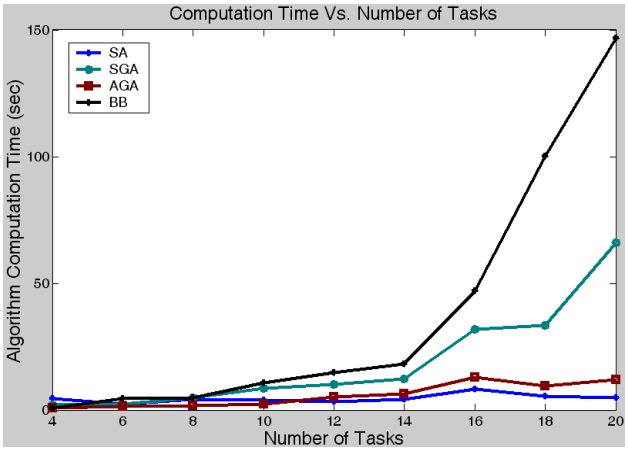


Figure 4: Computation Time of different Algorithms

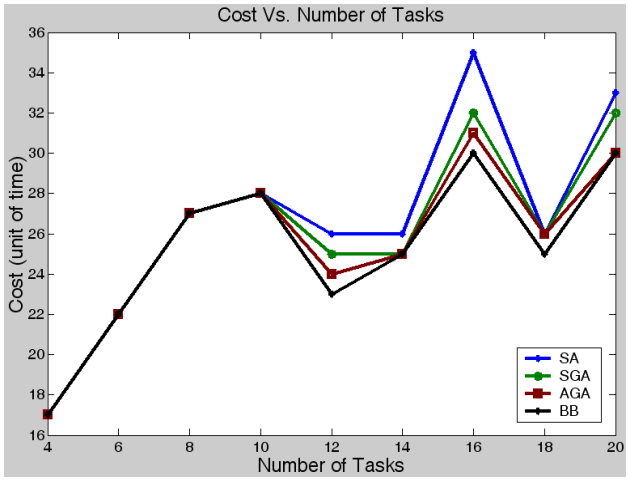


Figure 5: Turnaround time of the entire application

Figure 4 shows the computation time of the SA, SGA and AGA algorithms as a function of the number of tasks. From the figure, the SA finds a solution very fast compared with the SGA but has a comparable computation time with applying AGA. Figure 5 shows the quality of the results that obtained by the SA, SGA, and AGA in comparing with the optimal solution obtained by the BB algorithms [7]. It is clear that the quality of solutions that obtained by the AGA are better than that obtained by both the GA and the SA.

Figures 6, 7 and 8 show the simulation results of assigning other randomly generated instances onto a distributed computing system of 5 computers with LAN topology. Figure 6 shows the computation time of the SA, SGA and AGA algorithms as a function of the number of tasks for instances up to 50 tasks. Figure 7 shows the quality of solutions that are obtained by applying SA, SGA and AGA on the same instances. It is clear that AGA has the best results compared to SA and SGA. Figure 8 shows the workload distribution onto the 5 processors by applying SA, SGA, and AGA algorithms respectively. As shown in the figure, the workload distribution resulting by using AGA algorithm is better than that obtained by using SA and GA.

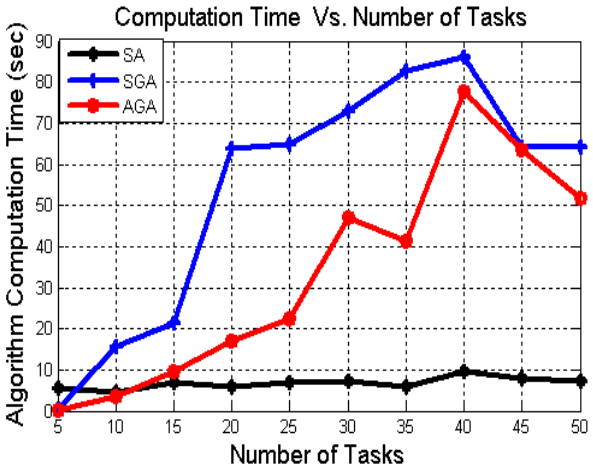


Figure 6: Computation Time of different Algorithms

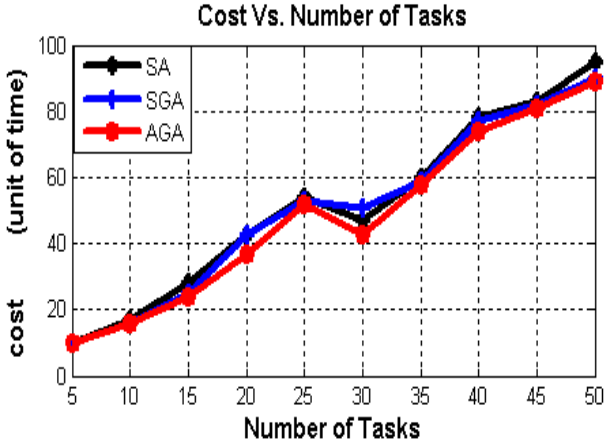
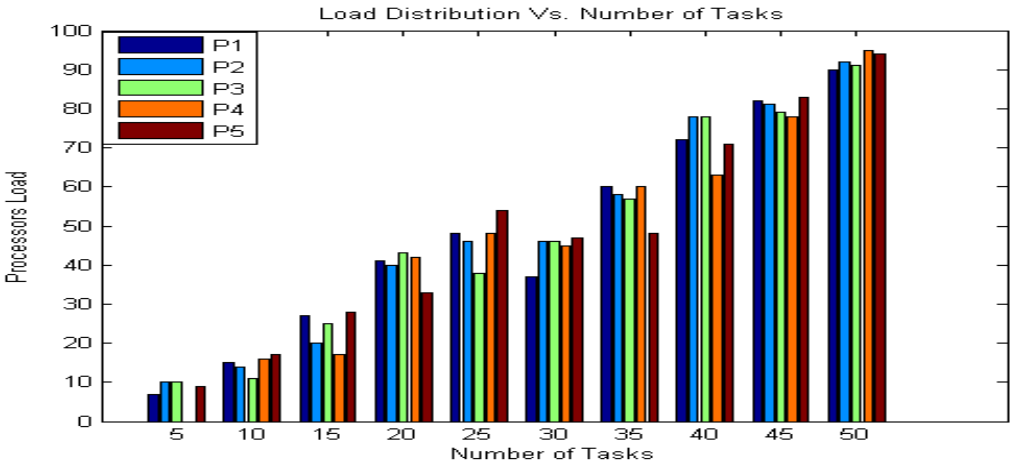
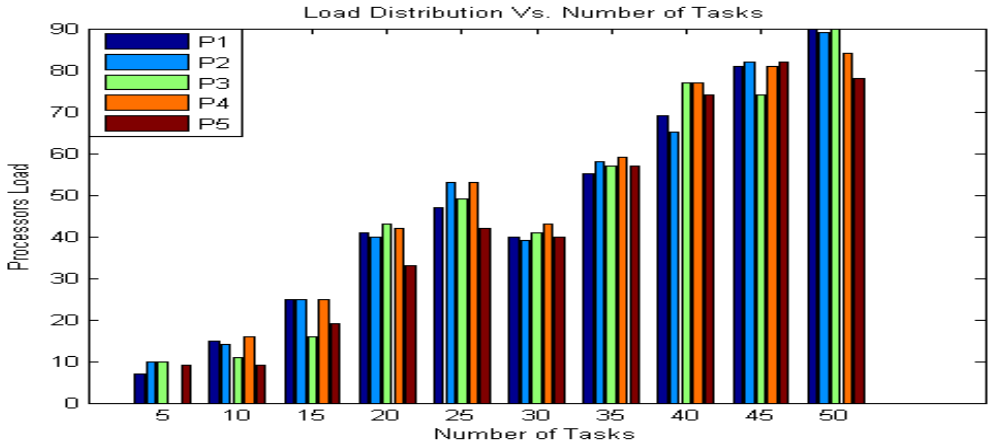


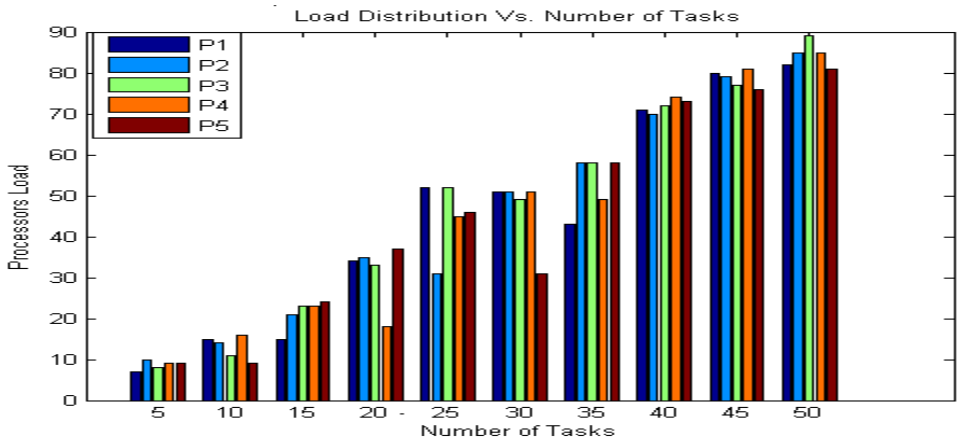
Figure 7: Turnaround Time of the entire application



(a) Workload Distribution using SA



(b) Workload Distribution using SGA



(c) Workload Distribution using AGA

Figure 8: Workload Distribution

6. CONCLUSIONS

In this paper, the load balancing problem is modeled as an optimization problem and an Adaptive Genetic Algorithm (AGA) is developed to solve the problem. The algorithm is based on adapting the mutation rate parameter (σ) of the well known Genetic Algorithm (GA) so as to solve load balancing problem in less computation time than that required by the GA. The proposed algorithm is tested and evaluated for a large number of randomly generated task graphs that being assigned to processors of a distributed computing system. The results shown that, the proposed algorithm provides high quality solutions in comparing with both the SA and the GA.

7. REFERENCES

[1] C.-C. Hui and S. T. Chanson, "Allocating Task Interaction Graph to Processors in Heterogeneous Networks," IEEE Transactions on Parallel and Distributed Systems, Vol. 8, No. 9, pp. 908-925, September 1997.

[2] M. Kafil and I. Ahmed "Optimal Task Assignment in Heterogeneous Distributed Computing Systems," IEEE Concurrency, Vol.6, No.3, pp.42-51, July-September 1998.

[3] A. Tom and C. S. R. Murthy "Optimal task allocation in distributed systems by graph matching and state space search," J. of Systems and Software, Vol. 46, No. 1, pp. 59–75, April 1999.

[4] Nirmeen A. Bahnasawy, Gamal M. Attiya, Mervat Mosa and Magdy A. Koutb, "A Modified A* Algorithm for Allocating Tasks in Heterogeneous Distributed Computing Systems" International Journal of Computing, Vol. 8, Issue 2, pp. 50-57, 2009.

[5] Y.-C. Ma and C.-P. Chung, "A Dominance Relation Enhanced Branch-and-Bound Task Allocation," J. of Systems and Software, Vol. 58, No. 2, pp.125-134, Sep. 2001

[6] G. Attiya and Y. Hamam "Static Task Assignment in Distributed Computing Systems," A book chapter in "Information processing: Recent Mathematical Advances in Optimization and Control", Chapter XIX, pages 241-258. Presses de l'Ecole des Mines de Paris, Paris, 2005, ISBN: 2911762568.

[7] G. Attiya and Y. Hamam. "Optimal Allocation of Tasks onto Networked Heterogeneous Computers using minimax Criterion," International Network Optimization Conference (INOC'03), pp. 25-30, Evry/Paris, France, 2003.

[8] P. Bourvy, J. Chassin, M. dobruck, L.Hluch , E. Luque, and T. Margalef, "Mapping and Load Balancing on Distributed Memory Systems," Proceedings of the Eight Symposium on Microcomputer and Microprocessor Applications, Vol. 1, pp. 315-324,1994.

[9] P. Bouvry, J. Chassin, and D. Trystram, "Efficient Solution for Mapping Parallel Programs," Proceedings of EuroPar'95, LNCS: 379-390, August 1995.

[10] L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski, "Task Matching and Scheduling in Heterogeneous Computing Environments Using a Genetic-Algorithm-Based Approach," J. of Parallel and Dist. Computing, vol.47, pp.8–22, 1997.

[11] J. Aguilar and E. Gelenbe, "Task Assignment and Transaction Clustering Heuristics for Distributed Systems," Information Sciences, Vol. 97, No.1-2, pp.199–219, March 1997.

[12] M. H. Zaharia, F. Leon, D. Gâlea, "Parallel Genetic Algorithms for Cluster Load Balancing," Proceedings ECIT2004 - Third European Conference on Intelligent Systems and Technologies, Iasi, Romania, July 21-23, 2004.

[13] Bibhudatta Sahoo, Sudipta Mohapatra, and Sanjay Kumar Jena, "A Genetic Algorithm Based Dynamic Load Balancing Scheme for Heterogeneous Distributed Systems" Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2008), Las Vegas, Nevada, USA, July 14-17, 2008. ISBN: 1-60132-084-1

[14] Kamaljit Kaur, Amit Chabra, and Gurvinder Singh, "Modified Genetic Algorithm for Task Scheduling in Homogenous Parallel System using Heuristics," International Journal of Soft Computing 5 (2), pp. 42-51, 2010. ISSN: 1816-9503.

[15] M.shouman, G.M.Attiya, and I.Z.Morsi, "Two Heuristic Approaches for Mapping Parallel Application on Distributed Computing Systems" Menoufia Journal of Electronic Engineering Research (MJEER), Vol. 18, no. 2, pp. 85-98, July 2008.

[16] Y. Hamam and K.S. Hindi, "Assignment of Program Modules to Processors: A Simulated Annealing Approach," European Journal of Operational Research, Vol. 122, No. 2, pp.509-513, April 2000.

[17] Gamal Attiya and Yskandar HAMAM, "Task Allocation for maximizing reliability of distributed Systems: A Simulated Annealing Approach", Journal of parallel and distributed computing (JPDC), Vol. 66, No. 10, pp. 1259-1266, October 2006.

[18] G. Attiya and Y. Hamam, "Two Phase Algorithm for Load Balancing in Heterogeneous Distributed Systems," IEEE Proceedings of 12th Euromicro Conference on Parallel, Distributed and Network based Processing (PDP2004), pp. 434-439, A Coruna, Spain, Feb. 11-13, 2004.

[19] M. A. Senar, A. Ripoll, A. Corts, E. Luque, "Clustering and Reassignment–Based Mapping Strategy for Message-Passing Architectures," Journal of System Architecture, Vol. 48, No. 8-10, pp.267-283, March 2003.

[20] Á. E. Eiben, R. Hinterding, and Z. Michalewicz, "Parameter control in evolutionary algorithms," IEEE Trans. Evol. Comput., vol. 3, pp. 124-141, Jul. 1999.