PARALLEL MIN-MAX ANT COLONY SYSTEM (MMAS) FOR DYNAMIC PROCESS SCHEDULING IN DISTRIBUTED OPERATING SYSTEMS CONSIDERING LOAD BALANCING

M. Nikravan and M. H. Kashani
Department of Electrical Computer
Islamic Azad University, Shahriar Shahreqods Branch
Tehran, Iran
E-mail: moh.nikravan@gmail.com,mh.kashani@gmail.com

KEY WORDS
Distributed Systems, Scheduling, Ant Colony, Optimization, Load Balancing.

ABSTRACT

This paper presents and evaluates a new method for process scheduling in distributed systems .Scheduling in distributed operating systems has a significant role in overall system performance and throughput. An efficient scheduling is vital for system performance .The scheduling in distributed systems is known as an NP-complete problem, even in the best conditions, and methods based on heuristic search have been proposed to obtain optimal and suboptimal solutions. In this paper, we proposed an Ant-based algorithm to solve this problem considering dynamic load balancing efficiently. We evaluate the performance and efficiency of the proposed algorithm using simulation results.

INTRODUCTION

Scheduling in distributed operating systems is a critical factor in overall system efficiency. A Distributed computing system (DCS) comprising a set of Computers (Processors) connected to each other by communication networks. Process scheduling in a distributed operating system can be stated as allocating processes to processors so that total execution time will be minimized, utilization of processors will be maximized, and load balancing will be maximized. Process scheduling in a distributed system is done in two phases: in the first phase processes are distributed on computers and in the second, processes execution order on each processor must be determined. Process scheduling in distributed systems has known to be NP-complete.
Several methods have been proposed to solve scheduling problem in DCS. The proposed methods can be generally classified into three categories: Graph-theory-based approaches (Shen and Tsai 1985), mathematical models-based methods (Ma et al. 1982), and heuristic Techniques (Park 2004), (Park and Choe 2002), (Woodside and Monforton 1993), (Sarje and Sagar 1991). Heuristics can obtain suboptimal solution in ordinary situations and optimal solution in particulars. Since the scheduling problem has

Known to be NP-complete, using heuristic Techniques can solve this problem more efficiently. Three most well-known heuristics are the iterative improvement algorithms (Lin and Yang 1999), the probabilistic optimization algorithms, and the constructive heuristics. In the probabilistic optimization group, GA-based methods (Lin and Yang 1999), (Martino 2002), (Martino and Mililotti 2003), (Moor 2003), (Oh and Wu 2004), (Wang and Korfhage 1995), (Zomaya et al. 1999) and simulated annealing (Salleh and Zomaya 1999) are considerable which extensively have been proposed in the literature.
One of the crucial aspects of the scheduling problem is load balancing. While recently created processes randomly arrive into the system, some processors may be overloaded heavily while the others are under loaded or idle. The main objectives of load balancing are to spread load on processors equally, maximizing processors utilization and minimizing total execution time (Salleh and Zomaya 1999). In dynamic load balancing, processes must be dynamically allocated to processors in arrival time and obtain a near optimal schedule, therefore the execution of the dynamic load balancing algorithm should not take long to arrive at a decision to make rapid process assignments. (Lan and Yu 1995) have proposed scheduling algorithms considering load balancing.
The ACO meta-heuristic was first described by Dorigo and was inspired by the ability of real ant colonies to efficiently organize the foraging behavior of the colony using external chemical *pheromone* trails acting as a means of communication. The ants deposit on the ground *pheromone* while traveling. Upon arrival at an intersection, ants make their choice of the path to follow according to a probability that is biased by the quantity of pheromones on each trail (Stutzle and Hoos 2000). Ant-based algorithms have emerged as powerful tools to solve NP-complete constrained optimization problems.
In this paper we have applied the relatively new meta-heuristic ant colony optimization (ACO) to efficiently solve this problem considering dynamic load balancing. In The proposed algorithm each ant starts with the set of

unscheduled processes and iteratively builds a complete solution (schedule) that shows the execution order of all existing unscheduled processes on processors. The fittest solutions are solutions which their corresponding schedules have less total execution time and communication cost, better load-balance and processor utilization. When a new solution is generated, before pheromone trails are updated a local search is applied on it to improve the quality of solution, if it is possible. We assume that the distributed system is not uniform and not preemptive, that is, the processors may be different, and a processor completes current process before executing a new one. The load-balancing mechanism used in this paper only schedule processes without process migration and is centralized.

PROBLEM DESCRIPTION AND FORMULATION

In order to schedule the processes in a distributed system, we should know the information about the input processes and distributed system itself such as: Network topology, processors speed, communication channels speed and so on. Since we study a deterministic model, a distributed system with *m* processors, *m* > 1 should be modeled as follows:

- P $=\{p_1, p_2, p_3, ..., p_m\}$ is the set of processors in the distributed system. Each processor can only execute one process at each moment, a processor completes current process before executing a new one, and a process can not be moved to another processor during execution. R is an $m \times m$ matrix, where the element r_{uv} $1 \le u, v \le m$ of R, is the communication delay rate between p_u and p_v. H is an $m \times m$ matrix, where the element h_{uv} $1 \le u, v \le m$ of H, is the time required to transmit a unit of data from p_u to p_v. It is obvious that $h_{uu} = 0$ and $r_{uu} = 0$.
- $T = \{t_1, t_2, t_3, ..., t_n\}$ is the set of processes to execute. A is an $n \times m$ matrix, where the element a_{ij} $1 \le i \le n$, $1 \le j \le m$ of A, is the execution time of process t_i on processor p_j .In homogeneous distributed systems the execution time of an individual process t_i on all processors is equal, that means : $1 \le i \le n$ $a_{i1} = a_{i2} = ... = a_{im}$. D is a linear matrix, where the element d_i $1 \le i \le n$ of D, is the data volume for process t_i to be transmitted, when process t_i is to be executed on a remote processor.
- F is a linear matrix, where the element f_i $1 \le i \le n$ of F, is the target processor that is selected for process t_i to be executed on. C is a linear matrix, where the element c_i $1 \le i \le n$ of C, is the processor that the process t_i is presented on just now.

The problem of process scheduling is to assign for each process $t_i \in T$ a processor $f_i \in P$ so that total execution time will be minimized, utilization of processors will be maximized, and load balancing will be maximized. In such systems there are finite numbers of processes, each having a process number and a execution time and placed in a process pool from which processes are assigned to processors. The main objective is to find a schedule with minimum cost. The following definitions are also needed:

Definition 1

The processor load for each processor is the sum of processes execution times allocated to that processor. However, as the processors may not always be idle when a schedule is evaluated, the current existing load on individual processors must also be taken into account, therefore (1):

$$Load(p_i) = \sum_{j=1}^{\substack{No.\,of\ allocated\\ processes\ on\\ processor\ i}} a_{j,i} + \sum_{k=1}^{\substack{No.\,of\ New\ Assigned\\ processes\ to\\ processor.\,i}} a_{k,i} \tag{1}$$

Definition 2

The length or *make span* of a schedule T is the maximal finishing time of all processes or maximum load. Also communication cost (CC) to spread recently created processes on processors and the Completion Time (ct) of a process t_i on processor p_j must be computed (2,3,4):

$$\max span(T) = \max(Load(p_i)) \quad \forall\, 1 \le i \le Number\ of\ \Pr ocessors \tag{2}$$

$$CC(T) = \sum_{i=1}^{number\ of\ new\ processes} \left(r_{c_i f_i} + h_{c_i f_i} \times d_i\right) \tag{3}$$

$$ct_{ij} = a_{ij} + \sum_{k=1}^{\substack{number\ of\ processes\\ in\ processor\ j's\ queue}} a_{kj} \tag{4}$$

Definition 3

The Processor utilization for each processor is obtained by dividing sum of processing times by maxspan, and the Average of processors utilization is obtained by dividing sum of all utilization by number of processors (5, 6):

$$U(p_i) = {Load(p_i)}\big/{\max span} \tag{5}$$

$$AveU = \left(\sum_{i=1}^{No\,of\ processors} U(p_i)\right) \Big/ Number\,Of\,Processors \quad (6)$$

Definition 4

Number of Acceptable Processor Queues (NoAPQ): We must define thresholds for light and heavy load on processors. If the processes completion time of a processor (by adding the current system load and those contributed by the new processes) is within the light and heavy thresholds, this processor queue will be acceptable. If it is above the heavy threshold or below the light-threshold, then it is unacceptable, but what is important is average of number of acceptable processors queues, which is achievable by (7):

$$AveNoAPQ = NoAPQ \big/ Number\,Of\,Processors \quad (7)$$

Definition 5

A Queue associated with every processor, shows the processes that processor has to execute.

THE PROPOSED ANT-BASED ALGORITHM

Pheromone Trail Defining

As discussed before, the process scheduling problem can be stated as allocating processes on processors in a way that satisfies the mentioned objectives. Therefore the pheromone value $\tau_k(t_i, p_j)$ was selected to represent the desirability of allocating process t_i on processor p_j in *k*' th iteration. We define a process-processor pheromone trail matrix which each process-processor pair has a single entry in the matrix. In MMAS we initialize the pheromone trails in such a way that after the first iteration all pheromone trails correspond to $\tau_{\max}(1)$. This can easily be achieved by initially setting τ_0 to some arbitrarily high value. After the first iteration of MMAS, the trails will be forced to take values within the imposed bounds; in particular, they will be set to $\tau_{\max}(1)$. This type of trail initialization is chosen to increase the exploration of solutions during the first iterations of the algorithm.

The Heuristic

The heuristic used here to select the next process to be scheduled is a modification to what is presented in (Sarje and Sagar 1991); this heuristic is called 'Relative Cost' (RC). According to our objectives in scheduling, it is better to consider two factors, when a process is allocated to a processor. First, matching, that means a process should be allocated to a processor that will completes it fastest possible. Second, load-balancing that means load should be balanced over all processors. In order to take into account both these factors … RC method defines to measures, *static relative cost (SRC) and dynamic relative cost (DRC)*. The SRC of a job processor pair (t_i, p_j) is simply the execution time of process t_i on processor p_j divided by the average execution time of process t_i on all processors, as shown in (8), and is fixed for the entire run. The DRC is re-calculated after each process is scheduled, and is the completion time, *c*t, of each process t_i on processor p_j, divided by the average *ct* of t_i on all processors, shown in (9).

$$SRC(t_i, p_j) = \frac{a_{ij}}{\left(\sum_{k=1}^{m} a_{ik}\right)/m} \quad (8)$$

$$DRC(t_i, p_j) = \frac{ct_{ij}}{\left(\sum_{k=1}^{m} ct_{ik}\right)/m} \quad (9)$$

The DRC and SRC and The best suitable processor are calculated for each unscheduled process each iteration. The best suitable processor for a process t_i is the processor p_j that maximizes SRC and DRC … and obtained as follows:

$$p_{best}(t_i) = \max_{\forall p_j}((\alpha \times SRC(t_i, p_j)) \times (\beta \times DRC(t_i, p_j))) \quad (10)$$

The heuristic use by the ants for each t_i is obtained as follows:

$$\eta(t_i) = (\alpha \times SRC(t_i, p_{best}(t_i))) \times (\beta \times DRC(t_i, p_{best}(t_i))) \quad (11)$$

Where these equations α and β are parameters used to control the effect of each value.

Fitness Function

As discussed before, the main objective of scheduling is to find a schedule with optimal cost while load balancing, processors utilization and cost of communication are considered. We take into account all objectives in following equation. The fitness of a Schedule T (12):

$$fitness(T) = \frac{(\gamma \times AveU) \times (\theta \times AveNoAPQ)}{(\alpha \times \max span(T)) \times (\beta \times CC(T))} \quad (12)$$

Which $0 < \alpha, \beta, \gamma, \theta \le 1$ are control parameters to control effect of each part according to special cases and their default value is one. This equation shows that a fitter solution (Schedule) has less make span, less communication cost, higher processor utilization and higher Average number of acceptable processor queues.

Updating The Pheromone Trail

As discussed before In MMAS only one single ant is used to update the pheromone trails in each iteration. Therefore, pheromone trails are updated as follows:

$$\tau_{k+1}(t_i, p_j) = \begin{cases} \rho \times \tau_k(t_i, p_j) + \Delta & \textit{if } t_i \textit{ is allocated on} \\ & \textit{processor } p_j \textit{ in } s^{ib} \\ \rho \times \tau_k(t_i, p_j) & \textit{otherwise} \end{cases}$$

Where $\Delta = \dfrac{f(s^{ib})}{f(s^{gb})}$ and $f(s)$ is the cost of solution *S*.

(s^{ib}) Is the *iteration-best* solution and (s^{gb}) is the *global-best* solution. It is to be noted that when a better solution than the *global-best* solution is found, the *global-best* solution is set to this solution. In MMAS search stagnation may occur. This can happen if at each choice point, the pheromone trail is significantly higher for one choice than for all the others. In such a situation the ants construct the same solution over and over again and the exploration of the search space stops. Obviously, such a stagnation situation should be avoided. By limiting the influence of the pheromone trails one can easily avoid the relative differences between the pheromone trails from becoming too extreme during the run of the algorithm. To achieve this goal, MMAS imposes explicit limits $\tau_{\min}$ and $\tau_{\max}$ on the minimum and maximum pheromone trails such that for all pheromone trails $\tau_k(t_i, p_j)$, $\tau_{\min} \le \tau_k(t_i, p_j) \le \tau_{\max}$. So after each pheromone updating if $\tau_k(t_i, p_j) < \tau_{\min}$ then we set $\tau_k(t_i, p_j) = \tau_{\min}$, and if $\tau_k(t_i, p_j) > \tau_{\max}$ then we set $\tau_k(t_i, p_j) = \tau_{\max}$. It is to be noted that always $\tau_{\min} > 0$.

Building a Solution

Each ant starts with an empty solution, and iteratively adds components to the solution until the solution is completed (all processes are scheduled).each iteration the next process to be scheduled is probabilistically selected according to heuristic value and pheromone trial information, and then the selected process is allocated to its best processor. The probability of process t_i to be selected is obtained as follows (13):

$$probability(t_i) = \frac{[\tau(t_i, p_{best}(t_i))]^{\alpha}.[\eta(t_i)]^{\beta}}{\sum\limits_{\substack{\forall unscheduled \\ task \quad t_k}} [\tau(t_k, p_{best}(t_k))]^{\alpha}.[\eta(t_k)]^{\beta}} \qquad (13)$$

Where these equations α and β are parameters used to control the effect of each value.

Local Search

When a complete solution is made by an ant before updating pheromone trails, a local search is applied on the solution to improve its quality, if it is possible. The local search is performed on every ant, every iteration, so it needs to be fairly fast. A simple approach is to check if any jobs could be swapped between processors which would result in a lower make span.

Mutation

In dynamic load balancing, processes must be dynamically allocated to processors in arrival time and obtain a near optimal schedule, therefore the execution of the dynamic load balancing algorithm should not take long to arrive at a decision to make rapid processes assignments. As ACO algorithms are population-based, each time that the algorithm has to schedule a new set of processes, a large amount of time may be consumed to construct ant colony and then ants' sequentially construct their solutions. Therefore a parallel version of this algorithm is more efficient. To run the ACO algorithm in parallel, we propose two models.

Message Passing Model

In this model we have one *Master* ant, and a number of *subordinate* ants. The master works as a coordinator. It sets the parameters and initializes the pheromone trails. Each subordinate ant is placed on an individual processor and is run on it. In start of each iteration the master sends the pheromone trail matrix to all subordinates, then building solutions by subordinate ants is done in parallel, finally solutions are returned to master. Master updates pheromone trails using solutions; this cycle is repeated until termination condition is met

Island Model

The island model is inspired from GA theory. In the island model, the whole colony is divided to a number of sub colonies. Each sub colony is placed on an individual processor. All colonies work in parallel after a defined number of iterations the colonies interchange the best local solutions they found. When a colony receives a solution that is better than its best solution, its best solution is set to the solution which received, and pheromone trails are updated according to new solution.

Termination Condition

We can apply multiple choices for termination condition. Max number of generation, algorithm convergence, equal fitness for fittest selected in respective iterations.

The Structure of Proposed GA-Based Algorithm

Our proposed Ant Colony-Based algorithm starts with a fixed number of ants. An individual ant constructs candidate solutions by starting with an empty solution and then iteratively adding solution components until a complete candidate solution is generated. A certain fitness function is used to evaluate the fitness of each solution. After the solution construction is completed, the ants give feedback on the solutions they have constructed by depositing pheromone on solution components which they have used in their solution. Typically, solution components which are part of better solutions or are used by many ants will receive a higher amount of pheromone, and hence, will more likely be used by the ants in future iterations of the algorithm. This process iterates until termination condition is satisfied, Figure 1.

```
Procedure ANT Scheduler
Begin
    Set parameters and initial pheromone trails;
    While (termination condition not met) do
        For i=1 to NOANTS do
            Cons. a Schedule $S_i$ and Local Search to
            improve $S_i$ ;
        End For;
        $s^{ib} = \left\{ s_j : f(s_j) = \max_{i=1}^{NOANTS} (f(s_i)) \right\}$ ;
        Update Trails Using $s^{gb}$ and $s^{ib}$ ;
        if $f(s^{ib}) > f(s^{gb})$ then $f(s^{gb}) = f(s^{ib})$;
    End While;
    Return The best solution found $s^{gb}$ ;
End
```

Figure 1: The Structure Of Proposed Algorithm

EXPERIMENTAL RESULTS

In this section, we have used the simulation results to show the performance of the proposed ACO based algorithm. Current solution techniques are concentrated on scheduling tasks with precedence constraints so our approach is not completely comparable with them. The parameter values used in ACO algorithms are often very important in getting good results, however the exact values are very often entirely problem dependent , and cannot always be derived from features of the problem itself. We have tried different values of the population size (*POPSIZE*), the extent to which pheromone information is used (α), the extent to which heuristic information is used (β), to find which values would steer the search towards the best solution. The best result achieved is as follows: $\alpha = 2$, $\beta = 20$, *NOANTS*=10,*NOGEN* =50, *m*=10 (number of processors), *n=100...900* (number of processes). Measurement of performance of these algorithms was based on three metrics: total completion time and average processor utilization. The default parameters were varied and the results collected from test runs were used to study the effects of changing these parameters.

Changing The Number of Processes

We have studied the effect of increasing number of processes on total completion time and average processor utilization. The Obtained results are shown in Figure 2.and, Figure 3. A considerable point in Figure 3 is that when number of processes is increased, higher utilization is obtained.

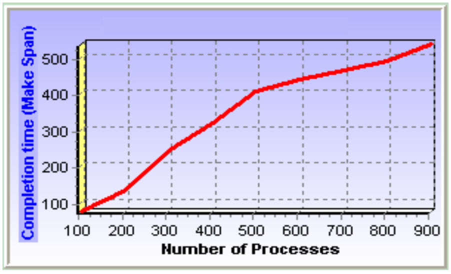

Figure 2: Total Completion Time

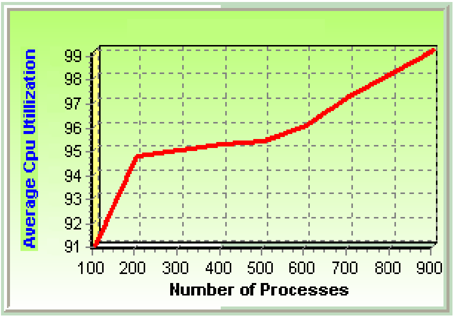

Figure 3: Average Processor Utilization

Changing The Number of Generations

When the number of generations was increased our proposed algorithm had a better function. The Obtained results are shown in Figure 4 and, Figure 5. While the number of generations was increased the total completion time was reduced, it is because that the components of fitter solutions are selected by more ants, so the amount of deposited pheromone on these components reinforced and, therefore they more likely will be selected in next generations. The result is that the quality of the generated process assignment improves after each generation. A considerable point in these figures is that when the number of generations was increased, due to its high convergence, higher utilization is obtained.

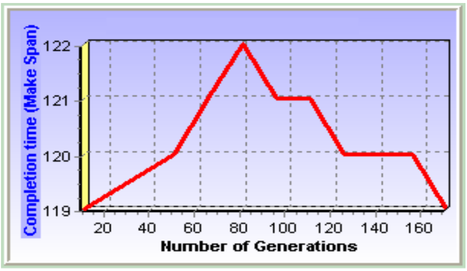

Figure 4: Total Completion Time

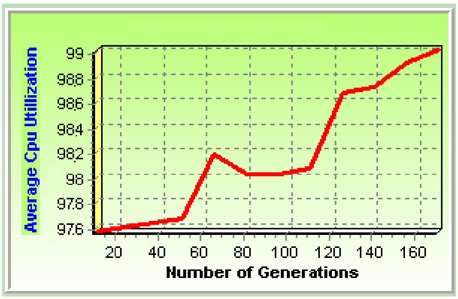

Figure 5: Average Processor Utilization

Changing The Size of Population

Changing the size of population is also considerable in terms of total completion time, processor utilization. The Obtained results are shown in Figure 6 and, Figure 7. While the size of population was increased the total completion time was decreased and, average processor utilization was increased. This is because that while the number of ants increased the search space of solutions exploited better, and it leads to better schedules.

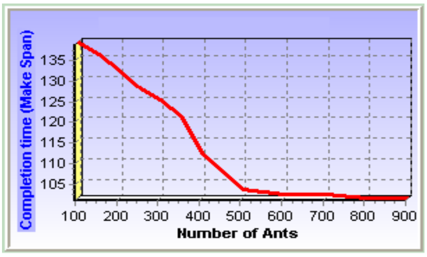

Figure 6: Total Completion Time

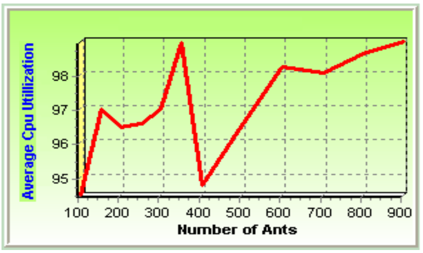

Figure 7: Average Processor Utilization

CONCLUTIONS

Scheduling in distributed operating systems has a significant role in overall system performance and throughput. We have presented and evaluated a relatively new meta-heuristic ant colony optimization method to solve this problem. This algorithm considers multi objectives in its solution evaluation and solves the scheduling problem in a way that simultaneously minimizes, makes span, and maximizes average processor utilization and load-balance.

REFERENCES

Cheng-Fa, Tsai and Chun-Wei Tsai. 2002. ”A new approach for solving large traveling salesman problem using evolutionary ant rules”. *Proceedings of the 2002 International Joint Conference on Neural Networks (IJCNN '02)*, pp. 1540-1545.

Lan , Y. and T. Yu. 1995. " A Dynamic Central Scheduler Load-Balancing Mechanism". *Proc. IEEE 14th Ann. Int'l Phoenix Conf. Computers and Comm.*, pp. 734-740.

Lin, M. and L.T. Yang. 1999. " Hybrid Genetic Algorithms for Scheduling Partially Ordered Tasks in a Multi-processor Environment". *In Proceedings of the 6th International Conference on Real-Time Computer Systems and Applications*, pp. 382 –387.

Ma, P.Y.R.; E.Y.S. Lee; and J. Tsuchiya. 1982. "A Task Allocation Model for Distributed Computing Systems". *IEEE Trans. On Computers*, Vol. 31, No. 1, pp. 41-47.

Martino, V. D. and M. Mililotti. 2002. " Scheduling in a Grid computing environment using Genetic Algorithms". *In Proceedings of the IEEE International Parallel and Distributed Processing Symposium*.

Martino, V. D. 2003. "Sub Optimal Scheduling in a Grid using Genetic Algorithms". *In Proceedings of the IEEE International Parallel and Distributed Processing Symposium*.

Moor, M. 2003. "An Accurate and Efficient Parallel Genetic Algorithm to Schedule Tasks on a Cluster". *In Proceedings of the IEEE International Parallel and Distributed Processing Symposium*.

Oh, J. and C. Wu. 2004. "Genetic-algorithm-based Real-time Task Scheduling With Multiple Goals". *International Journal of Systems and Software*, Vol. 71, pp. 245-258.

Park, C.I. and T.Y. Choe. 2002. "An optimal scheduling algorithm based on task duplication". *IEEE Transactions on Computers*, Vol. 51 ,No. 4 ,pp. 444–448.

Park, G.L. 2004. "Performance Evaluation of a List Scheduling Algorithm In Distributed Memory Multiprocessor Systems", *Journal of Future Generation Computer Systems*, Vol. 20, pp. 249-256.

Salleh, S. and A.Y. Zomaya. 1999. " Scheduling in Parallel Computing Systems: Fuzzy and Annealing Techniques", *Kluwer Academic*.

Sarje, A.K. and G. Sagar. 1991. "Heuristic Model for Task Allocation in Distributed Computer Systems", *Proceedings of the IEEE* , Vol. 138, No. 5, pp. 313-318.

Shen, C. C. and W.H. Tsai. 1985. "A Graph Matching Approach to Optimal Task Assignment in Distributed Computing Using a Minimax Criterion ". *IEEE Trans. On Computers*, Vol. 34, No. 3, pp. 197-203.

Solnon, C. 2002. "Ants Can Solve Constraint Satisfication Problems". *IEEE Trans. On Evolutionary Computation*, Vol. 6, No. 4, pp. 347-357.

Stutzle, T. and H. H. Hoos. 2000. "Max-Min Ant System". *Journal of Future Generation Computer Systems*, Vol. 16, pp. 889-914.

Wang, P. and W. Korfhage. 1995. ”Process Scheduling Using Genetic Algorithms”. *IEEE. Symposium on Parallel and Distributed Processing*, pp. 638-641.

Woodside, C.M. and G.G. Monforton. 1993. "Fast Allocation of Processes in Distributed and Parallel Systems". *IEEE Trans. On Parallel and Distributed Systems*, Vol. 4, No. 2, pp. 164-174.

Zomaya, A. Y.; C. Ward; and B. Macey. 1999. " Genetic Scheduling for Parallel Processor Systems: Comparative Studies and Performance Issues ". *IEEE Trans. On Parallel and Distributed Systems* , Vol. 10, No. 8, pp. 795-812.

Zomaya, A. Y. ,and Y. Teh. 2001. "Observations on Using Genetic Algorithms for Dynamic Load-Balancing ". *IEEE Trans. On Parallel and Distributed Systems*, Vol. 12, No. 9, pp. 899-911.