A GENETIC ALGORITHM FOR 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, genetic algorithm, simulated annealing , 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, using the power of genetic algorithms we solve this problem considering 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) is comprised of 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 been 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 [23], mathematical models-based methods [24], and heuristic Techniques [2, 6, 18, 21].

Heuristics can obtain suboptimal solution in ordinary situations and optimal solution in particulars. Since the scheduling problem has been known to be NP-complete, using heuristic Techniques can solve this problem more efficiently. Three most well-known heuristics are the iterative improvement algorithms [13],the probabilistic optimization algorithms, and the constructive heuristics. In the probabilistic optimization group, GA-based methods [1,4,5,11,13,14,17] and simulated annealing [12,20] 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 [12]. 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 task assignments. [9,15,16,19,22] have proposed scheduling algorithms considering load balancing .

A GA starts with a generation of individuals, which are encoded as strings known as chromosomes. A chromosome corresponds to a solution to the problem. A certain fitness function is used to evaluate the fitness of each individual. Good individuals survive after selection according to the fitness of individuals. Then the survived individuals reproduce offspring through crossover and mutation operators. This process iterates until termination condition is satisfied [10]. GA-based algorithms have emerged as powerful tools to solve NP-complete constrained optimization problems, such as traveling salesman problem, job-shop scheduling and flow-shop scheduling, machine learning, VLSI technology, genetic synthesis and etc [3, 7].

In this paper using the power of genetic algorithms we solve this problem considering load balancing efficiently. The proposed algorithm maps each schedule with a chromosome that shows the execution order of all existing processes on processors. The fittest chromosomes are selected to reproduce offspring; chromosomes which their corresponding schedules have less total execution time, better load-balance and processor utilization. We assume that the distributed system is non-uniform and non-preemptive, that is, the processors may be different, and a processor completes current process before executing a new one. The load-

Proceedings 21st European Conference on Modelling and Simulation
Ivan Zelinka, Zuzana Oplatková, Alessandra Orsoni ©ECMS 2007
ISBN 978-0-9553018-2-7 / ISBN 978-0-9553018-3-4 (CD)

balancing mechanism used in this paper only schedule processes without process migration and is centralized.
The remainder of this paper is organized as follows: The problem description and formulation is given in Section 2, In Section 3, we describe the proposed algorithm. Section 4, gives the performance evaluation of the proposed algorithm in comparison with other similar algorithms. Section 5 concludes this research.

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 chromosome (schedule) is evaluated, the current existing load on individual processors must also be taken into account, therefore (1):

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

Definition 2

The length or *maxspan* 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 must be computed (2,3) :

$$\max span(T) = \max\left(Load(p_i)\right) \quad \forall 1 \le i \le Number\ of\ \text{Processors} \qquad (2)$$

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

Definition 3

The Processor utilization for each processor is obtained by dividing the sum of processing times by maxspan, and the average of processors utilization is obtained by dividing the sum of all utilizations by number of processors (4, 5):

$$U(p_i) = Load(p_i) \Big/ \max span \qquad (4)$$

$$AveU = \left(\sum_{i=1}^{No\ of\ processors} U(p_i)\right) \Big/ Number\ Of\ \text{Processors} \qquad (5)$$

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 (6):

$$AveNoAPQ = NoAPQ / Number\ Of\ \text{Processors} \qquad (6)$$

Definition 5

A Queue associated with every processor, shows the processes that processor has to execute. The execution order of processes on each processor is based on queues.

THE PROPOSED GA-BASED ALGORITHM

Genetic algorithms, as powerful and broadly applicable stochastic search and optimization techniques, are the most widely known types of evolutionary computation methods today. In general, a genetic algorithm has five basic components as follows [3]:

1. An encoding method, that is a genetic representation (genotype) of solutions to the program.
2. A way to create an initial population of individuals (chromosomes).
3. An evaluation function, rating solutions in terms of their fitness, and a selection mechanism.
4. The genetic operators (crossover and mutation) that alter the genetic composition of offspring during reproduction.
5. Values for the parameters of genetic algorithm.

Genotype

In the GA-Based algorithms each chromosome corresponds to a solution to the problem. The genetic representation of individuals is called *Genotype*. Many *Genotypes* have been proposed in [3] .In this paper a chromosome consists of an array of *n* digits, where *n* is the number of processes. Indexes show process numbers and a digit can take any one of the *1..m* values, which shows the processor that the process is assigned to. If more than one process is assigned to the same processor, the left to-right order determines their execution order on that processor.

Initial Population

A genetic algorithm starts with a set of individuals called initial population. Most GA-Based algorithms generate initial population randomly. Here, each solution *i* is generated as follows: one of the unscheduled processes is randomly selected, and then assigned to one of the processors. The important point is the processors are selected circularly, it means that they are selected respectively form first to last and then come back to first. This operation is repeated until all of processes have been assigned. An initial population with size of *POPSIZE* is generated by repeating this method.

Fitness Function

As discussed before, the main objective of GA 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 function of a Schedule T (7):

$$fitness(T) = \frac{(\gamma \times AveU) \times (\theta \times AveNoAPQ)}{(\alpha \times \max span(T)) \times (\beta \times CC(T))} \quad (7)$$

Which $0 < \alpha, \beta, \gamma, \theta \leq 1$ 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 maxspan, less communication cost, higher processor utilization and higher Average number of acceptable processor queues.

Selection

The selection process used here is based on spinning the roulette wheel, which each chromosome in the population has a slot sized in proportion to its fitness. Each time we require an offspring, a simple spin of the weighted roulette wheel gives a parent chromosome. The probability p_i that a parent T_i is selected is given by (8):

$$P_i = \frac{F(T_i)}{\sum_{j=1}^{POPSIZE} F(T_j)} \quad (8)$$

where $F(T_i)$ is the fitness of chromosome T_i.

Crossover

Crossover is generally used to exchange portions between strings. Several crossover operators are described in the literature [10]. Crossover is not always affected, the invocation of the crossover depends on the probability of the crossover Pc. We have implemented two crossover operators. The GA uses one of them, which is decided randomly.

Single-Point Crossover

This operator randomly selects a point, called *Crossover point*, on the selected chromosomes, then swaps the bottom halves after crossover point, including the gene at the crossover point and generate two new chromosomes called children.

Proposed Crossover

This operator randomly selects points on the selected chromosomes, then for each child non-selected genes are taken from one parent and selected genes from the other.

Mutation

Mutation is used to change the genes in a chromosome. Mutation replaces the value of a gene with a new value from defined domain for that gene. Mutation is not always affected, the invocation of the Mutation depend on the probability of the Mutation *Pm*. We have implemented two mutation operators. The GA uses one of them, which is decided randomly.

First Mutation Operator
This operator randomly selects two points on the selected chromosome, then generates a chromosome by swapping the genes at the selected points.

Second Mutation Operator
The other approach is to check if any jobs could be swapped between processors which would result in a lower make span. If we want to test every possible swap, it would be computationally very intensive, and in larger problems would take an unfeasible amount of time. It also seems unreasonable to consider swapping processes on processors which their load is significantly below the make span, therefore we try to swap processes between overloaded and under loaded processors. This concept can be implemented as follows:

1. First, select a processor, say p_v, which has the latest finish time.
2. Second, select a processor, say p_u, which has least finish time.
3. Third, try to transfer a process form p_v to p_u or swap a single pair of processes between p_v and p_u that improves the make span of both processors the most.
4. This procedure is repeated until no further improvement is possible.

Replacement Strategy

When genetic operators (crossover, mutation) are applied on selected parents T_1, T_2 two new chromosomes *T'* and *T''* are generated. These chromosomes are added to new temporary population. By repeating this operation a new temporary population with size of 2**POPSIZE* is generated. After that fitter chromosomes are selected from current population and new temporary population, at last selected chromosomes made new population and algorithm restarts.

Termination Condition

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

The Structure of Proposed GA-Based Algorithm

Our proposed GA-Based algorithm starts with a generation of individuals. A certain fitness function is used to evaluate the fitness of each individual. Good individuals survive after selection according to the fitness of individuals. Then the survived individuals reproduce offspring through crossover and mutation operators. This process iterates until termination condition is satisfied. It is Considerable to say that parameters such as $P_C, P_m, POPSIZE, NOGEN, \alpha, \beta, \gamma, \theta$ must be determined before GA is started. Figure 1 shows this operation.

```
Procedure GA-Based algorithm;
Begin
  initialize P(k);  {create an initial population}
  evaluate P(k);  {evaluates all individuals in the
population}
  Repeat
      For i=1 to 2*POPSIZE do
          Select two chromosomes as parent 1
          and parent 2 from population;
          Child 1 and Child 2 ← Crossover( parent1,
          parent2);
          Child 1 ← Mutation ( Child 1 );
          Child 2 ← Mutation ( Child 2 );
          Add (new temporary population, Child 1, Child 2
);
      End For;
      Make (new population, new temporary population,
population );
     Population = new population;
  While (not termination condition);
Select Best chromosome in population as solution and
return it;
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 GA-based algorithm. Current solution techniques are concentrated on scheduling tasks with precedence constraints so our approach is not completely comparable with them. We have implemented more than 3000 lines of C++ program to simulate all of the proposed algorithms. All simulation experiments are run on a Pentium III 800, 256 MB RAM, IBM PC. We have tried different values of the population size (*POPSIZE*), mutation Probability (*Pm*),and crossover probability (*Pc*),to find which values would steer the search towards the best solution. The measurement of performance of these algorithms was based on three metrics: total completion time, average processor utilization and, cost of communication. 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. Brief justifications for the values used are given below. When the values discussed were tested 'base values' for each of the parameters where used to help isolate the performance of the parameter in

hand. These values were: *Pc*=0.9, *Pm*=0.1, *POPSIZE*=50, *NOGEN*=50, *m*=10 (number of processors), *n=100...1000*.

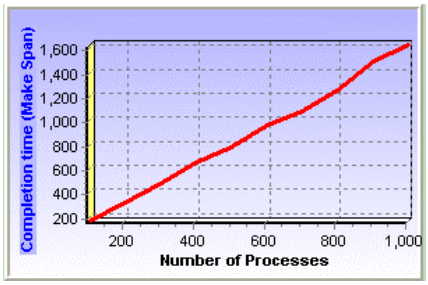

Figure 2: Total Completion Time

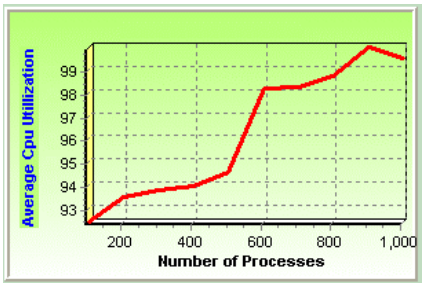

Figure 3: Average Processor Utilization

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, Figure 5 and, Figure 6. While the number of generations was increased the total completion time was reduced, it is because 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 higher utilization is obtained and, the cost of communication was decreased. Brief justifications for the values used are given below. These values were: *Pc*=0.9, *Pm*=0.1, *POPSIZE*=100, *NOGEN*=50…245, *m*=10 (number of processors), *n=300* (number of processes).

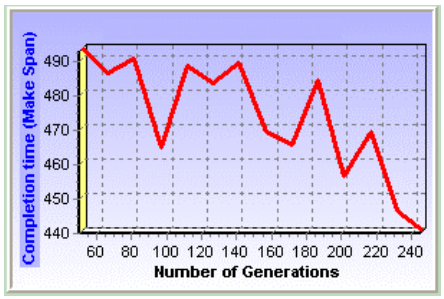

Figure 4: Total Completion Time

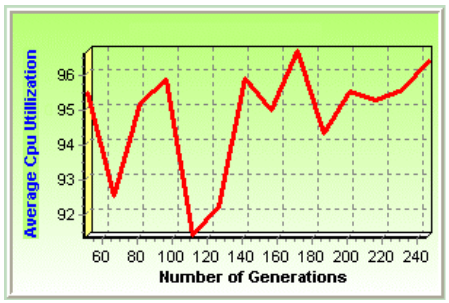

Figure5: Average Processor Utilization

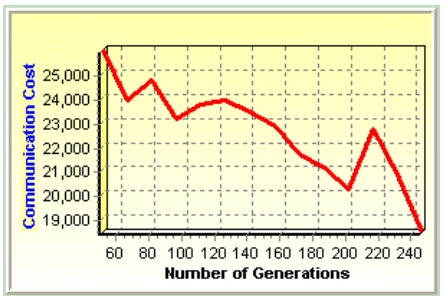

Figure 6: Cost Of Communication

Changing The Size of Population

Changing the size of population is also considerable in terms of total completion time, processor utilization and, cost of communication. The Obtained results are shown in Figure 7, Figure 8 and, Figure 9. While the size of population was increased the total completion time was decreased and, average processor utilization was increased. it is because that the number of the fitter chromosomes which are able to survive was increased, therefore fitter offspring may be generated and it leads to better schedules. According to above results while the size of population was increased higher processor utilization obtained and the cost of communication was decreased. Brief justifications for the values used are given below. *Pc*=0.9, *Pm*=0.1, *POPSIZE*=50…150, *NOGEN*=50, *m*=10 (number of processors), *n=300* (number of processes).

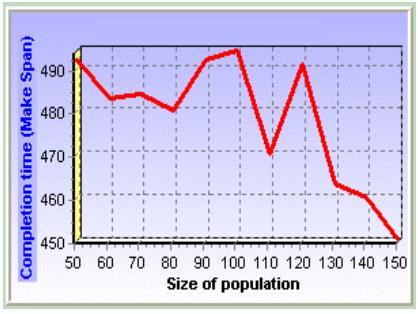

Figure 7: Total Completion Time

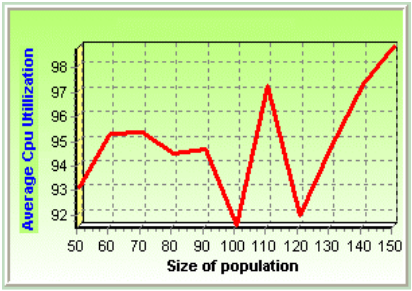

Figure 8: Average Processor Utilization

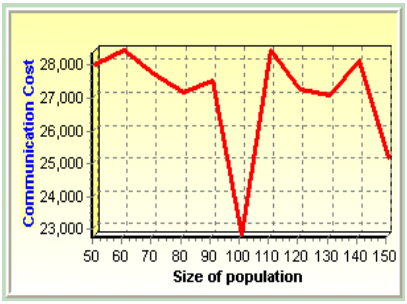

Figure 9: Cost Of Communication

CONCLUTIONS

Scheduling in distributed operating systems has a significant role in overall system performance and throughput. The scheduling in distributed systems is known as an NP-complete problem even in the best conditions. We have presented and evaluated new GA-Based 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 maxspan and communication cost, and maximizes average processor utilization and load-balance. Most existing approaches tend to focus on one of the objectives. Experimental results prove that our proposed algorithm tend to focus on all of the objectives simultaneously and optimize them.

REFERENCES

[1] W.Yao, J.Yao, & B.Li, "Main Sequences Genetic Scheduling For Multiprocessor Systems Using Task Duplication", *International Journal of Microprocessors and Microsystems, 28*, 2004, 85-94.

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

[3] A.T. Haghighat, K. Faez, M. Dehghan, A. Mowlaei, & Y. Ghahremani, "GA-based heuristic algorithms for bandwidth-delay-constrained least-cost multicast routing", *International Journal of Computer Communications 27*, 2004, 111–127.

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

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

[6] C.I.Park, & T.Y.Choe, "An optimal scheduling algorithm based on task duplication" , *IEEE Trans. on Computers, 51*(4), 2002, 444–448.

[7] A.T. Haghighat, K. Faez, M. Dehghan, A. Mowlaei, & Y. Ghahremani, "Multicast routing with multiple constraints in high-speed networks based on genetic algorithms" , *In ICCC 2002 Conf.*, India, 2002, 243–249.

[8] A.Y.Zomaya, & Y.Teh, "Observations on Using Genetic Algorithms for Dynamic Load-Balancing", *IEEE Trans .On Parallel and Distributed Systems, 12*(9), 2001, 899-911.

[9] K.Qureshi, and M.Hatanaka, "A Practical Approach of Task Scheduling and Load Balancing on Hetrogeneous Distributed Raytracing Systems", *Information Processing Letters 79*, 2001, 65-71.

[10] L.M.Schmitt, "Fundamental Study Theory of Genetic Algorithms" , *International Journal of Modelling and Simulation Theoretical Computer Science 259*, 2001, 1 – 61.

[11] A.Y.Zomaya, C.Ward, & B.Macey, "Genetic Scheduling for Parallel Processor Systems: Comparative Studies and Performance Issues", *IEEE Trans. On Parallel and Distributed Systems*, 10(8), 1999, 795-812.

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

[13] M.Lin, & L.T.Yang, "Hybrid Genetic Algorithms for Scheduling Partially Ordered Tasks in a Multi-processor Environment", *Proc. of the 6th IEEE Conf. on Real-Time Computer Systems and Applications*, 1999, 382–387.

[14] Sung-Ho Woo, Sung-Bong Yang, Shin-Dug Kim, Tack-Don Han, "Task scheduling in distributed computing systems with a genetic algorithm", *High-Performance Computing on the Information Superhighway*, HPC-Asia '97, 1997, p. 301.

[15] C. Xu, & F. Lau, "Load-Balancing in Parallel Computers : Theory and Practice", *Kluwer Academic*, 1997.

[16] Y. Lan, & T. Yu, "A Dynamic Central Scheduler Load-Balancing Mechanism", *Proc. of the 14th IEEE Ann. Int'l Phoenix Conf. on Computers and Communication*, 1995, 734-740.

[17] E.S.H.Hou, N.Ansari, & H.Ren, "A Genetic Algorithm for Multiprocessor Scheduling", *IEEE Trans. On Parallel and Distributed Systems, 5*(2), 1994, 113-120.

[18] C.M.Woodside, & G.G.Monforton, "Fast Allocation of Processes in Distributed and Parallel Systems", *IEEE Trans. On Parallel and Distributed Systems, 4*(2), 1993, 164-174.

[19] H.C. Lin, & C.S. Raghavendra, "A Dynamic Load-Balancing Policy with a Central Job Dispatcher (LBC)", *IEEE Trans. on Software Eng., 18*(2), 1992, 148-158.

[20] A.Nanda, et. al, "Scheduling Directed Task Graphs on Multiprocessors Using Simulated Annealing", *Proc. Int'l. Conf. On Distributed Systems*, 1992, 20-27.

[21] A.K.Sarje & G.Sagar, "Heuristic Model for Task Allocation in Distributed Computer Systems", *Proc. of the IEE-E, 138*(5), 1991, 313-318.

[22] F. Bonomi, & A. Kumar, "Adaptive Optimal Load-Balancing in a Heterogeneous Multiserver System with a Central Job Scheduler", *IEEE Trans. on Computers, 39*(10) 1990, 1232-1250.

[23] C.C.Shen, & W.H.Tsai, "A Graph Matching Approach to Optimal Task Assignment in Distributed Computing Using a Minimax Criterion", *IEEE Trans. On Computers, 34*(3), 1985, 197-203.

[24] P.Y.R.Ma, E.Y.S.Lee, & J.Tsuchiya, "A Task Allocation Model for Distributed Computing Systems", *IEEE Trans. On Computers, 31*(1), 1982, 41-47.

[25] T.C.Hu, *combinatorial algorithms* (Addison-Wesley, 1982).