A New Approach for Task Scheduling in Distributed Systems Using Learning Automata

M. Jahanshahi
Department of Computer Engineering
Islamic Azad University, Central Tehran Branch
Tehran, Iran
mjahanshahi@iauctb.ac.ir

M. R. Meybodi and M. Dehghan
Department of Computer Engineering and Information Technology
Amirkabir University of Technology
Tehran, Iran
{mmeybodi,dehghan}@aut.ac.ir

Abstract - Tasks scheduling problem is a key factor for a distributed system in order to achieve better efficiency. The problem of tasks scheduling in a distributed system can be stated as allocating tasks to processor of each computer. The objective of this problem is minimizing Makespan and communication cost while maximizing CPU utilization. Scheduling problem is known as NP-complete. Hence, many genetic algorithms have been proposed to search optimal solutions from entire solution space. However, the existing approaches are going to scan the entire solution space without consideration to techniques that can reduce the complexity of the optimization. In other words, the main weakness of these methods is to spend much time doing scheduling and hence need to exhaustive time. In this paper we use Learning algorithm to cope with the weakness of GA based method. In fact we use the Learning automata as local search in the memetic algorithm. Experimental results prove that the proposed method outperforms the existent GA based method in terms of communication cost, CPU utilization and Makespan.

Index Terms - Task scheduling, Memetic algorithm, Learning automata.

I. INTRODUCTION

With many achievements such as technology, computer architectures, and software packages, distributed systems are used in a great variety of applications. The problem of task scheduling in these systems has received a large amount of attention recently. Task scheduling in a distributed system can be stated as allocating tasks to processors of each computer such that the optimum performance is obtained. The aim of task scheduling is minimizing Makespan (job completion time) and communication cost while maximizing CPU utilization. This problem is known as NP-complete [14].

There are two categories for task scheduling; static and dynamic. In dynamic scheduling, schedules create during run time and no knowledge of task is in hand until it arrives. While in static scheduling, schedules are created before run time and can not change. Similarly, tasks must all be known in advance. In other words a static task scheduling algorithm schedules a set of tasks with known processing and communication characteristics on processors to optimize some performance metric, such as Makespan, communication cost and CPU utilization. In this paper we focus on static scheduling.

Several methods have been proposed to solve scheduling problem. The proposed methods can be generally classified into three categories: Graph-theory-based approaches [15], mathematical models-based methods [16] and heuristic Techniques[17-20]. As mentioned above the scheduling problem has been known to be NP-complete. Therefore using heuristic Techniques can solve this problem more efficiently. Three most well-known heuristics are the iterative improvement algorithms [25], the probabilistic optimization algorithms, and the constructive heuristics. In the probabilistic optimization group, GA-based methods [21-27] are considerable. The main distinction among them is chromosomal representation used for a schedule. However, these approaches scan the entire solution space without consideration to techniques that can reduce the complexity of the optimization. In other words their main shortcoming is to spend much time doing scheduling. This shortcoming of GA-based methods can be reduced by combing GA with another optimization technique.

This paper proposes a new algorithm by using memetic algorithm to cope with this shortcoming of the GA based methods. We use Learning automata as local search in the memetic algorithm. Simulation results show that our proposed method outperforms the existent GA based method in terms of communication cost, CPU utilization and Make span.

The rest of this paper is organized as follows: In section 2 the Learning automata which is used as local search in the memetic algorithm is introduced. Proposed method comes in section 3. Simulation results are given in section 4. Section 5 concludes the paper.

II. LEARNING AUTOMATA

Learning automata is an abstract model that chooses an action from a finite set of its actions randomly and takes it [5-8]. In this case, environment evaluates this action and reacts to it by a reinforcement signal. Then, learning automata updates its internal information regarding both the taken action and received reinforcement signal. After that, learning automata chooses another action again. Figure 1 depicts the relationship between learning automata and environment.
Every environment is represented by $E = \{\alpha, \beta, c\}$, where $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$ is a set of actions, $\beta = \{\beta_1, \beta_2, \ldots, \beta_r\}$ is a set of outputs, and $c = \{c_1, c_2, \ldots, c_r\}$ is a set of penalty probabilities. Whenever set $\beta$ has just two members, environment is kind of stochastic.

Learning automata is classified into fixed structure and variable structure. Learning automata with fixed structure is represented by $\{\alpha, \beta, p, T\}$, where $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$ is a set of actions, $\beta = \{\beta_1, \beta_2, \ldots, \beta_r\}$ is a set of outputs, $p = \{p_1, p_2, \ldots, p_r\}$ is the action probability vector, and $p(n+1) = T(\alpha(n), \beta(n), p(n))$ is learning algorithm.

Learning automata operates as follows; learning automata chooses an action from its probability vector randomly ($p_i$) and takes it. Suppose that the chosen action is $\alpha_i$. Learning automata after receiving reinforcement signal from environment updates its action probability vector according to formulas 1 and 2 in case of desirable and undesirable received signals respectively. In formulas 1 and 2, $a$ and $b$ are reward and penalty parameters respectively. If $a = b$ then algorithm is named $L_{RP}$. Also, if $b < a$ then the algorithm is named $L_{RP}$. Similarly, if $b = 0$ then the algorithm is called $L_{RI}$. For $p(n+1) = p(n) + a.(1 - p(n))$ and $p(n+1) = p(n) - a.p(n)$ respectively.

$$p(n+1) = p(n) + a.(1 - p(n))$$
$$p(n+1) = p(n) - a.p(n)$$

### III. PROPOSED METHOD

In our model there are finite numbers of tasks, each having a task number and a execution time and placed in a task pool from which tasks are assigned to processors. Figure 2 shows the proposed chromosome. In this chromosome, tasks 1, 2, 3, 4, 5, 6, 7 and 8 are assigned to processors 2, 1, 2, 3, 3, 4, 2 and 1 respectively. Both tasks 1 and 3 are assigned to processor 2 but first task 1 is executed then 3.

Before we describe the proposed method it is necessary to state some definitions which have been stated in [1] as follows:

- $T = \{t_1, t_2, t_3, \ldots, t_n\}$ is the set of tasks to execute.
- $P = \{p_1, p_2, p_3, \ldots, p_m\}$ is a set of processors in the distributed system. Each processor can only execute one task at each moment, a processor completes current task before executing a new one, and a task can not be moved to another processor during execution.
- $R$ is an $m \times m$ matrix, where the element $r_{uv}$ such that $1 \leq u, v \leq m$ of $R$, is the communication delay rate between $p_u$ and $p_v$.
- $H$ is an $m \times m$ matrix, where the element $h_{uv}$ such that $1 \leq u, v \leq m$ 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 $h_{uu} = 0$.  

![Fig. 1 Interaction between learning automata and environment](image1)

![Fig. 2 An example of proposed chromosome](image2)
A is an $n \times m$ matrix, where the element $a_{ij}$, $1 \leq i \leq n$, $1 \leq j \leq m$, of A, is the execution time of task $t_i$ on processor $p_j$.

- D is a linear matrix, where the element $d_{ij}$, $1 \leq i \leq n$, of D, is the data volume for task $t_i$ to be transmitted, when task $t_i$ is to be executed on a remote processor.

- F is a linear matrix, where the element $f_{ij}$, $1 \leq i \leq n$, of F, is the target processor that is selected for task $t_i$ to be executed on.

- C is a linear matrix, where the element $c_{ij}$, $1 \leq i \leq n$, of C, is the processor that the task $t_i$ is worked on.

- The processor load for each processor is stated as follows:

$$\text{Load}(p_i) = \sum_{j=1}^{m} a_{ij} + \sum_{k=1}^{m} a_{kj},$$

(3)

- The Makespan of a schedule is the maximal finishing time of all processes or maximum load.

$$\text{makespan}(T) = \max \{ \text{Load} (p_i) \}$$

(4)

- Communication cost (CC) is computed as follows:

$$\text{CC}(T) = \sum_{i=1}^{n} r_{ij}f_i + h_{ij}h_i \times d_i$$

(5)

- The Processor utilization for each processor and the average of processors utilization are also computed as follows:

$$U(p_i) = \frac{\text{Load}(p_i)}{\max \text{span}},$$

(6)

$$\text{AveU} = \left( \frac{\sum_{i=1}^{n} U(p_i)}{\text{Number Of Processors}} \right)$$

(7)

- Number of Acceptable Processor Queues (NoAPQ): We must define thresholds for light and heavy load on processors. If the tasks completion time of a processor 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:

$$\text{AveNoAPQ} = \frac{\text{NoAPQ}}{\text{Number Of Processors}}$$

(8)

A Queue associated with every processor, shows the tasks that processor has to execute. The execution order of tasks on each processor is based on queues. Finally the fitness of the chromosome (Schedule T) can be computed as follows:

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

(9)

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 Makespan, less communication cost, higher processor utilization and higher Average number of acceptable processor queues.

Now we describe the proposed method in details. Figure 3 depicts the proposed method. We apply Learning automata as local search in our proposed memetic algorithm.

In this method the time LA learns to assign the tasks to processors such that the fitness of the chromosome will be better. In this method LA has $n$ actions ($n$ is the number of tasks). When the LA chooses action $n$, that means LA decides to assign another processor to task $n$; one round of the LA based algorithm works as follows:

Initially LA considering its action probability vector chooses an action (e.g. action $n$). Its action means another processor must be assigned to task $n$. Therefore the new chromosome will be created. After that LA updates its action probability vector according to the fitness of new chromosome and the older one. In the method based on LA with updating rule $L_{R-P}$, if the fitness of the new chromosome is better that the older one, then the selected action is rewarded. Otherwise it is penalized. After updating the action probability vector by the LA, the new chromosome will be added to population. In next iteration LA chooses another action again.

IV. Simulation Results

In this section we evaluate the proposed method compared with [1] regarding communication cost, CPU utilization and Makespan.

First experiment:

In this experiment we increase the number of tasks and compute average of all CPU utilization, communication cost and Makespan. Figures 4 through 6 depict simulation results. As it can obtain from figure 4 average of all CPU
utilization in GA-based method (reference [1]) is less than our methods.

Figure 5 shows that when we increase the number of the tasks both methods have approximately the same communication cost. But as it can be seen from figure 6 Makespan in GA-Base method in case of increasing the number of tasks is more than the proposed method.

Figure 8 also depicts average of CPU utilization in both methods. As it can be seen GA-based method has under utilization. Also figure 9 shows communication cost of both methods.

Third experiment:
Objective of this experiment is computation of the above metrics when we increase the number of generations. Figure 10 through 12 show the simulation results. Figure 10 shows the average of CPU utilization in both methods. It can be seen with increasing the number of generation CPU utilization in GA-based method is less than our method. In another words GA-based method has not scalability. Figure 11 depicts communication cost in both methods. Also figure 12 shows Makespan in GA-based method is more than our method when we increase the number of generation.
V. Conclusion

In this paper we applied Learning automata as local search in memetic. Experimental results demonstrate that the proposed method outperforms the existent GA based method in terms of communication cost, CPU utilization and Makespan.

Reference

[21] W.Yao, J.Yao and B.Li, "Main Sequences Genetic Scheduling For Multiprocessor Systems Using Task


