A New Method Based on Memetic Algorithm for Task Scheduling in Distributed Systems

M. H. Kashani
Computer Engineering Department,
Islamic Azad University, Shahr-e-Qods branch
Tehran, Iran
kashani@shahryariau.ac.ir

M. Jahanshahi
Computer Engineering Department,
Islamic Azad University, Central Tehran branch
Tehran, Iran
mjahanshahi@iauctb.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, these 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 shortcoming of these approaches is to spend much time doing scheduling and hence need to exhaustive time. Therefore, in this paper we use memetic algorithm to cope with this shortcoming. We apply Learning automata, simulated annealing, and Tabu search as local search in our proposed memetic algorithm. Extended simulation results demonstrate that the proposed methods outperform the existent GA-based method in term of communication cost, CPU utilization and makespan.

Keywords—Task scheduling, Memetic algorithm, Simulated annealing

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 [24].

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 cannot 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 [25], mathematical models-based methods [26] and heuristic Techniques [27-30].

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 [35], the probabilistic optimization algorithms, and the constructive heuristics. In the probabilistic optimization group, GA-based methods [31-37] 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. Hence this paper proposes a new algorithm by using memetic algorithm to cope with this shortcoming. We apply Learning automata with updating algorithm $L_{R-P}$ and $L_{R-I}$, Simulated annealing and Tabu search as local search in memetic algorithm. Simulation results demonstrate that our proposed methods outperform the existent GA-based method in term of communication cost, CPU utilization and Makespan.

The rest of this paper is organized as follows: In section 2 used methods as local search in the proposed memetic algorithm are presented. Proposed methods come in section 3. Simulation results are given in section 4. Section 5 concludes the paper.
II. USED METHODS AS LOCAL SEARCH IN THE PROPOSED ALGORITHM

In this section we describe the methods which have been used as local search in our proposed memetic algorithm.

Learning automata is an abstract model that chooses an action from a finite set of its actions randomly and takes it [15-18]. In this case, environment evaluates this taken 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.

![Figure 1. Interaction between learning automata and environment](image)

Every environment is represented by $E = \{\alpha, \beta, c\}$, where $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$ is a set of inputs, $\beta = \{\beta_1, \beta_2, \ldots, \beta_p\}$ 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 $p$. In this environment $\beta_1=1$, $\beta_2=0$ are considered as penalty and reward respectively. Similarly, environment in kind of $S$ has infinite number of members. $c_i$ is the penalty probability of taken action $\alpha_i$.

Learning automata is classified into fixed structure and variable structure. Learning automata with variable structure is introduced as follows; Learning automata with variable structure is represented by $\{\alpha, \beta, p, T\}$, where $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$ is a set of actions, $\beta = \{\beta_1, \beta_2, \ldots, \beta_p\}$ is a set of inputs, $p = \{p_1, p_2, \ldots, p_p\}$ 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 $L_{R-P}$. Also, if $b << a$ then the algorithm is named $L_{RD-P}$. Similarly, if $b = 0$ then the algorithm is called $L_{R-I}$.

\begin{align}
 p_i(n+1) &= p_i(n) + a(1 - p_i(n)) \\
 p_j(n+1) &= p_j(n) - a \cdot p_j(n) \quad \forall j \neq i
\end{align}

\begin{align}
 p_i(n+1) &= (1 - b) \cdot p_i(n) \\
 p_j(n+1) &= \frac{b}{r-1} + (1 - b) \cdot p_j(n) \quad \forall j \neq i
\end{align}

Simulated annealing is a generalization of a Monte Carlo method for examining the equations of state and frozen states of n-body systems [2,3]. The concept is based on the manner in which liquids freeze or metals recrystallize in the process of annealing. In an annealing process a melt, initially at high temperature and disordered, is slowly cooled so that the system at any time is approximately in thermodynamic equilibrium. As cooling proceeds, the system becomes more ordered and approaches a “frozen” ground state at $T=0$. Hence the process can be thought of as an adiabatic approach to the lowest energy state. If the initial temperature of the system is too low or cooling is done insufficiently slowly the system may become quenched forming defects or freezing out in metastable states (i.e. trapped in a local minimum energy state). The original Metropolis scheme was that an initial state of a thermodynamic system was chosen at energy $E$ and temperature $T$, holding $T$ constant the initial configuration is perturbed and the change in energy $dE$ is computed. If the change in energy is negative the new configuration is accepted. If the change in energy is positive it is accepted with a probability given by the Boltzmann factor $\exp(-dE/T)$. This processes is then repeated sufficient times to give good sampling statistics for the current temperature, and then the temperature is decremented and the entire process repeated until a frozen state is achieved at $T=0$. By analogy the generalization of this Monte Carlo approach to combinatorial problems is straightforward: the current state of the thermodynamic system is analogous to the current solution to the combinatorial problem, the energy equation for the thermodynamic system is analogous to at the objective function, and ground state is analogous to the global minimum.

The basic concept of Tabu Search as described in [4-11] is a meta-heuristic superimposed on another heuristic. The overall approach is to avoid entainment in cycles by forbidding or penalizing moves which take the solution, in the next iteration, to points in the solution space previously visited (hence "Tabu"). The Tabu search begins by marching to a local minima. To avoid retracing the steps used the method records recent
moves in one or more Tabu lists. The original intent of the list was not to prevent a previous move from being repeated, but rather to ensure it was not reversed.

*Genetic algorithms (GA)* are search methods that take their inspiration from natural selection and survival of the fittest in the biological world [12-14]. GAs differ from more traditional optimization techniques in that they involve a search from a "population" of solutions, not from a single point. Each iteration of an GA involves a competitive selection that weeds out poor solutions. The solutions with high "fitness" are "recombined" with other solutions by swapping parts of a solution with another (crossover). After selection and crossover, there is a new population full of individuals. Some are directly copied, and others are produced by crossover. In order to ensure that the individuals are not all exactly the same, you allow for a small chance of mutation. Solutions are also "mutated" by making a small change to a single element of the solution. Recombination and mutation are used to generate new solutions that are biased towards regions of the space for which good solutions have already been seen. Mutation is, however, vital to ensuring genetic diversity within the population.

Memetic algorithms are a form of genetic algorithm that apply a local search process to refine solutions to hard problems [19-22].

In this paper we apply Learning automata with updating algorithm $L_{R-P}$ and $L_{R-J}$, simulated annealing and Tabu search as local search in memetic.

### III. PROPOSED METHODS

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.

![Figure 2. An example of proposed chromosome](image)

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

- $T = \{t_1, t_2, t_3, ..., t_n\}$ is the set of tasks to execute.
- $P = \{p_1, p_2, p_3, ..., p_m\}$ 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 cannot be moved to another processor during execution.
- $R$ is an $m \times m$ matrix, where the element $r_{uv}$ $1 \leq u,v \leq m$ of $R$, is the communication delay rate between $p_u$ and $p_v$.
- $H$ is an $m \times m$ matrix, where the element $h_{uv}$ $1 \leq u,v \leq m$ 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 $r_{uu} = 0$.
- $A$ 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 $d_i$ $1 \leq i \leq n$ 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 $f_i$ $1 \leq i \leq n$ 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 $c_i$ $1 \leq i \leq n$ of $C$, is the processor that the task $t_i$ is worked on.
- The processor load for each processor is stated as follows:

\[ Load(p_i) = \sum_{j=1}^{n} a_{ij} + \sum_{k=1}^{m} a_{kj} \]

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

\[ makespan(T) = \max \left( \text{Load} \left( p_i \right) \right) \]

\[ \forall 1 \leq i \leq \text{Number of Processors} \]

- Communication cost (CC) is computed as follows:

\[ CC(T) = \sum_{i=1}^{n} \left( r_{c_if_i} + h_{c_if_i} \times d_i \right) \]

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

\[ U(p_i) = \frac{\text{Load}(p_i)}{\text{max span}} \]

\[ \text{AveU} = \left( \sum_{i=1}^{\text{No of processors}} U(p_i) \right) / \text{Number Of Processors} \]
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:

\[ (8) \quad \text{AveNoAPQ} = \frac{\text{NoAPQ}}{\text{NumberOf Processors}} \]

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:

\[ \text{fitness}(T) = \frac{(\gamma \times \text{AveU}) \times (\theta \times \text{AveNoAPQ})}{(\alpha \times \text{max span}(T)) \times (\beta \times \text{CC}(T))} \quad (9) \]

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 methods in details. Figure 3 depicts the proposed method. We apply Learning automata with updating algorithm \( L_{R-P} \) and \( L_{R-1} \), Simulated annealing and Tabu search as local search in our proposed memetic algorithm.

In this section we describe the methods which are used as local search in our memetic algorithm.

A. LA-based methods

We applied Learning automata (LA) with updating rules \( L_{R-P} \) and \( L_{R-1} \) as local search in our memetic algorithm. In these methods during 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. Similarly, in method based on LA with updating rule \( L_{R-1} \), if the fitness of the new chromosome is better that the older one, then the selected action is rewarded. 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.

B. TABU SEARCH BASED METHOD

In each step of iteration a task is selected (index of the chromosome). Now two situations may occur:

- If the selected task in not in Tabu list then a new processor is assigned to it. Hence a new chromosome is created. If the created chromosome is better than the older one then it is added to population. After that the number of task (index of the chromosome) is added to the Tabu list.
- If the selected task is in the Tabu list then the above algorithm continues at most 5 times.

In our simulations length of the Tabu list is set to 3.

C. SIMULATED ANNEALING BASED METHOD

In each step of iteration a task in the chromosome is selected (index of the chromosome). Then a new processor is randomly assigned to it. Now a new chromosome is created. If the created chromosome is better than the older one then it is added to population; otherwise we use boltzman function \( e^{\frac{-\Delta f}{kT}} \) to accept or reject the new chromosome.

IV. SIMULATION RESULTS

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

First experiment:

In this experiment we try to increase 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 4. Average of CPU utilization in all methods considering number of task

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

Second experiment:

Similarly in this experiment the aim is computing the makespan, communication cost and CPU utilization of all processors. It can be seen in figure 7 that makespan in GA-Based method is more than our propose methods. That means if we increase population size, then the trend of makespan in GA-based method will be increased. In the other hand it can be obtain GA-based method has not scalability.

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

Third experiment:

Objective of this experiment is compute the above metrics when we increase the number of generations. Figure 10 through 12 shows the simulation results. Figure 10 shows the average of CPU utilization in all methods. It can be seen with increasing the number of generation CPU utilization in GA-based method is less than all methods. In another words GA-based method has not scalability.
Figure 11 depicts communication cost in all methods. As it can be seen communication cost in SA-based method is less than other methods. Also figure 12 shows makespan in GA-based method is more than other methods when we increase the number of generation.

![Communication Cost Comparison](image1)

![Makespan Comparison](image2)

V. CONCLUSION

In this paper we used the memetic algorithm for task scheduling. We applied Learning automata with updating algorithm $L_{R-P}$ and $L_{R-I}$, simulated annealing and Tabu search as local search in memetic. Extended simulation results demonstrate that our proposed methods outperform the existent GA-based method in term of communication cost, CPU utilization and Maxspan.

REFERENCE