A PSO-based Genetic Algorithm for Scheduling of Tasks in a Heterogeneous Distributed System

Yan Kang
Department of Software Engineering, School of Software, Yunnan University, Kunming, Yunnan, China
Email: yankang@ynu.edu.cn

He Lu and Jing He
Department of Software Engineering, School of Software, Yunnan University, Kunming, Yunnan, China
Email: hejing@ynu.edu.cn

***Abstract*—The static task scheduling problem in distributed systems is important because of optimal usage of available machines and accepted computation time for scheduling algorithm. A PSO-based hybrid algorithm is presented to schedule the tasks represented by a Directed Acyclic Graph (DAG) to a bounded number of heterogeneous processors such that its schedule length is optimized. The algorithm first generate feasible initial solutions by using some effective list scheduling strategy and then evolve the solution by using crossover and mutation operator. These operators are modified to ensure the validity of the solutions. And a PSO strategy is also used to improve the quality of the solutions by using local and global information together. The performance of the algorithm is demonstrated by comparing the scheduling length ratio with the existing effectively scheduling algorithms, Heterogeneous Earliest Finish Time and Genetic Scheduling algorithm.**

***Index Terms*—scheduling algorithm; heterogeneous; heuristic; distributed system**

I. INTRODUCTION

Huge Applications can cost-effectively utilize the underlying parallelism on the available distributed resources by partitioning the application into multiple independent tasks. The problem is generally addressed in terms of task scheduling, where tasks are the schedulable units of an application, and resources are a network of processors. The scheduling of a certain number of tasks to the parallel processors is critical for achieving high performance in distributed systems, especially for high performance computing.

Task scheduling problem can be represented by a Directed Acyclic Graph (DAG). Each node of a DAG represents a task and a directed edge corresponds to the precedence constraint between two tasks. The general task scheduling problem includes the problem of processing each task on suitable processor and the problem of sequencing tasks so that precedent. The problem of optimal scheduling of tasks have been shown to be NP-complete in general [1,2], and optimal solutions can be achieved only after an exhaustive search. A typical performance indicator for efficient scheduling of a set of tasks on the available processors is the make span, i.e., the time needed to complete all the jobs with task precedence requirements being satisfied.

This problem has been extensively studied, and various algorithms which are mainly for homogeneous processors have been proposed in the literature. Heterogeneous environment [3] is a diverse set of heterogeneous processors interconnected with a high-speed network, and then makes high speed processing of computationally scientific and commercial applications like weather prediction, image processing, high-definition television, real-time and distributed database systems. Recently many researches are focus on the specific context of heterogeneous systems. Application scheduling is critical for achieving high performance in heterogeneous computing systems.

And then, many heuristics have been proposed for giving a near optimization in polynomial time. These heuristics are classified into a variety of categories such as list scheduling algorithms [4-8], clustering algorithms [9-11], Genetic algorithms [12-16] and task duplication based algorithms [17-19].

In list scheduling algorithms [4-8], the task in a list is constructed by assigning priority to it and each task is assigned to the processor based on its priority. List scheduling algorithms are generally preferred since they have been shown to have a good cost-performance trade-off between its cost and its performance. Several variant list scheduling algorithms have been proposed to deal with heterogeneous system, for example Mapping Heuristic (MH) [4], Levelized-MinTime (LMT) [5], Heterogeneous Earliest Finish Time (HEFT) [6]. The HEFT algorithm significantly outperforms the MH and LMT algorithm in terms of average schedule length ratio, speedup, etc. Ref. [7] first uses the (HEFT) algorithm to find an initial schedule and iteratively improves it. low complexity A Performance Effective Task Scheduling (PETS) algorithm [8] presents a low complexity algorithm for heterogeneous computing systems.

Clustering algorithms [9-11] reduce the communicating cost by scheduling heavily communicating tasks onto the same processor instead schedule them on other available processors. They try to balance the parallelism and the inter-task communication as three phase scheduling. Firstly, use linear or nonlinear

doi:10.4304/jsw.8.6.1443-1450

clustering heuristics to group tasks with large communication cost into a set of clusters (unbounded) and secondly, use communication sensitive or insensitive heuristics to assign the clusters onto the set of available processors. Thirdly do cluster merging or de-clustering based on the available number of processors. Task Duplication based scheduling Scheme (TDS) [10] and Clustering for Heterogeneous Processors (CHP) [11] are some well known clustering approach for solving task scheduling algorithm.

Genetic algorithms have recently been applied to the task scheduling problem and Job-shop scheduling problem as robust stochastic search in algorithms [12-16]. This class of methods is based on the principles of natural selection and natural genetics that combine the notion of survival of the fittest, random and yet structured search, and parallel evaluation of nodes in the search space. GA has been used to directly evolve task assignment and order in processors. GA is generally used in combination with other list scheduling techniques and to evolve the actual assignment and order of tasks into processors. Ref. [12] uses a genetic approach to match and schedule the tasks, Problem–Space Genetic Algorithm (PSGA) [13] is proposed for heterogeneous processors and an incremental genetic algorithm (GA) [14] for the homogeneous processors. A GA is used to evolve individuals consisting of multiple lists in [15]. A genetic algorithm is present in [16] for solving Job-shop Scheduling Problem (JSP). They are differ from each other for different coding scheme, initial population generation, chromosome selection and offspring generation strategies.

In task duplication based algorithms [17-19], the waiting time of the dependent tasks by duplicating the tasks on more than one processor. Critical Path Fast Duplication (CPFD) [17], Heterogeneous Critical Node First (HCNF) [18] and Task duplication Algorithm for Network of Heterogeneous system (TANH) [19] are a few algorithms proposed in the literature for heterogeneous system using task duplication.

This paper presents an efficient hybrid algorithm to solve the task scheduling problem that the tasks represented as a DAG are assigned onto heterogeneous processors. Although there a few PSO algorithms in the literature for flexible job-shop scheduling problems (FJSP), we present a genetic scheduling algorithm which is incorporated into PSO strategy. The objective of our algorithm is to find a schedule that minimizes the maximum completion time of all tasks. The genetic algorithm first uses some list scheduling strategies to generate feasible initial solution to lessen useless search. Crossover operator used in this paper can guarantee the valid of the new population. A new mutation operator is presented to maintain population diversity and overcome premature convergence. PSO strategy is used to improve the search efficiency by combining local search (by self experience) and global search (by social experience). A new PSO operator effectively reduces the makespan of the schedule by using the local minimum execution time and global minimum one as heuristic strategy. Our hybrid algorithm has been shown to be a valid and effective approach for task scheduling algorithm on heterogeneous distributed system.

This paper is organized as follows: The next section briefly describe the scheduling problem, and give the solution representation and fitness function. The detailed operators are given in Section 3. In Section 4 we study the performance of the algorithm for heterogeneous systems. Section 5 concludes the paper.

II. TASK SCHEDULING PROBLEM

A. Task scheduling Problem

A scheduling model $G=(T,P,C,E)$ consists of a set of tasks of an application, a target computing system and an evaluating rule. G can be stated as follows:

$T=\{ t_i, 1 \leqslant i \leqslant n\}$ is a set of n tasks,

$P=\{ p_j, 1 \leqslant j \leqslant m \}$ is a set of m independent different types of processors,

$E=\{ e_{i,j}, 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m\}$ is a matrix of computation times. $e_{i,j}$ is the computation time of task t_i on processor p_j, may be different on different processor depending on the processors computational capability.

$C=\{ c_{i,j}, 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m\}$ is a matrix of communication times. The communication cost of edge $e_{i,j}$ which is for transferring data from task t_i (scheduled on processor p_m) to task t_j (scheduled on processor p_n) is as in the equation (1).

$$c_{i,j} = \begin{cases} d_{i,j} / r_{m,n}, & if\ \mathrm{t_i}\ \text{is a predecessor of}\ \mathrm{t_j}, m \neq n; \\ 0, & else. \end{cases} \quad (1)$$

where $d_{i,j}$ is the amount of data required to be transmitted from task t_i to task t_j, and $r_{m,n}$ is the link communication speed between two processors p_m and processor p_n. In this study, the channel initialization time is assumed to be negligible. Otherwise, $d_{i,k} = 0$ when both the tasks t_i and t_j are assigned on the same processor. Further, for illustration, the data transfer rate for each link is assumed to be 1.0 and thus communication cost and amount of data to be transferred will be the same.

In a given task graph, a task without any parent is called an entry task and a task without any child is called exit task. Without loss of generality, it is assumed that there is one entry task to the DAG and one exit task from the DAG. If there are more than one exit (entry) task, they are connected to a pseudo-exit (pseudo-entry) task with zero computation time and communication time.

Hypotheses considered in this paper are the following:

- Setting up times of processors are negligible,
- A limited number of fully connected heterogeneous processors are independent from each other,
- Processors are continuously available and can only execute one task at a given time without interruption.
- There are precedence constraints among the tasks, and then the task cannot be scheduled until its predecessor tasks have been completed.
- Each task requires exactly one processor without interruption.

A set of processor assignments and execution times for the each task of an application is called a schedule. Given a schedule, the processor onto which a task t_i is assigned is denoted as $P(t_i)$, and the start time and finish time of the task ti on the processor $P(t_i)$ is referred to by $S(t_i)$ and $F(t_i)$ respectively. A schedule is obtained by generating a set of tasks assignments and execution sequences.

B. Solution Representation

In this paper, each solution for the problem is a vector of length 2n where n is the number of tasks to be scheduled. Each vector is a schedule $S=\{R,A\}$, where $R=\{ri, 1\leqslant i\leqslant n,\}$ is the execution order of the tasks to be scheduled and $A=\{a_i, 1\leqslant i\leqslant n,\}$ is the assignment of tasks onto the processors. The cells on a solution S determine which tasks are assigned to which processors. The order in which the cells appear on R determines the order in which the tasks will be performed on each processor. Individuals are read from left to right to determine the ordering of tasks on each processor.

TABLE I
A SAMPLE SCHEDULE

P1	t1			t3			T7
P2		t2	t6		t5	t4	

Table I illustrates the schedule for assigning a task graph with seven tasks to two processors according to a schedule $S=\{\{1,2,6,3,5,4,7\}, \{1,2,2,1,2,2,1\}\}$.

C. Fitness Function

The fitness function will be used the following genetic algorithm is the objective function of the task scheduling problem. It can use the throughput, finishing time and processor utilization for evaluating the quality of the solution. The fitness function used for our algorithm is to schedule the tasks of an application to heterogeneous processors such that the makespan of the tasks is minimized. Since reproduction operator of the genetic operators will try to maximize the fitness function, the fitness value of a schedule, S, is defined as Eq. (2)

$$f(S)=\sum_{i\in t}(\max t_i+\max c_i)-makespan(S) \quad (2)$$

where $\max t_i$ is the maximum execution time of the task i, and $\max c_i$ is the average communication time of the task i.

III. PSO-BASED GENETIC SCHEDULING ALGORITHM

Genetic algorithms are of the most widely studied guided random search techniques for the task scheduling problem. A simple example is provided in Fig. 1 to illustrate the operation of our generate operator. If the number of individuals breaking the precedence restrictions is relatively small, then it may be acceptable to allow the operator to generate infeasible individuals and discard them later. If the number of infeasible individuals is comparable to those feasible individuals, however, then too much computational time will be wasted in generating the infeasible individuals and checking them. In this paper, modified operators presented are guaranteed to be feasible by using some heuristic strategy. And a PSO strategy is presented by mapping the particle of PSO to the scheduled task. The detailed operator is described as follows.

A. Popolation Initialization

In the standard GA, initial population is often generated randomly. But we need to design the initial individual carefully as the individual representation space and the search space is not in one-to-one correspondence. In this paper, a new genetic operator will be presented to a feasible initial solution.

Since the randomly generated R in schedule S cannot maintain the precedence constraints among the tasks, we effectively use list scheduling strategies to generate feasible solutions instead directly applying local search strategy to position information. Our initiate operator can generate valid initial population effectively as list scheduling strategy is good at finding near optimum in low cost.

The initiate operator can be described as:

1) A set of random execution times $RE=\{re_i, 1\leqslant i\leqslant n,\}$which is lessen than the maximum execution time of all tasks is given to a set of tasks, re_i is thought as the average execution time of the task i in the heterogeneous processors.
2) The priority p_i of the task t_i is calculated as Eq. (3):
$$p_i=\max_{j\in succ(ti)}\{p_j+re_i\} \quad (3)$$
where $succ(t_i)$ is the set of immediate successor tasks of task i. The priorities of the tasks are calculated from the exit task to entry task based on the longest path to an exit node on the DAG.
3) The permutation of R is generated by sorting the tasks based on priority value.

Different combinations of priority values can be generate by giving random execution time to each task. Our generate operator can guarantee the validity of the schedules by using basic list scheduling techniques.

Fig.1. shows a DAG with seven tasks and eight edges which is labeled by the communication time. A simple example is provided in Fig. 1 to illustrate the operation of our generate operator.

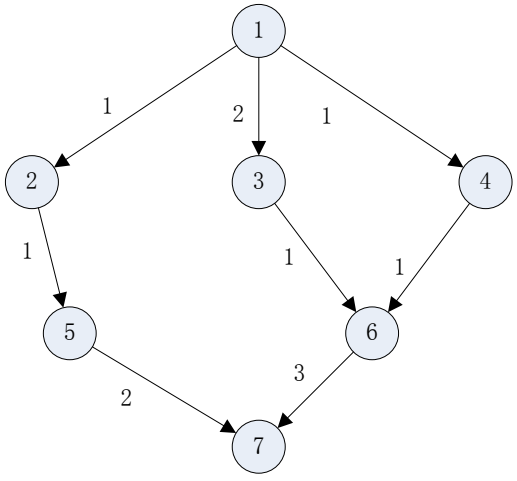

Figure 1. A sample task graph with 7 tasks

In the instance (n=7), priority information is P=[19, 12,14,16,10,12,7] based on the random average execution times of all task as {3,2,2,4,3,5,7}. p_1=7 is the priority of the task 7 as task7 is assigned a random value 7. A task order *R={1,4,3,2,6,5,7}* is obtained by using our generate operator, and task 1 is assigned firstly as p_1=12 is the priority of the task 1. If one task's priority is equal to another, the lower order number of task has priority.

Assume there are two processors available in the heterogeneous computing system, a random individual is obtained by combing *R={1,4,3,2,6,5,7}* with *A={1,2,2,1,1,2,2}*. Table II illustrates the schedule obtained by our generated operator.

The assignment permutation of S is generated by

TABLE II
SCHEDULE OF FIG. 1

P1	t1			t2	t6		
P2		t4	t3			t5	t7

assigning the random numbers which are less than the *m*, the number of the processors, to each element in the A. According to the schedule generated by the above strategy, we use an **ES** strategy to schedule the task t_i in *R* onto processor p_m in *A* in the same order. The start time and end time of task t_i , $S(t_i)$ and $P(t_i)$, are generated as its earliest start time and its earliest finish time on the processor p_j, $ES(t_{i,j})$ and $EF(t_{i,j})$. $ES(t_{0,j}) = 0$ for the entry task t_0, and $ES(t_{i,m})$ for the other tasks are computed recursively, starting from the entry task, as shown in Eq. (4).

$$ES(t_{i,m}) = \max\{c_{k,i} + ES(t_{k,n}), Av(p_m)\} \quad (4)$$

where $Av(p_m)$ *is* the earliest time that processor p_k completed the execution of the last assigned task, or the idle period between the assigned tasks that is enough to execute the task *i*. $EF(t_{i,m})$for the other tasks are computed as shown in Eq. (5).

$$EF(t_{i,m}) = w_i + ES(t_{i,m}) \quad (5)$$

where w_i is the random execution time of the task t_i, the task t_k is the immediate predecessor tasks of task t_i and $Av(p_m)$ is the earliest finish time of the task assigned on the processor p_m, or the earliest idle period between the already assigned tasks which is enough to complete the task t_i without avoiding the precedence restraints.

The **ES** schedule procedure is to assign the task in *R* from left to right onto the processor in *A* according to its earliest start time. The execution time of all tasks in Fig 1. is given in Table III.

Based on Table II and Table III, the start time and

TABLE III
EXECUTION TIME OF FIG 1.

V	1	2	3	4	5	6	7
P1	2	8	5	4	8	4	7
P2	6	7	3	6	6	3	5

finish time of all the tasks obtained by the **ES** strategy is showed in Table IV. The makespan of the schedule showed in Table IV is 26.

TABLE IV
SCHEDULE OF FIG1. BY ES STRATEGY

V	t1	t4	t3	t2	t6	t5	t7
P1	0-2			2-10	14-18		
P2		3-10	10-13			13-16	21-26

B. Crossover Operator

We use a crossover operator to evolve individuals consisting of multiple schedules, with first permutation in the schedule representing each task's priority and second permutation representing its assigned processor. We use random one-point crossover which randomly chosen a crossover point for the two parent individuals to generate the offspring.

For completing the unassigned positions on the operation sequence of the offspring, check all the operations of second parent from left to right. If corresponding operation is already assigned in the substring from first parent, skip to the next operation in operation sequence of second parent. Otherwise, place corresponding operation of the second parent for the position in offspring. The operations taken from second parent are the ones that proto-child needs.

1) Select a random point which is less than *n*, the number of the tasks, and cut the *R* into two halves,
2) Maintain the left half of the elements in the first individual *R1*,
3) Check *R2* from left to right. If the element is assigned in the let half of *R1*, skip to the next element in *R2*. Otherwise, place corresponding element of *R2* in *R1*,.
4) Obtain *A1* by maintaining the original mapping between the *R1* and *A1*,
5) Obtain *R2*, *A2* according to the steps 2), 3), 4).

Give the parent{{1,2,4,3,6,5,7},{1,2,3,1,2,2,1}}, and {{1,3,2,4,5,6,7},{1,1,2,2,1,1,2}}, child {{1,2,4,**3,6,5,7**}, {1,2,3,**1,2,2,1**}} and child {{1,3,2,**4,6,5,7**}, {1,1,2,**3,1,2,2**}} can be generated according to the random crossover is 3. The bold numbers in child show the elements generated by the crossover operator.

It is noted that a feasible solution is generated according the crossover operator since the precedence constraints are maintaining.

C. Mutation Operator

Mutation can be thought as an effectively escape method for premature convergence by randomly change the value of an individual. For maintaining the feasibility of the new generated individual, mutation operators here generate new individuals in the following two methods.

1) Mutation operator changes the assigned processor of the task within a single individual to another processor based on the mutation probability. In this way, an offspring with different assignment will be generated.

2) Mutation operator randomly changes the execution time of a task based on the mutation probability, and generate a permuted priority value by using the Eq. (3). Therefore the tasks will be scheduled in different order. In this way, an offspring with different scheduled order will be generated.

The mutation probability should be small to avoiding the large fluctuating of the quality of the population.

D. PSO Operator

The above operators which ensure the feasibility of individuals maybe restricts the actions of genetic operators. As a result, some parts of the search space maybe unreachable. To guarantee an evolved population with certain quality, a PSO operator based on Particle swarm optimization (PSO) [20] is presented in this paper. PSO is an optimization technique stimulating social behavior of the flying birds and their methods of information exchange. PSO algorithm improves the search efficiency by using the evolutionary computation which combining local best solution (local search) and global best solution (global search) together. PSO has been presented as an optimization technique in job shop problem [21]. In this paper, we use the PSO strategy to solve the task scheduling problem.

In PSO, each individual in the initial solutions called a flying particle whose velocity is dynamically changed according to the flying records of its local and its neighbors global. During the past few years, several models of PSO algorithm have been explored by researchers [22]. Normally the velocity is computed as:

$$v_{i,j+1} = wv_{i,j} + c1r1(LB_{i,j} - x_{i,j}) + c2r2(GB_j - x_{i,j}) \quad (6)$$

where *c1* and *c2* are constants called acceleration coefficients, *w* is called the inertia factor, *r1* and *r2* are two independent random numbers uniformly distributed in the range of [0, 1]. $LB_{i,j}$ is the local best position for particle *i* in the *jth* iteration, GB_j is the global best position for all particles in the *jth* iteration.

Thus, the position of each particle is updated in each generation according to the following equation:

$$x_{i,j+1} = x_{i,j} + v_{i,j+1}, j = 1,2,\cdots,d \quad (7)$$

The most difficult part in applying PSO successfully to our problem is to effectively map problem and generate solution. For finding good solutions for the task scheduling problem in acceptable time, we find a suitable coding between task scheduling problem solution and PSO particle. We sequence all the tasks of an application according to the position of the tasks order of processing time. The particle position can be generated stochastically according to the order of execution time of tasks on different processors. A particle position is corresponding to the assignment of the task. By generating the priority for each task, we convert the continuous position of particles to a permutation of tasks, *R* which is the first part of the schedule S. We compute the velocity of each task as the following model equations:

$$v_{i,j+1} = wv_{i,j} + c1r1(LB_{i,j} - t_{i,j}) + c2r2(GB_j - t_{i,j}) \quad (8)$$

The velocity of the task *i* in the *j+1th* iteration is calculated by using its local best execution time and its global best execution time in the *jth* iteration. The average execution time of the task *i* in the *j+1th* iteration is calculated as follows:

$$t_{i,j+1} = t_{i,j} + v_{i,j+1}, j = 1,2,\cdots,d \quad (9)$$

According to the Eq. (2), the priority of each task is generated by using the new average execution time. Then the tasks will be move toward a new position in permutation, *R*, and a schedule will be obtained by combing *R* and a random permutation *A*.

Generally, the value of each component in t_i by Eq. (8) can be clamped to the range [− *tmax*, *tmax*] to control excessive roaming of particles outside the search space, where *tmax* is the maximum execution time of all tasks. This process is repeated until a user-defined stopping criterion is reached.

E. Reproduction

Reproduction is a commonly used genetic operator. The reproduction process selects individuals from the current population to form the next population of individuals based on their fitness value. We generate the next generation of population by combining three groups of individuals together instead of just selecting individuals with better fitness value with a higher probability.

1) The first group of pNum×A% individuals is generated by using crossover operator,.
2) the second group of pNum × B% individuals is generated by doing PSO operator, they all selected individuals based on the roulette wheel.
3) The third group of pNum×C% individuals is generated by passing the individuals with best fitness value in the current generation to the next generation.

A,B,C are random numbers uniformly distributed in interval [0, 1], and their sum is equal to 1 . pNum is the number of initial population. These modifications will increase the performance of the genetic algorithm.

F. Algorithm Description

The process of implementing the PGA algorithm is as follows:

1) Initialize feasible population in the problem space according to our generate operator.
2) For each individual, schedule the tasks onto the processors and evaluate the desired optimization fitness function.
3) Crossover each individual and generate feasible new individual.
4) For each individual, schedule the tasks onto the processors.
5) Evaluate the desired optimization fitness function, and select pNum × A% individuals based on the roulette wheel.
6) Mutate the individual with a small probability.
7) Compare the execution time of each task with its LB. If current value is better than LB, then set LB value equal to the current value, and the LB position equal to the current position in problem space.
8) Compare the execution times of each task with its best execution time obtained so far. If current value is better than GB, then reset GB to the current execution time of the task.

9) Generate feasible new individual by using PSO operator.
10) Evaluate the desired optimization fitness function, and select pNum×B% individuals based on the roulette wheel.
11) Select pNum×C% individuals with best fit fitness.
12) Loop to step 2) until the fitness value cannot be improved or a specified number of generations is exceeded.

IV. Performance Analyses and Discussion

We have used Intel Xeon processors with 1 GHz speed for randomly generated application graphs. The comparisons of the algorithms are based on the following objectives:

1) *L(A)*, the makespan of the schedule *A*.
2) The improvement ratio (*IR(A,B)*), which is the ratio of the makespans difference between two schedules to the makespan of the first schedule.

$$IR(A,B) = (L(A) - L(B))/L(A) \quad (10)$$

Random DAGs with various characteristics will be generated according to the following parameters.

1) Number of tasks in the graph is a random number between 6 and 80.
2) The ratio of the depth to the width of the graph (DWR) can determine different shape of the graph. A dense graph is a short graph with high parallelism, while a sparse dense graph is a longer graph with a low parallelism. The value is a uniform distribution with range [0.8, 3].
3) The number of each task's successors is ranging from 1 to 4.
4) The ratio of the average communication time to the average execution time (CER). It can be thought as a computation-heavily application or communication-heavily application based on the ratio value that is between 0.1 and 2.
5) The range of execution time of each task (EC) , is a real random number in the range [0.5, 2]. It implies the range of the execution time in heterogeneous processors.

The genetic algorithm used the following parameters throughout the simulations:

population size = 50
crossover probability = 0.8
mutation probability = 0.04
maximum number of iterations = 500.

Tables V compare the near optimal schedule obtained by our PSA algorithm, and genetic algorithm [14] on random task graphs with no known optimal solutions. Tables V compare the near optimal schedule obtained by our PSA algorithm, and the list scheduling algorithm, HEFT [6] on random task graphs with no known optimal solutions.

HEFT algorithm selects the task with the so-called highest upward rank value at each step and assigns the selected task to the processor which minimizes its earliest finish time with an insert-based policy. The best obtained makespan by PGA and GA algorithm is presented in Table V. And he best obtained makespan by PGA and HEFT algorithm is presented in Table VI.

TABLE V
COMPARISON OF PGA AND GA ON RANDOM DAGS

Nodes	DWR	CER	L(GA)	L(PGA)	IR(GA,PGA)%
Group 1: EC=0.8					
7	0.9	0.2	28	27	3.57
17	1.2	1.8	45	44	2.22
23	0.8	0.9	61	58	4.92
36	0.9	1.7	80	75	6.25
41	1.4	1.1	118	113	4.24
59	2.8	0.7	179	174	2.79
63	2.1	1.3	246	239	2.85
75	1.8	0.4	321	308	4.05
Group 1: EC=1.8					
7	0.9	0.2	28	27	3.57
17	1.2	1.8	45	41	8.89
23	1.8	0.9	61	56	8.2
36	0.9	0.7	80	75	6.25
41	1.4	1.1	118	112	5.08
59	2.8	0.7	179	172	3.91
63	2.1	1.3	246	236	4.07
75	1.8	0.4	321	302	5.92

TABLE VI
COMPARISON OF PGA AND HEFT ON RANDOM DAGS

Nodes	DWR	CER	L(GA)	L(PGA)	IR(GA,PGA)%
Group 1: Ec=0.8					
7	0.9	0.2	29	27	6.9
17	1.2	1.8	47	43	8.51
23	0.8	0.9	62	58	6.45
36	0.9	1.7	81	75	7.41
41	1.4	1.1	123	113	8.13
59	2.8	0.7	187	174	6.95
63	2.1	1.3	265	239	9.81
75	1.8	0.4	334	308	7.78
Group 1: Ec=1.8					
7	0.9	0.2	29	27	6.9
17	1.2	1.8	47	41	12.8
23	1.8	0.9	62	56	9.68
36	0.9	0.7	81	71	12.3
41	1.4	1.1	123	112	8.94
59	2.8	0.7	187	172	8.02
63	2.1	1.3	265	236	10.9
75	1.8	0.4	334	302	9.58

The benchmarks, varying independently in DWR, CER, and EC, allow PGA and GA to be fairly evaluated by a test bench that does not take advantage of any special generating strategy.

For Tables V, the solution obtained by the PGA algorithm is better than the GA algorithm and is within 9% of the optimal schedule. For Tables VI, the solution obtained by the PGA algorithm is better than the HEFT algorithm and is within 13% of the optimal schedule with substantial improvements of at least 5%. As EC increased, the relative performance of the PGA algorithm increased as the improved performance of PGA is due to the heterogeneous. It is conceivable that as CER, and consequently the communication cost increases, the effects of PGA's become obvious relative to HEFT. It is similarly expected that as DWR increases the parallelism of the DAG will improving the performance of PGA.

V. Conclusion

In this paper, the problem of task scheduling is stated as minimizing the schedule length for the tasks scheduled on heterogeneous processors. A hybrid algorithm(PGA) is proposed by combining the PSO strategy and modified genetic operators together. Genetic algorithm is a well-studied heuristic for the task scheduling problem dedicated to minimize makespan of the schedule. A fairly expressive solution representation and the incorporation of smart heuristics, resulted in increased complexity. A generate operator is presented in this paper to generate feasible initial population by effectively using list scheduling strategy. And modified crossover and mutation operators facilitates the generation of the offspring and guarantees the feasibility of the new population. PSO strategy is incorporated to guiding the search more effectively based on local and global information together. A mapping mechanism is presented between the particle of swarm and the scheduled task and is used in the order of tasks to be scheduled without interference to the assignment of tasks.

For DAGs randomly generated under various parameters, PGA algorithm is experimentally shown to outperform a leading genetic algorithm and a leading list scheduling algorithm.

Acknowledgment

This work was supported in part by a grant from the Open Foundation of Key Laboratory in Software Engineering of Yunnan Province under Grant NO. 2011SE03, National Natural Science Foundation of China (Grant No. 60763008), and "CDIO-based software system modeling and design research and implementation" (Grant No. Rj14).

References

[1] Graham, R.L., L.E. Lawler, J.K. Lenstra and A.H. Kan, "Optimization and approximation in deterministic sequencing and scheduling: A survey." Ann. Discrete Math., pp. 287-326, 1979.

[2] Cassavant. T. and J.A. Kuhl, "Taxonomy of scheduling in general purpose distributed memory systems." IEEE Trans. Software Engg., vol. 14, pp. 141-154, 1988.

[3] Hui, C.C. and S.T. Chanson, "Allocating task interaction graphs to processors in heterogeneous networks." IEEE Trans. Parallel and Distributed Systems, vol. 8, pp. 908-926, 1997.

[4] EI-Rewini, H. and T.G. Lewis, "Scheduling parallel program tasks onto arbitrary target machines." J. Parallel and Distributed Computing, vol. 9, pp. 138-153, 1990.

[5] Iverson, M., F. Ozguner and G. Follen, "Parallelizing existing applications in a distributed heterogeneous environments." Proc. Heterogeneous Computing Workshop, pp: 93-100, 1995.

[6] Topcuoglu, H., S. Hariri and M.Y. Wu, "Performance effective and low-complexity task scheduling for heterogeneous computing." IEEE Trans. on Parallel and Distributed Systems, vol. 13(3), 2002.

[7] Liu G. Q., Poh K. L., and Xie M., "Iterative list scheduling for heterogeneous computing." J. Parallel and Distributed Computing, vol. 65, pp. 654-665, 2005.

[8] Ilavarasan E. and Thambidurai P., "Low complexity performance effective task scheduling algorithm for heterogeneous computing environments", J. Computer Sciences, vol. 3(3) , pp. 94-103, 2007.

[9] Kafil, M. and I. Ahmed, "Optimal task assignment in heterogeneous distributed computing systems." IEEE Concurrency,vol. 6, pp. 42-51, 1998.

[10] Ranaweera, A. and D.P. Agrawal, "A task duplication based algorithm for heterogeneous systems." Proc. IPDPS, pp. 445-450, 2000.

[11] Cristina Boeres, Jos´e Viterbo Filho and Vinod E. F. Rebello, "A cluster-based strategy for scheduling task on heterogeneous processors." Proc. 16th Symp. on Computer Architecture and High Performance Computing (SBAC-PAD), 2004.

[12] Wang, L., H.J. Siegel, V.P. Rowchoudhry and A.A. Maciejewski, "Task matching and scheduling in heterogeneous computing environments using a genetic algorithm-based approach." J. Parallel and Distributed Computing, vol. 47, pp. 8-22, 1997.

[13] Dhodhi, M.K., I. Ahmad, A. Yatama, "An integrated technique for task matching and scheduling onto distributed heterogeneous computing systems." J. Parallel and Distributed Computing, vol. 62, pp. 1338-1361, 2002.

[14] Annie, S.W., H. Yu, S. Jin, K.-C. Lin, "An incremental genetic algorithm approach to multiprocessor scheduling." IEEE Trans. on Parallel and Distributed Systems, vol. 15, pp. 824-834. 2004.

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

[16] Ye Li, Yan Chen, "A Genetic Algorithm for Job-Shop Scheduling." Journal of software, vol. 5(3) pp. 269-274, 2010.

[17] Braun, T.D., H.J. Siegel, N. Beck and L.L. Boloni et al., "A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems." Proc. 8th Workshop on Heterogeneous Processing, pp. 15-29, 1999.

[18] Ahmed, I. and Y. Kwok, "On exploiting task duplication in parallel program scheduling. IEEE Trans. on Parallel and Distributed Systems," vol. 9, pp. 872- 892, 1998.

[19] Basker, S. and SaiRanga, P.C. "Scheduling directed a-cyclic task graphs on heterogeneous network of workstations to minimize schedule length. "Proc. ICPPW, 2003.

[20] Kennedy J, Eberhart RC, and ShiY. Swarm intelligence. San Francisco: Morgan Kaufmann Publishers; 2001.

[21] Weijun X., and Zhiming W., "An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems", vol. 48, PP. 409-425, 2005

[22] Bou L., Ling W., and Yi-Hui J., " An effective hybrid PSO-based algorithm for flow shop scheduling with limited buffers", vol. 35, PP. 2791-2806, 2008

Yan Kang, Sichuan province, 1972.6. She received the BS degree and MS degree from Yunnan University, Yunnan and PhD from Institute of Software, Chinese Academy of Sciences, Beijing. She is the associate professor of Software Engineering Department, Yunnan University. She has more than 15 years of experience in the field of heuristic strategy and algorithm design. She is currently working on scheduling problem, and data mining.