Implementation of multi-objective evolutionary algorithm for task scheduling in heterogeneous distributed systems

Yuanlong Chen, Dong Li, Peijun Ma
Harbin Institute of technology ,Heilong jiang, China
E-mail:cyuanlong@126.com

***Abstract*—This paper presents an effective method for task scheduling in heterogeneous distributed systems. Its objective is to minimize the last task's finish time and to maximize the system reliability probability. The optimum is carried out through a non-domination sort genetic algorithm. The experimental results based on both randomly generated graphs and the graphs of some real applications showed that, when compared to two well known previous methods, such as heterogeneous earliest finish time (HEFT) algorithm and Critical Path Genetic Algorithm, this algorithm surpasses previous approaches in terms of both last task's finish time and the system reliability probability.**

***Index Terms*—DAG scheduling, ask graphs, heterogeneous system, non-domination sort genetic algorithm**

I. INTRODUCTION

Software engineer play an important role in control software especially in some safety critical system. Correct implementation of software ensures proper operation of these systems. EXCELLENCE task scheduling strategy will be to reduce the probability of these system's software error.

With the recent advancements in massive parallel processing technologies, the problem of scheduling tasks in multiprocessor system is becoming increasingly important. The problem of scheduling task graph of a parallel program ONTO a parallel and distributed computing system is a well defined NP-complete problem. This problem involves mapping a Directed Acyclic Graph (DAG) for a COLLECTION of computational tasks and their data precedence onto parallel processing systems.

Over the past few years, they have become the most attractive option for high performance computing and information processing. They have been increasingly employed for critical applications such as aircraft control, industrial process control, and etc. Increased commercialization of heterogeneous distributed computational systems pertains to the fact that ensuring system reliability is of critical importance. Therefore, the goal of a task scheduler is to assign tasks to available processors such that precedence requirements for these tasks are satisfied, with the overall execution length (i.e., makespan) minimized, while the reliability of the system is maximized.

Some scheduling algorithms are therefore proposed to deal with the heterogeneous SYSTEMS; for example, Mapping heuristic (MH)[1], dynamic level scheduling (DLS) algorithm, levelized min time (LMT) algorithm[2], Critical-path-on-a-Machine (CPOP) algorithm, and heterogeneous earliest finish time algorithm[3]. The HEFT algorithm significantly outperforms the DLS, MH, LMT and CPOP algorithms in terms of average schedule length ratio [4,5].The HEFT algorithm SELECTS the tasks with the so-called highest upward rank value at each step, and assigns the selected task to the processor to minimize its earliest finish time. Tasks mean computational time on all processors and the mean communication rates on all links were used to compute the upward rank value.

Recently, Genetic Algorithms (GAs) was widely reckoned as a useful meta-heuristics for obtaining high quality solutions for a broad range of combinational optimization problems which included task scheduling [6][7]. The GA operates on a number of solutions. Another merit of genetic is that its inherent parallelism can be exploited to further reduce its running time. However, Standard GA algorithms for task scheduling are monolithic, as they attempt to scan the entire solution. To enable the GA algorithm search the solution more effectively, CPGA was proposed .CPGA is based on standard GA algorithm with some heuristic principles that has been added to improve its performance [8].

Unfortunately, most of these algorithms can not minimize the execution length, and at the same time maximize the system's reliability. While this problem requires the simultaneous optimization of more than one non-commensurable and competing criterion, solutions to the multi-objective optimization problem are usually computed by combining them into a single criterion to be optimized. In this paper, a new modified GA algorithm is proposed, namely, HEFT-No dominated Sorting Genetic Algorithm (HEFT-NSGA) which seek to solve the multi-objective optimization problem in task scheduling.

This paper is organized as follows: Section 2 surveys the related work of our study. In section 3, our HEFT-NSGA for task SCHEDULING is presented. Experimental results are provided in Section 4 followed by conclusion in Section 5.

II. RELATED WORK

A task scheduling system model consists of an application, a target computing environment, and a performance criteria for scheduling.

doi:10.4304/jsw.7.6.1367-1374

A. Related definition

An application is represented by a directed acyclic graph, $G=(V,E)$, where V is the set of tasks and E is the set of edges between the tasks. Each edge $e(i,j)\in E$ represents the precedence constraint such that, task v_i should complete its execution before task v_j starts. $Data$ is an $m\times m$ (m is the number of tasks) matrix of communication data, where $data_{i,j}$ is the amount of data required to be transmitted from task v_i to v_j .

In a given task's graph, a task without any parent is called entry task, and a task without any child is called an exit task. In this paper, we have asserted that the task graph is a single-entry-single-exit task graph. If there are more than one entry (exit) task, they are connected to a zero-cost pseudo entry (exit) task with zero-cost edges, which do not affect the schedule.

We assume that the target computing environment consists of n heterogeneous processors, $p_1, p_2 \cdots p_n$, connected in a fully connected topology. Let W be an $m\times n$ computation cost matrix in which each $w_{i,j}$ gives the execution time to complete task v_i on processor p_j .

Each processor may fail due to hardware fault which result in task's failure. These faults may be transient or permanent and are independent. Each independent fault results in the failure of only one processor.

Basic Terminologies:

1) The feasible schedule S ensures that task's constraints between tasks of all tasks are met. A partial schedule is one which does not contain all tasks.

2) For a task v_i , $St(v_i)$ and $ft(v_i)$ are scheduled start time and scheduled finish time respectively. For a processor p_j , $St(p_j)$ and $ft(p_j)$ are the processor's start time (the time it takes to run a task) and FINISH time (the time it takes to complete a task) respectively.

3) For a task v_{ij} , p_j is its scheduled processor.

4) For a communication $e_{i,j}$, $comu_{i,j}$ is the communication's delay between task v_i and v_j , if TASKS v_i and v_j are scheduled on different processors, that is

$$comu(i,j)=data(i,j)\times d(p_k,p_h) \tag{1}$$

where task v_i is mapped onto processor p_k , task v_j is mapped onto processor p_h , and $d(p_k,p_h)$ is the time required to send a unit length data from p_k to p_h .

5) Processor p_j's ready time $R(p_j)$ is it's available time when it runs a task.

6) For a task v_i , $EST(i,j)$ and $EFT(i,j)$ are the scheduled earliest start time of task v_i on processor p_j , and the scheduled earliest finish time of task v_i on processor p_j respectively.

$$EST(i,j)=\max(\max(ft(k)+comu(k,i)) \tag{2}$$

$$R(p_j))v_k\in parent(v_i) \tag{3}$$

$$EFT(i,j)=EST(i,j)+w(i,j) \tag{4}$$

7) The Data Arrival Time (DAT) of v_i at processor p_j is defined as:

$$DAT(i,j)=\max(ft(k)+comu(i,k))$$

$$v_k\in parent(v_i) \tag{5}$$

If tasks v_k and v_i are scheduled on the same processor, then $comu(i,k)$ equals zero.

8) The parent task that maximizes the above expression is called the favored predecessors of v_i and it is denoted by $favored(v_i,p_j)$.

9)Let $F(a)=<f1(a),f2(a),f3(a)>$ and $F(b)=<f1(b),f2(b),f3(b)>$ be the vector values of the cost function F for solutions a and b respectively. Then, a dominates b if $fi(a)\le fi(b)$ for all $i(i=1,2,3)$ and either $f1(a)<f1(b)$ or $f2(a)<f2(b)$ or $f3(a)<f3(b)$.

10) Reliability Probability of Processor: The reliability probability of processor p during a time interval t is $e^{-\lambda_p t}$ [9, 10]. Under a task allocation S, the time required to execute all the tasks assigned to processor p is $\sum_{i=1}^{N}X_{ip}\cos t(i,p)$, if task v_i is scheduled on processor p , then $X_{ip}=1$, otherwise $X_{ip}=0$, then the corresponding processor reliability can be formulated as formula (7);

$$PR_p(S)=e^{-\lambda_p\sum_{i=1}^{N}X_{ip}E_{ip}} \tag{6}$$

11) Reliability probability of path e_{pq} during a time interval t is $e^{-\mu_{pq}t}$ [9, 10]. Under a task allocation S, the time required for data communication between the terminal processors p and q is $\sum_{i=1}^{N}\sum_{j\neq i}X_{ip}X_{jq}(data_{ij}/d(p,q))$; then, the corresponding path reliability can be given by formula (7);

$$PR_{pq}(S)=e^{-\mu_{pq}\sum_{i=1}^{N}\sum_{j\neq i}X_{ip}X_{jq}(dataq_{i,j}/d(p,q))} \quad (7)$$

12) System's reliability probability with the task allocation S is computed as follows:

$$R(S)=\prod_{p=1}^{P}R_p(S)\prod_{p=1}^{P}\prod_{q\neq p}R_{pq}(S)=e^{-count(X)} \quad (8)$$

$$count\ (S)=\sum_{i=1}^{N}\sum_{p=1}^{P}\lambda_p X_{ip}E_{ip}+\sum_{i=1}^{N}\sum_{j\neq i}^{N}\sum_{p=1}^{P}\sum_{q\neq p}\mu_{pq}X_{ip}X_{jq}\left(\frac{C_{ij}}{W_{pq}}\right) \quad (9)$$

The first term of the function $count(S)$ reflects the unreliability caused by the execution of tasks on processors of various reliabilities, and the second term reflects the unreliability caused by the inter-processor communication through different paths of various reliabilities.

Maximizing the system reliability is equivalent to minimizing $count(S)$.

B. The Heterogeneous-Earliest-Finish-Time (HEFT) Algorithm

The HEFT algorithm has two major phases: a task prioritizing phase for computing the priorities of all tasks and a processor selection phase for selecting the tasks in the order of their priorities, thereby scheduling each selected task on its best processor, which minimizes the task's finish time.

Task Prioritizing Phase: This phase requires the priority of each task to be set with the upward rank value, $rank_u$ which is based on mean computation and mean communication costs. The task list is generated by sorting the tasks by decreasing order of $rank_u$ .It can easily be shown that the decreasing order of $rank_u$ values provides a topological order of tasks, which is a linear order that preserves the precedence constraints.

Processor Selection Phase: The HEFT algorithm schedules the task on the processor on which the task has the earliest finish time.

The upward rank of a task v_i is recursively defined by formula (11) and (12).

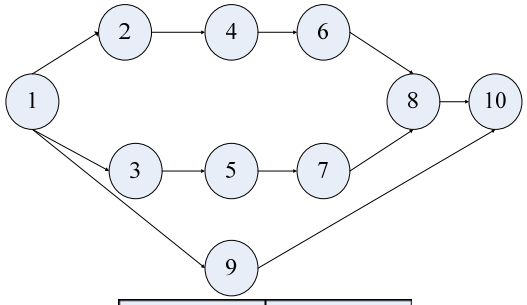
$$rank_u(n_i)=\overline{w_i}+\max_{n_j\in suc(n_i)}(\overline{w_{i,j}}+rank_u(n_j)) \quad (11)$$

$$rank_u(n_{exit})=\overline{w_{exit}} \quad (12)$$

C. The Critical Path Genetic Algorithm (CPGA)

The CPGA algorithm is considered as a hybrid of SGA principles and heuristic principles. The same principles and operators which are used in Standard Genetic Algorithm are used in the CPGA algorithm.

SGA algorithm is started with an initial population of feasible solutions. Therefore, by applying some operators, the best solution can be obtained after some generations. The selection of the best solution is determined according to the value of the fitness function. According to this, the chromosome is divided into two sections, the mapping and the scheduling sections. The mapping section contains the processor indices where tasks are to be run. The scheduling section determines the sequence for processing the tasks.Figure2 shows an example of such representation of the chromosome.



1	3
2	4
3	4
4	3
2	3
4	2
6	7
10	8
7	4

Figure 1. A task graph and the computation time on different processors

Sched j	1	3	2	9	5	4	7	6	8	10
Alloc j	p1	p1	p2	p1	p1	p2	p1	p2	p2	p1

Figure 2. Chromosome encoding

The same principles and operators which are used in SGA algorithm have been used in the CPGA algorithm. The encoding of the chromosome is the same as in SGA, but, in the initial population, the second part (schedule) of the chromosome can be constructed using ALAP [16].

In CPGA, three modifications have been applied in the SGA to improve the scheduling performance. These modifications are:

Reuse idle time:

The idle time of the processor is used to assign some tasks to idle time slots

Priority of the CPNs:

According to the modification, the initial population is produced using the following steps:

Initially, the entry task is the selected task and it is marked as a critical path task. An immediate task is marked as a critical path task. An immediate successor (of the selected task) that has the highest priority value is selected and it is marked as a critical path task. This process is repeated until the exit node is repeated. In each

generation of population, the critical path task is scheduled as early as possible.

Load balance:

The aim of load balance modification is to obtain the minimum schedule length and, at the same time, satisfy the load balance.

III. HEFT-NO DOMINATED SORTING GENETIC ALGORITHM (HEFT-NSGA)

Task scheduling is also a class of optimization problems. Existing scheduling algorithms handle task scheduling as a single objective optimization. But in many practical applications, multi-objectives need to be optimized at the same time.

Evolutionary algorithm has successfully been applied to the field of multi-objective optimization. In order to achieve global search evolutionary algorithm, maintain the composition of the population of potential solutions between generations， this means that, population to population is effective for searching best solutions to multi-objective optimization problems.

In the case of multiple objectives, there may not be one solution which is best in comparison to all other objectives. In a typical multi-objective optimization problem, there exist a set of solutions which are superior to the rest of the solutions in the search space when all objectives are considered, but they are inferior to other solutions in the space in one or more objectives. These solutions are known as Pareto-optimal solutions or non-dominated solutions. The rest of the solutions are known as dominated solutions. Since none of the solutions in the non-dominated set is absolutely better than any other, any one of them is an acceptable solution.

NSGA-Ⅱ is by far one of the best evolutionary multi-objective optimization algorithm[11].

The developed HEFT-NSGA algorithm is considered as a hybrid of the NSGA and the heuristic principles .On the other hand, the same principles and operators which are used in NSGA algorithm are also used in the HEFT-NSGA algorithm .The encoding of the chromosome is the same as in SGA.

A. Generating initial population

By performing the following steps, chromosomes would be created.

1) Set st(m)(processor's start processing time)=0 ft(m)(processor's finish processing time)=0;

2) Select a task v_i whose predecessors are scheduled;

3) Select the processor in which task v_i has the earliest finish time, however, if two or more processors have earliest finish times, select one of them in random;

4) if task v_i scheduled on processor p_k , set $ft(p_k) = ft(v_i)$;

5) Repeat steps 2 to 4 until all tasks are scheduled and new chromosomes are generated;

6) check the tasks in the chromosomes weather they satisfy the logical requirements;

7) Repeat steps 1 to 6 using the number of initial population.

B. HEFT-NSGA algorithm

The initialized population is sorted based on non-domination sort into each front. The first front being completely non-dominated set in the current population, and the second front being dominated by the individuals in the first front only and the front goes on and on. Each individual in the fronts are assigned rank (fitness) values based on the front which they belong to. Individuals in first front are given a fitness value of 1 and individuals in second are assigned fitness value as 2 and so on.

In addition to fitness value, a new parameter called crowding distance is calculated for each individual. The crowding distance is a measure of how close an individual is to its neighbors. Large average crowding distance will result in better diversity in the population.

Parents are selected from the population by using binary tournament selection based on the rank and crowding distance. An individual selected has either its rank lesser than the other or its crowding distance greater than the other. The selected population generates offspring from crossover and mutation operators.

The current parent population and current offspring is sorted again based on non-domination and only the best N individuals are selected, where N is the population size. The selection is based on rank and the crowding distance on the last front.

Step 1: Initialize the parameter and encode the chromosome;

Step 2: Generate Initial Population

Step 3: Non-Dominated sort: The initialized population is sorted based on non-domination;

Step 4: While stop criterion is not satisfied, do begin.

4.1) $P_{new} \leftarrow P_{current}$

4.2) repeat for ($N_P/2$) times,

$P_{Dad} \leftarrow select(P_{new})$;

$P_{mom} \leftarrow select(P_{new})$;

$P_{new} \leftarrow crossover(P_{Dad}, P_{mom})$;

End repeat;

4.3) for each chromosome $\in Pnew$ do begin

Mutate (chromosome);

End for.

4.4)Non-Dominated sort ($P_{current}$, P_{new}).

Step 5: Return the best N_p chromosomes

The non-dominated fast sort algorithm is described as below.

For each individual p in population P, do the following:

Initialize $S_p = \Phi$. (This set contains all individuals that are dominated by p .)

Initialize $n_p = 0$. (This is the number of individuals that dominate p .)

For each individual q in P,

If p dominated q then,

Add q to the set S_p i.e. $S_p = S_p \cup \{q\}$,

Else if q dominates p then

Increase the domination counter for p i.e. $n_p = n_p + 1$,

If $n_p = 0$ i.e. no individual dominate p then p belongs to the first front;

Set rank of individual p to one, i.e. $p_{rank} = 1$.

Update the first front set by adding p to front one, i.e. $F1 = F1 \bigcup \{p\}$.

This is carried out for all the individuals in the main population P.

Initialize the front counter to one.

While the ith front is non-empty, i.e. $F_i \neq \Phi$;

Set Q = ∅ . The set for storing the individuals for (i + 1)th front.

for each individual p in front F_i ,for each individual q in S_p (S_p is the set of individuals dominated by p), $n_q = n_q - 1$, decrease the domination count for individual q.

If $n_q = 0$, then none of the individuals in the subsequent fronts would dominate q.

Hence, set $q_{rank} = i + 1$.

Update the set Q with individual q i.e. $Q = Q \bigcup q$.

Increase the front counter by one.

Now the set Q is the next front and hence $F_i = Q$.

Crowding Distance: Once the non-dominated sort is complete, a crowding distance is assigned; the individuals are selected based on their ranks, and all individuals in the population are assigned a crowding distance value.

Comparing the crowding distance between two individuals in different front is meaningless. The crowing distance is calculated as follows;

For each front F_i , n is the number of individuals. Initialize the distance to zero for all the individuals i.e. $F_i(d_j) = 0$, where j corresponds to the jth individual in front F_i. For each objective function m, sort the individuals in front F_i based on objective m i.e. $I = sort(F_i, m)$ .Assign infinite distance to boundary values for each individual in F_i . $I(d_1) = \infty$ and $I(d_n) = \infty$

For $k = 2to(n-1)$,

$$I(d_k) = I(d_k) + \frac{I(k+1)*m - I(k-1)*m}{f_m^{\max} - f_m^{\min}} \quad (13)$$

Where, $I(k)m$ is the value of the mth objective function of the k^{th} individual in I .

The basic idea behind the crowding distance is to find the Euclidian distance between each individual in a front based on their m objectives in the m dimensional hyper space. Individuals in the boundary are always selected since they have infinite distance assignment.

Selection: Once the individuals are sorted based on non-domination and with crowding distance assigned, the selection is carried out using a crowded comparison-operator ($\prec_n$). The comparison is carried out as follows based on:

1) Non-domination rank p_{rank} i.e. individuals in front F_i will have their rank as $p_{rank} = i$.

2) Crowding distance $F_i(d_j)$, $p \prec_n q$,If $p_{rank} < q_{rank}$, or if p and q belong to the same front F_i then $F_i(d_p) > F_i(d_q)$ i.e.

The individuals are selected by using a binary tournament selection with crowed-comparison- operator.

Since standard genetic algorithm may require some time to find an ideal result, it is necessary to modify some principles .In HEFT-NSGA, once a task is selected to schedule on a processor, some steps are altered, and the pseudo code of this algorithm is as follows:

1) $\forall RT[P_j] = 0$, and RT is the ready time of the processors;

2) Let LT be a list of tasks according to the topological order of DAG;

3) For i=1 to m, and m is the number of tasks in DAG;

a) Remove the first task ti from list LT,

b) For j=1 to n ,and n is number of Processors,

If P_j can make task v_i complete as early as possible, scheduled v_i on p_j ,

$$ST[v_i] = \max\{RT[p_j], DAT(i, j)\}$$

$$FT[v_i] = ST[v_i] + w_{i,j}$$

$$RT[p_j] = FT[v_i]$$

END If

END For

END For

IV. Experimental Results and Discussion

In this section, we compare the performance of the HEFT-NSGA algorithm with two well-known scheduling algorithms in heterogeneous distributed system: the HEFT and CPGA algorithms. We consider two sets of graphs as the basis for testing the algorithms: randomly generated application graphs and graphs that represent some of the numerical real world problems.

A. comparison metrics

Comparisons of the algorithms are based on the following three metrics:

Makespan or scheduling length is defined as:

Makespan = EFT (v_{exit}),

Where EFT (v_{exit}) is the earliest finish time of the schedule exit task.

Schedule Length Ratio [SLR]. The main performance measure of a scheduling algorithm on a DAG graph is the schedule length (makespan) of its output schedule. Since a large set of application graphs with different properties are used, it is necessary to normalize the schedule length to the lower bound, which is called the Schedule Length Ratio (SLR). The SLR is defined as

$$SLR = \frac{makespan}{\sum_{v_i \in CP_{\min}} \min\{\cos t(v_i)\}} \qquad (14)$$

Where, $CP_{\min}$ is the critical path of the DAG when the task node weights are evaluated as the minimum computation cost among the eligible processors.

Reliability probability: The application reliability probability can be evaluated by the reliability of the exit task, and is defined as follows

Reliability probability = $p[V_{exit}]$.

B. Randomly generated application graph

In this study, we first considered the randomly generated application graphs such that, a random graph generator was implemented to generate weighted application DAGs with various characteristics that depended on several input parameters. The simulation based framework allows assigning set of values to the parameters used in the random graph generator.

For the generation of random graphs which are commonly used to compare scheduling algorithms [4，5, 12,13], five fundamental characteristics of the DAG are considered:

DAG size （ m ） :The number of tasks in the application DAG

Communication to computation cost ratio (CCR): this is defined as the ratio of the average communication cost to the average computation cost.

Computational cost heterogeneity factor, h : higher h value indicates higher variance for the computation cost of a task, with respect to the processor in the system and vice versa [5].

In all the experiments, only graphs with a single entry node and a single exit node were considered, as the input parameters were restricted to the following values:

$v \in \{20,40,60,80,100,120\}$

$h \in \{0.5,1.0,2.0\}$

$CCR \in \{0.5,1.0,1.5,2.0\}$

C. Random application performance results

The goal of these experiments is to compare the proposed HEFT-NSGA algorithm with the other two algorithms, HEFT and CPGA.

The performance of the algorithms was compared with respect to various graph characteristics. The first set of experiments compares the performance of the algorithms with respect to various CCR and graph size. The results are shown in Fig.3-5.According to the results ,when the CCR<1 the SLR-based ranking of the algorithms is {HEFT-NAGS,CPGA,HEFT},when the CCR>1 the SLR-based ranking of the algorithms is {HEFT-NSGA,HEFT,CPGA}.We also observe from results that HEFT-NSGA outperforms CPGA and HEFT algorithms in terms of the makespan, SLR, and reliability probability. By multiobjective genetic algorithm, HEFT-NSGA trying to find solutions that the schedule system can reach a better balance than other algorithms between objectives. We also can observe from Fig 4 and Fig 5 that from a certain indicators, HEFT-NSGA have improved not so obvious, but together, all indicators are better than other algorithms to improve.

D. Application Graphs of Real Word Problems

Using real applications to tests the performance of algorithms is very common [4,5,13,16,14 ,15].Hence, in addition to randomly generated DAGs, we also simulated two real-word problems: Gaussian elimination[4， 5， 13， 16],Fast Fourier transformation(FFT)[5,14].

For the experiments of Gauss elimination application, the same CCR and range percentage values(given in Section 5.2) were used. Since the structure of the application graph is known, we do not need the other parameters. A new parameter, matrix size(l),is used in place of m(the number of tasks in the graph).The total number of tasks in a Gaussian eliminateion graph is equal to $\frac{l^2 + l - 2}{2}$ [5].

For the comparison of SLR and the reliability probability, the matrix size used in the experiments is varied from 6 to 18, with an increment step of 2, and the number of processors is set to 4.The average SLR and reliability probability produced by each scheduling algorithm related to matrix size are shown in Fig 6.From Fig 6 we also observe that HEFT-NSGA outperforms CPGA and HEFT algorithms significantly.

The FFT algorithm consists of two parts: recursive calls and the butterfly operation. The task graph can be divided into two parts recursive call tasks and butterfly operation tasks. For an input size of vector ,there are recursive call tasks and butterfly operation tasks. Each path from the start task to any of the exit tasks in an FFT task graph is a critical path since the computation costs of tasks in any level are equal and the communication costs of all edges between two consecutive levels are equal[5].

For the FFT-related experiments, only the CCR and range percentage parameters, among the parameters given in section D,were used , as in the Gauss elimination application. According to the values of CCR we want, we generate DAGs with different number of tasks .Fig 7 demonstrates that NSGA-HEFT algorithms outperforms CPGA and HEFT algorithms in terms of makespan ,SLR and reliability probability.

V. CONCLUSION

With the development of parallel computing, distributed system applications have been greatly expanded. In some applications, scheduling system does not require only the fastest scheduling task, since the scheduling result is required to ensure the system's reliability probability to maximize at the same time. Often, existing algorithms do not take into account the mandate of the earliest completion time and system's reliability, since scheduling results obtained by these algorithms may be outstanding in one aspect, but not ideal for other areas. In this paper, multi-objective evolutionary algorithm and heuristic algorithm combined to make multiple objectives simultaneously optimized. The performance of HEFT-NSGA is compared to two of the existing scheduling algorithms: the HEFT and CPGA algorithms. The comparison is based on both randomly generated application DAGs and two real-world Gaussian elimination problems, and fast Fourier transformation .This simulation experiment results show that HEFT-NSGA algorithm outperforms both HEFT and CPGA algorithms in terms of scheduling length(makespan),scheduling length ratio(SLR),and reliability probability.

REFERENCE

[1] H. El-Rewini, T.G. Lewis, Scheduling parallel program tasks onto arbitrary target machines, J. Parallel Distrib. Comput. 9 (2) (1990) 138-153.

[2] M. Iverson, F. Ozuner, G. Follen, Parallelizing existing applications in a distributed heterogeneous environment, in: Proceedings of Heterogeneous Computing Workshop, 1995, pp. 93-100.

[3] P.Y.R. Ma, E.Y.S. Lee, M. Tsuchiya, A task allocation model for distributed computing systems, IEEE Trans. Comput. 31 (1) (1982) 41-47

[4] G.Q. Liu, K.L. Poh, M. Xie, Iterative list scheduling for heterogeneous computing, J. Parallel Distrib. Comput. 65 (5) (2005) 654-665

[5] H. Topcuoglu, S. Hariri, M.-Y. Wu, Performance-effective and low complexity task scheduling for heterogeneous computing, IEEE Trans. Parallel Distrib. Syst.13 (3) (2002) 260-274.

[6] Wu, A.S., H. Yu, S. Jin, K.-C. Lin, and G. Schiavone, 2004. "An Incremental Genetic Algorithm Approach to Multiprocessor Scheduling," IEEE Trans. Parallel and Distributed Systems, 15: 824-834.

[7] Kwok, Y. and I. Ahmad, 1999,"Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors," ACM Computing Survey, 31: 406-471.

[8] Fatma A.Omara,Mona M.Arafa,Genetic algorithms fo task scheduling problem.Journal of Parallel and Distributed Computing,70(2010):13-22

[9] Attiya, G., Hamam, Y., 2006. Task allocation for maximizing reliability of distributed systems: a simulated annealing approach. Journal of Parallel and Distributed Computing 66, 1259–1266

[10] Shatz, S.M., Wang, J.P., Goto, M., 1992. Task allocation for maximizing reliability of distributed computer systems. IEEE Transactions on Computers 41, 1156–1168.

[11] Deb K, Pratap A, Agarwal S, Meyarivan T. A fast and elitist multi-objective genetic algorithm: NSGA-II. IEEE Trans. On Evolutionary Computation, 2002,6(2):182−197.

[12] A. Dogan, F. Özgüner, Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing, IEEE Trans. Parallel Distrib. Syst. 13 (3) (2002) 308-323

[13] Mohammad I. Daoud, Nawwaf Kharma, A high performance algorithm for static task scheduling in heterogeneous distributed computing systems, J.Parallel Distrib. Comput. 68 (4) (2008) 399⊥ 409.

[14] Y. Chung, S. Ranka, Application and performance analysis of a compile-time optimization approach for list scheduling algorithms on distributed memory multiprocessors, in: Proc. Super Computing, 1992, pp. 512-521.

[15] C.M. Woodside, G.G. Monforton, Fast allocation of processes in distributed and parallel systems, IEEE Trans. Parallel Distrib. Syst. 4 (2) (1993) 164⊥ 174

[16] M. Wu, D. Dajski, Hypertool: a programming aid for message passing systems, IEEE Trans. Parallel Distrib. Syst. 1 (3) (1990) 330-343

Yuanlong Chen was born in 1981, and received his M.S. degree in 2007. He now works in school of computer science and technology,Harbin Institute of technology. He is a doctor and He is engage mainly in systems engineering and parallel computing.

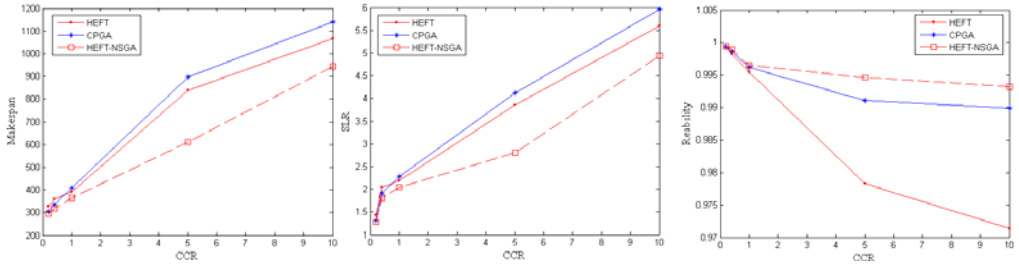


Figure 3. Makespan, SLR and Reliability probability of HEFT ,CPGA and HEFT-NSGA

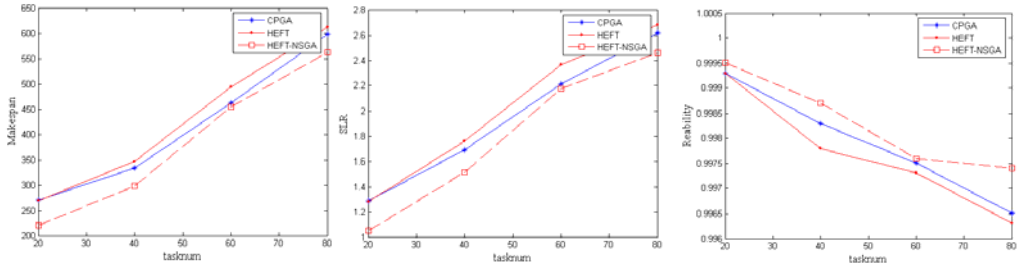


Figure 4. Makespan,SLR and Reliability probability of HEFT ,CPGA and HEFT-NSGA for CCR=0.4

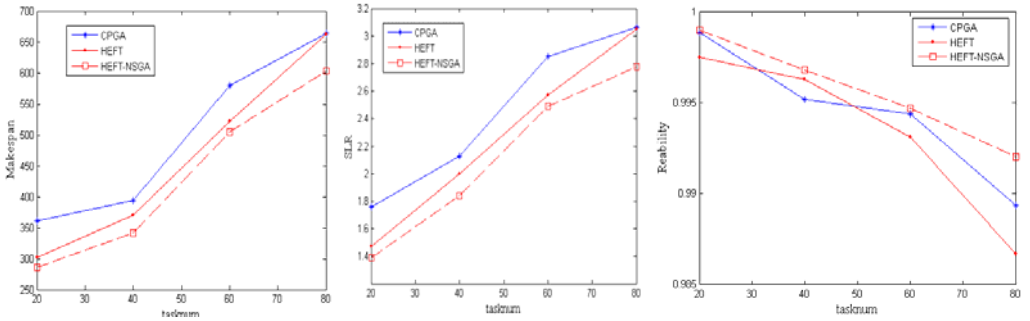


Figure 5. Makespan, SLR and Reliability probability of HEFT, CPGA and HEFT-NSGA for CCR=1.2

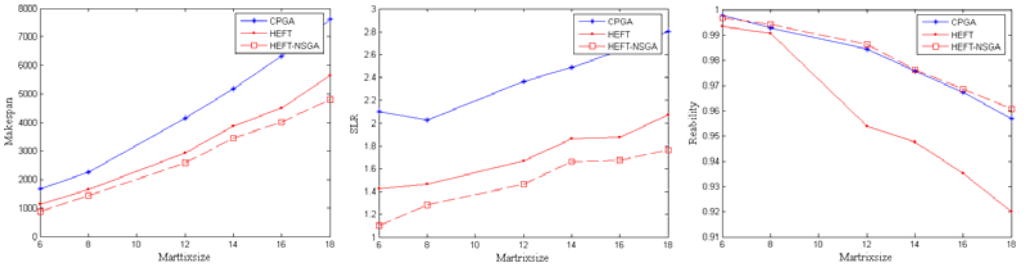


Figure 6. Makespan SLR and Reliability probability of CPGA,HEFT and HEFT-NSGA for Gaussian elimination

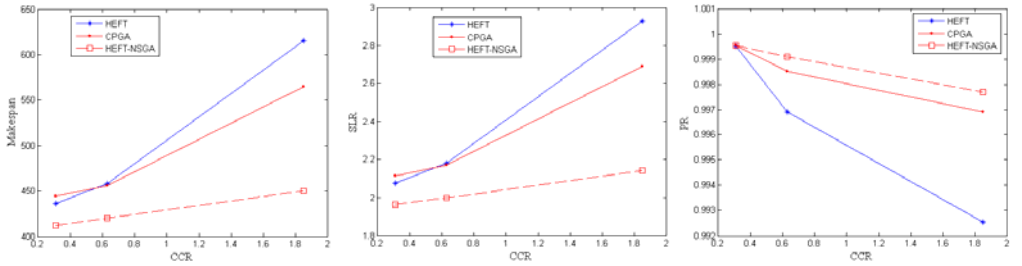


Figure 7. Makespan SLR and Reliability probability of CPGA,HEFT and HEFT-NSGA for FFT