Volume 4, Issue 2, February 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Task Scheduling in Distributed Systems using Discrete Particle Swarm Optimization

S.Sarathambekai*, K.Umamaheswari
IT Department,
PSG College of Technology, India

Abstract: Finding an optimal schedule of tasks for an application in distributed environment is critical in general. Task assignment is an extremely NP complete problem. This type of problem can be resolved by heuristic algorithms efficiently because the traditional methods such as dynamic programming and the back tracking need more time for solving this NP complete problem. Particle Swarm Optimization (PSO) is a newly developed meta-heuristic global optimization technique. It was originally designed only for continuous optimization problems. In task scheduling, the particles are represented as discrete values. It is obvious that the classical PSO cannot be used to solve discrete problems directly because its positions are real-valued. Some conversion techniques are needed to operate PSO in discrete domain. This paper presents a modified PSO called Discrete PSO (DPSO). In DPSO, no conversion techniques are needed because the velocity and positions are redefined to operate the PSO in a discrete domain directly. In this paper, the scheduler aims at minimizing make span, flow time and reliability cost simultaneously in distributed systems for scheduling of independent tasks using DPSO. Benchmark instances of Expected Time to Complete (ETC) model are used to test the DPSO. Based on the simulations and comparisons, the DPSO algorithm is viable approach for the task scheduling problem.

Keyword- Distributed system, Heterogeneous systems, Heuristic, Task Scheduling, Particle Swarm Optimization

I. INTRODUCTION

Heterogeneous Computing (HC) systems consist of mixed group of machines, communication protocols and programming environments and offer a diversity of architectural capabilities that has different execution requirements. One of the key challenges of HC system is the task scheduling problem. In general, scheduling is concerned with distribution of limited resources to certain tasks to optimize few performance criterions, like the completion time, waiting time. Task assignment problems can be classified into two categories based on the types of tasks [1]: scheduling a meta-task composed of independent tasks with no data dependencies and assigning an application composed of tasks with precedence constraints. There are more than a few conflicting objectives in multi-objective optimization problems to be optimized and it is hard to identify the best solution. For example, a bike manufacturer wish to maximize its turnover and minimize its manufacturing cost at a time. These objectives are conflicting to each other. A higher turnover would raise the manufacturing cost. There is no single optimal solution. The most traditional approach to solve a multi-objective optimization problem is to summative the objectives into a single objective by using a weighting sum.

Particle Swarm Optimization (PSO) is a population based heuristic robust stochastic optimization algorithm proposed by Kennedy and Eberhart [2] in 1995, motivated by the flocking behaviour of birds. This has been applied in wide area and different fields such as engineering, physics, mathematics, chemistry and etc. In PSO, each particle is a candidate solution in the search space. All particles have fitness values calculated by a fitness function, and have velocities to direct the flying of the particles. Compared with Genetic Algorithm (GA), PSO has some striking characteristics [3]. It has memory, and the knowledge of good solutions is shared by all particles. In this way, PSO can update its particles' positions according to individuals' memory and swarm's greatest information iteratively. With the collective intelligence of these particles, the swarm can converge to an optimum or near-optimum. PSO has a flexible and well-balanced method to improve and adjust to the global and local exploration and exploitation abilities within a short computation time. These characteristics make PSO highly reasonable to be used for solving single objective and also multi-objective optimization problems.

The performance of PSO greatly depends on its control parameters such as inertia weight and acceleration coefficients. Slightly different parameter settings may direct to very different performance. A significant development in the performance of PSO with adaptive inertia weight over the generations was suggested by J C Bansal [4]. The adaptive control parameter concepts have been used in PSO [1], [5] with Single Objective Optimization (SOO) problems. Initially the development of PSO has intended in continuous search space. Recently many researchers proposed different conversion techniques such as Smallest Position Value (SPV), Ranked-Order-Value (ROV) and Truncation of the real values for mapping continuous positions of particles in PSO to the discrete values. So, that the original PSO algorithm spends a lot of computation time in conversion of real values to integer values. Kang [1] proposed PSO called Discrete PSO, which can update their particles in the discrete domain directly without any conversion techniques. This paper uses

Discrete PSO with adaptive inertia weight to optimize multiple objectives and also uses the weighted aggregation method [6] for calculating the fitness value. The remainder of the paper is organized as follows: The Section specifies a problem statement. Section 3 reviews related algorithms for task scheduling problem. Section 4 presents the brief introduction to PSO. The proposed DPSO is presented in Section 5. Experimental results are reported in Section 6. Finally, Section 7 concludes the paper.

II. PROBLEM DEFINITION

A Heterogeneous Computing (HC) system consists of a number of connected heterogeneous Processor Elements (PEs). Let T = $\{T_1, T_2\ldots, T_n\}$ indicate the n number of independent tasks to be scheduled on m processors P = $\{P_1, P_2\ldots, P_m\}$. Because of the heterogeneous nature of the processors and disparate nature of the tasks, the expected execution times of a task executing on different processors are different. Every task has an Expected Time to Compute (ETC) on a specific processor. The ETC values are assumed to be known in advance. An ETC matrix is an *n x m* matrix in which m is the number of processors and n is the number of tasks. One row of the ETC matrix represents estimated execution time for a specified task on each PE also one column of the ETC matrix consists of the estimated execution time of a specified PE for each task.

The task scheduling problem is formulated based on the following assumptions:

1. All tasks are non pre-emptive
2. Every processor can execute only one task at a time.
3. Every task is processed on one processor at a time.

This paper presents the scheduling of independent tasks on a set of heterogeneous processors in order to minimize the make span, reliability cost and flow time simultaneously.

Most popular optimization criterion is minimization of make span [1] i.e. the finishing time of the newest task. Make span computes the throughput of the HC system. Assume that $C_{i,j}$ (i ε{1,2,...,n}, j ε {1,2,...,m}) is the execution time for performing i[th] task in j[th] processor and W_j (j ε {1,2,...,m}) is the previous workload of P_j. According to the above definition, make span can be estimated using the equation (1).

Make span=max { $\sum C_{i,j} + W_j$ } j ε (1, 2, 3…, m) (1)

$\forall$ task i allocated to processor j

Reliability is defined to be the probability that the system will not fail during the time that it is executing the tasks. The Reliability Cost [7, 8] as like a meter of how reliable a given system is when a group of tasks are allocated to it. The lesser the reliability cost increases the reliability. In this model, processor failures are assumed to be independent, and follow a Poisson Process with a constant failure rate. Failures of communication links are not considered here. The reliability cost of a task T_i on a processor P_j is the product of P_j's failure rate (PFR) λ_j and T_i 's execution time on j .Thus, the reliability cost of a schedule is the summation over all tasks' reliability costs based on the given schedule. According to the above definition, the reliability cost is defined in the equation (2), where $X(T_i) = j$ indicates that task T_i is allocated to P_j

Reliability Cost = $\sum_{j=1}^{m} \sum_{X(T_i)=j} \lambda_j C_{ij}(T_i)$ (2)

Flow time [9] is the sum of the finishing times of tasks. Flow time measures the Quality of Service of the HC system. The flow time can be estimated using the equation (3), where $F_{i,j}$ is the finishing time of $task\ T_i$ on a processor P_j

Flow time= $\sum_{j=1}^{m} \sum F_{i,j}$ (3)

$\forall$ task i allocated to processor j

III. RELATED WORK

In general, finding optimal solutions for the task assignment problem in a HC system is NP-complete. Therefore, only small-sized instances of the problem can be solved optimally using precise algorithms. For large scale instances, most researchers have spotlighted on developing heuristic algorithms that give up near-optimal solutions within a reasonable computation time. Braun [10] elucidated 11 heuristics for scheduling tasks and assessed them on different types of heterogeneous computing environments. The 11 heuristics examined are Opportunistic Load Balancing, Minimum Execution Time, Minimum Completion Time, Min-min, Max-min, Duplex, Genetic Algorithm, Simulated Annealing, Tabu, and A* .The authors illustrated that the Genetic Algorithm can obtain better results in comparison with others. The above stated heuristics intended to minimize a single objective, the make span of the schedule.

Izakian [11] recommended an efficient heuristic called min-max for scheduling meta-tasks in heterogeneous computing systems. The effectiveness of proposed algorithm is investigated with 5 popular pure heuristics min-min, max-min, LJFR-SJFR, sufferage, and Work Queue for minimizing make span and flow time. The author also considers the effect of these pure heuristics for initializing Simulated Annealing (SA) meta-heuristic approach for task scheduling on heterogeneous environments.

Meta-heuristic algorithms have been initiated to reach a better solution quality for the task scheduling problem such as SA, Tabu Search, GA and Swarm Intelligence (SI). SI consists of two successful techniques of Particle Swarm Optimization (PSO) and Ant Colony Optimization algorithm (ACO). Abraham [12] stated the usage of a number of nature inspired meta-heuristics (SA, GA, PSO, and ACO) for task scheduling in computational grids using single and multi-objective optimization techniques. PSO yields faster convergence when compared to GA, because of the balance between exploration and exploitation in the search space.

The inertia weight in PSO significantly affects the convergence and exploration-exploitation of search process. So, that the performance of PSO greatly depends on its control parameter such as inertia weight. Slightly different parameter setting may direct to very different performance in PSO. Kennedy [2] developed PSO with no inertia weight. Shi and Eberhart [13] first time presented the concept of inertia weight with constant value. Further, many researchers introduced dynamical adjusting of inertia weight which can increase the capabilities of PSO. Bansal [5] presented a comparative study on 15 strategies to set inertia weight in PSO. The author concluded chaotic inertia weight is the best strategy for better accuracy and random inertia weight strategy is best for better efficiency. Kaushik [14] proposed an adaptive inertia weight which is the Euclidean distance of the particles of a particular generation from the global best. Xin [15] presented Linearly Decreasing Inertia weight for enhancing the efficiency and performance of PSO.

The main advantages of PSO algorithm are précised as: simple concept, easy implementation, robustness to control parameters, and computational effectiveness when compared with mathematical algorithm and other heuristic optimization techniques [16]. However, these greater characteristics make PSO a highly feasible candidate to be used for solving multi-objective optimization problems. In fact, there have been several recent proposals to extend PSO to handle multi-objectives: The swarm metaphor of Ray and Liew [17], Dynamic neighbourhood PSO proposed by Hu and Eberhart [2], the Multi-objective PSO (MOPSO) by Coello and Lechuga [18].

Different criteria can be used for evaluating the effectiveness of scheduling algorithms. All the existing works investigated a number of these heuristics for minimizing make span or make span and flow time, However no attempts has been made to minimize make span, flow time and reliability cost simultaneously for scheduling meta tasks on heterogeneous systems using DPSO.

IV. PARTICLE SWARM OPTIMIZATION

PSO is an optimization algorithm based on population. The system is initialized with a population of random solutions (particles). The population in PSO is called a swarm. Each particle moves in the D-dimensional problem space with a velocity. The velocity is dynamically changed based on the flying knowledge of its own (Personal best) and the knowledge of the swarm (Global best). The velocity of a particle is controlled by three components, namely, inertial momentum, cognitive, and social. The inertial component simulates the inertial behaviour of the bird to fly in the previous direction. The cognitive component models the memory of the bird about its previous best position, and the social component models the memory of the bird about the best position among the particles.

PSO is different from other GA. It does not have the selection, crossover and mutation operators. It means that the members of the entire swarm are preserved through the search procedure, so that information is socially shared between particles to direct the search towards the optimum position in the search space. PSO can be easily implemented because it has no filtering operators (selection, crossover and mutation). It is computationally economical because its memory and CPU speed necessities are low [19].

The movement of the particle towards the best solution is directed by updating its velocity and position characteristics. The velocity and position of the particles are updating by using the equation (4) and (5), where i=1, 2, 3…POP, j=1, 2, 3…D, POP is the number of particles in the swarm, W is the inertia weight which is used to control the impact of the previous history of velocities on the current velocity of a given particle, $V_i^t(j)$ is the j^{th} element of the velocity vector of the i^{th} particle in t^{th} iteration which determines the direction in which a particle needs to move, $present_i^t(j)$ is j^{th} element of i^{th} particle (solution) in t^{th} iteration. r_1 and r_2 are random values in range[0, 1] sampled from a uniform distribution, C_1 and C_2 are positive constants, called acceleration coefficients which control the influence of Personal best (Pbest) and Global best (Gbest) on the search process.

$$V_i^{(t+1)}(j) = W V_i^t(j) + C_1 r_1 (Pbest_i^t(j) - present_i^t(j)) + C_2 r_2 (Gbest^t(j) - present_i^t(j)) \quad (4)$$

$$present_i^{(t+1)}(j) = V_i^{(t+1)}(j) + present_i^t(j) \quad (5)$$

The pseudo code of classical PSO algorithm for task scheduling is given in Fig 1.

```
begin
    Randomly initialize the swarm;
    Position and velocity of the particle is initialized randomly;
    Calculate fitness value of each particle and find the Pbest and the Gbest;
      repeat
         Velocity and Position of each particle is updated using (4) and (5).
         Evaluate fitness value of each particle.
        Update Pbest for each particle.
        Update Gbest.
      until stopping condition is true;
```

Fig 1.Pseudo code of classical PSO algorithm

V. Proposed Discrete Particle Swarm Optimization

The classical PSO was originally designed only for continuous optimization problems. This cannot be used to solve discrete problems directly because its positions are real-valued, so that the conversion techniques are needed to operate PSO in discrete domain. In proposed PSO, no conversion techniques are needed because the velocity and positions are redefined to operate in a discrete domain directly and hence much computation time can be saved. The asymptotic complexity of the proposed algorithm has same as the original PSO because the algorithm follows the same pseudo code of the classical PSO in Fig 1. Only the way of updating the velocity and position are different from classical PSO. The proposed DPSO called Linearly decreasing Inertia weight DPSO (LIDPSO) because the value of inertia weight (W) is varying linearly from large value to small value instead of constant value in classical PSO. The flow diagram of LIDPSO is shown in Fig.2

The equation (6) and (7) shows the updation of the particle's velocity and position in discrete domain. In LIDPSO, the above equation (4) and (5) are rewritten in equation (6) and (7). [11], [22]

$$V_i^{(t+1)}(j) = WV_i^t(j) \cup C_1 r_1 (Pbest_i^t(j) - present_i^t(j)) \cup C_2 r_2 (Gbest^t(j) - present_i^t(j)) \quad (6)$$

$$present_i^{(t+1)}(j) = present_i^t(j)\ (swap)\ V_i^{(t+1)}(j) \quad (7)$$

The $swap$ operator in equation (7) is defined as follows:

Consider the multi-processor scheduling with n tasks and N particles (N is a population size). A particle P is a list of n tasks. A new particle P' is obtained when exchanging task n_i and task n_j in particle P. The swap operator, for example (n_i, n_j) then the swap operation is denoted in equation (8)

$$P' = P + SO(n_i, n_j) \quad (8)$$

For example:

Particle 1: (1, 2, 3, 4)

Particle 1's new position: (1, 2, 3, 4) + SO (1, 3) = (3, 2, 1, 4)

A Set of Swap Operator (SSO) is created, when $SO_1, SO_2, \dots SO_n$ are imposed to a particle continuously. This process can be depicted in equation (9).

$$SSO = (SO_1, SO_2, \dots SO_n) \quad (9)$$

The equation (8) is rewritten in equation (10).

$$P' = P + SSO \quad (10)$$

Assume A and B are two vectors. An impose of SSO to A and B when updating the position of the particle.(i.e.) $B = A + SSO$. So minus operation in equation (6) between two vectors is defined in equation (11)

$$B = A + SSO \Leftrightarrow A - B = SSO \quad (11)$$

In equation (6), $(Pbest_i^t(j) - present_i^t(j))$ and $(Gbest^t(j) - present_i^t(j))$ are defined as SSO_2 and SSO_3 respectively. The new velocity $V_i^{(t+1)}(j)$ consists of three SSO's: old velocity (SSO_1), $Pbest_i^t(j) - present_i^t(j)$ and $Gbest^t(j) - present_i^t(j)$. The "U" operator act as a merging of three SSO's into single SSO called new velocity. The equation (6) is rewritten in equation (12).

$$V_i^{(t+1)}(j) = SSO_1 \cup SSO_2 \cup SSO_3 \quad (12)$$

Inertia Weight plays an important role to attain a good balance between the exploration and the exploitation of the search space. A high inertia weight is more suitable for global search and a small inertia weight helps local search. The LIDPSO uses the inertia weight in (6), which linearly decreases from large value to small value through the search process of identifying the global optima. The linearly decreasing inertia weight is calculated in equation (13), where $iter\_max$ is the maximum number of iterations and $Curr\_iter$ is the current iteration number. Typically, this algorithm started with a large inertia weight (W_{max}), which is decreased over time. The value of W_t is permitted to reduce linearly with iteration from W_{max} to W_{min}.

$$W_t = (W_{max} - W_{min}) \frac{Curr\_iter}{iter\_max} + W_{min} \quad (13)$$

The performance of LIDPSO is compared with Constant control parameters DPSO called CDPSO and Linearly decreasing Inertia with Time varying acceleration DPSO called LTDPSO. In CDPSO, the control parameters W, C_1 and C_2 are constant during the whole run of the algorithm. In LTDPSO algorithm, the inertia weight is calculated using equation (13) and acceleration coefficient C_1 and C_2 are calculated using equation (14) and (15), where $iter\_max$ is the maximum number of iterations and $Curr\_iter$ is the current iteration number. Larger values of C_1 guarantee larger deviation of the particle in the search space, while the larger values of C_2 signify the convergence to the present global best (gbest). C_1 has been permitted to reduce from its initial value of C_{1_final} to $C_{1\_initial}$ while C_2 has been raised from C_{2_final} to $C_{2\_initial}$.

$$C_{1\_t} = \left(C_{1\_final} - C_{1\_initial}\right)\frac{Curr\_iter}{iter\ max} + C_{1\_initial} \quad (14)$$

$$C_{2\_t} = \left(C_{2\_final} - C_{2\_initial}\right)\frac{Curr\_iter}{iter\ max} + C_{2\_init} \quad (15)$$

A. Particle representation and population initialization

The DPSO begins from a random initial population (Swarm) like other evolutionary algorithms. Population initialization consists of two parts: Particle generation and Processor allocation. Number of tasks and population size are required to generate particles. The initial population consists of randomly generated particles. The individual position is obtained from the task permutation algorithm. For the permutation form, the position of a task in the permutation vector represents the sequence the task is scheduled, and the corresponding value of each element indicates a node index number.

In this paper, the solutions (Particles) are represented by permutation-based definition.

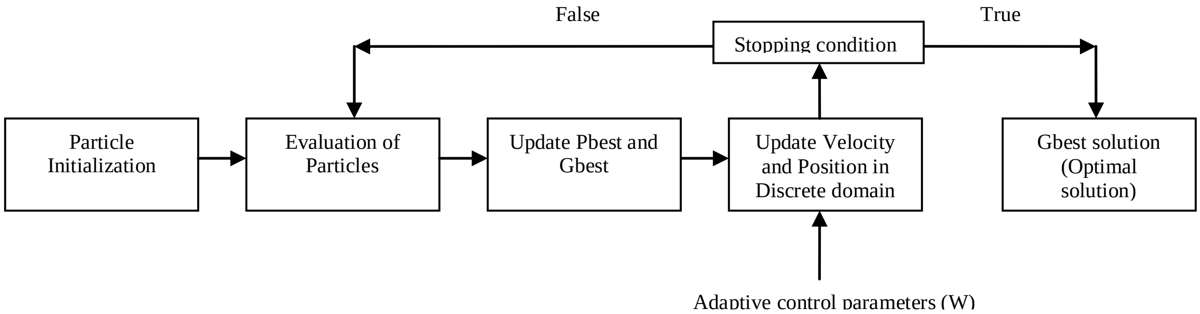


Fig.2.Flow diagram for LIDPSO

This definition provides n! number of solutions. The example is represented in Fig 3(a).

After generating the particles, the processors are allocated to the task in a particle, randomly. Generated particles and number of processors are required to generate processor allocation matrix. If a task assigned to a processor is represented by '1' further it's not assigned to any other processors. The example is represented in Fig 3(b).It shows the scheduling for the generated particle 1 in Fig 3(a) can be scheduled on 2 processors P1 and P2.

B. Particle Evaluation

The three objectives, make span, flow time and reliability cost are calculated as given in equation (1), (2) and (3). Randomly Assigned Weighted Aggregation (RAWA) method [2] is used to calculate the weights for DPSO. For the RAWA, the weights can be generated in the equation (16), (17) and (18).

$$W_1\ (t) = random\,(\lambda) \quad (16) \qquad W_2\ (t) = (1.0 - W_1\ (t))random(\lambda) \quad (17)$$

$$W_3\ (t) = 1.0 - W_1\ (t) - W_2 \quad (18)$$

The function $fit(swarm)_{sum}$ is a sum of three objectives, the make span, reliability cost and flow time. For three objective functions, the weighted single objective function $fit(swarm)_{sum}$ is obtained using the equation (19).

$$fit\,(swarm)_{sum} = W_1\ Makespan + W_2\ Flowtime + W_3\ Reliability\ Cost \quad (19)$$

C. Particle's Movement

The particle position is updated during the each iteration based on two types of experiences: personal best and global best experiences. The personal best experience ($Pbest_i^t$) is the experienced position by particle $present_i^t$ which obtains the smallest fitness value during flying. The $Gbest$ represents the best particle found in the entire population of each generation. For each iteration, the particle modifies its velocity and position through each dimension j by referring to $Pbest_i^t$ and the swarm's best experience $Gbest^t$ using equation (6) and (7).

VI. EXPERIMENTAL EVALUATION

The experimental results are attained using a set of benchmark instances [20] for the distributed heterogeneous systems. All algorithms are coded in C and executed on an Ubuntu platform.

A. *Benchmark description*

The simulation is performed on the benchmark [20] instances which are categorized in 12 types of ETC's based on the 3 following metrics: task heterogeneity, machine heterogeneity and consistency. In this benchmark, quality of the ETC matrices are varied in an attempt to simulate various possible heterogeneous computing environments by setting the values of parameters $mean_{task}$, V_{task} and $V_{machine}$, which represent the mean task execution time, the task heterogeneity, and the machine heterogeneity respectively. In ETC matrices, the amount of variance among the execution time of tasks in the meta-task for a given processor is defined as task heterogeneity. Machine heterogeneity represents the distinction among the execution times for a given task across all the processors [20]. The Coefficient of Variation Based (CVB)

ETC generation method gives a larger control over the spread of execution time values than the common range based method proposed by Braun [13].

The CVB type ETC matrices generation method works as follows: First, a column vector of the expected task execution time with the preferred task heterogeneity, s, is created following gamma distribution with mean mean_{task} and stand deviation $\mathrm{mean}_{task} \times V_{task}$. The input parameter V_{task} is desired coefficient of variation of values in s. The value of V_{task} is high for high task heterogeneity, and small for low task heterogeneity. Each element of s is then used to produce one row of the ETC matrix following gamma distribution with mean q[i] and standard deviation s[i] $\times V_{machine}$ such that the desired coefficient of variation of values in each row is $V_{machine}$. The value of $V_{machine}$ is large for high machine heterogeneity, and small for low machine heterogeneity. Task and machine heterogeneities are modelled by using different V_{task} and $V_{machine}$ values: high heterogeneity is represented by setting V_{task} and $V_{machine}$ equal to 0.6, and low heterogeneity is modelled using V_{task} and $V_{machine}$ equal to 0.1. [1]

a) Particle Initialization:

- Number of Task= 4 (1,2,3,4)
- Population Size= 3

Particle 1: 1 2 3 4
Particle 2: 3 1 2 4
Particle 3: 2 4 3 1

b) Processor Allocation:

P1: 1 0 1 0
P2: 0 1 0 1

ETC Matrix: (Execution time in seconds)

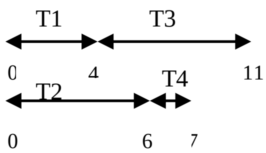
	P1	P2
T1	4	5
T2	3	6
T3	7	4
T4	8	1

c) Objective Calculation:

PFR for P1=0.00000095; PFR for P2=0.000001s

Particle 1: **P1**

P2



Make span=11 secon

Reliability cost:

P1=(4*0.00000095)+(7*0.00000095) =0.00001045
P2=(6*0.000001)+(1*0.000001)=0.000007

Reliability cost= 0.00001045+0.000007=0.00001745

Flow time=4+6+11+7=28 seconds

Fig.3. (a) Particle Initialization (b) Processor allocation (c) Objective calculation

To capture other possible characteristics of real scheduling problems, three different ETC consistencies namely consistent, inconsistent and semi-consistent are used. An ETC matrix is considered consistent if a processor P_i executes task T_j faster than processor P_j, then P_i executes all the jobs faster than P_j. Inconsistency indicates that a processor is quicker for a few jobs and slower for some others. An ETC matrix is considered semi-consistent if it includes a consistent sub-matrix. A semi consistent ETC matrix is characterized by an inconsistent matrix which has a consistent sub-matrix of a predefined size.

B. *Algorithms comparison*

Simulations were carried out to compare the performance analysis of LIDPSO with respect to: a) CDPSO b) LTDPSO

All the algorithms are stochastic based algorithms. Each independent run of the same algorithm on a particular problem instance may yield a different result. To make a good comparison of the algorithms each experiment was repeated 10 times with different random seeds and the average of the results are reported.

C. *Parameter setup*

The following parameters are initialized for simulating the CDPSO, LIDPSO and LTDPSO algorithms.

- Population size (N) = 100 and Number of iteration =50 for all the algorithms.
- The Failure rate for each processor is uniformly distributed [10, 11] in the range from 0.95×10^{-6} /h to 1.05×10^{-6}/h.
- The values of control parameters for CPSO are W=0.8, C_1 = 1 and C_2=1
- Values for linearly decreasing inertia weight and time varying acceleration coefficients [5]:
  - Inertia weights W_{max} =0.8 and W_{min} =0.
  - Acceleration coefficients $C_{1\_initial}$ = 2.5, C_{1_final} = 0.5, $C_{2\_initial}$ = 0.5, C_{2_final} = 2.5. C_1 has been allowed to decrease from its initial value of 2.5 to 0.5, while C_2 has been increased from 0.5 to 2.5

D. *Performance comparisons*

To make the comparison fair, the swarms for all the methods were initialized using the same random seeds. All instances consisting of 20 tasks and 2 or 3 processors are classified into 12 different types of ETC matrices according to the 3 metrics. All the algorithms are applied on all 12 problem instances and the results plotted from Fig.4 to Fig.10.

The instances are labelled as g_a_bb_cc as follows:

- g means gamma distribution used in generating the matrices.
- a shows the type of inconsistency; c means consistent, i means inconsistent, and s means semi-consistent.
- bb indicates the heterogeneity of the tasks; hi means high and lo means low.
- cc represents the heterogeneity of the machines; hi means high and lo means low.

The average Relative Percentage Deviation (RPD) [1] is used for comparing the results of the proposed LIDPSO with CDPSO and LTDPSO. It is calculated in equation (20), where P is the average result of the proposed algorithm and AC_i is the average result provided by CDPSO and LTDPSO for each instance.

$$RPD = (AC_i - P)/P * 100 \qquad (20)$$

Table I shows the comparison of the LIDPSO with CDPSO and LTDPSO in terms of the average fitness value for scheduling the meta-tasks on 2 processors and 3 processors. In most of the benchmark instances, the LIDPSO provides better results than CDPSO and LTDPSO. On average fitness value, the improvement of LIDPSO over CDPSO and LTDPSO is 1.19%, 4.77% respectively for scheduling tasks on 2 processors and the improvement for scheduling tasks on 3 processors is 1.48%, 5.05% respectively across all instances.

In Fig 4, the result of CDPSO is improved compared with LIDPSO and LTDPSO in most of the benchmark instances for scheduling tasks on 2 processors. On average fitness value under consistent model, the improvement of CDPSO over LIDPSO and LTDPSO is 0.29% and 2.06% across all instances respectively. The LIDPSO provides better results than CDPSO and LTDPSO are shown in Fig 5. On average fitness value under consistent model, the improvement of LIDPSO over CDPSO and LTDPSO is 3.81% and 1.71% across all instances respectively.

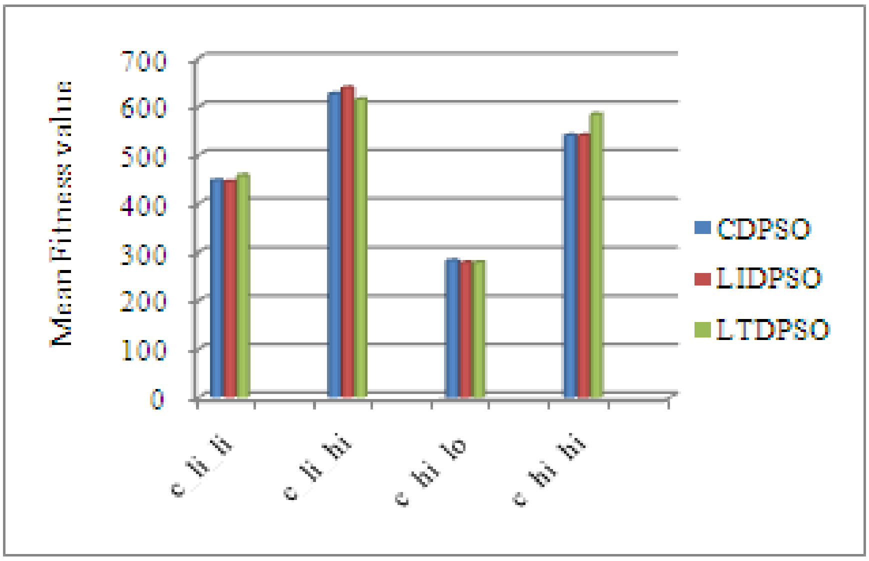


Fig.4.Comparison of average fitness value on 2 processors with consistent model

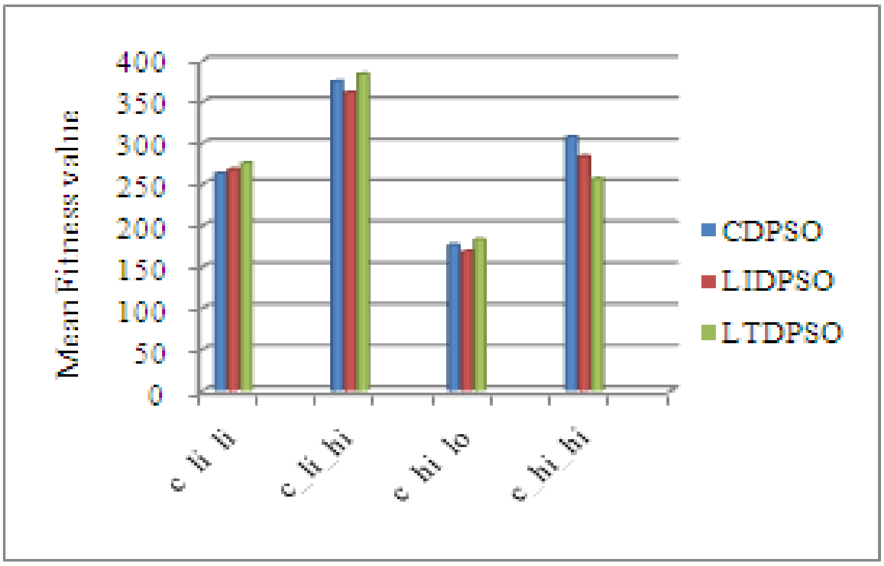


Fig.5.Comparison of average fitness value on 3 processors with consistent model

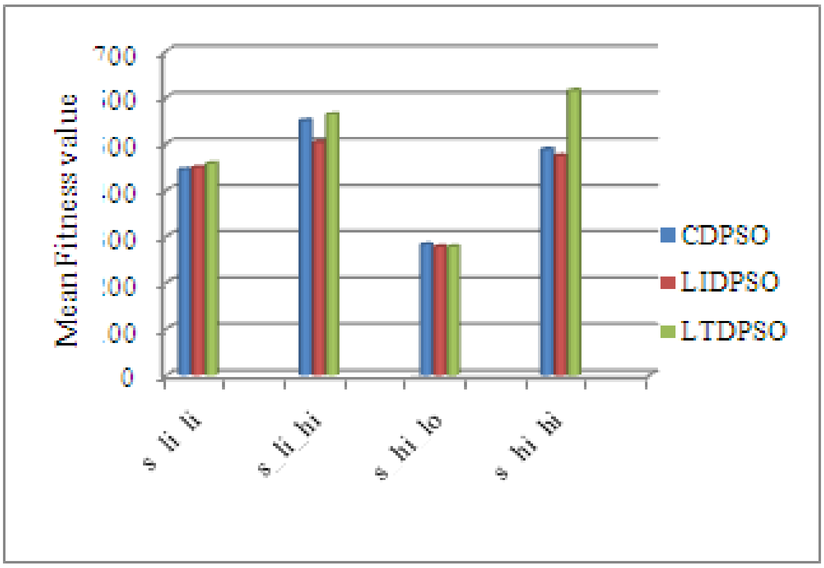


Fig.6.Comparison of average fitness value on 2 processors with semi-consistent model

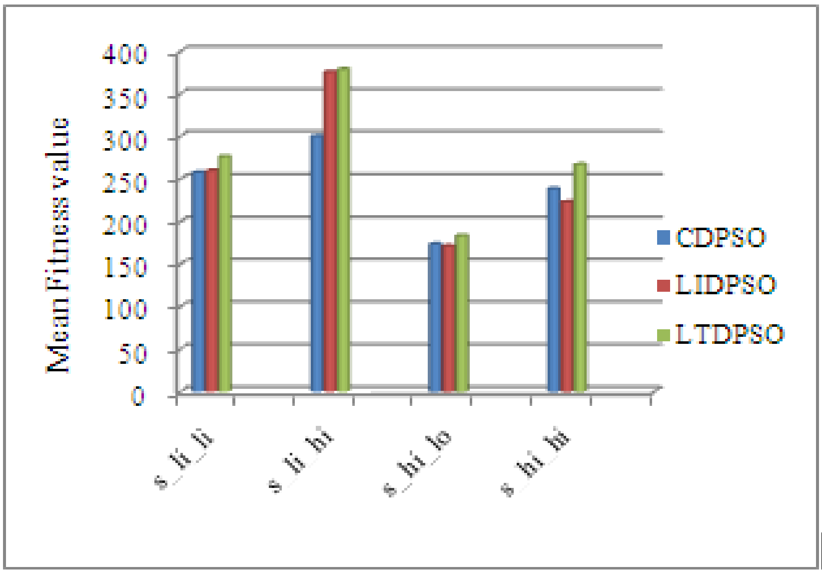


Fig.7.Comparison of average fitness value on 3 processors with semi-consistent model

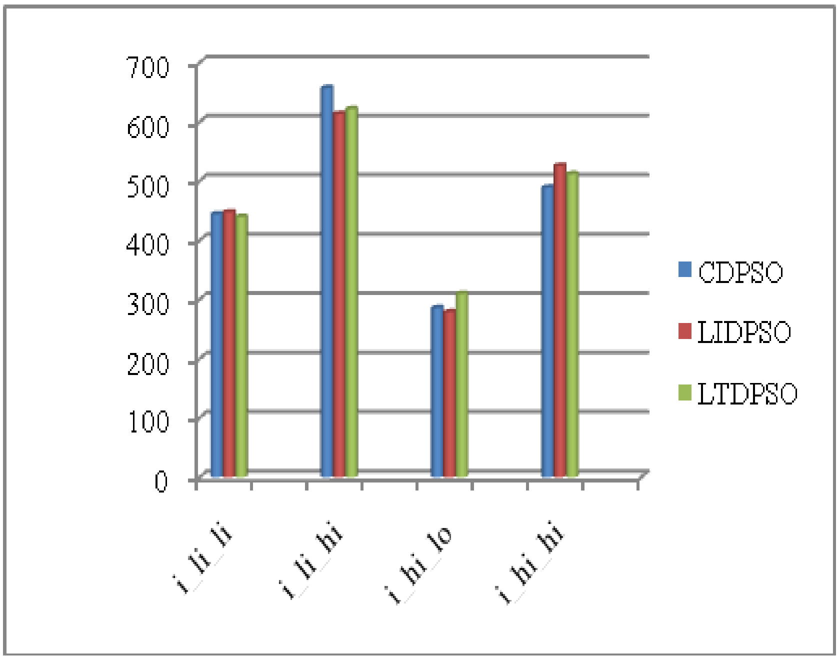


Fig.8.Comparison of average fitness value on 2 processors with inconsistent model

TABLE I COMPARISON OF AVERAGE RESULTS BETWEEN LIDPSO WITH CDPSO AND LTDPSO OVER FITNESS VALUE

Type of heterogeneity	2 Processors			3 Processors		
	CDPSO	LIDPSO	LTDPSO	CDPSO	LIDPSO	LTDPSO
c_li_li	447.705765	**445.1753235**	457.808594	**263.558884**	267.920212	275.9220735
c_li_hi	626.0338745	638.5205995	**615.036285**	374.7586215	**361.160538**	384.057648
c_hi_lo	282.208679	**278.112961**	278.450821	176.7502975	**168.6026155**	183.1452105
c_hi_hi	540.369217	**540.0888065**	584.13913	307.465439	283.694763	**256.791458**
s_li_li	**443.5552825**	448.1753235	457.808594	**257.743935**	259.7317045	276.1615905
s_li_hi	549.531677	**502.535034**	563.104904	**299.751999**	375.9031065	378.9714815
s_hi_lo	282.09082	**276.9756775**	277.3663485	173.3772965	**170.766464**	182.801636
s_hi_hi	487.420395	**473.897522**	614.3197635	238.143692	**222.6656495**	266.9108125
i_li_li	444.03923	447.440613	**439.7098235**	256.5464095	**248.7847065**	258.365059
i_li_hi	657.205078	**613.578644**	621.9611815	384.8706205	**356.4942475**	378.811142
i_hi_lo	285.6050875	**278.723343**	309.4141845	180.922913	180.3042145	**180.279396**
i_hi_hi	**488.8827365**	526.415497	511.6748045	264.345665	**235.6744615**	267.6668095

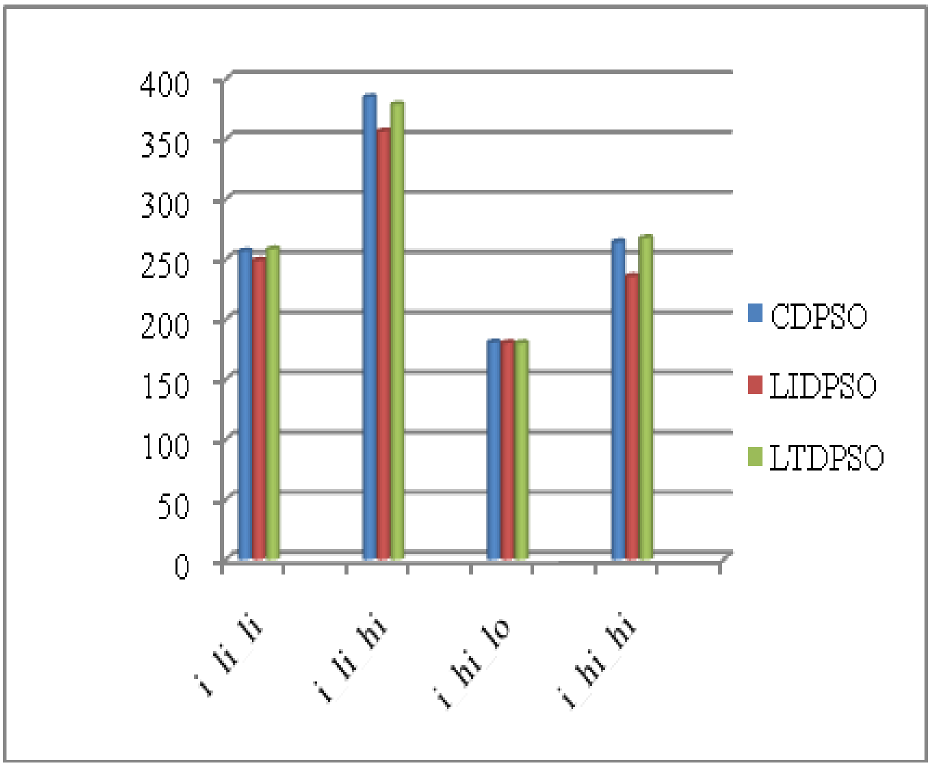


Fig.9.Comparison of average fitness value on 3 processors with inconsistent model

TABLE II COMPARISONS OF AVERAGE RESULTS BETWEEN LIDPSO WITH CDPSO AND LTDPSO OVER MAKE SPAN IN SECONDS

Type of heterogeneity	2 Processors			3 Processors		
	CDPSO	LIDPSO	LTDPSO	CDPSO	LIDPSO	LTDPSO
c_li_li	**15133**	15159	15413.5	7350.5	**7323.5**	7490.5
c_li_hi	19336.5	18630	**18482**	8892	**8497.5**	9245.5
c_hi_lo	9612.5	**8945.5**	9273.5	4697	4770	**4690.5**
c_hi_hi	**13089**	14520.5	15914	**5272.5**	6076	6451.5
s_li_li	**14908.5**	15159	15413.5	7311.5	**7279.5**	7509
s_li_hi	18557.5	**17023.5**	18731.5	**8481.5**	9023	9838.5
s_hi_lo	9660	**8788**	9358.5	4913	4785	**4748.5**
s_hi_hi	**15067.5**	16854.5	19263	**5539**	5686	7173.5
i_li_li	**14638**	15083	14881.5	7261	**7039**	7290
i_li_hi	19961	**18533.5**	18823.5	10514	**9445.5**	10345
i_hi_lo	9579.5	**9034.5**	10307	4867	4602	**4565**
i_hi_hi	**15598.5**	16543.5	16606.5	5782.5	**5869.5**	6726.5

The improvement of LIDPSO is increased compared with CDPSO and LTDPSO for scheduling task on 2 processors. This is shown in Fig 6. On average fitness value under semi-consistent model, the improvement of LIDPSO over CDPSO and LTDPSO is 3.58% and 12.40% across all instances respectively. In Fig 7, the CDPSO provides better results than LIDPSO and LTDPSO in most of the benchmark instances for scheduling tasks on 3 processors. On average fitness value under semi-consistent model, the improvement of CDPSO over LIDPSO and LTDPSO is 6.20% and 14.02% across all instances respectively. On average fitness value under inconsistent model, the improvement of LIDPSO over CDPSO and LTDPSO is 0.51%, 0.90% respectively for scheduling on 2 processors and the improvement for scheduling on 3 processors is 6.40% and 6.25% across all instances respectively.

TABLE III COMPARISONS OF AVERAGE RESULTS BETWEEN LIDPSO WITH CDPSO AND LTDPSO OVER FLOW TIME IN SECONDS

Type of heterogeneity	2 Processors			3 Processors		
	CDPSO	LIDPSO	LTDPSO	CDPSO	LIDPSO	LTDPSO
c_li_li	5145	**5143**	5248	**4055.5**	4169.5	4364
c_li_hi	**8401**	8674	8466	5942.5	**5905**	6085
c_hi_lo	**3176**	3359.5	3295	3061	**2479**	3177
c_hi_hi	**6172.5**	6224	6964.5	**3631.5**	5524	4397.5
s_li_li	5149	**5143**	5248	**3887**	3922.5	4302
s_li_hi	6516.5	**6031.5**	6502.5	**4604.5**	5273.5	6229.5
s_hi_lo	3126	**3055.5**	3204.5	**2681**	2552.5	3123.5
s_hi_hi	**6556.5**	7095.5	7794.5	3813.5	**3453.5**	4546.5
i_li_li	5329	**5138.5**	5059.5	3877.5	**3765**	3990

i_li_hi	8942	**8533**	8452.5	6287	**6052.5**	6242.5
i_hi_lo	**3345**	3358	3592.5	2964.5	**2844.5**	3061
i_hi_hi	6334.5	**6250**	6569.5	5029.5	**3448**	3899.5

Table II shows the comparisons of the LIDPSO with CDPSO and LTDPSO in terms of the average make span value for scheduling on 2 processors and 3 processors. The performance of LIDPSO is improved than CDPSO and LTDPSO in most of the benchmark instances. The improvement of LIDPSO over CDPSO and LTDPSO is 0.28% and 3.51% respectively on 2 processors and 0.90%, 8.16% of improvement on 3 processors across all instances respectively.

TABLE IV COMPARISONS OF AVERAGE RESULTS BETWEEN LIDPSO WITH CDPSO AND LTDPSO OVER RELIABILITY COST

Type of heterogeneity	2 Processors			3 Processors		
	CDPSO	LIDPSO	LTDPSO	CDPSO	LIDPSO	LTDPSO
c_li_li	0.009857	**0.0098145**	0.009897	**0.009558**	0.0096935	0.009767
c_li_hi	0.012482	0.012341	**0.012304**	0.011588	**0.0112975**	0.0122315
c_hi_lo	**0.0059885**	0.0062055	0.006136	0.006213	**0.006116**	0.0063265
c_hi_hi	**0.010528**	0.010657	0.0110025	**0.00756**	0.0085895	0.007647
s_li_li	**0.009778**	0.0098145	0.009897	0.009612	**0.0095815**	0.009881
s_li_hi	0.012496	**0.011405**	0.012339	**0.01115**	0.0112155	0.013345
s_hi_lo	**0.005932**	0.006108	0.006049	0.006211	**0.0061745**	0.0063535
s_hi_hi	**0.010532**	0.011302	0.011545	0.007461	**0.0074205**	0.008656
i_li_li	0.0097135	0.009800	**0.009702**	0.009489	**0.009367**	0.0095305
i_li_hi	0.0127195	0.012262	**0.012202**	0.0135	**0.0124725**	0.013187
i_hi_lo	**0.0059875**	0.006200	0.006296	**0.0062245**	0.006232	0.0061065
i_hi_hi	0.010888	0.0110365	**0.010523**	0.0078425	**0.006996**	0.0080025

TABLE V COMPARISONS OF AVERAGE OBJECTIVE RESULTS OF CDPSO, LIDPSO AND LTDPSO UNDER ETC CONSISTENCIES

Objectives	**Mean value of ETC consistency**	**Appropriate Algorithm for scheduling on 2 processors**	**Percentage of Improvements (%)**	**Appropriate Algorithm for scheduling on 3 processors**	**Percentage of Improvements (%)**
Make span	Consistent	CDPSO	2.21%,4.71%	CDPSO	8.31%,7.80%
	Semi-consistent	**LIDPSO**	0.11%,6.68%	CDPSO	1.44%,21.46%
	Inconsistent	**LIDPSO**	2.90%,1.70%	**LIDPSO**	12.72%,6.72%
Flow time	Consistent	CDPSO	0.15%,3.34%	CDPSO	1.73%,6.36%
	Semi-consistent	**LIDPSO**	0.64%,8.54%	CDPSO	2.01%,11.52%
	Inconsistent	**LIDPSO**	0.98%,2.41%	**LIDPSO**	5.45%,7.31%

Reliability cost	Consistent	CDPSO	0.16%,1.25%	CDPSO	2.22%,3.01%
	Semi-consistent	**LIDPSO**	0.29%,3.12%	**LIDPSO**	0.13%,11.18%
	Inconsistent	LTDPSO	1.51%,1.49%	**LIDPSO**	5.67%,5.02%

The comparison of the LIDPSO with CDPSO and LTDPSO in terms of the average flow time for scheduling the meta-tasks on 2 processors and 3 processors are shown in Table 3.From Table III, the improvement of LIDPSO over CDPSO and LTDPSO is 0.50%, 4.70% respectively for scheduling tasks on 2 processors and the improvement for scheduling tasks on 3 processors is 0.60%, 7.06% respectively.

Table IV shows the comparisons of the LIDPSO with CDPSO and LTDPSO in terms of the average reliability cost value. The performance of LIDPSO is better than CDPSO and LTDPSO in most of the benchmark instances for scheduling the meta-tasks on 2 processors and 3 processors.

TABLE VI COMPARISON OF OVER ALL AVERAGE FITNESS VALUE OF ALL THE THREE ALGORITHMS FOR 2 PROCESSORS VERSUS 3 PROCESSORS

Mean fitness value of all types of heterogeneity instances	2 Processors	3 Processors	2 Processors	3 Processors	2 Processors	3 Processors
	CDPSO	CDPSO	LIDPSO	LIDPSO	LTDPSO	LTDPSO
	461.2206535	264.852981	455.803279	260.975224	477.5662	274.15703

The improvement of LIDPSO over CDPSO and LTDPSO is 0.05%, 0.90% respectively for scheduling tasks on 2 processors and 1.20%, 5.60% of improvement for scheduling tasks on 3 processors across all instances respectively. The results obtained in Table V shows the LIDPSO gives better results compared with CDPSO and LTDPSO in most of the ETC consistencies. The improvement of LIDPSO over CDPSO and LTDPSO is 2.67% and 9.82% across all instances respectively. These results indicate that the LIDPSO is a viable alternative for task scheduling problem.

Table VI and Fig 10 show the comparisons of over all average fitness value for scheduling the meta-tasks on 2 processors versus 3 processors of LIDPSO, CDPSO and LTDPSO across all instances. All the three algorithms are found to be more efficient when tasks are being scheduled in 3 processors.

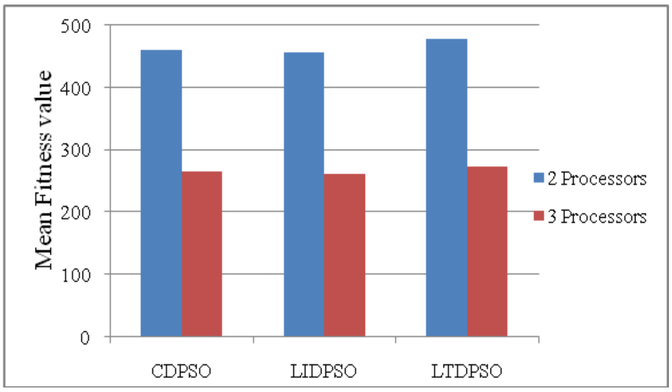


Fig.10 Comparison of over all average fitness value for scheduling on 2 processors versus 3 processors

The improvement of CDPSO with 3 processors over 2 processors is 74.14%, LIDPSO with 3 processors over 2 processors is 74.65% and LTDPSO with 3 processors over 2 processors is 74.20% across all instances.

All the above results show that the proposed algorithm is a feasible substitute for task scheduling problem in distributed system.

VII. CONCLUSION

Achieving an optimal solution for scheduling of tasks on processors is crucial for distributed systems due to their highly heterogeneous nature. In this paper a new, efficient Discrete PSO algorithm called LIDPSO has been successfully applied to the multi-objective multiprocessor task scheduling problem to find optimal schedules for meta-tasks to minimize the make span, flow time and reliability cost simultaneously. In the discrete PSO, the representation of position and velocity of the particle is extended from continuous value vector to discrete value vector. As a result, the mapping from continuous to discrete for particles are not needed and hence much computation time can be saved. The LIDPSO includes the inertia weight which linearly decreases from large value to small value through the search process of identifying the global optima. This adaptiveness of inertia weight permits it to reach an excellent balance between the exploration and the exploitation of the search space. The proposed algorithm was tested with 12 different types of ETC matrices available in the literature. The simulation results and comparisons prove that the LIDPSO is better compared to other algorithms which have constant control parameters and time varying control parameters. Over all the CDPSO, LIDPSO and LTDPSO algorithms are found to perform efficiently when tasks are being scheduled on 3 processors.

The future work will investigate scheduling tasks with large data set and the task with precedence constraint which are pre-emptive in nature or in dynamic environments.

REFERENCES

[1] Qinma Kang, and Hong He,"*A novel discrete particle swarm optimization algorithm for meta-task assignment in heterogeneous computing systems*", Elsevier Microprocessors and Microsystems ,pp 10–17,2011

[2] Kennedy,J, and Eberhart.R ,"*Particle swarm optimization*", *In proceeding of the fourth IEEE International conference on Neural Networks*,Perth,Australia.IEEE Service Center,1995

[3] S.Sarathambekai and K.Umamaheswari, "*Comparison among four Modified Discrete Particle Swarm Optimization for Task Scheduling in Heterogeneous Computing Systems*", *International Journal of Soft Computing and Engineering (IJSCE)*,ISSN: 2231-2307, Vol-3, Iss-2,2013

[4] J. C. Bansal, P. K. Singh,Mukesh Saraswat, Abhishek Verma, Shimpi Singh Jadon and Ajith Abraham, "*Inertia Weight Strategies in Particle Swarm Optimization*", IEEE Third World Congress on Nature and Biologically Inspired Computing (NaBIC), ISBN 978-1-4577-1122-0,pp 633 – 640,2011

[5] S.N.Sivanandam, P.Visalakshi and A.Bhuvaneswari, "*Multiprocessor Scheduling Using Hybrid Particle Swarm Optimization with Dynamically Varying Inertia*", International Journal of Computer Science & Applications, Vol. 4 Issue 3, pp 95-106,2007

[6] Yaochu Jin, Tatsuya Okabe and Bernhard Sendhoff ,"*Solving three objective optimization problems using Evolutionary dynamic weighted aggregation: Results and Analysis*", Genetic and Evolutionary Computation Conference. Springer, Berlin, pages 636—637,2003

[7] Xiao Qin,Hong Jiang,"*Dynamic, Reliability-driven Scheduling of Parallel Real-time Jobs in Heterogeneous Systems*", IEEE International conference on Parallel Processing,pp 113-122,2001

[8] Wei Luo, Xiao Qin, and Kiranmai Bellamssss,"*Reliability-Driven Scheduling of Periodic Tasks in Heterogeneous Real-Time Systems*", IEEE International Symposium on Embedded Computing,Ontario, Canada,2007

[9] Hesam Izakian, Behrouz Tork Ladani, Ajith Abraham,Vaclav Snasel, "*A Discrete Particle Swarm Optimization Approach For Grid Job Scheduling*", International Journal of Innovative Computing, Information and Control, ISSN 1349-4198 Volume 6, Number 9,2010

[10] H.J. Braun et al, "*A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems*", Journal of Parallel and Distributed Computing, Vol 61, No.6, 2001.

[11] H. Izakian, A. Abraham and V. Snasel, "*Performance comparison of six efficient pure heuristics for scheduling meta-tasks on heterogeneous distributed environments*", Neural Network World, vol.19,no.6, pp.695-710,2009

[12] A. Abraham, H . Liu, C. Grosan, F. Xhafa, "*Nature inspired meta-heuristics for grid scheduling: single and multi-objective optimization approaches*", Studies in Computational Intelligence, Springer Verlag: Heidelberg, Germany, pp. 247–272,(2008)

[13] Y. Shi and R. Eberhart., "*A modified particle swarm optimizer*", In Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., pages 69–73,2002

[14] Kaushik Suresh, Sayan Ghosh, Debarati Kundu, Abhirup Sen, Swagatam Das and Ajith Abraham,"*Inertia-Adaptive Particle Swarm Optimizer for Improved Global Search*", IEEE International conference on Intelligent systems and design,pp 253-258,2008

[15] J. Xin, G. Chen, and Y. Hai.,"*A Particle Swarm Optimizer with Multistage Linearly-Decreasing Inertia Weight*",IEEE International Joint Conference on Computational Sciences and Optimization, volume 1, pages 505–508,2009

[16] J. Park, K. Lee, J. Shin, and K. Y. Lee, "*A Particle Swarm Optimization for Economic Dispatch with Nonsmooth Cost Function*", IEEE Trans. on Power Systems, Vol. 20, No.1, pp. 34-42,2005

[17] T. Ray, and K. Liew, "*A swarm metaphor for multiobjective design optimization*", Engineering Optimization, Vol. 34, pp. 141-153,2002.

[18] C.A. Coello Coello, and M. Salazar Lechuga, "*MOPSO: A proposal for multiple objective particle swarm optimization*", Congress on Evolutionary Computation IEEE Service Center, Piscataway, New Jersey, pp. 1051-1056,2002

[19] Ozgur Uysal1 and Serol Bulkan2, "Comparison Of Genetic Algorithm And Particle Swarm Optimization for Bicriteria Permutation Flowshop Scheduling Problem", *International Journal of Computational Intelligence Research,* ISSN 0973-1873 Vol.4, No.2, pp.159–175 ,2008

[20] S. Ali, H.J. Siegel, "*Representing task and machine heterogeneities for heterogeneous computing systems*", Tamkang Journal of Science and Engineering, Vol. 3, No. 3, pp. 195-207, 2000

[21] Kaushik Suresh, Sayan Ghosh, Debarati Kundu, Abhirup Sen, Swagatam Das and Ajith Abraham, "*Inertia-Adaptive Particle Swarm Optimizer for Improved Global Search*", IEEE International conference on Intelligent systems and design,pp 253-258,2008

[22] Xiaohong Kong1,2, Jun Sun1 and Wenbo Xu1, "*Permutation-based Particle Swarm Algorithm for Tasks Scheduling in Heterogeneous systems with Communication Delays*", International Journal of Computational Intelligence Research ISSN 0973-1873 Vol.4, No.1, pp. 61–70,2008

