Hybrid Real-Time Task Scheduling Algorithm in Overload Situation for Multiprocessor System

A. Hatami[1,2], S. Chuprat[2], H. Md Sarkan[2] and N. Firdaus Mohd Azmi[2]
[1]*Faculty of Computing, University of Teknologi Malaysia, 81310 Johor Bahru, Johor, Malaysia.*
[2]*Advanced Informatics School, Universiti Teknologi Malaysia, Kuala Lumpur, Malaysia.*
Amir.hatami.h@live.com

***Abstract*—Real-time systems are reactive systems which should meet major constraints in scheduling tasks like time limitation and resources allocation for scheduling the task effectively when the system in overloaded condition. Failure of system in scheduling tasks when system is overloaded can result in catastrophic impacts. The goal of this research is to propose a task scheduling algorithm that able to perform better than traditional Earliest Deadline First (EDF) and minimize the overall completion time when the system in overloaded condition. The proposed scheduling algorithm is built based on three new improved scheduling algorithms namely: (1) Hybrid Particle Swarm Optimization (PSO) and Hybrid Invasive Weed Optimization (HPIO), (2) Enhanced Initial Swarm (EIS), and (3) Hybrid EDF, EIS and HPIO Optimization (HEDFPIO). The author proves that more successful tasks is scheduled by using HPIO in multiprocessor system in over loaded situation among PSO and ACO. The author uses EIS algorithm in order to improve local search in HPIO and have fair load balance among processors. Finally the author presents a new hybrid algorithm that combines HPIO, EIS and EDF which is called HEDFPIO, It is observed that we could achieve higher successful ratio in task scheduling and with shorter calculation time in overloaded situation.**

***Index Terms*—Enhanced Initial Swarm; Hybrid; Invasive Weed Optimization; Particle Swarm Optimization Overload.**

I. INTRODUCTION

A real-time scheduling system contains scheduler, clock and processor. Tasks are assigned to the processors and it will be executed in a specific time and specified deadline by the characteristic of the scheduling algorithm. There are many scheduling techniques and the interest is to find the most optimum algorithms. In this paper, it is presented the hybrid algorithms which uses best part of the selected optimal algorithms and finally analyses the performance of the newly introduced hybrid algorithms.

One of these optimal algorithms is Particle Swarm Optimization (PSO) which is based on swarm intelligence. This algorithm simulates the behavior of individual particle in a group to optimize the survival of species. One of the most advantages of PSO is its robustness in controlling parameters and its high computational efficiency [1].

Invasive Weed Optimization (IWO) is a stochastic algorithm that simulates the behavior of weeds. Also this algorithm is presented by Mehrabian and Lucas [2]. IWO has shown successful results in many fields and solved many problems such as optimization and tuning of a robust controller [2].

In this paper, it is shows that HPIO achieves better results by increasing the number of successful scheduled task and decreasing calculation time. The author could achieve better result in comparison with other algorithms that will be reviewed in this paper such as Earliest Deadline First (EDF), Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO) and Invasive Weed Optimization (IWO) in overloaded situation. Also for this study the author implements and tests all algorithms with both uniprocessor and multiprocessor system. The author uses a method to have fair load balance among the processor to have better CPU utilization and improve local search in PSO and IWO. As illustrated in graphs of experiment shows that new presented algorithms have better performance by increasing successful tasks with improved calculation time.

The author considers using homogenous processors to compare the performance of the algorithms with previous research. By using homogenous processors, rate of all the tasks will be same in identical processor. Also, there is no constraint on requested time since the tasks model is based on sporadic model.

II. RELATED WORKS

This research is categorized in three main parts. The first part author studies the current suggested solution by ACO, PSO and IWO in uniprocessor. In second part, the author checks the best fair load balance algorithm to combine the output with multiprocessor systems and enhance initial swarm to avoid HPIO getting trapped in local search. And in the final part, author checks the feasibility of improved task scheduling by Earliest Deadline First algorithm.

Shah and Kotecha [3, 5], and Shah et al. [4] have used ACO and EDF algorithm and introduced a Hybrid algorithm that performs very well in comparison with normal EDF algorithm. The suggested adaptive framework is using EDF algorithm in "under load" situation and when system is "overloaded", it switches to the ACO algorithm for scheduling the tasks. "When a system is assigned to schedule an amount of task which is more than the available system resource can handle is called overloaded situation".

Therefore, execution of tasks will depend on the pheromone value laid on each scheduled task and heuristic function. The Adaptive ACO framework schedules tasks in lesser execution time when compared to normal ACO and EDF in overload situation. The weakness of adaptive framework is observed when the number of the tasks is increased ACO algorithm, adaptive framework requires more time to calculate which does not make good candidate for real-time task scheduling systems.

Karimi [6] used particle swarm optimization for task scheduling in Grid computing. PSO is considered as a

population based stochastic optimization method that simulates the behaviors of bird flocking [7]. In this method, the best result will be calculated based on "follow the bird which is nearest to the food".

Karimi [6] used PSO to have better result in task scheduling by repeating same method until better result is found. In PSO model all the possible scenario which is helping to solve the problem is considered as a bird or particle. Each particle has a fitness value which is calculated by fitness function. In PSO model it is needed to define a problem space and all particle fly through the problem space. Each particle has a velocity which will be recalculated in iteration. Velocity will be calculated to follow the current optimum particle.

PSO algorithm has set of random particles that are created and then an optimal particle will be selected in each iteration. Two parameters play an important role in PSO algorithm which is called pBest and gBest. pBest or Personal best is considered as best fitness which has achieved and gBest or neighborhood best position which is tracked by the particle swam optimizer, pBest and gBest are those which are obtained so far by any particle in the population [8].

Karimi's [6] project design is in grid computing, the challenge for the author was assigning tasks to the resource and the problem arises when the system is overloaded. Therefore, Maryam used PSO algorithm to reduce execution time and utilize maximum resource. PSO is performing fast enough; she was looking for an algorithm to have fair destitute in Grid system. She used few algorithms which were benchmarked in many researches. These algorithms are Opportunistic Load Balancing (OLB), Min-min, Max-min and Discrete Particle Swarm Optimization (DPSO).

The outcome of the mentioned research was HDPSO which was combination of Min-Min and DPSO which could achieve better results when compared to other algorithms like OLB, Max-min and DPSO. Max-min heuristic is efficient only when most of the jobs arriving to the grid system are shortest [8].

Ghalenoei et al. [9] introduced a novel swarm base optimization algorithm which is inspired from Invasive Weed Optimization. (IWO) to do task scheduling of unmanned aerial vehicles (UAVs). The authors compared the result of IWO with Genetic Algorithms (GA) that is based on the simulation of result. In this experiment, IWO obtains better performance in comparison with GA.

According to Mehrabain and Lucas [2], IWO has three main parts. These parts are initialization, reproduction and spatial dispersal. In the first step, sample population is created based on initial seeds randomly. In second part, each individual seed is growing and it is allowed to reproduce new seeds and linearly depending on their own. In third part, the generated seeds are being randomly scattered with a normal distribution over the search space. The meaning of distribution is equal to the location of parent plant, but standard deviation (SD), σ, will be reduced from a specified initial value, σ_initial, to the final value, σ_final, according to Equation (1).

$$\sigma_{iter} = \frac{(\text{iter}_{max} - iter)^n}{(\text{iter}_{max})^n}(\sigma_{initial} - \sigma_{final}) + \sigma_{final} \qquad (1)$$

where σ_iter is the standard deviation at the present step, and σ_initial,σ_final, iter_(max) (maximum number of iterations), and n (modulation index) are other parameters. This nonlinear modification has shown satisfactory performance in many simulations [2]. This assumption means the seeds will be randomly distributed such that they lie close to the parent plant [10]

In next step, each weed allows to produce seeds and spreads them as mentioned in previous steps. Then all the seeds and their parents are ranked based on their fitness function. After that those seeds which are having lesser fitness are eliminated from the list. This method is based on "survival of the fittest" idea [11] (a common concept in evolutionary algorithms) gives a chance to plants with lower fitness to reproduce, and if their off springs have good fitness, they can survive in their offspring's existence [2].

Finally, if maximum number of iteration has been reached then the result is considered as best fitness and nearest to optimal result.

Table 1
IWO parameters

Symbol	Quantity	Value
N_0	Number of initial population	10
$iter_{max}$	Maximum number of iterations	400
dim	Problem dimension	18
P_{max}	Maximum number of plant	40
S_{max}	Maximum number of seeds	3
S_{min}	Minimum number of seeds	1
n	Nonlinear modulation index	3
σ_{init}	Initial value of standard deviation	1
σ_{final}	Final value of standard deviation	0.008

Ghalenoei et al. [9] follows all the mentioned steps to design his framework but he used different spatial dispersal module. His module designed to random selection of solution from a neighboring hypercube in the discrete space of solutions around the plant with a normal distribution. The sample of his pseudo code is provided for your reference. Please refer to Table 1 for parameters which have been used and Figure 1 regarding IWO pseudo code.

1. Genearte random population of N_0 individuals from the set of feasible solutions
2. i=: 1
3. do
   a. Compute maximum and minimum fitness in the colony
   b. For each individul $w \in W$
      i. Compute the number of seeds for w, corresponding to its fitness
      ii. Randomly select the seeds from the feasible solutions around the parent plant (w) in a neighborhood with normal distribution
      iii. Add the generated seeds to the solution set, W
   c. If $|W| = N > p_{max}$
      i. Sort the population N in descending order of their fitness
      ii. Truncate population of weeds with smaller fitness until $N = p_{max}$
   d. i=: i+1
4. Repeat 3 until the maximum number of iterations

Figure 1: Pseudo-code of HPIO

Ghalenoei et al. [9] compared his results with various algorithms such as ACO, PSO and IWO. But IWO could achieve better result.

III. HYBRID PSO WITH IWO (HPI)

As it was mentioned before, the objective of this research was to improve minimum time cost and the author plans to achieve it by combining PSO and IWO and use their strengths to introduce a new algorithm. PSO could schedule tasks very

fast and accurate [6, 12] and IWO could have better result in comparison with PSO [9] but IWO algorithm requires more time to reproduce and eliminate seeds with lower fitness. In this research the author proves that, by using new algorithm we can schedule more tasks when compared to PSO and take lesser time in comparison with IWO.

In our experiment, we create HPI algorithm based on Figure 3. The main difference in this algorithm with IWO is the way particle is created and the particles which are eliminated are with less fitness. We created more initial particle in the initial sample and then truncate them in each iteration. Therefore, search space has become bigger to find the best order of the task scheduling. This makes the algorithm work faster than the IWO as it is not required to generate sample particle again. Removing sample in each iteration is based on the truncate value.

Truncate process will be continuous until swarm size becomes greater than truncate value. By creating bigger initial population, we could achieve optimal result in comparison with previous results of PSO and IWO in shorter time as shown in Figure 2. Optimal result means we can have more successful tasks scheduled in comparison with other algorithms. In Figure 2, point "Z" in graph is consider as an optimal result.

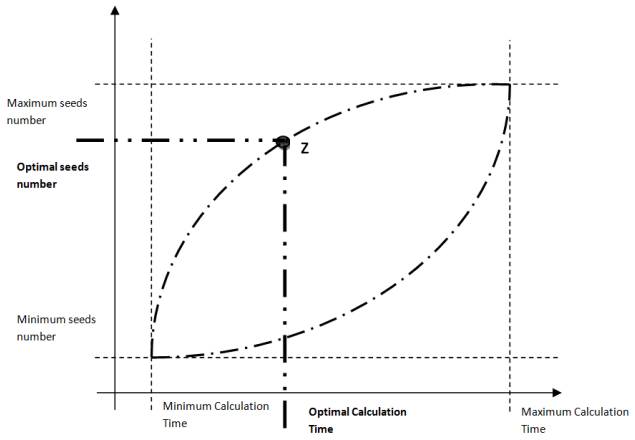


Figure 2: Calculation time changes based on seeds (particle) number

Meanwhile, Figure 3 shows the pseudo code of HPIO. In Line 1 and 2, swarm sample will be created and store in swarm list then in line 3 and 4 fitness value of each swarm will be calculated. From line 5 to 16, pBest and gBest will be calculated. In line 16, new voracity will be calculated and replace in system according to equation2 and after that in line 17 new locations will be evaluated based on equation3 and apply in line 18. Then fitness is calculated accordingly in line 19. In line 20 to 22. The parameter used in HPIO algorithm is mentioned in Table 2 which is based on the trial and error experiment and best parameter selected for Table 2.

$$vel_i^{\alpha+1} = \omega vel_i^{\alpha} + c_1 R_1 * (pBest_i - x_i^{\alpha}) + c_2 R_2 * (gBest_i - x_i^{\alpha}) \quad (2)$$

$$Loc_i^{\alpha+1} = Loc_i^{\alpha} + Loc_i^{\alpha+1} \quad (3)$$

$$\omega = \omega_{max} - \frac{\omega_{max} - \omega_{min}}{T} \times t \quad (4)$$

```
HPIO( TruncateValue )
1     for each Swarm in SwarmList
2           Initialize random
      end for
3     for each Swarm in SwarmList
4           Calculate fitness value
      end for
5     Itteration := 0
6     while( Iteration < Max_Iteration and !AllTasksScheduled) do
7           /* Update pBest*/
8           for each Swarm in Swarm List
9                 if ( pBest < Swarm.Fitness) then
10                      pBest := Swarm.Fitness
11                      pBestLocation := Swarm.location
                  end if
            end for
12          /* Update gBest */
13          if( Itteration ==0  or SwarmList[BestSwarmIndex]< gBest) then
14                gBest := SwarmList[BestSwarmIndex].Fitness
15                gBestLocation := SwarmList[BestSwarmIndex].Location
            end if
16          Calculate New Velocity based on Velocity equation (2)
17          Calculate New Location according to Location equation (3)
18          Apply new Velocity and Location
19          Update Fitness
20          If( SwarmList.Size > TruncateValue )then
21                Truncate population of swarm with smaller fitness
            end if
22          Itteration := Itteration + 1
      end while
```

Figure 3: Pseudo code of HPIO

Table 2
Parameters of HPIO algorithm

Parameter	Description
Swarm Size	60
Number of initial population	same as number of tasks
Maximum Number of iteration	10
Problem Dimension	2
self-recognition coefficient (*C1*)	2
Social coefficient (*C2*)	2
W_UPPERBOUND	1.0
W_LOWERBOUND	0.0
Truncate population	2

IV. USING EIS IN HPIO ALGORITHM

Below pseudo code illustrates the EIS algorithm that we used to have fair load balance in multiprocessor system and improve the local search in HPIO algorithm. The author expects to have better load balance after using EIS algorithm in the output result. As part of experiment we are using EIS in multiprocessor and compared the output with other algorithm such as ACO and PSO. Please refer to Figure 4 for more information.

EIS algorithms has been customized in order to receive list of the tasks as input and then based on number of processor rearrange them to have almost same task work load among all the processors. The EIS works better by queuing shorter execution time of tasks for scheduling and by having lower complexity it enhances the initial particles and improve the result.

```
EIS()
Begin
1       Initialize processorIndex List & board
2       minTaskProcessValue := double.MaxValue;
3       minTaskNumber := -1;
4       minProcess := -1;
5       for each task "i" in Task list
6               minValue := double.MaxValue;
7               minIndex := -1;
8               for each processor "j" in processor list
9                       if (board[i][j] < minValue) then
10                              minIndex := j;
11                              minValue := board[i][j];
                        end if
                end for
12              Add minIndex to ProcessorIndex list
13              if ( minValue < minTaskProcessValue) then
14                      minTaskNumber := i;
15                      minTaskProcessValue := minValue;
16                      minProcess := minIndex;
                end if
        end for
17      Add task to Processor in Processor List with minimum waiting time
18      UpdateData(minTaskNumber, minProcess);
END

UpdateData(minTaskNumber, minProcess)
 Begin
19      tolerance := board[minTaskNumber][minProcess];
20      for each task "i" in Task list
21              for each processor "j" in processor list
22                      if (j == minProcess) then
23                              board[i][j] += telorance;
                        end if
24      Remove minTaskNumber from Task list
25      Remove minTaskNumber from Board list
End
```

Figure 4: Pseudo-code of EIS

V. HYBRID EDF, EIS AND HPIO

The final research design framework is as shown in Figure 5. In order to improve calculation time, we tried hybrid EDF and HPIO. As shown below, if a task set has ability to be scheduled then it will be scheduled by using EDF otherwise it will be scheduled using HPIO. Based on EDF general schedulable formula, a task set which a set of "n" independent real-time tasks {τ_1,τ_2,…,τ_n,} is schedulable if and only if the total utilization of the task sets less than or equal to 2.

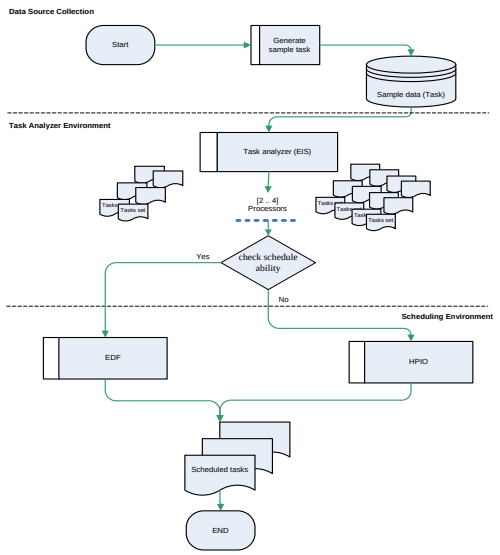


Figure 5: Hybrid HPIO, EIS and EDF

In equation 5, "U" shows the total utilization of the task sets and Ci represents execution time of task τ_i, and Ti will be period of task τ_i. Those task sets have a condition to be scheduled by the EDF and will be sent to EDF algorithm and remaining task will proceed to schedule by HPIO.

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \quad (5)$$

VI. RESULT AND DISCUSSION

A. Compare ACO and HPIO in uniprocessor

The author performs task generation for testing purpose according to previous studies. He selects a set of random tasks which contains 7, 14, 20, 25, 35, 50, 75, 100, 150, 200, 300, 350, 500, 750, 1000, 1500 and 2000 task sets to create similar condition for previous algorithms and proposed algorithm.

In part A, the author performs task scheduling for all the task sets and based on the Figure 6 it shows number of successful task in uniprocessor by using HPIO, PSO and ACO. As it can be observed in most of the cases HPIO could schedule more tasks in comparison with ACO and PSO algorithms. Based on Figure 6, in task set 150,300 and 350 ACO could schedule more task than HPIO and PSO. Table 3, shows the data related to timing each algorithm requires to finish the calculation. ACO performs better that other algorithms but in terms of calculation time, ACO takes more time than PSO and HPIO which is not acceptable in real-time scheduling systems. HPIO could schedule 525 tasks while ACO scheduled 485 tasks and PSO scheduled 461 tasks. HPIO improved the result to 8% in comparison with ACO and 14% improvement in comparison with PSO.

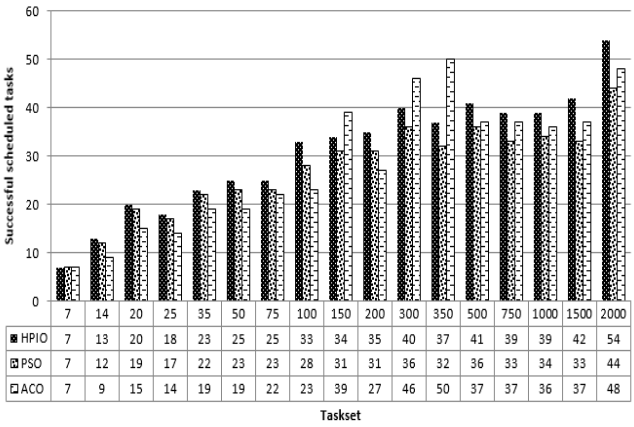


Figure 6: Successful Scheduled Tasks for ACO, PSO and HPIO in uniprocessor

Table 3
Calculation time for ACO, PSO and HPIO in uniprocessor

Task set

Algorithm	7	14	20	25	35	50	75	100	150	200	300	350	500	750	1000	1500	2000
ACO	24	9	47	136	688	4K	29K	120K	892K	5M	26M	61M	112M	166M	230M	439M	504M
HPIO	2	13	8	9	10	13	18	26	34	48	79	86	142	259	1010	2210	5474
PSO	1	2	3	3	7	7	11	14	23	30	50	61	93	168	587	1229	2839

Calculation Time (ms)

B. Enhanced HPIO by EIS Algorithm in Multiprocessor

Figure 7 shows total successful tasks using three algorithms

of HPIO-EIS and PSO-EIS and ACO-EIS in uniprocessor, dual, triple and quad processor. As illustrated HPIO could schedule more tasks in comparison with PSO and ACO algorithm in all processors. In this experiment for uniprocessor we did not use EIS algorithm but for dual, triple and quad processor EIS is combined with HPIO, PSO and ACO. Based on result it is observed that by increase in number of processors and using EIS we could achieve more successful tasks. Total number of tasks as input is 7076. Based on Figure 7, HPIO-EIS improved the result by almost 6% when compared to PSO-EIS and ACO-EIS in dual processors. HPIO-EIS improve the result by 12% when compared to PSO-EIS and HPIO-EIS improved by 10% when compared to ACO in triple processor. This improvement for HPIO-EIS in quad core processor is 4% when compared to PSO-EIS and 7% when compared to ACO-EIS.

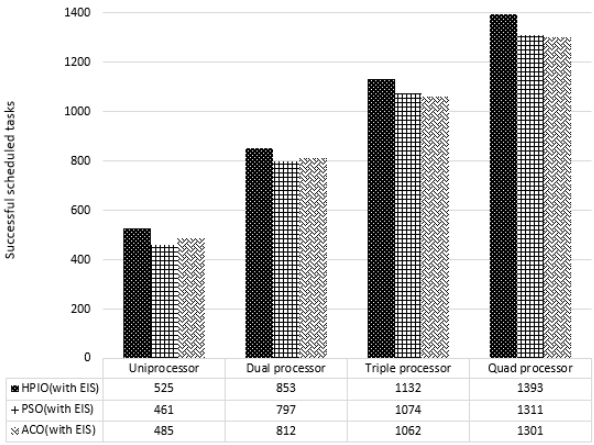


Figure 7: Total successful task by using EIS in different processor

Figure 8 shows details related to successful ratio in dual processor. Figure 9 and Figure 10 shows the data related to triple processor and quad processor. As it is shown in Figure 7, HPIO performs better when compared to other algorithms; the same expected result is presented in Figure 8, 9 and 10 also. This enhancement in result is due to usage EIS to arrange the inputs of the processor and effect on algorithms and have a divergent result in the local search.

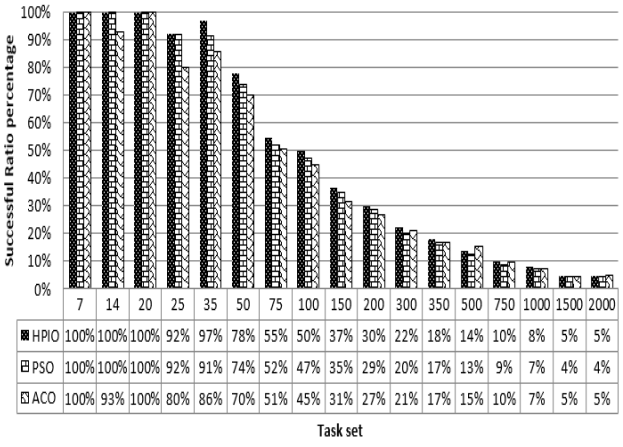


Figure 8: Successful ratio in dual processor

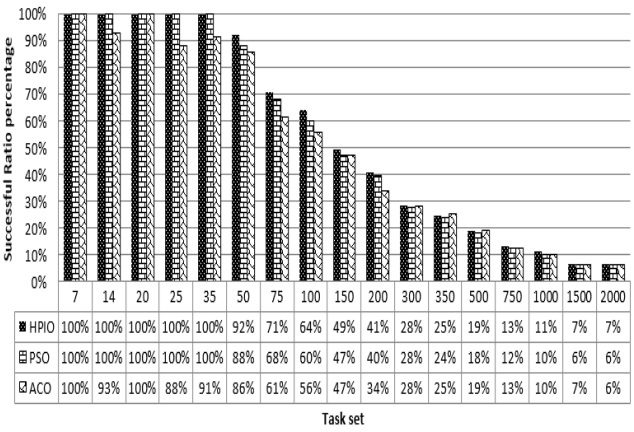


Figure 9: Successful ratio in triple processor

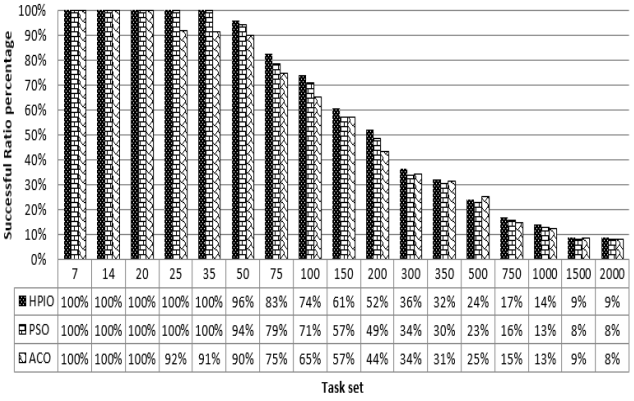


Figure 10: Successful ratio in quad processor

Figure 9 shows the successful ratio of scheduled tasks in triple core processor. In this test also HPIO with EIS could achieve higher result in comparison with PSO and ACO algorithm. As you might observe ACO perform well if the tasksets are big but it is highlighted that ACO require long time to process the data in comparison with PSO or HPIO. Since time is an important factor in this research therefore ACO cannot be a good candidate.

VII. HYBRID EDF, EIS AND HPIO

In this part, we explain the result related to Hybrid EDF, EIS and HPIO. Figure 11 shows how the algorithm switch between EDF and HPIO when load is increasing. As it can be observed until task 35 the system could handle all the tasks by using EDF algorithm which is performed so fast but after that when load is increased to system then it goes to overloaded situation and from EDF slowly it switches to the HPIO algorithm.

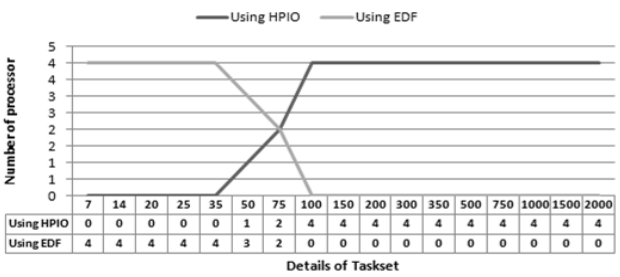
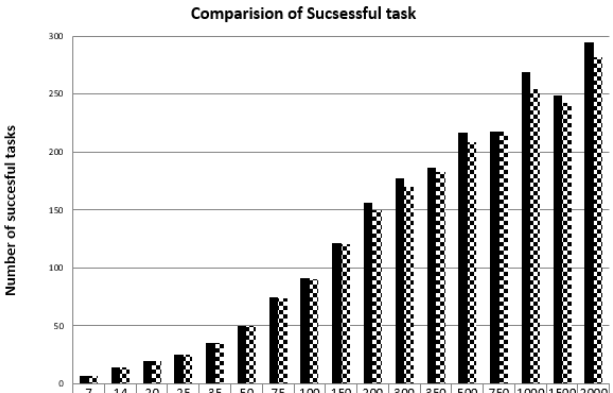


Figure 11: Processor allocation for EDF, EIS and HPIO

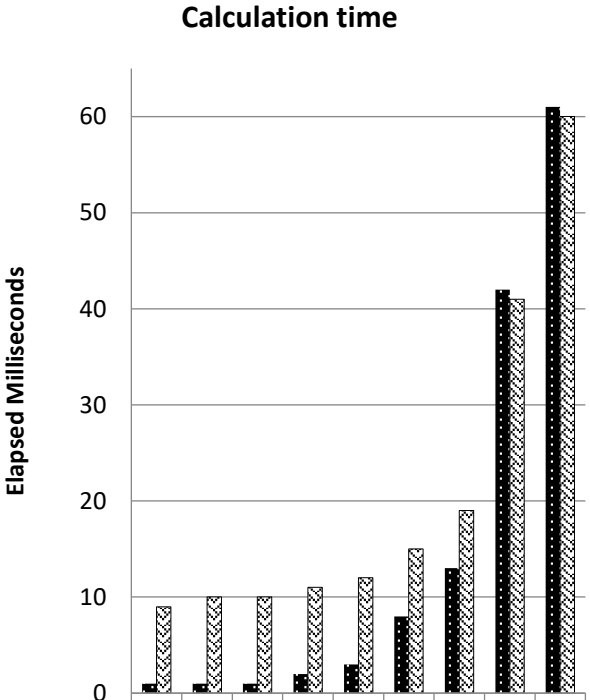
Figure 12, shows the comparison of number of successful tasks in HPIO and HEDFPIO algorithm. As we can observe that we had some improvement in number of tasks successfully scheduled. In most of the cases HEDFPIO could schedule more tasks by combining EDF, EIS and HPIO.



	7	14	20	25	35	50	75	100	150	200	300	350	500	750	1000	1500	2000
HEDFPIO	7	14	20	25	35	50	75	91	121	156	177	186	217	218	269	249	295
HPIO	7	14	20	25	35	50	74	90	120	150	170	183	208	218	254	242	282

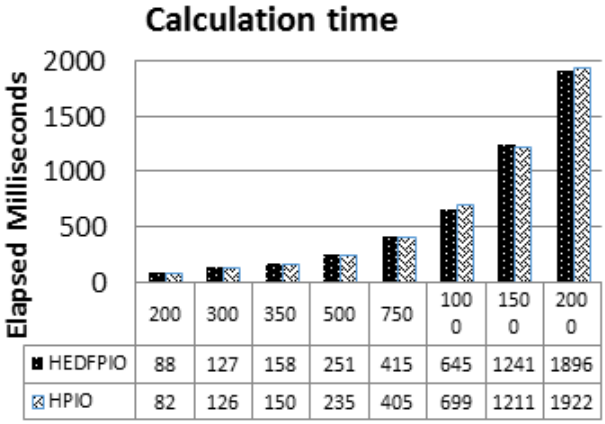
Figure 12: Comparison of Successful Task between HEDFPIO and HPIO

In Figure 13 and 14, we can observe that when system is not overloaded, we can save calculation time since EDF perform so fast in comparison with other algorithm. Hence it is understood that EDF decrease calculation time from task set 7 to 75 while overloaded situation start from 30 tasks in this experiment. As shown in Figure 11, system behavior is also change from task set 35 and it shows that system require more resource for scheduling tasks. By using EDF, EIS and HPIO, total successful schedule tasks improved around 3% and completion time decreases by 1.2%.



	7	14	20	25	35	50	75	100	150
HEDFPIO	1	1	1	2	3	8	13	42	61
HPIO	9	10	10	11	12	15	19	41	60

Figure 13: Calculation time for task set 7 to 150



	200	300	350	500	750	1000	1500	2000
HEDFPIO	88	127	158	251	415	645	1241	1896
HPIO	82	126	150	235	405	699	1211	1922

Figure 14: Calculation time for task set 200 to 2000

VIII. CONCLUSION

In conclusion, as it is observed HPIO can achieve better results in comparison with ACO and PSO for number of successful scheduled tasks. By using EIS algorithm with HPIO, the author could improve the initial population and therefore, better result achieved in multiprocessor. The author combined EDF algorithm with EIS and HPIO for improving the calculation time in multiprocessor. HEDFPIO could be performing better and faster in comparison with HPIO. The author conducts many research to achieve the result and all the algorithms to be implemented in C#.

ACKNOWLEDGMENT

This research was partly funded by Research University Grant of Universiti Teknologi Malaysia (Vote: 07462).

REFERENCES

[1] M. Karimi and H. Motameni, "Tasks scheduling in computational grid using a hybrid discrete particle swarm optimization," *International Journal of Grid and Distributed Computing*, vol. 6, no. 2, pp. 29–38, 2013.

[2] A. R. Mehrabain and C. Lucas, "Novel Numerical optimization algorithm inspired from invasive weed colonization," *Ecological Informatics.*, vol. 1, no. 4, pp. 355–366, 2006.

[3] A. Shah and K. Kotecha, "ACO based dynamic scheduling algorithm for real-time multiprocessor systems," *Internationa. Journal.of Grid and High Performance Computing.*, vol. 3, no. 3, pp. 20–30, 2011.

[4] A. Shah, K. Kotecha, and D. Shah, "Dynamic Scheduling for real-time distributed systems using ant colony optimization," *International Journal of Intelligent Computing and Cybernetics.*, vol. 3, no. 2, pp. 279–292, 2010.

[5] A. Shah and K. Kotecha, "Adaptive scheduling algorithm for real-time multiprocessor systems," in *Proc. IEEE International Conference Advance Computing*, Patiala, India, 2009, pp. 6–7.

[6] M. Karimi, "Hybrid discrete particle swarm optimization for task scheduling in grid computing," *International Journal of Grid and Distributed Computing*, vol. 7, no. 4, pp. 93–104, 2014.

[7] J. Nandanwar and U. Shrawankar, "An adaptive real time task scheduler," *International Journal of Computer Science Issues*, vol. 9, no. 6, pp. 335–340, 2012.

[8] F. Xhafa and A. Abraham, "Nature inspired schedulers in computational grids," *CSI Communications*, vol. 34, no. 11, pp. 28–30, 2011.

[9] M. R. Ghalenoei, H. Hajimirsadeghi, and C. Lucas, "Discrete invasive weed optimization algorithm: Application to cooperative multiple task assignment of UAVs," in *Proceedings of the 48th IEEE Conference on Decision and Control*,Shanghai, 2009, pp. 1665–1670.

[10] V. E. Balas, L. C. Jain, and B. Kovačević, "Soft computing applications," in *Proceedings of the 6th International Workshop Soft Computing Applications*, vol. 1, pp. 338–345, 2015.

[11] K. M. Passino and T. D. Seeley, "Modeling and analysis of nest-site selection by honeybee swarms: The speed and accuracy trade-off," *Behav. Ecol. Sociobiol.*, vol. 59, pp. 427–442, 2006.

[12] M. Karimi and H. Motameni, "Tasks scheduling in computational grid using a hybrid discrete particle swarm optimization," *International Journal of Grid and Distributed Computing*, vol. 6, no. 2, pp. 29–38, 2013.