ACO Based Dynamic Scheduling Algorithm for Real-Time Multiprocessor Systems

Apurva Shah, G H Patel College of Engg & Tech, India

Ketan Kotecha, Nirma University, India

ABSTRACT

The Ant Colony Optimization (ACO) algorithms are computational models inspired by the collective foraging behavior of ants. The ACO algorithms provide inherent parallelism, which is very useful in multiprocessor environments. They provide balance between exploration and exploitation along with robustness and simplicity of individual agent. In this paper, ACO based dynamic scheduling algorithm for homogeneous multiprocessor real-time systems is proposed. The results obtained during simulation are measured in terms of Success Ratio (SR) and Effective CPU Utilization (ECU) and compared with the results of Earliest Deadline First (EDF) algorithm in the same environment. It has been observed that the proposed algorithm is very efficient in underloaded conditions and it performs very well during overloaded conditions also. Moreover, the proposed algorithm can schedule some typical instances successfully which are not possible to schedule using EDF algorithm.

Keywords: *Ant Colony Optimization (ACO), Dynamic Scheduling Algorithm, Multiprocessor Systems, Real-Time Systems, Scheduling*

INTRODUCTION

Real-time systems are defined as those systems in which the correctness of the system depends not only on the logical results of computations but also on the time at which the results are produced (Ramamritham & Stankovik, 1994). The objective of real-time system is to guarantee the deadline of tasks in the system as much as possible when we consider soft real-time system and for achieving this goal vast research on real-time task scheduling has been conducted.

DOI: 10.4018/jghpc.2011070102

Real-time scheduling techniques can be broadly divided into two categories - Static and Dynamic. Static algorithms assign all priorities at design time and it remains constant for the lifetime of the task. Dynamic algorithms assign priority at runtime, based on execution parameters of tasks.

The ACO algorithms are computational models inspired by the collective foraging behavior of ants. Each ant is an autonomous agent that constructs a path. There might be one or more ants concurrently active at the same time. Ants do not need synchronization. Forward ant moves to the good-looking neighbor from the current node, probabilistically. Probabilistic choice is biased by Pheromone trails previously deposited and heuristic function. Without heuristics information, algorithm tends to converge towards initial random solution. In backward mode, ants lay down the pheromone. In ACO algorithm, pheromone is added only to arcs belonging to the global best solution (Dorigo & Gambardella, 1997). Pheromone intensity of all the paths decreases with time, called pheromone evaporation. It helps in unlearning poor quality solution (Dorigo & Stutzle, 2004).

ACO based scheduling algorithm has been presented for single processor real-time system (Shah & Kotecha, 2010). Moreover, ACO has been applied for heterogeneous multiprocessor systems (Turneo et al., 2008; Chang, Wu, Shann, & Chung, 2008) and reconfigurable parallel processing system (Saad, Adawy, & Habashy, 2006). In this paper, ACO based dynamic scheduling algorithm for multiprocessor real-time operating system has been proposed.

RELATED WORK

A multiprocessor system is tightly coupled so that global status and workload information on all processors can be kept current at a low cost. The system has shared memory and generally uses centralized scheduler. If system uses separate scheduler for each processor, the decisions and actions of the schedulers of all the processors are coherent. Multiprocessor systems are divided in two basic types, homogeneous and heterogeneous. In homogeneous system, processors can be used interchangeably and in contrast, heterogeneous processors cannot be used interchangeably. Homogeneous processors can be subdivided in two types: identical and uniform. In identical processors, it is assumed that all processors are equally powerful whereas uniform multiprocessor machine is characterized by a speed (Baruah, Funk, & Goossens, 2003).

There are two main strategies when dealing with the problem of multiprocessor scheduling: partitioning strategy and global strategy (Oh & Son, 1995). In a partitioning strategy, once a task is allocated to a processor, all of its instances are executed exclusively on that processor. In a global strategy, any instance of a task can be executed on any processor, or even be preempted and moved to a different processor before it is completed (Lopez, Diaz, & Garcia. 2004).

EDF (Earliest Deadline First) and LLF (Least Laxity First) algorithms are proved optimal under the condition that tasks are preemptive, there is only one processor and it is not overloaded (Dertouzos & Ogata, 1974; Mok, 1983). EDF is appropriate algorithm to use for on-line scheduling on uniform multiprocessors (Funk, Goossens, & Baruah, 2001). However, many practical instances of multiprocessor real-time system are NP-complete, i.e. it is believed that there is no optimal polynomial-time algorithm for them (Ramamritham, Stankovik, & Shiah, 1990; Ullman, 1973).

The scheduling is considered as on-line, if scheduler makes scheduling decision without knowledge about the task that will be released in the future. On-line scheduling algorithms cannot work efficiently in overloaded conditions. Researchers have proved that, for single processor system, the competitive factor of an on-line scheduling algorithm is at most equal to 0.25 when the system is highly overloaded and its value can't be more than 0.385 when the system is slightly over-loaded. They have further derived the upper limit of competitive factor for dual processor system and it is 0.5 during overloading condition (Liu, 2001; Baruah, Koren, Mishra,, Raghunath, Roiser, & Shasha, 1991). It has been also proved that multiprocessor system is schedulable with guarantee when utilization is below 0.37482 (Lundberg, 2002).

Advantages of ACO based systems are high parallelism and can exhibit high level of robustness, scalability, fault tolerance and effectiveness on unquantifiable data along with simplicity of individual agent (Dorigo & Caro, 1999; Ramos, Muge, & Pina, 2002). This technique combines global and local heuristics to allow step-by-step decisions by a group of cooperating agents and it seems particularly suitable to efficiently explore the search space of this class of problems that can be formulated as stochastic decisions making processes as well as the traveling salesman problem for which this method has been introduced (Dorigo & Gambardella, 1997; Dorigo, Maniezzo, & Colorni, 1996). Several characteristics make ACO a unique approach: it is constructive, population-based metaheuristic which exploits an indirect form of memory of previous performance. These combinations of characteristics are not found in any of the other metaheuristic (Dorigo & Stutzle, 2004).

EDF performs very efficiently during underloaded conditions but it has been experimentally proved that its performance is quite poor when the system is even slightly overloaded (Saini, 2005; Locke, 1986). The proposed algorithm is found quite efficient in underloaded and slightly overloaded conditions. Its performance is satisfactory in highly overloaded conditions also.

SYSTEM AND TASK MODEL

The system assumed here is tightly coupled multiprocessor real-time system with shared memory and soft timing constraints. The processors assumed are homogeneous and identical. The system has adopted global strategy along with centralized scheduler. The system permit task preemption and inter-processor migrations (i.e. a job executing on a processor may be interrupted at any instant and its execution resumed later on the same or a different processor with no cost or penalty).

A task set consists of m independent periodic tasks {T1,...,Tm}. In the periodic model of real-time task Ti = (Ei, Pi), a task Ti is characterized by two parameters: an execution requirement Ei, and a period Pi - with the interpretation that the task generates a job at each integer multiple of Pi and each such job has an execution requirement of Ei units and must complete by a deadline equal to the next integer multiple of Pi (Baruah & Goossens, 2004; Liu, 2001).

In soft real-time systems, each task has a positive value. The goal of the system is to obtain as much value as possible. If a task succeeds, then the system acquires its value. If a task fails, then the system gains less value from the task (Locke, 1986). In a special case of soft real-time systems, called a firm real-time system, there is no value for a task that has missed its deadline, but there is no catastrophe either (Koren & Shasha, 1995; Kotecha & Shah, 2008). The algorithm proposed in this paper applies to firm real-time system and the value of the task has been taken same as its computation time required.

APPLICATION OF ACO IN REAL-TIME MULTIPROCESSOR SYSTEMS

Pseudo-code of the proposed algorithm is given in Box 1.

The detail flowchart of the algorithm shown in Box 1 is given in Figure 1.

Tour Construction

Each schedulable task is considered, as a node and different ants will start the journey starting from one of the node. Ants will continue to traverse till all nodes are visited. Nodes are assumed having memory and they store the value of pheromone.

The starting point of the journey will be different node for each ant. The number of ants taken is same as number of executable tasks the system is having at that time. Therefore, from all the different nodes, one ant will start the journey. All the ants traverse through all the nodes based on probability (pi) of each node. Ants traverse in such a way that all the nodes are visited and no node is repeated.

Probability (pi) of a node at particular time t can be found using following equation:

Box 1. The_New_scheduling_algorithm

```
 {
i.        Construct tour of each  ant and produce different task execution
sequences
ii.        Analyze the task execution sequences generated for available number
of processors
iii.        Update the value of pheromone on each node
iv.        Decide probability of each task and select the task for execution
}
```

$$p_i(t) = \frac{\left(\tau_i(t)\right)^{\alpha} * \left(\eta_i(t)\right)^{\beta}}{\sum_{l \in R_1} \left(\tau_l(t)\right)^{\alpha} * \left(\eta_l(t)\right)^{\beta}}$$

where,

- pi (t) is the probability of ith node at time t; i Є R1 and R1 is set of nodes (schedulable tasks) at time t.
- τi(t) is pheromone on ith node at time t.
- ηi(t) is heuristic value of ith node at time t, which can be determined by,

$$\eta_i = \frac{K}{D_i - t}$$

here, t is current time, K is a constant (suitable range is 5 to 20) and Di is absolute deadline of ith node.

α and β are constants which decide importance of τ and η.

Ants construct their tour based on the value of p of each node as per following:

Ant 1: Highest p → second highest p→ third highest p→…

Ant 2: Second highest p→ highest p→ third highest p→…

Ant 3: Third highest p→ highest p→ second highest p→…

….

Suppose at time t, there are 4 schedulable tasks. Each task will be considered as a node and from each node; one ant will start its journey. If the priorities of all the nodes are assumed in decreasing order of A, B, C, D; ants will traverse different nodes as per following:

Ant 1: A→ B→ C→ D
Ant 2: B→ A→ C→ D
Ant 3: C→ A→ B→ D
Ant 4: D→ A→ B→ C

Analyze the Journey

After all ants have completed their tour, evaluate the performance of different ants' journey for the available number of processors.

The journey of each ant gives us different possibilities of scheduling of the tasks. Find out the number of missed tasks during the journey of each ant (i.e. for possible scheduling order) and decide the best journey produced by an ant during which minimum number of tasks missed the deadline. Update the value of pheromone on each node as per following.

Pheromone Update

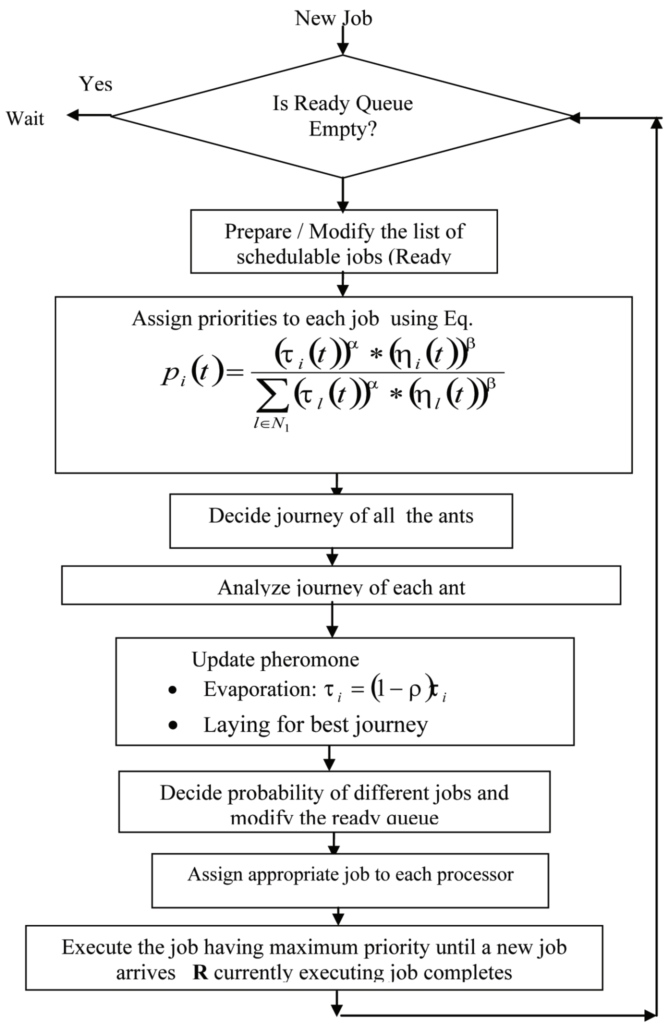
Pheromone updating on each node is done in two parts.
Pheromone Evaporation: Pheromone evaporation is required to forget bad journey of ants and to encourage new paths (Ramos, Muge, & Pina, 2002). Pheromone is evaporated on each node as per following equation:

$$\tau_i = (1 - \rho)\tau_i$$

where,

- ρ is a constant. (suitable range is 0.2 to 0.4)
- i Є R2; R2 is set of all (schedulable and non-schedulable at that time) tasks.
- Pheromone Laying: Pheromone will be laid only for the best journey of ants. Select the

Figure 1. Flowchart of the proposed algorithm



New Job
Is Ready Queue Empty?
Yes
Wait
Prepare / Modify the list of schedulable jobs (Ready
Assign priorities to each job using Eq.
$p_i(t)=\frac{(\tau_i(t))^\alpha * (\eta_i(t))^\beta}{\sum_{l\in N_1}(\tau_l(t))^\alpha * (\eta_l(t))^\beta}$
Decide journey of all the ants
Analyze journey of each ant
Update pheromone
Evaporation: $\tau_i = (1-\rho)\tau_i$
Laying for best journey
Decide probability of different jobs and modify the ready queue
Assign appropriate job to each processor
Execute the job having maximum priority until a new job arrives R currently executing job completes

best journey as discussed earlier and lay pheromone. Amount of pheromone (Δτ) laid will be different at each node and it depends on the order of visited node. The nearest node will get highest amount of pheromone and far most node will get the least. Pheromone is added on each node as per following equation

$$\tau_i = \tau_i + \Delta\tau_i$$

where,

- i Є N1, N1 is set of nodes at time t.
- $\Delta\tau = \frac{ph}{s}$

Here,

- $ph = C * \frac{Number\ of\ Sucessed\ Tasks}{Number\ of\ Missed\ Tasks + 1}$

- s is sequence number of the node visited by the ant during the best journey.
- value of C is constant (preferably 0.1).

Selection of Task to Be Executed By Each Processor

After updating pheromone again find out the value of p for each task. Depending on the available number of processors, select the tasks for execution starting from the highest value of p to its lower values.

Important Points about the Algorithm

- Each schedulable task is considered as a node, and it stores the value of τ i.e. pheromone. Initial value of τ is taken as one.
- Number of ants required to construct the tour is an important design criteria. During simulation, we have taken number of ants same as number of executable tasks the system is having at that time. Therefore, from each node one ant will start its journey during tour construction.
- Value of α and β decide importance of τ and η. During simulation, we got best results with $\alpha = 1$ and $\beta = 2$.

Unlike the real ants, the pheromone is laid only for the best journey of ants. Amount of pheromone laid is also not uniform. Moreover, for forgetting the poor journey, pheromone is evaporated.

SIMULATION METHOD

The proposed algorithm and EDF algorithms are simulated for the real-time multiprocessor system with homogeneous processors and preemptive environment. The results are taken using periodic tasks. The relative deadline of the task is considered the same as its period i.e. before the arrival of the next job; present job is expected to be completed. The task set generated for each load value consist of 4 different values of period. For taking result at each load value, we have generated 200 task sets each one containing 3 to 90 tasks.

For periodic tasks, load of the system can be defined as summation of ratio of required computation time and period of each task. The results are taken from underloaded conditions ($0.5 \leq$ load) to highly overloaded conditions (load ≤ 3) and tested on more than 1 lakh scheduling instances.

The system is said to be overloaded when even a clairvoyant scheduler cannot feasibly schedule the tasks offered to the scheduler. A reasonable way to measure the performance of a scheduling algorithm during an overload is by the amount of work the scheduler can feasibly schedule according to the algorithm. The larger this amount the better the algorithm. Because of this, we have considered following two as our main performance criteria:

In real-time systems, deadline meeting is most important and we are interested in finding whether the task is meeting the deadline. Therefore the most appropriate performance metric is the Success Ratio (SR) and defined as (Ramamritham, Stankovik, & Shiah, 1990):

$$SR = \frac{Number\ of\ tasks\ successfully\ scheduled}{Total\ number\ of\ tasks\ \ arrived}$$

It is important that how efficiently the processors are utilized by the scheduler especially during overloaded conditions. Therefore the other performance metric is Effective CPU Utilization (ECU) and defined as:

$$ECU = \sum_{i \in R} \frac{V_i}{T}$$

where,

- V is value of a task and,
  - value of a task = Computation time of a task, if the task completes within its deadline.

Table 1. Task Set for first case

Task	Arrival Time	Absolute Deadline	Required Exe. Time
A	0	10	3
B	0	10	4
C	0	11	10

    - value of a task = 0, if the task fails to meet the deadline.
- R is set of tasks which are scheduled successfully i.e. completed within their deadline.
- T is total time of scheduling.

An on-line scheduler has a competitive factor Cf if and only if the value of the schedule of any finite sequence of tasks produced by the algorithm is at least Cf times the value of the schedule of the tasks produced by an optimal clairvoyant algorithm (Liu, 2001). Since maximum value obtained by a clairvoyant scheduling algorithm is a hard problem, we have instead used a rather simplistic upper bound on this maximum value, which is obtained by summing up the value of all tasks (Baruah, Koren, Mishra, Raghunath, Roiser, & Shasha, 1991). Therefore, value of ECU for clairvoyant scheduler has been considered as 100%.

Finally, the results are obtained, compared with EDF algorithm in the same environment and shown in Figure 3 to Figure 5.

DEMONSTRATION OF SCHEDULING OF SOME TYPICAL INSTANCES

In this section, two cases consisting of different task sets are given and their scheduling with EDF algorithm and the proposed algorithm are discussed. First case discusses scheduling of a task set of aperiodic tasks and second case discusses the same for task set consisting of periodic tasks.

First task set is made up of aperiodic tasks and shown in Table 1. The system is considered having two processors. EDF schedule of the task set is shown in Figure 2(a) and it can be observed that task C missed the deadline

Scheduling using the proposed Algorithm:

1. At T = 0, the proposed algorithm works as shown in Table 2. Out of 3 eligible tasks, processor 1 (P1) selects task C and processor 2 (P2) selects task A for execution. Intermediate steps along with values of variables are shown in the Table 2.

Therefore, the scheduling by the proposed algorithm results as shown in Figure 2(b). We can observe that EDF algorithm has failed as it missed the deadline for task C, but the proposed algorithm has been succeeded.

2. Second task set is made up of periodic tasks and shown in Table 3. The system is considered having three processors and load of the system is 0.95.

Using both algorithms results are taken and measured in terms of SR and ECU. For the proposed algorithm we get SR = 100% & ECU = 95.32%, whereas EDF gives SR = 96.38% & ECU = 74.43%. For EDF algorithm, value of SR is less than 100% as it is failed to schedule all the tasks, but the proposed algorithm has got value of SR as 100%. Therefore, it has been concluded that proposed algorithm has scheduled all the tasks successfully.

RESULTS

Figure 3 shows the results of the proposed algorithm and EDF algorithm during simulation with two processors. The results are taken from

Figure 2. Scheduling for First Case Using (a) EDF Algorithm (b) New Algorithm

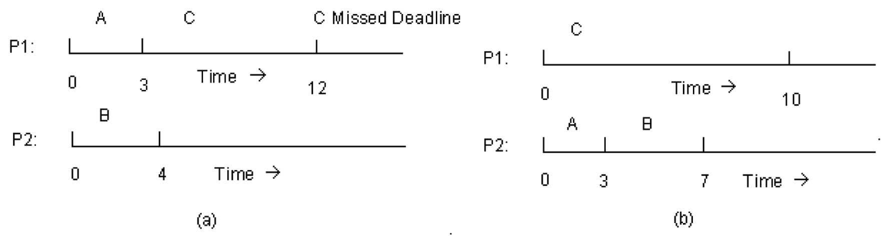


Table 2. Steps of the proposed algorithm at time T =0

Tour of Ant	Path	Analyze Journey		Imp. After Pheromone Update	Select Task at T=0
		Succ. Tasks	Miss. Tasks		
1	A,B,C	2	1	C(max) A B(min)	P1: C P2: A
2	B,A,C	2	1		
3	C,A,B	3	0		

Table 3. Task set for second case

Task No.	Arrival Time	Period	Relative Deadline	Required Exe. Time
1	0	10	10	2
2	0	10	10	2
3	1	3	3	1
4	2	5	5	1
5	3	3	3	1
6	2	10	10	1
7	2	3	3	1
8	0	11	11	10
9	0	8	8	2

underloaded condition (load ≤ 0.5) to highly overloaded condition (load ≥ 3).

Results show that the proposed algorithm performs very well when the system is not overloaded. During overloaded condition, as expected ECU and SR of EDF fall down rapidly but proposed algorithm works exceptionally well. The number of processors is increased during simulation and the results are shown in Figure 4 and Figure 5 with number of processors is 5 and 15 respectively.

For underloaded condition (i.e. load ≤ 1), performance of the proposed algorithm is quite well and nearer to optimal clairvoyant scheduling algorithm. As the load increases just beyond 1.00, EDF is not able to schedule. Even a spike of overload can affect the system seriously for long period of time. But we can observe that

Figure 3. Load Vs % SR and Load Vs %ECU with number of processor = 2

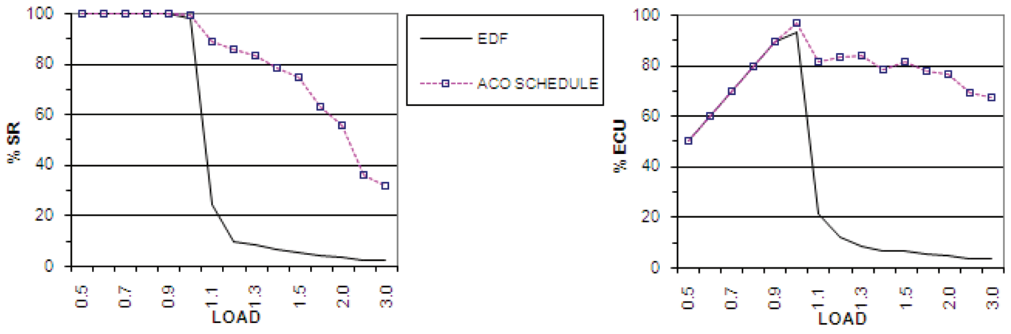


Figure 4. Load Vs % SR and Load Vs %ECU with number of processor =5

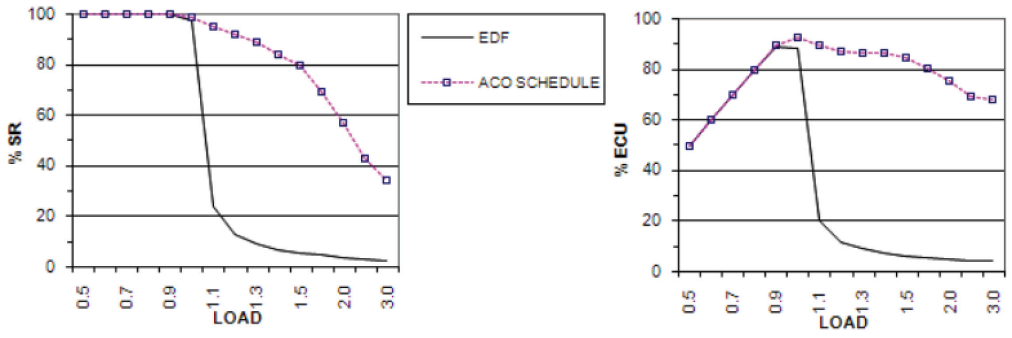
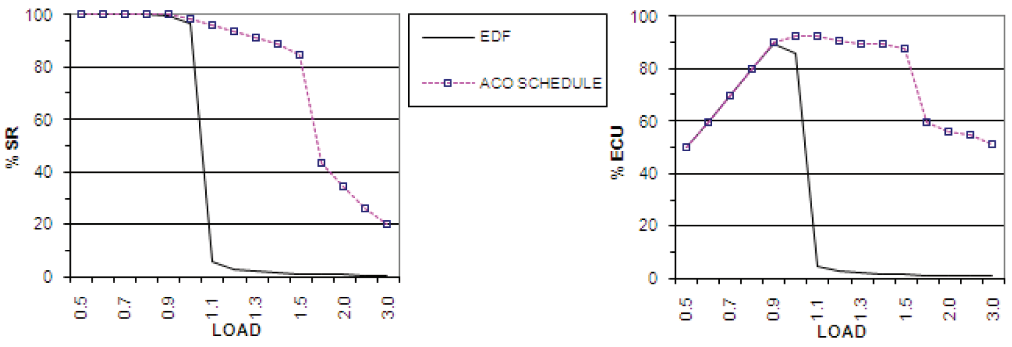


Figure 5. Load Vs % SR and Load Vs %ECU with number of processor =15



ACO based algorithm is working effectively and sustain the overload. It gives value of ECU more than 75% up to the load value 1.5. Because of this characteristic, the proposed algorithm is more suitable for the situation when future work load of the system is unpredictable. During highly overloaded situations (load ≥ 2) also the proposed algorithm works satisfactorily and gives value of ECU more than 50% up to the load value 2.50.

ECU measures that how efficiently the processors are utilized. By considering maximum value of ECU (i.e. 100%) for clairvoyant scheduler, the competitive factor achieved by the proposed algorithm as well as EDF algorithm have been found. For the proposed algorithm, competitive factor is at least 0.91 up to load value is equal to 1 whereas for EDF its value is 0.83. Moreover, we find that the competitive factor of the proposed algorithm is more than 0.75 and 0.55 when loading factor is 1.50 and 2.00 respectively where as in the same conditions competitive factor achieved by EDF algorithm is less than 0.06 and 0.04 respectively.

The algorithm is dynamic. During simulation periodic tasks are considered but it can handle aperiodic tasks as well. For periodic task, even if arrival time is changed, the algorithm can work effectively. Moreover, the algorithm can work with available number of processors. Therefore, even if one processor (except master) fails in between, the system works normally.

CONCLUSION

The algorithm discussed in this paper is dynamic (or online) scheduling algorithm for real-time multiprocessor systems using the concept of ACO. The algorithm finds importance of each schedulable task using pheromone value on a task and heuristic function, which is proportional to deadline of the task. We can conclude the following from results achieved during simulation:

The proposed algorithm performs efficiently for the multiprocessor system having homogenous, identical processors and soft timing constraints. The algorithm performs very well during underloaded conditions and slightly overloaded conditions. During highly overloaded conditions also it performs quite satisfactorily. Many instances, which are not possible to schedule using EDF algorithm, can be scheduled successfully using the proposed algorithm. The only disadvantage of the algorithm is: Time required for decision of scheduling is more, compared to EDF algorithm. But the inherent parallelism provided by the algorithm reduces the intensity of this disadvantage.

REFERENCES

Baruah, S., Funk, S., & Goossens, J. (2003). Robustness results concerning EDF scheduling upon uniform multiprocessors. *IEEE Transactions on Computers*, *52*(9), 1185–1195. doi:10.1109/TC.2003.1228513

Baruah, S., & Goossens, J. (2004). Rate-monotonic scheduling on uniform multiprocessors. *IEEE Transactions on Computers*, *52*(7), 966–970. doi:10.1109/TC.2003.1214344

Baruah, S., Koren, G., Mishra, B., Raghunath, A., Roiser, L., & Shasha, D. (1991). On-line scheduling in the presence of overload. In *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science* (pp. 100-110).

Chang, P. C., Wu, I. W., Shann, J. J., & Chung, C. P. (2008). ETAHM: An energy-aware task allocation algorithm for heterogeneous multiprocessor. In *Proceedings of the 45th Conference on Design Automation* (pp. 776-779).

Dertouzos, M., & Ogata, K. (1974). Control robotics: The procedural control of physical process. In *Proceedings of the IFIP Congress* (pp. 807-813).

Dorigo, M., & Caro, G. (1999). The ant colony optimization metaheuristic . In Corne, D., Dorigo, M., & Glover, G. (Eds.), *New ideas in optimization*. New Delhi, India: McGraw-Hill.

Dorigo, M., & Gambardella, L. M. (1997). Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, *1*(1), 53–66. doi:10.1109/4235.585892

Dorigo, M., Maniezzo, V., & Colorni, A. (1996). The ant system: Optimization by a colony of cooperative agents. *IEEE Transactions on Systems, Man, and Cybernetics. Part B, Cybernetics*, *26*(1), 29–41. doi:10.1109/3477.484436

Dorigo, M., & Stutzle, T. (2004). *Ant colony optimization*. Cambridge, MA: MIT Press.

Funk, S., Goossens, J., & Baruah, S. (2001). On-line scheduling on uniform multiprocessors. In *Proceedings of the IEEE Real-Time Systems Symposium* (pp. 183-192).

Koren, G., & Shasha, D. (1995). DOver: An optimal on-line scheduling algorithm for overloaded real-time systems. *SIAM Journal on Computing*, *24*, 318–339. doi:10.1137/S0097539792236882

Kotecha, K., & Shah, A. (2008). Efficient dynamic scheduling algorithms for real-time multiprocessor system. In *Proceedings of the International Conference on High Performance Computation Networking and Communications* (pp. 21-25).

Liu, W. S. (2001). *Real-time systems*. New Delhi, India: Pearson Education.

Locke, C. D. (1986). *Best effort decision making for real-time scheduling*. Unpublished doctoral dissertation, Carnegie-Mellon University, Pittsburgh, PA.

Lopez, J. M., Diaz, J. L., & Garcia, D. F. (2004). Minimum and maximum utilization bounds for multiprocessor rate monotonic scheduling. *IEEE Transactions on Parallel and Distributed Computing*, *15*(7), 642–653. doi:10.1109/TPDS.2004.25

Lundberg, L. (2002). Analyzing fixed-priority global multiprocessor scheduling. In *Proceedings of the Eighth IEEE Real-Time and Embedded Technology and Applications Symposium* (pp. 145-153).

Mok, A. (1983). *Fundamental design problems of distributed systems for the hard-real-time environment.* Unpublished doctoral dissertation, Massachusetts Institute of Technology, Cambridge, MA.

Oh, Y., & Son, S. H. (1995). Allocating fixed-priority periodic tasks on multiprocessor systems. *Real-Time Systems*, *9*(3), 207–239. doi:10.1007/BF01088806

Ramamritham, K., & Stankovik, J. A. (1994). Scheduling algorithms and operating support for real-time systems. *Proceedings of the IEEE*, *82*(1), 55–67. doi:10.1109/5.259426

Ramamritham, K., Stankovik, J. A., & Shiah, P. F. (1990). Efficient scheduling algorithms for real-time multiprocessor systems. *IEEE Transactions on Parallel and Distributed Systems*, *1*(2), 184–190. doi:10.1109/71.80146

Ramos, V., Muge, F., & Pina, P. (2002). Self-organized data and image retrieval as a consequence of inter-dynamic synergistic relationships in artificial ant colonies. In *Proceedings of the Second International Conference on Hybrid Intelligent System*, Santiago, Chile (pp. 500-512).

Saad, E. M., Adawy, M. E., & Habashy, S. M. (2006). Reconfigurable parallel processing system based on a modified ant colony system. In *Proceedings of the 23rd National Conference on Radio Science*, Menoufiya, Egypt (pp. 1-11).

Saini, G. (2005). Application of fuzzy-logic to real-time scheduling. In *Proceedings of the 14th IEEE Real-Time Conference* (pp. 60-63).

Shah, A., & Kotecha, K. (2010). Dynamic scheduling algorithm for real-time operating system using ACO. In *Proceedings of the International Conference on Computing Intelligence and Communication Networks* (pp. 617-621).

Turneo, A., Pilato, C., Ferrandi, F., Sciuto, D., & Lanzi, P. L. (2008). Ant colony optimization for mapping and scheduling in heterogeneous multiprocessor systems. In *Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation* (pp. 142-149).

Ullman, J. D. (1973). Polynomial complete scheduling problems. *Operating Systems Review*, *7*(4), 96–101. doi:10.1145/957195.808055