International Journal of Intelligent Information Systems
2015; 4(2-1): 13-17
Published online January 28, 2015 (http://www.sciencepublishinggroup.com/j/ijiis)
doi: 10.11648/j.ijiis.s.2015040201.13
ISSN: 2328-7675 (Print); ISSN: 2328-7683 (Online)

On-line scheduling algorithm for real-time multiprocessor systems with ACO

Cheng Zhao, Myungryun Yoo, Takanori Yokoyama

Department of Computer Science, Tokyo City University, Tokyo, Japan

Email address:
zcs88122@gmail.com (Cheng Zhao), yoo@cs.tcu.ac.jp (Myungryun Yoo), yokoyama@cs.tcu.ac.jp (Takanori Yokoyama)

To cite this article:
Cheng Zhao, Myungryun Yoo, Takanori Yokoyama. On-Line Scheduling Algorithm for Real-Time Multiprocessor Systems with ACO. *International Journal of Intelligent Information Systems*. Special Issue: Logistics Optimization Using Evolutionary Computation Techniques. Vol. 4, No. 2-1, 2015, pp. 13-17. doi: 10.11648/j.ijiis.s.2015040201.13

Abstract: The Ant Colony Optimization algorithms (ACO) are computational models inspired by the collective foraging behavior of ants. By looking at the strengths of ACO, they are the most appropriate for scheduling of tasks in soft real-time systems. In this paper, ACO based scheduling algorithm for real-time operating systems (RTOS) has been proposed. During simulation, results are obtained with periodic tasks, measured in terms of Success Ratio & Effective CPU Utilization and compared with Kotecha's algorithm in the same environment. It has been observed that the proposed algorithm is equally optimal during underloaded conditions and it performs better during overloaded conditions.

Keywords: Real-Time Systems, Scheduling, ACO, EDF

1. Introduction

In recent year, applications of real-time systems are spreading. For example, the automotive, mobile phone, plant monitoring systems and air traffic control systems.

There are two types of real-time systems: *Hard real-time* systems and *Soft real-time* systems. *Hard real-time* systems are defined as those systems in which the correctness of the system depends not only on the logical result of computation, but also on the time at which the results are produced[1]. *Soft real-time* systems are missing an occasional deadline is undesirable, but nevertheless tolerable. Our interest in this question stems from the increasing prevalence of applications such as networking, multimedia, and immersive graphics systems that have only *Soft real-time* systems.

The objective of real-time task scheduler is to reduce the deadline of tasks in the system as much as possible when we consider soft real time system. To achieve this goal, vast researches on real-time task scheduling have been conducted. Mostly all the real time systems in existence use preemption and multitasking. Real time scheduling techniques can be broadly divided into two categories: Off-line and On-line.

Off-line algorithms assign all priorities at design time, and it remains constant for the lifetime of a task. On-line algorithms assign priority at runtime, based on execution parameters of tasks. On-line scheduling can be either with static priority or dynamic priority. RM (Rate Monotonic) and DM (Deadline Monotonic) are examples of On-line scheduling with static priority [2]. EDF (Earliest Deadline First) and LST (Least Slack Time First) are examples of On-line scheduling with dynamic priority. EDF and LST algorithms are optimal under the condition that the jobs are preemptable, there is only one processor and the processor is not overloaded [3][4]. But the limitation of these algorithms is, their performance decreases exponentially if system becomes slightly overloaded [5].

Several characteristics make ACO a unique approach: it is constructive, population-based meta-heuristic which exploits an indirect form of memory of previous performance. [6][7]. Therefore in this paper, the same approach has been applied for real-time operating systems.

The rest of this paper is organized as follows. In Sec. 2, our system model is presented. In Sec. 3, our proposed algorithm is described and discussed. In Sec. 4, a simulation-based evaluation of proposed algorithm and kotecha's algorithm[8] . Sec. 5 is conclusions.

2. System Model

The system knows about the deadline and required computation time of the task when the task is released. The task set is assumed to be preemptive. We have assumed that

the system is not having resource contention problem. Moreover, preemption and the scheduling algorithm incur no overhead.

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 [8]. 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 [9]. Here, we propose an algorithm that applies to firm real-time system. The value of the task has been taken same as its computation time required [10].

3. Related Work

3.1. Ant Colony Optimization

Social insects that live in colonies, such as ants, termites, wasps, and bees, develop specific tasks according to their role in the colony. One of the main tasks is the search for food. Real ants, when searching for food, can find such resources without visual feedback, and they can adapt to changes in the environment, optimizing the path between the nest and the food source. This fact is the result involves positive feedback, given by the continuous deposit of a chemical substance, known as pheromone.

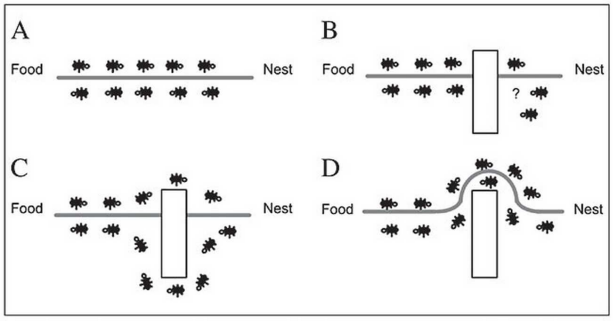

Figure 1. *Ant colony optimization.*

A classic example of the construction of a pheromone trail in the search for a shorter path is shown in Fig. 1 and was first presented by Colorni[11]. In Fig. 1A there is a path between food and nest established by the ants. In Fig. 1B an obstacle is inserted in the path. Soon, ants spread to both sides of the obstacle, since there is no clear trail to follow (Fig. 1C). As the ants go around the obstacle and find the previous pheromone trail again, a new pheromone trail will be formed around the obstacle. This trail will be stronger in the shortest path than in the longest path, as shown in Fig. 1D.

3.2. Kotecha's Algorithm

The scheduling algorithm is required to execute when a new task arrives or presently running task completes. The main steps of the proposed algorithm are given as following and the flowchart of the algorithm has been shown in Fig.2:

- Construct tour of different ants and produce the task execution sequence
- Analyze the task execution sequences generated for available number of processor
- Update the value of pheromone
- Decide probability of each task and select the task for execution

The detailed description of four main steps is as follows:

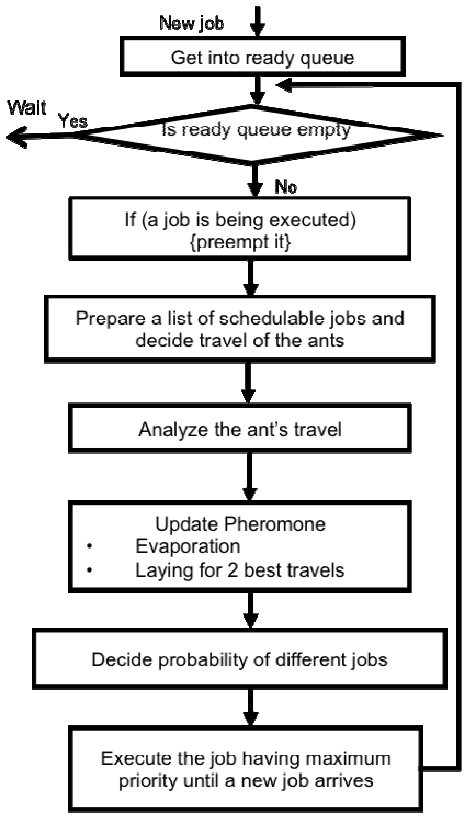

Figure 2. *Flowchart of algorithm*

3.2.1. Tour Construction

First, find probability of each node using (1). Each schedulable task is considered as a node and probability of each node to be selected for execution is decided using pheromone τ and heuristic value η.

$$p_i(t) = \frac{(\tau_i(t))^{\alpha} * (\eta_i(t))^{\beta}}{\sum_{l \in R1} (\tau_i(t))^{\alpha} * (\eta_i(t))^{\beta}} \quad (1)$$

where,

pi(t) is the probability of ith node at time t; i∈ N_1 and N1 is set of schedulable tasks at time t.

τi(t) is pheromone on ith node at time t.

η_i is heuristic value of i^{th} node at t, which can bedetermined by (2)

$$\eta_i = \frac{K}{D_i - t} \quad (2)$$

Here, t is current time ,K is constant 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 > second highest p > highest p >….

…

Suppose at time t, there are 4 schedulable tasks. As shown in Figure 1, each task will be considered as a node and from each node; one ant will start its journey. If we consider the priorities of all the nodes are 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

3.2.2. Analyze the Journey

After all ants have completed their tour, evaluate the performance of different ants's travel. We have analyzed this based on ratio of number of success tasks and number of missed tasks. Find out maximum two best journeys of ants and update the value of pheromone accordingly.

3.2.3. Pheromone Update

Pheromone updating on each node is done in two steps:

- Pheromone Evaporation :Pheromone evaporation is required to forget bad travel of ants and to encourage new paths. Value of τ is updated using(3)

$$\tau_i(t+1) = (1-\rho)\tau_i(t) \tag{3}$$

where,

ρ is a constant.

$i \in R_l$; R_l is set of all tasks.

- Pheromone Laying: Pheromone will be laid only for two best journeys of ants. Select the best journey and put pheromone depending on their order of visited node. Amount of pheromone (Δτ) laid will be different at each node i.e. the nearest node will get highest amount of pheromone and far most node will get least.

$$\tau_i(t+1) = \tau_i(t) + \Delta\tau_i \tag{4}$$

where,

$i \in N_2$; N_2 is set of tasks executed by the ant.

$$\Delta\tau = \frac{ph}{s} \tag{5}$$

Here,

$$ph = C * \frac{Number\ of\ Successed\ Jobs}{Number\ of\ Missed\ Jobs+1} \tag{6}$$

s is sequence number of node visited by the ant during the best travel.

Value of C is constant (preferably 0.1)

3.2.4. Selection of Task for Execution

After updating pheromone, again find out probability of each node using (1) and select the task for execution having the highest probability value.

3.2.5. 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 for all nodes.

Value of α and β decide importance of τ and η. During simulation, both values are taken as one.

Number of ants which construct the tour, is important design criteria. During simulation, number of ants taken is same as number of executable tasks the system is having at that time.

3.3. Proposed Algorithm

In Kotecha's algorithm ,The part of Tour Construction, the heuristic value ηi depend on absolute deadline that is most similar to EDF algorithm. However, EDF's performance decreases exponentially if system becomes slightly overloaded[12]. Since we consider the heuristic value should be depend on a ratio of remain execute time and absolute deadline at time t. This can make more helpful heuristic value to increase the Success ratio of jobs .

And in the part of pheromone evaporation, executable tasks and executed tasks without distinction when the stage of updating pheromone. We consider the search history of a previous time when scheduler startup is a reference value. The tasks are periodic, they are released by each period, when they exit run once , they should be released in the near future, Therefore, selected tasks have high probability be selected again. At the pheromone evaporation stage, we only chose the schedulable tasks to evaporate. The selected tasks are still with high pheromone until next period.

We proposed a new pheromone evaporation that don't make all tasks forget traveled nodes, only the task which schedulable at time t. It as follows :

3.3.1. New Tour Construction

First, find probability of each node using (7). Each schedulable task is considered as a node and probability of each node to be selected for execution is decided using pheromone τ and heuristic value η.

$$p_i(t) = \frac{(\tau_i(t))^{\alpha} * (\eta_i(t))^{\beta}}{\sum_{l \in R1} (\tau_i(t))^{\alpha} * (\eta_i(t))^{\beta}} \tag{7}$$

where,

pi(t) is the probability of ith node at time t; i∈ N_1 and N1 is set of schedulable tasks at time t.

τi(t) is pheromone on ith node at time t.

ηi is heuristic value of ith node at t, which can be determined by (8),

$$\eta_i = \begin{cases} 0, & otherwise \\ \frac{K*D_i}{E_i(t)}, & if \ \frac{D_i}{E_i(t)} \le 1 \end{cases} \tag{8}$$

Here, t is current time, K is constant and Di is absolute deadline of ith node, $E_i(t)$ is remain execute time of ith node.

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

3.3.2. New Pheromone Evaporation

Pheromone evaporation is required to forget bad travel of ants and to encourage new paths. Value of τ is updated using (9).

$$\tau_i(t+1) = (1-\rho)\tau_i(t) \quad (9)$$

where,

ρ is a constant.

$i \in N_1$; N_1 is set of schedulable tasks at time t.

4. Simulation and Results

We have implemented our algorithm & Kotecha's algorithm and have run simulations to accumulate empirical data. We have considered periodic tasks for taking the results. For periodic tasks, load of the system can be defined as summation of ratio of executable time and period of each task. For taking result at each load value, we have generated 200 task sets each one containing 3 to 9 tasks. The results for 5 different values of load are taken ($0.7 \leq$ load ≤ 2.5) and tested on more than 35,000 tasks. Results are shown in Table 1 and Fig. 2.

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:

1) 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 and defined as (10) [5],

$$SR = \frac{Number\ of\ Jobs\ Successefully\ schedled}{Total\ number\ of\ Jobs\ arrived} \quad (10)$$

2) 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 (11):,

$$ECU = \sum_{i \in R} \frac{V_i}{T} \quad (11)$$

Where,

V is value of task and

- value of a task = Computation time of a task, if the task completes within its deadline.
- value of a task = 0, if the task miss the deadline.

R is set of tasks, which are completed within their deadline.

T is total time of scheduleing.

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 [7]. 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 [14]. Therefore, we have considered value of ECU for clairvoyant scheduler is 100%.

Finally, the results are obtained, compared with Kotecha's algorithm in the same environment and shown in Fig.3 and Fig.4.

Table 1. *Results: Success Ratio of job*

Load	Kotecha's Algo	New Algo.
0.7	100%	100%
1.0	100%	100%
1.5	61.60%	66.70%
2.0	47.30%	54.70%
2.5	36.70%	48.80%

Fig.3 and Table1 shows the results achieved by the proposed algorithm and Kotecha's algorithm. We can obseve that the proposed Algorithm have a same performance with Kotecha's algorithm. However, we find that proposed algorithm is definitely more than 5% and12% when load values are 1.5 and 2.5.

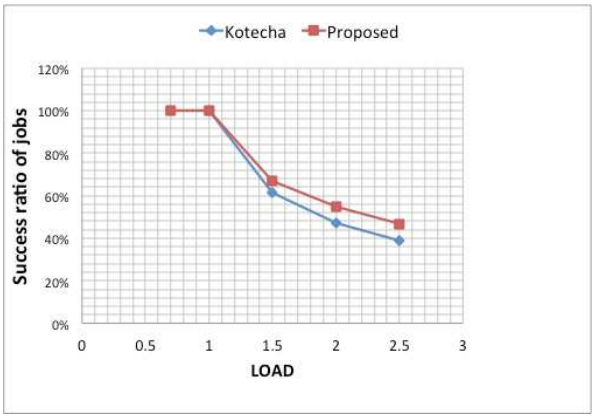

Figure 3. *Success ratio of jobs*

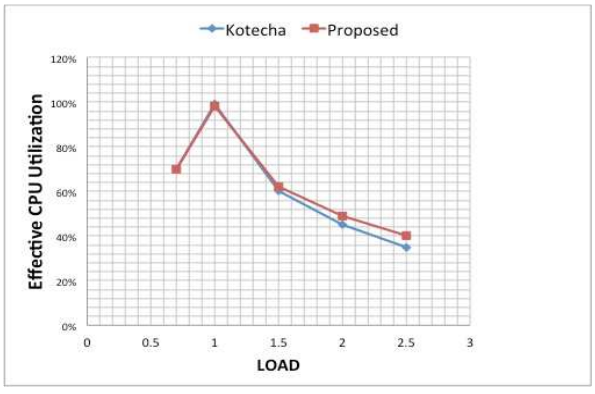

Figure 4. *Effective CPU Utilization.*

Fig. 4 and Table2 shows the results of Effective CPU Utilization achieved by the proposed algorithm and Kotecha's algorithm. We can observe that the proposed Algorithm have a same performance with Kotecha's algorithm. However, we

find that proposed algorithm is definitely more than 5% and12% when load values are 1.5 and 2.5.

Table 2. *RESULTS: Effective CPU Utilization*

Load	Kotecha's Algo	New Algo.
0.7	100%	100%
1.0	100%	100%
1.5	61.60%	66.70%
2.0	47.30%	54.70%
2.5	36.70%	48.80%

5. Conclusions

The algorithm discussed in this paper is for scheduling of soft real-time system with single processor and preemptive task sets. For scheduling, the concept of ACO has been introduced. The algorithm is simulated with periodic task sets; results are obtained and compared with Kotecha's algorithm.

From the results of simulation we can conclude that the proposed algorithm performs equally optimal for single processor, preemptive environment when the system is underloaded. We can also observe that the proposed algorithm during overloaded conditions performance is better than Kotecha's algorithm.

Acknowledgements

This work is supported in part by JSPS KAKENHI Grant Number 24500046.

References

[1] K. Ramamritham and J. A. Stankovik, "Scheduling algorithms and operating support for real-time systems", Proceedings of the IEEE, vol. 82, pp. 56-76, January 1994.

[2] C. L. Liu and L. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment", Journal of ACM, vol.20, pp: 46-61, January 1973.

[3] M. Dertouzos and K. Ogata, "Control robotics: The procedural control of physical process," Proc. IFIP Congress, pp. 807-813, 1974.

[4] A. Mok, "Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment," Ph.d. thesis, MIT, Cambridge, Massachusetts, May 1983.

[5] G. Saini, "Application of Fuzzy logic to Real-time scheduling", Real-Time Conference, 14th IEEE-NPSS.pp.113-116, 2005.

[6] M. Dorigo and G. Caro, "The Ant Colony Optimization Metaheuristic in D. Corne, M. Dorigo and F. Glover(eds)", New Ideas in Optimization, McGraw Hill, 1999.

[7] V. Ramos, F. Muge, and P. Pina, "Self-organized data and image retrieval as a consequence of inter-dynamic synergistic relationships in artificial ant colonies", In Second International Conference on Hybrid Intelligent System, IOS Press, Santiago, 2002.

[8] Kotecha and A Shah, "Scheduling Algorithm for Real-Time Opeating Systems using ACO", 2010 Intelligence and Communication Networks, pp. 617–621, Nov 2010.

[9] C. D. Locke, "Best Effort Decision Making for Real-Time Scheduling", Ph.d. thesis, Computer Science Department, Carnegie-Mellon University, 1986.

[10] G. Koren and D. Shasha, "Dover: An optimal on-line scheduling algorithm for overloaded real-time systems", SIAM Journal of Computing, 24(2): 318-339 April 1995.

[11] A Shah, K Kotecha and D Shah, "Adaptive scheduling algorithm for real-time distributed systems", To appear in International Journal of Intelligent Computing and Cybernetics.

[12] A. Colorni, M. Dorigo, and V. Maniezzo, "Distributed optimization by ant colonies," In: Proceedings of European Conf. on Artificial Life. Elsevier, Amsterdam, pp. 134-142, 1991.

[13] K. Ramamritham, J. A. Stankovik, and P. F. Shiah, "Efficient scheduling algorithms for real-time multiprocessor systems", IEEE Transaction on Parallel and Distributed Systems, vol. 1, April 1990.

[14] S. Baruah, G. Koren, B. Mishra, A. Raghunath, L. Roiser, and D. Shasha, "On-line scheduling in the presence of overload," In FOCS, pp. 100–110 1991.