On-Line Scheduling Algorithm for Real-Time Multiprocessor Systems with ACO and EDF

Cheng Zhao, Myungryun Yoo, Takanori Yokoyama
Department of computer science, Tokyo City University
1-28-1 Tamazutsumi, Setagaya-ku
Tokyo, Japan

***Abstract*—Preemptive Earliest Deadline First (EDF) has been proved to be optimal scheduling algorithm for single-processor systems. 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 & context switch and compared with Kotecha's algorithm in the same environment. It has been observed that the proposed algorithm is equally optimal during underload conditions and it performs better during overloaded conditions.**

Keywords— Real-time system; sceduling; ACO; EDF

I. INTRODUCTION

In recent year, the applications of real-time systems have attracted attention. 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 miss 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. Rate Monotonic (RM) and Deadline Monotonic (DM) are examples of On-line scheduling with static priority [2]. Earliest Deadline First (EDF) and Least Slack Time First (LST) 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 section 2, our system model is presented. In section 3 related work is described. In section 4 our proposed algorithm is described and discussed. In section 5 a simulation-based evaluation of proposed algorithm and kotecha's algorithm [8]. Section 6 is conclusions.

II. 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.

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

III. RELATED WORK

We will discus about ACO and EDF. Kotecha's algorithm is combination of two scheduling algorithms: EDF algorithm and ACO based Scheduling algorithm.

A. 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

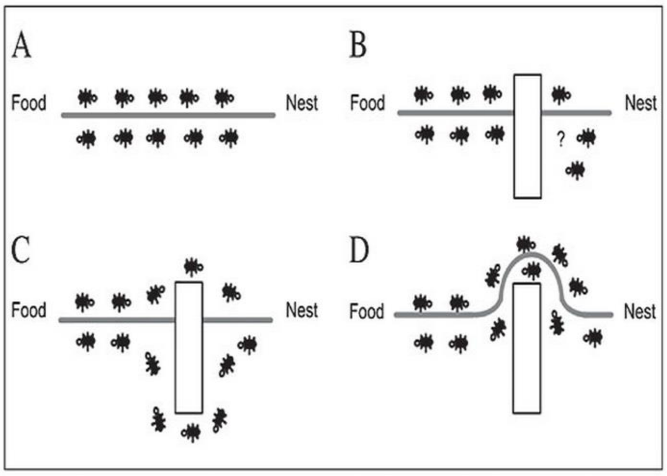

Fig.1. Ant colony optimization

by the continuous deposit of a chemical substance, known as pheromone.

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.

B. EDF algorithm

The priority of each task is decided based on value of its deadline. The task with the nearest deadlines has the highest priority. Number of tasks equivalent to number of processors is selected for execution on different processors by centralized scheduler.

C. ACO Based scheduling 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:

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

The detailed description of four main steps is as follows:

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,

$p_i(t)$ is the probability of ith node at time t; i$\in N_1$ and N_1 is set of schedulable tasks at time t.

$\tau_i(t)$ is pheromone on ith node at time t.

η_i is heuristic value of ith node at t, which can be determined by (2).

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

Here, t is current time, K is constant and D_i 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:

Ant1. Highest p > second highest p > third highest p >….

Ant2. Second highest p > highest p > third highest p >….

Ant3. 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:

Ant1. A > B > C > D

Ant2. B > A > C > D

Ant3. C > A > B > D

Ant4. D > A > B > C

2) Analyze the Journey

After all ants have completed their tour, evaluate the performance of different ants' 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) Pheromone Update

Pheromone updating on each node is done in two steps:

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

$$\tau_i(t+1) = (1-\rho)\tau_i(t) \quad (3)$$

Where,

ρ is a constant.

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

b) 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.

$$\tau_i(t+1) = \tau_i(t) + \Delta\tau_i \qquad (4)$$

Where,

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

$$\Delta\tau = \frac{ph}{s} \qquad (5)$$

Here,

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

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

Value of C is constant (preferably 0.1)

3) 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

4) 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.

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

D. Kotecha's algorithm

Kotecha's algorithm is combination of both of these algorithms and it works as per following:

a) During underload condition, the algorithm uses EDF algorithm and priority of jobs will be decided dynamically depending on its deadline.

b) During overloaded condition, it use ACO based algorithm, priority of jobs will be decided depending on pheromone value laid on each schedulable task and heuristic function.

Switching Criterion:

Initially the proposed algorithm uses EDF algorithm considering that the condition is not overloaded. But when two consecutive jobs miss the deadline, it will be identified, as overloaded condition and the algorithm will switch to ACO based algorithm. After 10 jobs have continuously achieved the deadline, again the algorithm will shift to EDF algorithm considering that overload condition that overloaded condition had been disappeared.

During underload condition, EDF algorithm is used for reducing execution time and during overloaded condition ACO based scheduling algorithm is used for achieving better performance. By this way, adaptive algorithm has taken advantage of both algorithms and overcome their limitations.

IV. PROPOSE ALGORITHM

In Kotecha's algorithm, the switching criterion is depend on the result of executed jobs, in this theory during underload condition, EDF algorithm is used for reducing execution time and during overloaded condition ACO based scheduling algorithm is used for achieving better performance. However, the switching criterion is not clear and it is difficult to identify system's condition. Moreover, when a task-set has a large numbers of tasks, the switching criterion will occur frequent switching bring unnecessary overhead of computation.

Therefore, we need more general switching criterion. We purpose two switching criterion depend on task schedulability analysis: Utilization Analysis and Response Time Analysis.

A. *Utilization Base Analysis*

We assume a task set τ of n periodic tasks to be scheduled on *m* identical processors. Each task τ_i =(C_i, D_i) is characterized by a worst-case computation time C_i, a relative deadline D_i. We will assume every task having constrained deadline, ie. Every deadline is equal to the corresponding period. The utilization of a task is defined as $U_i = \frac{C_i}{D_i}$, Let U_{max} be the largest utilization among all tasks. J. Goossens, S. Funk, and S. Baruah [11] examined that a system of independent periodic tasks can be scheduled successfully on m processors by EDF scheduling if it satisfies the formula as follows:

$$\sum_{\tau_i \in \tau} U_i \leq m(1 - U_{max}) + U_{max} \qquad (7)$$

We purpose a new switching criterion depend on (7), when scheduler star up we use (7), if it satisfied considering that the condition is not overloaded, the algorithm use EDF algorithm. Otherwise, it will be identified as overloaded condition and the algorithm will switch to ACO based algorithm. The scheduler will star-up when a task arrival or completed. By this way before a new task arrival, the total utilization is relative fixed. Therefore, algorithm will not frequent switching.

B. *Response Base Time Analysis*

Response Time Analysis (RTA) is an effective technique that has been widely used to derive schedulability tests and properties for various different models of task systems. M. Bertogna and M. Cirinei [12] examined that An upper bound on the response time of a task τ_k in an EDF-scheduled multiprocessor system can be derived by the fixed point iteration on the value R_k of the following expression, starting with $R_k = C_k$:

$$R_k \leftarrow C_k + \left\lfloor \frac{1}{m} \sum_{i \neq k} I_k^i(R_k) \right\rfloor \qquad (8)$$

Where,

I_k^i is interference of task τ_i on task τ_k

R_k is response time of task τ_k

C_k is Worst-case computation time of task τ_k

M. Bertogna and M. Cirinei [12] also examined A task set τ is schedulable with EDF on a system with m identical processors if, for all tasks are satisfies the formula as follows

$$D_k - R_k \geq 0 \qquad (9)$$

We purpose a new switching criterion depend on (9), when scheduler star up we use (9), if it satisfied considering that the condition is not overloaded, the algorithm use EDF algorithm. Otherwise, it will be identified as overloaded condition and the algorithm will switch to ACO based algorithm. The scheduler will star-up when a task arrival or completed. By this way before a new task arrival, the value of each task's response time is relative fixed. Therefore, algorithm will not frequent switching.

V. SIMULATION AND RESULT

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 7 different values of load are taken ($0.8 \leq$ load ≤ 2.0) and tested on more than 35,000 tasks.

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 criterion:

A. 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 [5],

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

B. In real-time systems, new task will occur any time. We count the "context switch" (i.e., the time need to save the status of the task being preempted and to load the preempting task) as overhead of each algorithm.

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

Fig. 2 and fig. 3 shown the results obtained in terms of %SR and context switch by each algorithm when number of processors are 2.The results are taken from underload condition(load ≥ 0.8) to highly overloaded condition(load $\leq$ 2). Fig. 2 shows the results of success ratio achieved by the two proposed algorithms 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 3% and8% when load values are 1.0 and 1.8. Fig. 3 shows the results of context switch achieved by the two proposed algorithm and Kotecha's algorithm. It observed that switching criterion depends on utilization analysis algorithm is occurred context switch fewer than Kotecha's algorithm during overloaded condition. Moreover, we find that switching criterion depend on response time analysis algorithm is occurred context switch fewer than Kotecha's algorithm when load from 1.0 to 1.6. The results remain consistent when we increase the number of the processors. Fig. 4 to fig. 7 demonstrates the same when numbers of processors are 4 and 8.

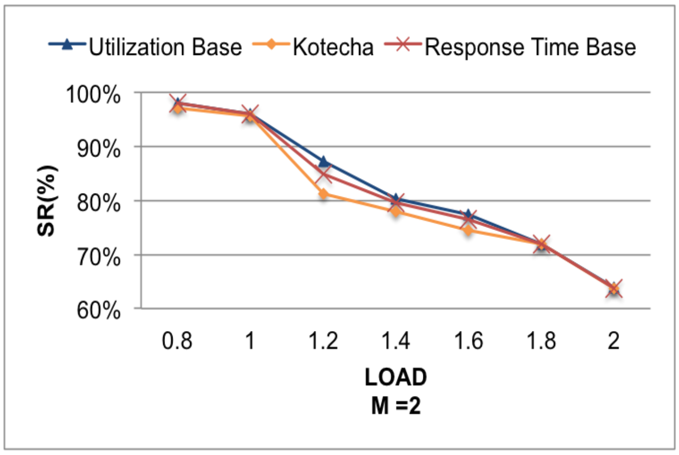

Fig. 2. Success ratio of jobs, when Number of Processors =2

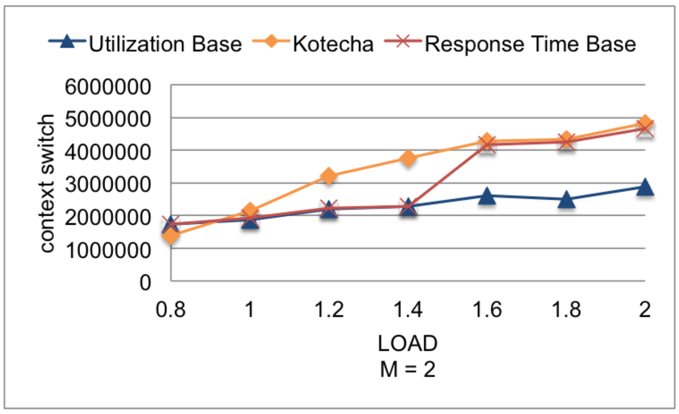

Fig. 3. Context Switch, when Number of Processors =2

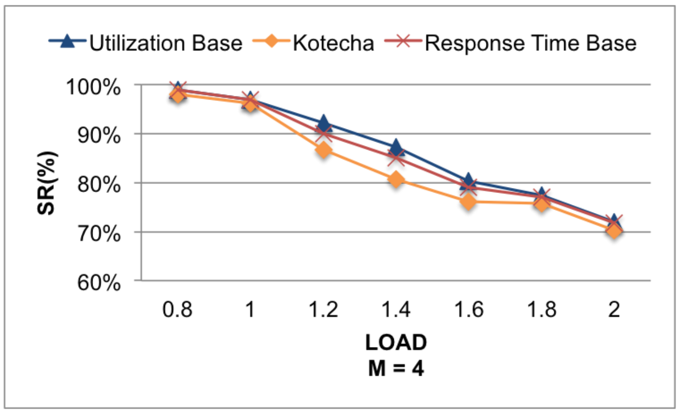

Fig. 4. Success ratio of jobs, when Number of Processors =4

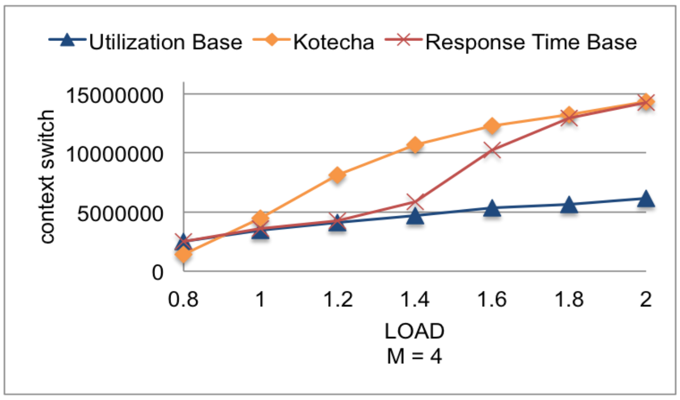

Fig. 5. Context Switch, when Number of Processors = 4

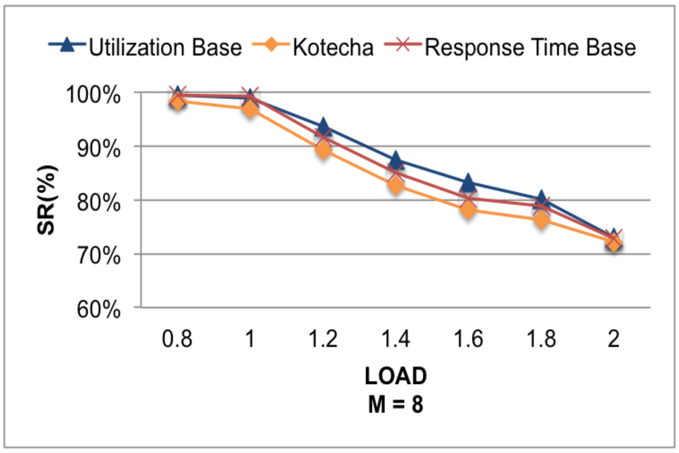

Fig. 6. Success ratio of jobs, when Number of Processors =8

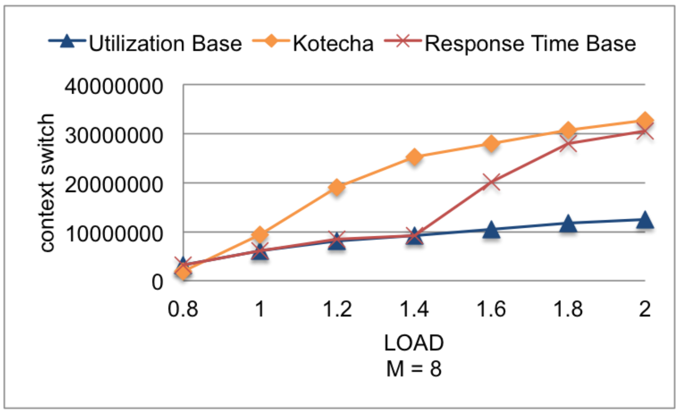

Fig. 7. Context Switch, when Number of Processors =8

VI. CONCLUSIONS

The algorithm discussed in this paper is for scheduling of soft real-time system with multiprocessor environment and preemptive task sets. The algorithm is simulated with periodic task sets; results are obtained and compared with Kotecha's algorithm.

The proposed algorithm works well in underload or overloaded condition. During underload condition, the success ratio and context switch of proposed algorithm is almost same as Kotecha's algorithm and during overloaded condition; it performance better than Kotecha's algorithm. Especially, the switching criterion depends on utilization analysis algorithm.

The algorithm can switch automatically between EDF algorithm and ACO based scheduling algorithm depend on utilization analysis or response time analysis. Therefore, the proposed algorithm is dynamic, during simulation only periodic tasks are considered but it can schedule aperiodic tasks also. The algorithm can work with available number of processors.

ACKNOWLEDGMENT

The authors would like to thank the developers of TOP-PERS/ATK1. This work supported in part by JSPSKAKENHI Grant Number 15K00084.

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, pp. 1-10, Santiago, 2002.

[8] K. Kotecha and A. Shah, "Scheduling Algorithm for Real-Time Opeating Systems using ACO," 2010 Intelligence and Communication Networks, pp. 617–621, Nov 2010.
[9] G. Koren and D. Shasha, "Dover: An optimal on-line scheduling algorithm for overloaded real-time systems," SIAM Journal of Computing, vol. 24, no. 2, pp. 318-339 April 1995.
[10] J. Goossens, S. Funk and S. Baruah, "Priority-Driven Scheduling of Periodic Task Systems on Multiprocessors," Real Time Systems, vol. 25, nos. 2/3, pp. 187-205, 2003.
[11] M. Bertogna and M. Cirinei, "Response-time analysis for globally scheduled symmetric multiprocessor platforms," 28th IEEE International Real-Time Systems Symposium, pp.149-160, 2007.
[12] A. Shah, K. Kotecha and D. Shah, "Adaptive scheduling algorithm for real-time distributed systems," in Biologically-Inspired Techniques for Knowledge Discovery and Data Mining, pp. 236-248, IGI global, 2014.
[13] A. Colorni, M. Dorigo and V. Maniezzo, "Distributed optimization by ant colonies," Proceedings of European Conf. on Artificial Life. Elsevier, Amsterdam, pp. 134-142, 1991.
[14] 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, no. 2, pp. 184-194, April 1990.