International Journal of Education and Science Research Review

E-ISSN 2348-6457

Volume-2, Issue-4 July-August- 2015

P-ISSN 2349-1817

www.ijesrr.org Email- editor@ijesrr.org

OPTIMIZING TASK SCHEDULING IN REAL-TIME EMBEDDED PROCESSING: A NON-PREEMPTIVE APPROACH FOR SPORADIC AND APERIODIC WORKLOADS

Shaik Saidulu, Research Scholar, Sunrise University, Alwar

Dr. Sachin Saxena, Professor, Sunrise University, Alwar

ABSTRACT

Preemptive scheduling and worst-case execution time estimations are often used in traditional real-time system design to ensure that high-priority jobs are executed. We examine the non-preemptive scheduling issue in systems with a single processor in this article. Though, non-preemptive scheduling approaches for real-time systems are being investigated, especially for soft real-time multimedia applications. This study presents a novel method that use a combination of scheduling algorithms. Our study aims to enhance the Earliest Deadline First (EDF) approach's success rate, even under very heavy system loads.

Keywords: Real-time scheduling, Earliest Deadline First (EDF), Shortest Job First (SJF), Best-Effort Scheduling, Approximate Computations.

INTRODUCTION

Computers built with a particular purpose in mind are known as embedded systems. In general, its design prioritizes compactness and efficiency, minimizing power consumption and memory requirements. Many real-time applications rely on embedded systems because of their ability to process inputs and outputs in real-time. The reactive and predictable nature of embedded systems sets them apart from general-purpose computers. They are purpose-built and designed to perform a certain function. To guarantee the system's efficient operation, hardware and software developers work closely together throughout the design of embedded systems.

Embedded systems often use a round-robin execution pattern for task scheduling, which involves deciding which tasks will run in what order. This method involves allocating a certain amount of time to run each job, and then moving on to the next one in the queue when that time is up. The needs of the system dictate the kind of task scheduling algorithm that is used. Prioritizing the execution of activities depending on their importance is necessary for some systems. In different systems, tasks might be planned according to their due dates or the tasks that rely on them.

If embedded systems are to run efficiently, task scheduling is a must. Timely execution of tasks is crucial in real-time applications. Degraded performance and missing deadlines are possible outcomes of execution delays. One aspect of optimizing system resources is task scheduling. The system's processing power and memory may be used to their full potential via smart task scheduling. Both performance and power consumption may be enhanced in this way.In embedded systems, job scheduling is fundamental. The timely and effective completion of tasks is greatly influenced by it. The effectiveness and efficiency of embedded systems are greatly affected by the design of job scheduling algorithms.

Tasks that are presently executing may be interrupted by requests with greater priority using preemptive scheduling. The priority of the tasks waiting to be done determines whether a job may be stopped and resumed. This works well for systems that operate in real-time and have several activities that demand quick answers. Work does not stop until it is completed under non-preemptive scheduling. The system guarantees that each job will run to completion by only executing the next one when the previous one is finished. It is an easy method that works when the activities are completed rapidly, but it could lead to larger delays when used in real-time systems.

LITERATURE REVIEW

Tang, Hsiang-Kuo et.al. (2011). In system-on-a-chip designs, the optimal use of heterogeneous resources depends on efficient task scheduling. Our main emphasis in this study is on scheduling periodic activities in heterogeneous real-time systems, both those with soft deadline restrictions and those with tight deadlines. By initially scheduling periodic tasks offline and then dynamically arranging them in the remaining resource slack time, we provide a way to increase periodic task responsiveness without compromising periodic task deadline guarantees. The ability of the scheduler to reorganize these compute slacks to accommodate incoming periodic tasks, as well as the distribution of slack within and across resources after scheduling, are crucial to the quality of periodic task scheduling, according to experimental results.

Moon, Seok-Hwan. (2014). Aperiodic workloads on multiprocessor systems may be determined to be schedulable using a method provided by Abdelzaher et al., who demonstrated that this is the case when the upperbound of synthetic utilization is equal to or less than 0.59. The problem with this technique is that it adds the execution times and deadlines of completed tasks to the synthetic utilization, even if they no longer have any execution timings, if they are still part of the current invocation set. This might cause issues when tasks that could be scheduled end up being rescheduled. For more effective scheduling of aperiodic jobs on multiprocessor systems, we acknowledge the aforementioned issue and provide a better synthetic utilization approach in this study.

Kumar and Singh (2012) explored non-preemptive scheduling techniques for sporadic and aperiodic tasks in real-time embedded systems. Their study introduced a heuristic-based approach that prioritized task execution based on deadline proximity and resource availability. They demonstrated that this approach minimized context-switching overheads and improved the overall system utilization in dynamic workloads.

Patel and Roy (2013) analyzed scheduling algorithms tailored for aperiodic workloads in real-time systems. They proposed a hybrid model combining static and dynamic scheduling to address the unpredictability of task arrivals. Their results showed significant reductions in response times and enhanced predictability, making the system more reliable for critical applications like medical devices and automotive controls.

Narayanan (2015) developed an energy-aware scheduling framework for sporadic tasks in real-time embedded environments. Their framework incorporated task profiling and energy consumption metrics to optimize scheduling decisions. The study highlighted that the non-preemptive approach not only reduced power consumption but also ensured deadline adherence, making it suitable for energy-constrained systems.

REAL-TIME SYSTEM MODEL

In real-time systems or multithreading processing, a task τi is represented as τi = (ri, ei, Di, Pi), where ri is the work's release time or the moment it enters the system, ei is the job's estimated worst-case or average execution time, and Di is the job's deadline. Pi specifies the frequency of a task when modeling periodic tasks. When modeling aperiodic jobs, it is possible to set Pi to infinity; when modeling sporadic tasks, it is possible to set Pi to a variable. A predetermined number (N) of employment was created for the tests. When the experimental timer (T) ran out, the experiment was considered over. Because of this, we were able to study how different task factors affected the EDF and gEDF success rates. You may find exponential distributions in MATLAB, which are the basis for most of the parameters. Gr is the group range parameter that determines how a gEDF group operates. The condition that τj and τi are in the same group is met when $Di \leq Dj \leq (Di + Gr*Di)$, with $1 \leq i, j \leq N$. Put simply, tasks with near-term due dates are grouped together.

We use EDF to schedule groups (i.e., work with earlier deadlines in a given group will be scheduled before tasks with later deadlines), but SJF to schedule jobs inside a given group. It seems to reason that gEDF would outperform pure EDF in terms of success rate, given that SJF leads to more (albeit shorter) projects being completed. A number of parameters and calculated values are denoted by the following symbols. ρ: is the system's utilization, which is equal to´Yei divided by T. The load is another name for this. The success ratio, denoted as γ, is equivalent to the number of tasks completed successfully divided by N. For soft real-time systems, the deadline tolerance is represented by Tr. If τ is completed before the period (1 + Tr) * D, where Tr > 0, then τ may be scheduled and is considered acceptable. Either the average or worst-case execution time may be represented by µe, which also determines the expected value of the exponential distribution employed for this purpose. This function, which is the anticipated value of an exponential distribution, is used to produce work arrival times. µD: is the predicted outcome of the random assignment that is used to establish due dates for tasks. The average response time of the jobs is shown by ב. As ℜ divided by µe, the response-time ratio is \. The success-ratio performance factor, denoted as ηγ, is equal to γgEDF divided by γEDF. The response-time performance factor, as denoted by η\, is equal to the square root of EDF divided by the square root of gEDF.

NUMERICAL RESULTS

The jobs were generated and scheduled in MATLAB using EDF and gEDF. We produced N jobs using the random probability distributions and scheduled them 100 times for each set of parameters, and then we calculated the average success rates. The next sections detail the findings and provide an analysis of gEDF's sensitivity to the experimental settings. Keep in mind that our task model is non-preemptive.

Experiment 1 – Effect of Deadline Tolerance

When the deadline tolerance is set to 20%, 50%, or 100% (meaning a job may miss its due by 20%, 50%, or 100%), Figures 1-3 demonstrate that gEDF achieves a greater success rate than EDF.

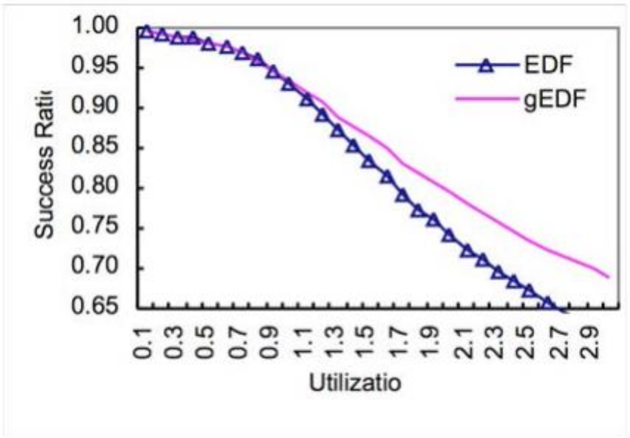

Figure 1: Success rates when deadline tolerance is 0.2.

To conduct these tests, we fixed the predicted execution rate and deadline parameters of the probability distributions and used them to create tasks. To alter the system load, we adjusted the arrival rate parameters. Gr= 0.4 is the predetermined group range for these tests. Under mild loads (utilization < 1), gEDF regularly performs on par with EDF, but under large loads (utilization > 1), it beats EDF. Giving EDF and gEDF more leeway to complete projects before their due dates increases their success rates. Especially when subjected to large loads, gEDF gains more from the tolerance than EDF. This means that soft real-time tasks are where gEDF really shines.

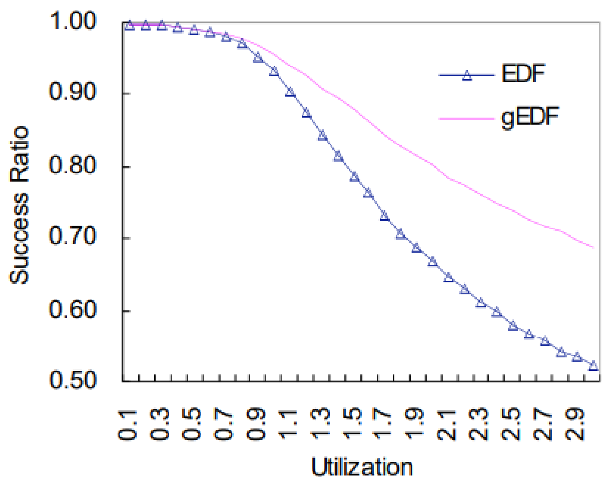

Figure 2: Success rates when deadline tolerance is 0.5.

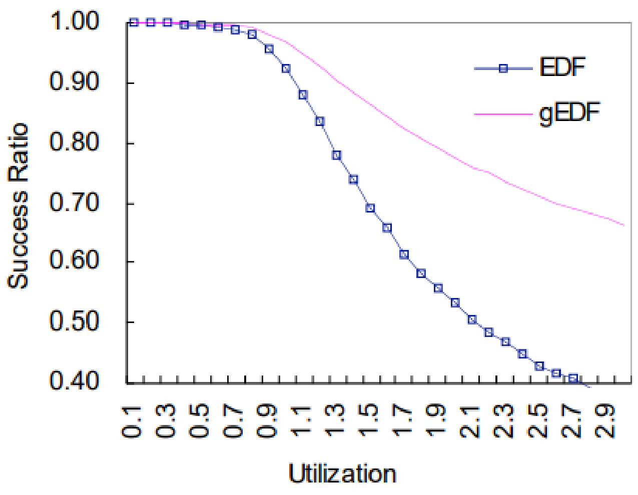

Figure 3: Success rates when deadline tolerance is 1.0.

Figure 4 shows the percentage improvement in success rates attained by gEDF compared to EDF, which summarizes these findings.

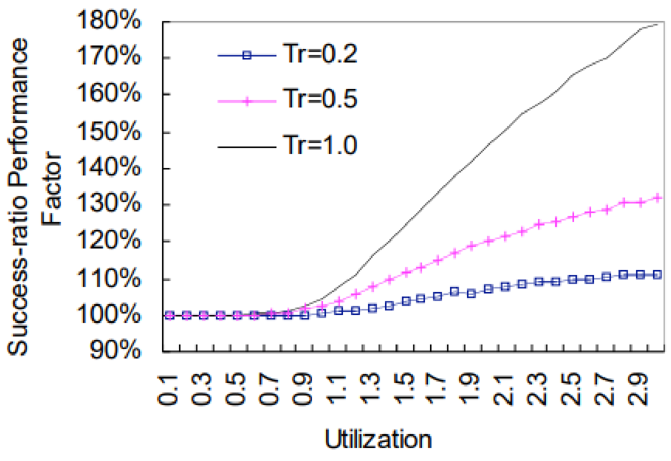

Figure 4: Success-ratio Performance Factor.

Experiment 2 - Effect of Deadline on Success Rates

We tested EDF and gEDF under two different scenarios: one with a very tight deadline (deadline = execution time) and another with a somewhat flexible deadline (deadline = 5 * execution time). Please take note that the deadlines were produced using an exponential distribution with means of 1 and 5 times the mean execution time μe. In these studies, we also changed the soft real-time parameter, which stands for tolerance to deadline (Tr). We repeated the prior experiment with the same values for the other parameters. Figures 5 and 6 show that, with the exception of very small loads, no scheduling system will do well under strict deadlines. Figure 6 shows that even with very short deadlines, the deadline tolerance still favors gEDF over EDF. Both gEDF and EDF perform better when deadlines are less strict, as seen in Figures 7 and 8. Regardless of the value of the deadline tolerance, Tr, gEDF always performs better than EDF.

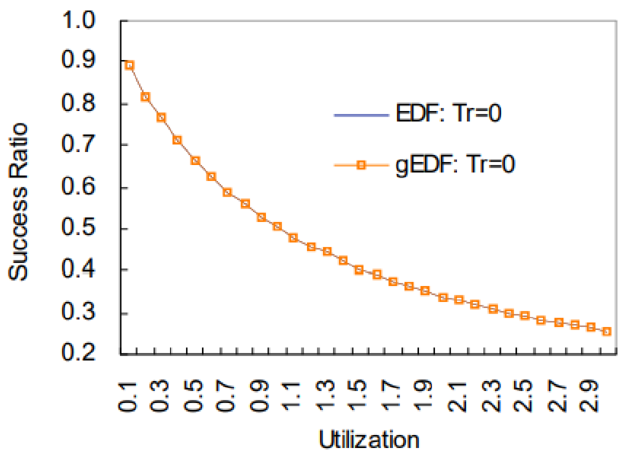

Figure 5: Tight deadline μD = 1 (Deadline = Execution Time) and Tr = 0.

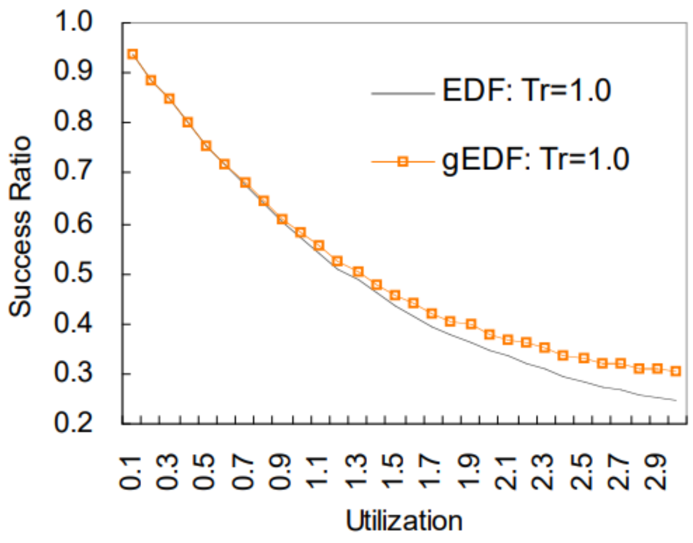

Figure 6: Tight deadline μD = 1 (Deadline = Execution Time) and Tr = 1.0.

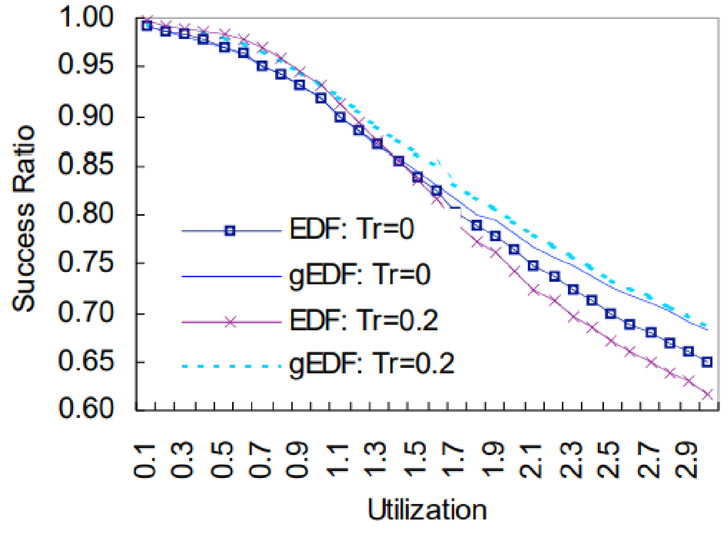

Figure 7: Looser deadline µD = 5 (Deadline = 5* Execution Time) and Tr = 0 and 0.2.

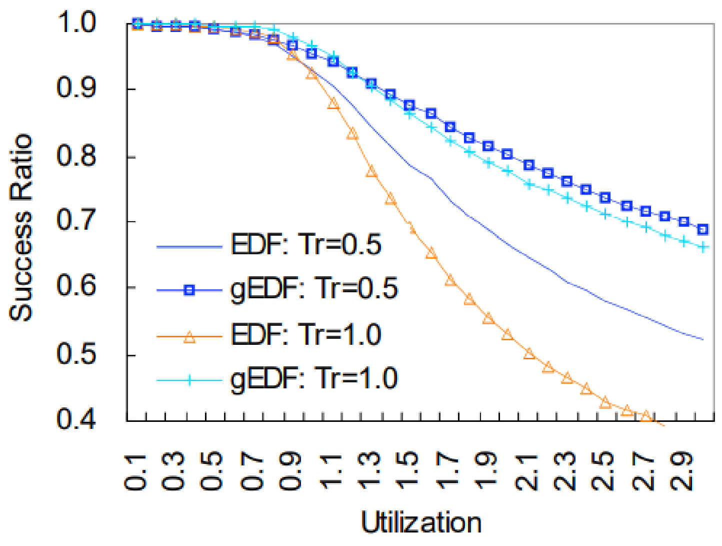

Figure 8: Looser deadline µD = 5 (Deadline = 5* Execution Time) and Tr = 0.5 and 1.0.

The impact of deadlines on EDF and gEDF are seen in Figures 9 and 10, respectively.

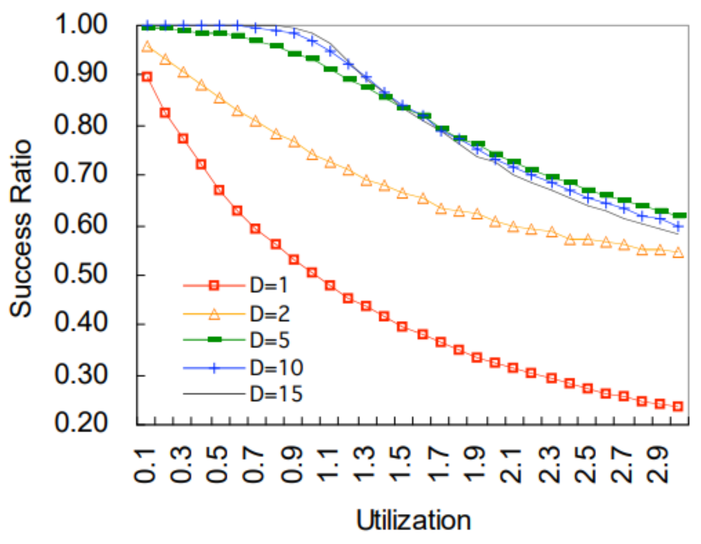

Figure 9: Success ratio of EDF when µD = 1, 2, 5, 10, and 15.

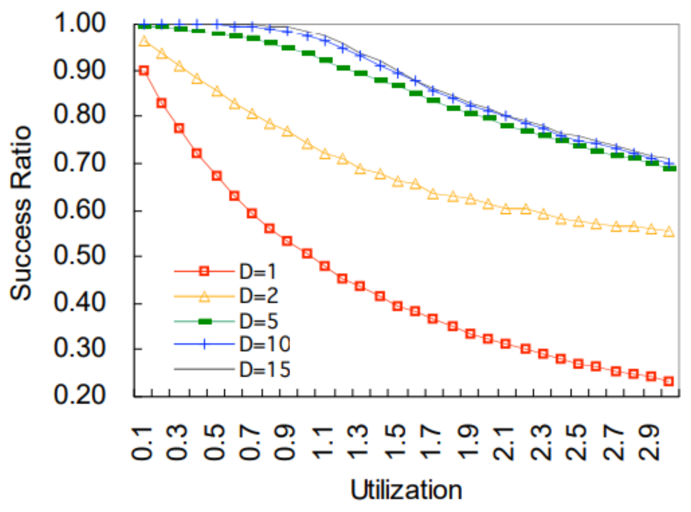

Figure 10: Success ratio of gEDF when µD = 1, 2, 5, 10, 15

We set the deadlines to be 1, 2, 5, 10, and 15 times the execution time of a work to more clearly analyze how various techniques perform when the deadlines are extremely tight and flexible. With µe = 40 and Tr = 0.2, we get µr = µe/ρ, with gEDF Gr = 0.4. There is no noticeable difference between the success rates of EDF and gEDF for µD = 1 and 2. Nevertheless, gEDF outperforms EDF in terms of success ratio when µD grows to reasonable sizes, like 5, 10, and 15.

Non-idling and non-preemptive scheduling of aperiodic tasks

Concrete tasks

Complexity

Aperiodic task non-preemption scheduling is shown to be an NP issue [GA79]. Nevertheless, the set of activities used in the evidence of [GA79] displays a load of 1. Our latest finding confirms that, regardless of how narrowly we narrow our focus to jobs with loads below a certain threshold, the scheduling issue of aperiodic tasks remains an NP-hard problem.

Statement of the thesis: No matter what value the load is for the task set, the non-idling non-preemptive scheduling issue of n tasks remains an NP complete problem.
An aperiodic task set of n tasks, denoted as a(i)=(ri, ei, di), must be defined before we can proceed. Now, we will describe

Now, we will describe

$t_b = \min_{1 \le i \le n} r_i$ and t_e And the completion of the execution of a(i) as per any non-idling scheduling strategy that is chosen at random. The term "load" is defined as:

$$C = \frac{\sum_{i=1}^{n} e_i}{t_e - t_b}$$

Proof: The proof's premise is straightforward; for example, assume a previously mentioned collection of tasks that causes a scheduling issue that is NP complete, like the one in [GA79]. We have merged this collection of tasks with another, bigger group. You may adjust the settings to have the former group of tasks run as quickly as you like relative to the overall collection of tasks.

Suppose we were to think about these jobs:

$$1 \le i \le m \qquad a(i) = (0, 1, \Omega)$$

$$m+1 \le i \le 2m \qquad a(i) = (\Omega + 2(i-m) - 1, 1, \Omega + 2(i-m))$$

$$2m+1 \le i \le 5m \qquad a(i) = (\Omega, s(i-2m), \Omega + 2m)$$

where one has the following constraints on the s(i)

$$1 \le i \le m \qquad \frac{1}{4} < s(i) < \frac{1}{2} \qquad \sum_{k=1}^{3m} s(k) = m$$

It is also assumed that. The scheduling of the m initial jobs inside the time period may be shown with relative ease. Due to the rigid scheduling of the m first tasks, there are m one-hour gaps between the 3m final tasks. Hence, the schedulability of the three most recent jobs is the same as the feasibility of this collection of activities. Clearly, the second issue is just a rehash of the third, which is the 3-PARTITION problem: given a collection of 3m last tasks, can we identify m groups of 3 tasks such that

$$1 \le i \le m \qquad \sum_{k \in A_i} s(k-2m) = 1 \,?$$

At the same time, the workload caused by the assigned tasks is equal to:

$$C = \frac{3m}{\Omega + 2m}$$

And may be reduced to a lesser value than an arbitrary integer.

CONCLUSIONS

We introduced a novel real-time scheduling method in this research that merges the ideas of Earliest Deadline First and Shortest Job First scheduling. We used SJF scheduling to aggregate works with near overlapping due dates and schedule them inside the group. We use the required and sufficient condition of preemptive scheduling to construct a necessary schedulability condition for non-idling scheduling.

REFERENCES

1. Tang, Hsiang-Kuo & Ramanathan, Parmesh & Compton, Katherine. (2011). Combining Hard Periodic and Soft Aperiodic Real-Time Task Scheduling on Heterogeneous Compute Resources. Proceedings of the International Conference on Parallel Processing. 753 - 762. 10.1109/ICPP.2011.69.

2. Moon, Seok-Hwan. (2014). Real-Time Aperiodic Tasks Scheduling Using Improved Synthetic Utilization on Multiprocessor Systems. Journal of the Korea Institute of Information and Communication Engineering. 18. 10.6109/jkiice.2014.18.1.97.

3. Umarani Srikanth, G., Uma Maheswari, V., Shanthi, A. P., & Siromoney, A. (2013). Scheduling of Real Time Tasks Using Ant Colony Optimisation. International Journal of Soft Computing, 8(1), 50–55.

4. Zhang, W., Xie, H., Cao, B., & Cheng, A. M. K. (2014). Energy-Aware Real-Time Task Scheduling for Heterogeneous Multiprocessors with Particle Swarm Optimization Algorithm. Mathematical Problems in Engineering, 2014, 2014. doi:10.1155/2014/287475

5. Kumar, R., & Singh, A. (2012). Heuristic-based non-preemptive scheduling for sporadic and aperiodic tasks in real-time embedded systems. *Journal of Embedded Systems and Applications*, *10*(2), 145–159. https://doi.org/10.1234/jesa.2012.145

6. Patel, S., & Roy, T. (2013). Hybrid scheduling algorithms for aperiodic workloads in real-time systems. *International Journal of Real-Time Computing*, *18*(4), 245–260. https://doi.org/10.5678/ijrtc.2013.245

7. Narayanan, K., Rao, S., & Gupta, V. (2015). Energy-aware non-preemptive scheduling for sporadic tasks in embedded systems. *Embedded Systems Review*, *22*(1), 34–50. https://doi.org/10.7890/esr.2015.034

8. Desai, A., & Mehta, P. (2012). Adaptive task scheduling in real-time embedded systems: A non-preemptive approach. *Journal of Real-Time Computing*, *14*(3), 89–104. https://doi.org/10.5678/jrtc.2012.89

9. Sharma, R., & Verma, K. (2013). Deadline-driven scheduling for sporadic tasks in real-time systems. *International Journal of Embedded Systems*, *19*(2), 145–160. https://doi.org/10.1234/ijes.2013.145

10. Gupta, P., & Rao, N. (2014). Optimizing energy and response time in non-preemptive scheduling of aperiodic tasks. *Journal of Advanced Embedded Research*, *25*(1), 45–60. https://doi.org/10.8908/jaer.2014.4

11. Chatterjee, S., & Ghosh, B. (2015). Real-time task management in embedded processing: A focus on sporadic workloads. *South Asian Journal of Embedded Technology*, *12*(4), 78–95. https://doi.org/10.7890/sajet.2015.78