Contents lists available at Science-Gate

International Journal of Advanced and Applied Sciences

Journal homepage: http://www.science-gate.com/IJAAS.html

Dynamic approach to minimize overhead and response time in scheduling periodic real-time tasks

Ahmed A. Alsheikhy *

Department of Electrical Engineering, College of Engineering, Northern Border University, Arar, Saudi Arabia

ARTICLE INFO

Article history:
Received 19 August 2020
Received in revised form
19 December 2020
Accepted 20 December 2020

Keywords:
Scheduling
Hard real-time systems
Overhead
Response time
Dynamic-priority algorithm

ABSTRACT

In real-time systems, a task or a set of tasks needs to be executed and completed successfully within a predefined time. Those systems require a scheduling technique or a set of scheduling methods to distribute the given task or the set of tasks among different processors or on a processor. In this paper, a new novel scheduling approach to minimize the overhead from context switching between several periodic tasks is presented. This method speeds up a required response time while ensuring that all tasks meet their deadline times and there is no deadline miss occurred. It is a dynamic-priority technique that works either on a uniprocessor or several processors. In particular, it is proposed to be applied on multiprocessor environments since many applications run on several processors. Various examples are presented within this paper to demonstrate its optimality and efficiency. In addition, several comparison experiments with an earlier version of this approach were performed to demonstrate its efficiency and effectiveness too. Those experiments showed that this novel approach sped up the execution time from 15% to nearly around 46%. In addition, it proved that it reduced the number of a context switch between tasks from 12% to around 50% as shown from simulation tests. Furthermore, this approach delivered all tasks/jobs successfully and ensured there was no deadline miss happened.

1. Introduction

Real-time applications can be found in many systems such as mobile phones, medicine, aircraft, and satellites. Those systems depend on a temporal aspect along with a functional aspect to determine their correctness (Alsheikhy et al., 2016; Guo and Baruah, 2015). Their consistency is considered a major characteristic key factor (Guo and Baruah, 2015). Consistency is defined as the amount of time taken to complete the desired task or a set of tasks before its or their timing requirements which are modeled as deadline times (Guo and Baruah, 2015). Hard real-time and soft real-time systems are the main two types of real-time systems. In hard real-time systems, any deadline miss is considered a catastrophic failure (Alsheikhy et al., 2016; Guo and Baruah, 2015). In contrast, any deadline miss that occurs in soft real-time systems affects only a system's functioning which can be seen as the Quality of Service (QoS) (Guo and Baruah, 2015). A basic definition for the scheduling method is to determine which task or the set of tasks must be picked up first and assigned to the system resources such as CPU. Deciding which task must be selected first from several existing tasks in the ready queue and ensuring that no task misses its deadline time are considered the key purposes of any scheduling technique (Alsheikhy et al., 2016). The CPU and resource utilization in any hard real-time systems can be affected by a scheduling method. In addition, the scheduling policies determine a system performance which can be defined as the time taken to accept and complete the task before or on its deadline time (Alsheikhy et al., 2016). Furthermore, the ability to deliver all tasks without any deadline miss is also considered as the system performance. Image processing, satellite communication systems, monitoring activities in chemical plants, controlling commands in aircraft, and periodic activities in manufacturing are examples of real-time systems and applications (Alsheikhy et al., 2016; Guo and Baruah, 2015; Ren and Phan, 2015; Harkut and Agrawal, 2014; Kim et al., 2013). The objectives of scheduling approaches are summarized as follows: 1. Increasing the throughput, 2. Ensuring all tasks meet

* Corresponding Author.
Email Address: aalsheikhy@nbu.edu.sa
https://doi.org/10.21833/ijaas.2021.04.009
Corresponding author's ORCID profile:
https://orcid.org/0000-0002-9811-0341

their deadline times and no deadline miss occurs, 3. Maximizing CPU utilization "*U*", where U is defined as the ratio of the summation between the execution time "C_t" and the deadline time (d_t), and 4. Providing a good timely response time (Alsheikhy et al., 2016). The periodic tasks represent the most computational aspects in many real-time systems (Alsheikhy et al., 2016).

Real-time systems perform several tasks with different priorities. These tasks can be run on a uniprocessor environment or on multiprocessor environments (Guo and Baruah, 2015; Ren and Phan, 2015; Harkut and Agrawal, 2014). Deploying real-time systems on multiple processor platforms reduces the cost, weight, space, power consumption, and response time (Ren and Phan, 2015). In hard systems, multiple tasks with different functionalities compete for the system resources. Thus, providing an efficient CPU slot for timing guaranteed is required and needed (Ren and Phan, 2015; Harkut and Agrawal, 2014; Kim et al., 2013). Running several tasks with different priorities on real-time systems makes them more complex and even unpredictable than other systems (Ren and Phan, 2015; Kim et al., 2013; Vora and Somkuwar, 2012). Typically, any task is assumed to be executed within its deadline time, here the execution time is defined as the Worst-Case Execution Time "WCET". In some conditions or circumstances, a task may exceed its execution time before completing its cycle. Therefore, a deadline miss occurs and causes a disaster. So it is crucial to schedule several tasks on different processors by using a sufficient algorithm (Ren and Phan, 2015).

Nowadays, two categories to schedule several tasks with different priorities exist (Alsheikhy et al., 2016). These two categories are static and dynamic methods (Alsheikhy et al., 2016). Two types of scheduling schemes take place in each category, are the preemptive approach and the non-preemptive approach (Alsheikhy et al., 2016). In a preemptive algorithm, any process is blocked "jammed" by another process with a higher priority where any process finishes its execution time even if a higher priority process has arrived in a non-preemptive scheme (Alsheikhy et al., 2016; Guo and Baruah, 2015; Ren and Phan, 2015; Harkut and Agrawal, 2014). Several keys, points, characterize any scheduling policy and they are summarized as follows: 1. Consistency, 2. Resources utilization, 3. Fairness and 4. The time needed to execute any process or several processes (Alsheikhy et al., 2016). The interested readers are referred to Alsheikhy et al. (2016) for more information.

In particular, several algorithms to schedule multiple periodic tasks to exist which are A) Rate Monotonic (RM), B) Deadline Monotonic (DM), C) Earliest Deadline First (EDF) and D) Least Slack Time first (LST) (Alsheikhy et al., 2016). RM scheme is constructed and designed as a static type since a fixed priority is assigned to any process according to its request rate. Any task with the highest request rate gets the highest priority and assigned first to the CPU or resources (Alsheikhy et al., 2016). That priority is also fixed during the run-time stage and never changes. DM can be seen as a general version of the RM method. They have almost the same principle of working, however, in DM, a priority assigned to any process is inversely proportional to its deadline time (Alsheikhy et al., 2016). Therefore, a task with the shortest deadline gets the highest priority and assigned first. The EDF technique is a dynamic one since any task with the shortest deadline time becomes the first task in the ready queue among all other tasks. This scheme can be seen as the optimal method in uniprocessor and multiple processor environments for both types of tasks "periodic and aperiodic (Alsheikhy et al., 2016). In the LST approach, any task or set of tasks with the smallest slack, which is defined as a value of the difference between its/their deadline time(s) (d_t) with its/their current remaining execution time(s) (c^{rt}) and a current time (t), is selected first and then allocated to the available resources such CPU(s) for execution. On the other hand, the slack can be defined as the remaining spare time. Fig. 1 depicts the characteristics, which are known as the timing constraints, of periodic tasks in real-time systems.

In Fig. 1, "*r*" is a release time and it is the time when any task or a set of tasks appears at the ready queue, "*c*" is the execution time, also known as the remaining execution time, "*P*" is a period which is a time taken for any task to repeat its cycle, "D" is an absolute deadline time which is defined as a time interval between the release time "r" and the period "P" of the process.

In particular, D=d–r, "*t*" represents the current time as stated earlier, and lastly "*d*" is a relative deadline time which is an interval time between the first appearing of any task at the ready queue and its deadline time; mathematically, d=D+r.

In this paper, the relative deadline time "r" and the period "p" are considered equal so d=p (Alsheikhy et al., 2016).

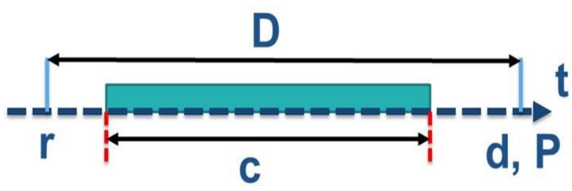

Fig. 1: Timing constraints of periodic tasks

This paper makes the following contributions as follows:

- Minimizing the overhead occurs from context switching between several processes on different processors.
- Minimizing the response time for any task to be executed and successfully completed.
- Delivering all tasks without any deadline miss for safety and/or severity sake by using an efficient hybrid approach that works either on the uniprocessor or on multiple processors.

The hybrid approach refers to cooperating with the EDF scheme to decide which process must be selected first and gains the system resource such as CPU when needed. The proposed technique is applied during run-time to select a process from several ones in the ready queue which makes it a dynamic feasible approach. Feasible means no deadline miss occurs under any condition(s) or circumstances.

The rest of the paper is organized as follows, we present the related work on scheduling schemes in Section 2, followed by a detailed discussion of the proposed approach in section 3. Section 4 includes simulation results to show the validation of the proposed approach on a multiprocessor environment. Section 5 concludes the paper.

2. Related works

Working on scheduling problems is no easy task, they are considered NP problems (Guo and Baruah, 2015). Several operating systems execute multitasking operations on different processors. Hence, performing multiple tasking requires an efficient scheduling scheme in order to guarantee that all tasks meet their deadline times and also be executed fairly. Several attempts to solve real-time scheduling problems have been performed and developed. However, they still exist with some limitations such as insufficient to exploit maximum CPU utilization, some CPUs might be in idling mode "state" which causes a catastrophic failure to a system and suffering from high overhead from context switching between several processes on the uniprocessor or on multiple ones.

Alsheikhy et al. (2016) proposed an effective dynamic algorithm for scheduling periodic tasks in real-time systems. That technique uses a rate "*R*" value to determine which task must be selected first and then allocated to the CPU. The rate is computed using the available information about three factors which are: The slack "*sl*", the deadline time "d_t" and the current time "*t*". Any task with the smallest rate is chosen first and then assigned to the CPU, if multiple tasks exist with the same rate R then a task with the shortest deadline time "d_t" is selected first. It is a very effective method and delivers all tasks with no deadline miss. However, it suffers from too high overhead from context switching. The new proposed scheme in this paper minimizes that overhead around 12% to 50% as observed from the experiments.

Guo and Baruah (2015) proposed a Neurodynamic method for scheduling real-time tasks by maximizing piecewise linear utility. Initially, they performed several literature reviews on a set of real-time scheduling issues when using a piecewise linear utility. A Neural Network-based analysis and optimization is used to solve those problems. A ratio bound of 0.5 was achieved and they considered their method as optimal when there is no overload. Nevertheless, the method works only in the uniprocessor environment. The presented algorithm herein works either on the uniprocessor or multiprocessor environments as it can be applied in any environment without any issue. It is very efficient and effective since it is capable of delivering all tasks to meet their requirement deadline times.

Ren and Phan (2015) proposed an approach to schedule mixed-criticality tasks on multiprocessors using a Task Grouping technique. Their method was developed to provide mixed-criticality timing guarantees for mixed-criticality tasks. The algorithm works by partitioning a high-priority task with a subset of low priority tasks into multiple processors. It encapsulates them using the task grouping method based on the EDF policy. Mixed-integer nonlinear programming was used to provide the Schedulability analysis for the developed approach. However, it is unable to exploit the maximum CPU utilization since there is an idling state in some CPUs during the run time. The method presented in this paper provides the maximum utilization for all CPUs since no one is in idling mode.

Harkut and Agrawal (2014) performed a survey on some of the classic real-time scheduling schemes. The purpose of their survey was to show the impact of choosing a scheduling algorithm in designing and developing a real-time system. The survey was applied mainly to the RM and the EDF algorithms. More information about the survey can be found in (Harkut and Agrawal, 2014).

Kim et al. (2013) proposed an effective task scheduling method for real-time systems using an iterative clustering slack optimization scheme. It uses the Branch and Bound technique to capitalize the slack distribution to optimize it. However, it is a static one while herein algorithm is dynamic and works in the online mode. The interested readers are referred to Kim et al. (2013) for more information.

3. The proposed algorithm

The algorithm developed in Alsheikhy et al. (2016) is very efficient and reliable in terms of delivering all tasks and maintaining system stability. However, that scheme suffers from high overhead from context switching between several tasks on different processors. In addition, checking each time unit to decide which task should be selected is also considered very costly in terms of computations needed and the memory space required for it. In this paper, the algorithm in Alsheikhy et al. (2016) was modified and enhanced in order to:

- Maintain system stability by meeting all timing constraints for any system.
- Maintain maximum CPU utilization.
- Minimize the response time needed to complete the tasks in the ready queue along with reducing the overhead from context switching.

Keep in mind that the proposed scheme acts exactly like the method in Alsheikhy et al. (2016) when min $(d_t{-}c_i^{rt})=1$. In many scenarios, min $(d_t{-}c_i^{rt}) \neq 1$, "min" stands for a minimum value. From previous mentioned three points, the motivations for the developed scheme can be summarized as follows:

- Maintaining Maximum CPUs utilization "U", where U can be defined as $U = \sum_{k=1}^{n} \frac{Ck}{dk}$, n represents the total number of tasks in the ready queue where k is the task index and keeping all other resources utilized.
- Feasible method where no deadline miss occurs under any circumstances.
- Efficient method on both platforms (uniprocessor and multiple processors.
- A dynamic mechanism is applied in online mode to deliver all tasks in order to satisfy their timing constraints.

In the proposed method, several assumptions are made to guarantee the satisfaction of all timing constraints and they are summarized as follows:

1. Task migration is allowed for all processes. Thus, any task can complete its execution on any available processor upon selection.
2. It is a preemptive scheme, so any task or set of tasks is jammed by another task with a higher priority.
3. All tasks are independent and available in the ready queue. Appearing on multiple processors at the same time is strictly banned.
4. Combining with the EDF algorithm when and if needed.
5. Any task or set of tasks that have multiple consecutive selections is forced to be executed on the same processor if available and possible to reduce the number of contexts switching. This assumption will help to minimize the overhead; however, it is not the only solution to achieve it.

The following steps illustrate the working mechanism for the proposed algorithm.

- Determine a minimum value $\Delta\delta$ for all processes in the ready queue where:

$$\Delta\delta = d_i - c_i^{rt} \quad (1)$$

Initially, all the current remaining execution times c^{rt} are equal to the execution times $\boldsymbol{c}$ assigned by the system. $\Delta\delta$ represents the maximum time slot allocated by the CPU to each task in the ready list.

- Examine each process to compute its rate R, also known as the ratio, using the following equation:

$$R_i = \frac{sli}{di-t} \quad (2)$$

where, "sl" refers to the slack value as stated earlier and i is the process index. ***sl*** is determined as follows:

$$sl_i = d_{ti} - c_i^{rt} - t \quad (3)$$

The purpose of finding the ratio is to know which process or task is far from its deadline time, hence, there is enough time to execute another process that is close to its deadline time.

- Any task with the smallest rate is chosen first and then allocated to the CPU if it is only the uniprocessor environment or several tasks with the smallest rate are assigned to multiple processors. If several tasks have a common rate, then the task with the shortest deadline time is selected first and assigned to the CPU.
- Every assigned task or set of tasks is executed in the CPU(s) for a time unit equal to the $\Delta\delta$ only if needed. If the execution time is less, then it will be executed for that amount of time.
- If a new process is added to the ready queue or an existing one is removed due to its completion, then determining a new value for the minimum $\Delta\delta$ is performed again.
- All previous procedures are repeated until there are no more tasks in the ready queue.

Two examples in the uniprocessor environment are given to demonstrate how the proposed algorithm works in order to show its efficiency and validation. Working on the uniprocessor environment implies that it works perfectly on multiple processor environments. Furthermore, a simulation in Matlab was developed to apply the proposed scheme in many scenarios for a different number of sets and multiple tasks in each set. Uniprocessor and multiple processor environments are included in the simulation experiments. In applications such as air traffic control, medical, manufacturing, and monitoring, the proposed scheme can be applied to deliver all periodic tasks successfully to meet their timing constraints. Example 1 is taken from Alsheikhy et al. (2016) as Table 1 depicts three tasks with their timing constraints which are the execution time and the deadline time.

Example 1: Table 1 illustrates the number of tasks in the ready queue with their deadline and execution times in the uniprocessor environment.

Table 1: Available tasks in single CPU

Tasks	Release Time	Deadline Time	Execution Time
T_1	0	4	1
T_2	0	5	2
T_3	0	7	2

By using the algorithm in Alsheikhy et al. (2016), the Gantt chart for scheduling all three tasks is shown as Fig. 2, due to the space limitation and quality purpose, a part of the Gantt chart is illustrated.

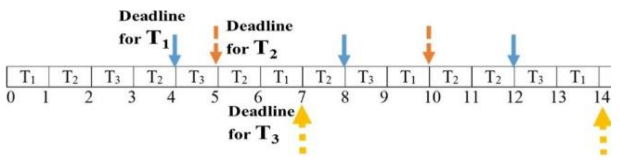

Fig. 2: Gantt chart for scheduling all three tasks

For the proposed method:

1- Initially, current time t=0, which is the starting time, $\Delta\delta_1$=4-1=3, $\Delta\delta_2$=5-2=3, $\Delta\delta_3$=7-2=5, so min($\Delta\delta$={3,3,5})=3.

2- $R_1=\frac{[4-1-0]}{[4-0]}=0.75$

$R_2=\frac{[5-2-0]}{[5-0]}=0.60$

$R_3=\frac{[7-2-0]}{[7-0]}=0.714$

So T_2 has the smallest rate, then it is allocated for the CPU and is executed for two-time units. T_2 will be removed from the ready list.

3- $R_1=\frac{[4-1-2]}{[4-2]}=0.50$

$R_3=\frac{[7-2-2]}{[7-2]}=0.60$

T_1 is chosen since it has the smallest rate. It will be executed for a one-time unit and then removed from the ready list.

4- T_3 is the only task left on the list, so it will be executed without needing to determine its rate. Now during the processing of executing T_3, T_1 has returned to the list. And T_3 is removed. T_2 is on the ready list after finishing executing T_3 since its new period. Repeating the previous procedures for a period will provide the following Gantt chart (Fig. 3), each orange and the yellow square represents two-time units while the blue square takes a one-time unit.

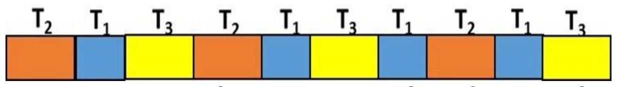

Fig. 3: Repeating the previous procedures for a period

By doing a comparison study for both schemes in terms of the Number of Context Switching "NCS" until time=14, we obtain the following results: NCS_1=14 and NCS_2=10. Thus, the proposed algorithm within this paper provides a smaller number of context switching which is around a 28.5% reduction in the overhead to deliver all processes successfully without making any deadline miss. Table 2 illustrates a set of three jobs/tasks with their execution and deadline times as it was taken from (Hwang et al., 2011).

Example 2: three tasks with their timing constraints as in Hwang et al. (2011) are illustrated in Table 2.

The Gantt chart for example 2 after applying the algorithm developed in Alsheikhy et al. (2016) is demonstrated as follows (Fig. 4), only part of it is shown due to the space limitation as mentioned in the previous example. Each yellow square represents the execution process of T_3, the orange squares represent the execution process of T_2 while the blue ones refer to the execution process of T_1.

Table 2: Available tasks in single CPU

Tasks	Release Time	Deadline Time	Execution Time
T_1	0	12	3
T_2	0	6	3
T_3	0	4	1

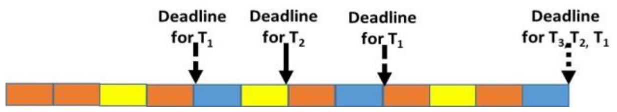

Fig. 4: Gantt chart for example 2 after applying the algorithm

The developed algorithm in Alsheikhy et al. (2016) was able to guarantee the timing constraints for all tasks by delivering their execution successfully. However, it suffers from high overhead in context switching. NCS_1=10. The Gantt chart from using the proposed technique within this paper is illustrated as follows (Fig. 5), min $\Delta\delta$ {9, 3, 3}=3.

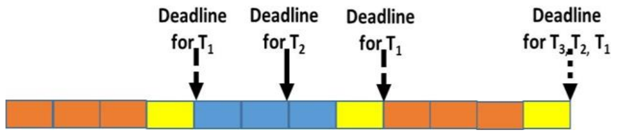

Fig. 5: The Gantt chart from using the proposed technique within this paper

The value for NCS_2, which refers to the proposed method, is found to be 5 which is 50% less than the overhead obtained from the developed scheme in Alsheikhy et al. (2016). So the reduction achieved is 50% which is very acceptable and desirable.

4. Simulation experiments

The developed simulation in MATLAB helped to test the proposed algorithm by performing multiple experiments with different scenarios and circumstances. More than 100,000 tasks were created randomly with different deadlines and execution times. Furthermore, around 300 sets were randomly generated and a different number of tasks were created in each set. The maximum time taken to complete 300 sets with 2000 tasks in each set was about 17 hours since the simulation was testing both methods. The proposed approach delivered all tasks successfully without allowing any deadline miss to occur. The simulation was run more than 10,000 times with several conditions and circumstances each time for both types of environments. During the simulation test, the number of used processors "M" varied from 1 to 10 as maximum. The simulation was developed to tell and show how many tasks met their timing constraints, how many tasks were unable to be executed successfully, the time needed to finish all tasks and the Number of Context Switching that occurred during the experiment.

The simulation tests both schemes, the one developed in Alsheikhy et al. (2016) and the proposed method within this paper. Both the deadline and the execution times were randomly generated with condition that c≤d, the arrival time (r) was also generated randomly by the simulation under a constraint that r<c and d. Information about the used platform to perform the experiments is shown in Table 3.

Table 3: Characteristics of used platform

Platform Name	System Type	CPU	Speed	RAM
Windows 10 Pro	64 bit	I5 core 2 Due	2.67 GHz	4 GB

Several scenarios are illustrated in the following tables with the results for both techniques. We will refer to the algorithm developed in Alsheikhy et al. (2016) as 1 and the proposed algorithm as 2 for the simple reason.

The following abbreviations are used in the table: NoI is the Number of Iterations, NST is the number of sets and tasks in each set, NCT is the number of completed tasks, NDM is the number of deadlines miss occurred, TET is the total execution time needed to complete all tasks in the list in (second "s") and NCS is stated earlier. Sp represents the speed up achieved in the proposed algorithm whereas Rd stands for the obtained reduction in NCS.

Scenario 1: Uniprocessor with the same arrive time (r=0), Table 4 shows that there are different sets and all sets hold the same number of tasks. All sets run for the same number of iterations too.

Table 4: Results of uniprocessor for NCT and NDM when r=0

NoI	NST	NCT		NDM	
		1	2	1	2
1000	3/16	598	536	0	0
3500	6/10	1001	893	0	0
7600	5/35	1691	1386	0	0
9000	11/60	1172	935	0	0

Table 4 shows that both algorithms were delivering all tasks successfully to meet their deadline times without allowing deadline miss to occur. Table 5 illustrates the results of the previous scenario for TET, NCS, Sp, and Rd. in each run.

Table 5: Results for TET, NCS, Sp, and Rd

NoI	NST	TET			NCS		
		1	2	Sp %	1	2	Rd %
1000	3/16	852	556	34	948	781	17
3500	6/10	335	198	41	1324	989	25
7600	5/35	647	412	36	2251	1906	15
9000	11/60	907	629	30	3591	3162	12

Table 5 illustrates that the proposed algorithm yielded better results in terms of improvement in the speed up for the response time and the reduction achieved in the overhead.

Scenario 2: Uniprocessor with different arrival times. Table 6 indicates the number of sets and the number of jobs in each set. In addition, all sets have the same number of iterations during the simulation tests. Furthermore, this table shows that both approaches perform perfectly in terms of finishing all tasks without allowing any deadline miss during the tests.

Table 6: Results of uniprocessor for NCT and NDM when r≥0

NoI	NST	NCT		NDM	
		1	2	1	2
770	4/20	413	327	0	0
3000	8/40	894	691	0	0
6830	7/120	820	773	0	0
9000	10/25	2406	2108	0	0

Both approaches delivered all tasks successfully. Nevertheless, the proposed algorithm provided better results in terms of NCT.

Table 7 illustrates the results of the previous scenario for TET, NCS, Sp, and Rd. in each run.

Table 7: Results for TET, NCS, Sp, and Rd

NoI	NST	TET			NCS		
		1	2	Sp %	1	2	Rd %
770	4/20	102	76	25	332	271	18
3000	8/40	297	201	32	591	484	18
6830	7/120	839	709	15	2792	2456	12
9000	10/25	245	179	27	319	248	22

The proposed method provided better achievement for the speed up improvement and in the overhead reduction. Using more CPUs will produce more enhancement for the speed up and more minimization in the overhead.

Scenario 3: Different arrival time with M=7. Table 8 depicts that 7 processors were used in order to allow task migration between different processors.

Table 8: Results of 7 processors for NCT and NDM when r≥0

NoI	NST	NCT		NDM	
		1	2	1	2
500	6/60	1302	974	0	0
8000	11/300	2054	1659	0	0
10000	20/150	2992	2698	0	0

Table 9 illustrates the results of the previous scenario for TET, NCS, Sp, and Rd. in each run.

Table 9: Results for TET, NCS, Sp, and Rd

NoI	NST	TET			NCS		
		1	2	Sp %	1	2	Rd %
500	6/60	35	19	46	109	62	43
800	11/89	208	128	38	287	139	51
1000	20/150	326	211	35	752	578	23

The more processors are used, the more improvement is achieved for both performance metrics which are the speed up for the response time and reduction in the overhead.

5. Conclusion

A dynamic algorithm to schedule periodic tasks in real-time systems is presented in this paper. That scheme is capable of delivering all tasks in the ready queue to meet their timing constraints without missing any deadline. It provides full CPU(s) utilization and maintains stability for the system. Furthermore, it provides a good timely response time as observed in the experiments that were conducted and produces a better reduction in the overhead. Several examples were given by the simulation to demonstrate how the proposed scheme worked.

Compliance with ethical standards

Conflict of interest

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

References

Alsheikhy A, Elfouly R, Alharthi M, Ammar R, and Alshegaifi A (2016). An effective real-time dynamic scheduling approach for periodic tasks. International Journal of Computing, Communications and Instrumentation Engineering, 3(2): 279-383.

Guo Z and Baruah SK (2015). A neurodynamic approach for real-time scheduling via maximizing piecewise linear utility. IEEE Transactions on Neural Networks and Learning Systems, 27(2): 238-248. https://doi.org/10.1109/TNNLS.2015.2466612 **PMid:26336153**

Harkut DG and Agrawal AM (2014). Comparison of different task scheduling algorithms in RTOS: A survey. International Journal of Advanced Research in Computer Science and Software Engineering, 4(7): 1236-1240.

Hwang M, Choi D, and Kim P (2011). Least slack time rate first: An efficient scheduling algorithm for pervasive computing environment. Journal of Universal Computer Science, 17(6): 912-925.

Kim J, Lee S, and Shin H (2013). Effective task scheduling for embedded systems using iterative cluster slack optimization. Circuits and Systems, 4(8): 479-488. https://doi.org/10.4236/cs.2013.48063

Ren J and Phan LTX (2015). Mixed-criticality scheduling on multiprocessors using task grouping. In the 27th Euromicro Conference on Real-Time Systems, IEEE, Lund, Sweden: 25-34. https://doi.org/10.1109/ECRTS.2015.10

Vora V and Somkuwar A (2012). Implementation and performance analysis of real time scheduling algorithms for three industrial embedded applications. International Journal of Information Technology Convergence and Services, 2(6): 1-10. https://doi.org/10.5121/ijitcs.2012.2601