An Effective Real-Time Dynamic Scheduling Approach for Periodic Tasks

Ahmed Alsheikhy[1], Raafat Elfouly[2], Mosleh Alharthi[3], Reda Ammar[1] and Abdulrahman Alshegaifi[1]

***Abstract*—**Real-time embedded systems are widely used in many applications such as control, monitoring and aviation. Several tasks are performed under strict time constraints. In such systems, deadline miss may lead to a fatal result so that all tasks (jobs) need to be scheduled to ensure that they meet their deadline times. Scheduling policy is one of various factors that affect their performance. It determines which task or a set of tasks should be selected first from ready queue to run. This paper presents an effective dynamic scheduling approach during run-time based on using a single value such as Worst-Case Execution Time (WCET) to schedule periodic tasks in either multiprocessor or uniprocessor environments. It selects which process or a set of processes must be selected for execution. The proposed algorithm works in any real-time system such as aviation, medical and process control for power plants or chemical plants.

***Keywords*—** Real-time embedded systems, scheduling approach, time constraints, deadline, performance, periodic tasks, dynamic, multiprocessor, uniprocessor, run-time.

I. Introduction

SCHEDULING which job(s) must be selected first and its or their parameters play a significant role on system performance. The main objective of scheduling is to decide which job is selected and run from ready queue and assigned to CPU [1]. Scheduling method affects CPU performance since it determines the CPU and resources utilizations [1],[2]. Two types of real-time systems exist nowadays and can be summarized as follows: 1) Hard real-time systems in which deadline miss means fail and could lead to a disaster result and 2) Soft real-time systems where deadline miss is tolerated and they still perform their functions. Scheduling can be defined as a method that specifies which task or a set of tasks is assigned to resources in order to complete a desired job.

A scheduler is responsible for scheduling activity; it is implemented to ensure that all resources are kept busy and to give users availability to share different resources effectively. In real-time systems, schedulers are developed to make sure all processes meet their deadlines for stability/severity sake.

Ahmed Alsheikhy[1], Reda Ammar[1] and Abdulrahman Alshegaifi[1] are with the Computer Science and Engineering Department, University of Connecticut, Storrs, CT 06268 USA.

Raafat Elfouly[2] is with the Computer Engineering Department, Cairo University, Egypt.

Mosleh Alharthi[3] is with the Electrical Engineering Department, Taif University, Saudi Arabia.

Static and dynamic methods for scheduling algorithms exist these days. In each category, there are two schemes of algorithms which are 1. Preemptive approach: where a task is blocked by a higher priority process and 2. Non preemptive approach: where the task completes its execution time even if a higher priority process has arrived [1],[2],[3]. A scheduling policy can be characterized by several factors which are summarized as follows: A. Do all processes meet their deadlines?, B. CPU Utilization: the percentage of the CPU being busy, C. Response time for every job and D. Fairness: which is dividing the CPU time equally among all available jobs [2],[3] and load balancing which is dividing the load equally among all processors/cores.

Multiple algorithms exist which can be summarized as follows: 1) Rate Monotonic (RM): is static type for periodic tasks since each process is assigned a priority based on its request rate. The higher request is the higher priority is. However, that priority does not change during run-time and that is why it is called static [3],[6], 2) Deadline Monotonic (DM): is a generalization version of rate monotonic. Their concepts are almost the same; however, the difference is that in deadline monotonic a priority assigned to each process is inversely proportional to its deadline. So a process with shortest deadline gets highest priority and assigned first to the resources [3],[6], 3) Earliest Deadline First (EDF): a process with shortest deadline time gets its turn first since it has a highest priority among all other processes. This algorithm is considered optimal and can be used for both types of tasks (periodic and aperiodic) in uniprocessor environments and 4) Least Slack Time first (LST): where a task with smallest slack is chosen first and assigned to available resources for execution. Slack is defined as the difference between the deadline (d) of any task with its remaining execution time (c^r) and current time (t) as depicted in figure 1 [2],[3],[4],[5].

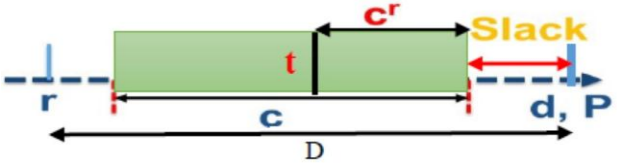

Fig. 1 time constraints of periodic tasks

Figure 1 shows time constraints of periodic tasks which are 1. Release time (r): which is a time at which a process becomes available at ready queue, 2. Execution time (c) which is considered to be the Worst-Case Estimated Time

"WCET", 3. Period (P): a time when the process repeats its cycle, 4. Absolute deadline (D): which is a time interval between release time and period of the process. In mathematical form, D = d – r and 5. Relative deadline (d): which is an interval time between first creation of the process and its deadline; mathematically, it is d = D + r. In many cases, P and d are the same.

Our contribution in this paper is done by developing a new hybrid scheduling algorithm for periodic tasks that works either on a uniprocessor or multiple processors systems; by hybrid we mean it cooperates with EDF algorithm when needed. It works during run-time to decide which task or a set of tasks should be selected first from ready queue and gains system resources such as CPU. The main objectives of this algorithm are to ensure that all processes meet their deadlines, keep a system stable, eliminate idle state of all existing CPUs and providing a good timely response time.

In the remainder of this paper, we present related work on scheduling schemes in Section 2, followed by a detailed discussion of the proposed approach in section 3. Section 4 includes two case studies to show the validation of the proposed approach on a multiprocessor environment. Section 5 includes simulation results. Section 6 is the conclusion of the paper.

II. Related work

In today's technology, many operating systems perform multitasking operations which mainly depend on scheduling algorithms [4]. These algorithms are used to ensure that all processes meet their deadline times and execute fairly [4],[5]. Multitasking can be defined as a concept of performing multiple operations at the same time. However, it does not imply that all tasks, also known as processes, are executed in parallel. In multitasking environment, decisions are made according to some criteria which can be summarized as follows: A. A new process was created, e.g., fork, B. A process switches from waiting to ready, e.g., I/O completion, C. A process switches from running to waiting, e.g., I/O was requested, D. A process switches from ready to running, e.g., CPU takes control of it and lastly E. A process was terminated.

In [4], M. Hwang, P. Kim and D. Choi proposed an optimized scheduling dynamic algorithm which uses a rate of the remaining execution time over the remaining time until deadline. The higher rate is the higher priority is; they claimed it is the optimal method for multiprocessor since it schedules all tasks correctly. It is very similar to our approach; however, our proposed approach yields better results since it has the capability to schedule periodic tasks with different arrival times whereas the method in [4] assumed all tasks with the same arrive times and this issue was the motivation for the proposed method. In many real-time systems, periodic tasks come with different arrival times and it is very rarely to have all processes with the same arrival times. So the algorithm in [4] is inapplicable for those systems. In [5], same procedures were done on different sets of tasks under various conditions by M. Hwang, P. Kim and D. Choi. Nevertheless, they did not include any information about their approach on tasks with different arrival times.

J. H. Anderson, V. Bud and U. C. Devi in [9] proposed a new scheduling algorithm based on EDF for multiple processors environment on soft real-time systems. Each task is either fixed or migrating which means it moves between different CPUs to accommodate its execution time along with deadline one. Their goal was to ensure bounded tardiness in EDF; however, some deadline miss occur which is not allowed in hard real-time systems. Our method works on hard real-time systems and ensures all CPUs in multiprocessor environments are fully utilized and no deadline miss occurs at all time which keeps system stable and fully functioning.

III. The Proposed Algorithm

The previous mentioned techniques such as RM, DM, LST and EDF are applicable on uniprocessor and are not preferred on multiple processors since they leave some CPUs with idle states and some deadline misses occur. Even the method described in [4] is unable to provide an optimal solution since it is useless on tasks with different arrival times. In the proposed algorithm, the motivations for it can be summarized as follows: 1. An efficient method on multiple processors and uniprocessor as well; working correctly on multiprocessor environments implies that it works too on uniprocessor without any issue, 2. Gives maximum CPUs utilization since it keeps all of them busy which delivers all tasks and no deadline miss occurs, 3. It is feasible approach which means it does what it is supposed to perform by ensuring stability under various circumstances and 4. An ability to develop an on-line dynamic scheduling technique which aims to prevent deadline miss at all times under several conditions or circumstances.

The objectives of the proposed algorithm are to A. Make sure all processes meet their deadline times and no deadline miss occurs and B. Keep all resources utilized.

Several assumptions are taken into consideration in order for the method to work in a right way and give the best results; the assumptions are as follows: A. Preemptive approach which means any task can be blocked by another task with higher priority, B. Task migration is allowed so the task finishes its execution on any available processor, C. All tasks in the ready queue are available upon selection and they are independent, D. Any process is not allowed to appear on multiple processors at the same time and E. Combines with EDF algorithm when and if needed.

The utilization U is computed as follows:

$$U=\sum_{i}^{n} \frac{ci}{di} \quad (1)$$

n represents number of processes available in the ready queue whereas i indicates the process index.

The following pseudo code describes how the proposed

scheme works.

Algorithm: Dynamic Scheduling Scheme

1. *All tasks in the ready queue are examined each time unit to decide which one should be selected first to assign to available resources* ***Initialization: Generate random numbers for the deadline times (d), arrival time (r) and the execution time (c)***
2. ***Initially, remaining execution time (C^r) = C, completed tasks (com) = 0 and deadline miss (dm) = 0***
3. ***While (ready queue != 0)***
4. *Compute $slack_i$, where $slack_i = d_i - t - c_i^r$, i is the process index and t represents current time*
5. *A rate or ratio R_i, where i denotes the task index, is computed using the following equation:*

   $R_i = \frac{slacki}{di-t}$, *$d_i$ is the relative deadline and t is the current time as stated earlier*
6. *A task with smallest rate gets the highest priority and assigned first to the system resources; if more than task have the same rate; then the task or set of tasks with shortest deadline is selected first.*
7. ***If (C^r(i) == 0), then***
8. *Remove C_i(t) from ready queue and com = com + 1*
9. *else*
10. *continue proceeding with remaining tasks*
11. *If (c_i^r > di) when process reaches its deadline, then*
12. *Deadline miss occurs and dm = dm + 1*
13. *If a new process arrives, insert it at its right place*
14. ***do***
15. *End*

IV. CASE STUDY

Two examples are presented in this paper to demonstrate how the proposed scheme performs. Both examples can be found in [5]. We will apply the proposed method on multiple processors environment since it is our concerned. First, we will perform EDF method on one example to show its weakness on multiprocessor and then perform our approach on it to show the difference in the results.

Case 1: Same arrival time for five processes in the ready queue as shown in table 1 where three CPUs are existed and used.

TABLE I
AVAILABLE PROCESSES IN THE READY QUEUE

Tasks	Release Time	Deadline Time	Execution Time
T_1	0	2	1
T_2	0	2	1
T_3	0	2	1
T_4	0	8	6
T_5	0	8	6

Using the EDF algorithm, the scheduling result is shown in table 2. P_i indicates the processor ID number.

TABLE II
EDF RESULT

Time / Processor ID	0	1	2	3	4	5	6	7	8
P_1	T_1	T_4	T_1	T_4	T_1	T_4	T_1	T_4	T_4
P_2	T_2	T_5	T_2	T_5	T_2	T_5	T_2	T_5	T_5
P_3	T_3	NOP	T_3	NOP	T_3	NOP	T_3	NOP	T_1

In processor 3, NOP represents no operation at that time which means it was idle. So T_4 and T_5 missed their deadlines. Both processors 1 and 2 were totally busy while processor 3 was only busy for about *55% of its time*. Table 3 represents the result of the proposed algorithm.

TABLE III
RESULTS OF THE PROPOSED ALGORITHM

Time / Processor ID	0	1	2	3	4	5	6	7
P_1	T_1	T_2	T_1	T_2	T_1	T_2	T_1	T_2
P_2	T_4	T_3	T_4	T_3	T_4	T_3	T_5	T_3
P_3	T_5	T_4	T_5	T_5	T_5	T_4	T_4	T_5

Table 3 indicates that all tasks met their deadlines and also all the three CPUs were fully busy and utilized.

Case 2: Same arrival time for nine processes in the ready queue as shown in table 4 with four CPUs.

TABLE IV
RELEASE TASKS

Tasks	Release Time	Deadline Time	Execution Time
T_1	0	15	8
T_2	0	6	5
T_3	0	10	4
T_4	0	4	3
T_5	0	4	2
T_6	0	3	1
T_7	0	3	1
T_8	0	5	1
T_9	0	60	7

All tasks met their deadline times as shown in table 5 which represents a snap shot of the whole answer due to the space limitation.

TABLE V
RESULT OF THE PROPOSED ALGORITHM

Time / Processor ID	0	1	2	3	4	5	6	7	8	9	10	11
P_1	T_1	T_1	T_2	T_1	T_2	T_1	T_2	T_1	T_2	T_1	T_2	T_1
P_2	T_2	T_2	T_3	T_2	T_4	T_2	T_3	T_2	T_3	T_2	T_4	T_3
P_3	T_4	T_4	T_4	T_5	T_6	T_4	T_4	T_6	T_4	T_4	T_5	T_7
P_4	T_5	T_6	T_7	T_8	T_7	T_5	T_5	T_7	T_8	T_5	T_6	T_8

The coming section includes simulation tests which were perfromed to test the algorithm under various condition in order to prove its validation.

V. SIMULATION EXPERIMENTS

We developed a simulation system to test the proposed algorithm under various conditions. The simulation proved validation of it and showed it provides the desired results. More than *7000 task sets* were tested with *average of 30 tasks in each set*; *each set ran for an average of 8000 times*. The simulation system works for uniprocessor and multiprocessor as well; several number of processors were used and the maximum number was set to *10 CPUs*. The simulation tells how many tasks met their deadlines, how many tasks missed their deadlines and the elapsed time to complete all sets. The execution time (C) and deadline time (d) were randomly generated by the simulation where d is greater than C and several tasks may have the same deadline times; the same applies on the execution time (C). The maximum deadline time was set to 60 and 100 to investigate how the algorithm behaves under several circumstances when the number of processes increases. The arrival time (r) was also generated randomly by the simulation under a constraint that r < c and d. Table 6 contains information about a device and platform we used to test the proposed algorithm.

TABLE VI
CHARACTERISTICS OF USED DEVICE

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

Case 1: Uniprocessor with the same arrive time (r = 0)

TABLE VII
RESULTS OF UNIPROCESSOR

Number of Iterations	Number of sets and tasks	Number of completed tasks	Number of deadline miss	time
1000	5/18	17	0	157s
3000	5/25	31	0	193s
7000	5/30	49	0	232s
10000	5/20	89	0	319s

Table 7 shows that the proposed algorithm was successfully scheduled all tasks and no task missed its deadline time. GUI in the simulation system took average of *24*s in each run which is included in the results in time column.

Case 2: Uniprocessor with different arrive time

TABLE VIII
RESULTS OF UNIPROCESSOR

Number of Iterations	Number of sets and tasks	Number of completed tasks	Number of deadline miss	time
8500	5/20	16	0	396s
9300	5/15	52	0	471s
10000	5/30	63	0	785s
10000	5/20	71	0	693s

There was no deadline miss as shown in table 8 and the arrival time values (r) influenced number of completed tasks which met their deadlines.

Case 3: Multiple processors with the same arrive time. M represents number of processors; M = 3.

TABLE IX
RESULTS OF 3 PROCESSORS WITH THE SAME ARRIVE TIME

Number of Iterations	Number of sets and tasks	Number of completed tasks	Number of deadline miss	time
3000	5/23	368	0	162s
5600	5/30	531	0	188s
7000	5/30	1197	0	448s
10000	5/50	1436	0	492s

Table 9 shows the results of using 3 processors under the same conditions we used for uniprocessor in case 1; no deadline miss occurred and number of completed tasks was doubled several times. The elapsed time reduced significantly which improved the delay and response times. Same test was repeated with different arrival times for each process; the number of processes which met their deadline times was varying from run to another one due to fact that different values were randomly generated.

Case 4: Same arrive time with M = 5 processors; same conditions in case 3 were applied.

TABLE X
RESULTS OF 5 PROCESSORS

Number of Iterations	Number of sets and tasks	Number of completed tasks	Number of deadline miss	time
3500	5/25	831	0	149s
4700	5/30	1011	0	155s
7000	5/30	1748	0	351s
10000	5/20	2793	0	406s

In the following case, the number of sets varied in each run and M was 7.

Case 5: different arrive time with M = 7

TABLE XI
RESULTS OF 7 PROCESSORS WITH DIFFERENT ARRIVAL TIMES

Number of Iterations	Number of sets and tasks	Number of completed tasks	Number of deadline miss	time
8500	6/20	2191	0	387s
9300	7/15	1998	0	422s
10000	5/30	2581	0	446s

The proposed algorithm was completely able to execute tasks as much as possible without any deadline miss. The following table shows a comparison analysis between the proposed approach and algorithm mentioned earlier in [5] on multiprocessor environments. In [5], the algorithm uses the

following equation to decide which task must be selected first; $R_i = C_i^r / (d_i - t)$. The higher rate is the higher priority is. The previous equation is quite similar to the proposed approach except that the task with smallest rate is selected based on the proposed method to allow CPUs to be busy all times to avoid any deadline miss. In addition, the slack quantity was chosen instead of the remaining time. The algorithm in [5] can be used only with all processes having the same arrival times which assumed to be "0" while the proposed algorithm can be used either with different arrival times or same arrival times.

The comparison includes the number of tasks completed their execution time without any deadline miss and the number of tasks missed their deadlines. #1 refers to the proposed algorithm while #2 represents the algorithm in [5]. The comparison was done under several conditions with the same number of sets, tasks and the same arrive time which was r = 0.

TABLE XII
RESULTS OF THE COMPARISON ANALYSIS

Number of Iterations and Processors	Number of sets and tasks	Number of completed tasks		Number of deadline miss	
		#1	#2	#1	#2
3000/4	5/23	780	779	0	0
4000/5	5/15	1281	1281	0	0
5000/6	5/30	1003	998	0	0
7000/7	5/20	2978	2977	0	0
10000/7	5/200	942	940	0	0

Table 12 shows clearly that both approaches performed very well in scheduling all tasks without any deadline miss. Nevertheless, the proposed algorithm provided more completed tasks. We applied many experiments by increasing number of tasks and sets while maintaining same number of processors M which was 10, the proposed algorithm produced more number of tasks which met their deadlines.

VI. CONCLUSION AND FUTURE WORK

This paper presented a method to schedule periodic tasks to meet their deadlines without allowing any deadline miss to occur. Also the proposed algorithm keeps all available CPUs in the system busy at all times to schedule more tasks. It keeps systems stable and provides a good timely response time as observed in the experiments done. Several examples were given to demonstrate how the proposed scheme works. Furthermore, we conducted comparative analysis between the proposed algorithm and algorithm in [5]; our scheme gave the best results in terms of number of completed tasks which met their deadlines under several conditions. Furthermore, both methods yielded no deadline miss in all our experiments. For future work, we will investigate a scheme to schedule periodic tasks based on their average expected execution time instead of using the *Worst Case Execution Time* "*WCET*" since the knowledge of knowing the WCET in many applications is unpredictable.

REFERENCES

[1] K. Li, X. Tang and B. Veeravalli, "Scheduling Precedence Constrained Stochastic Tasks on Heterogeneous Cluster Systems", IEEE Transactions on computers, Vol. 64, No. 1, pp. 191-204, January 2015. http://dx.doi.org/10.1109/TC.2013.205

[2] G. Muller, "Scheduling Techniques and Analysis", Buskerud University College, March 2013.

[3] B. Miller, F. Vahid and T. Givargis, "RIOS: A Lightweight Task Scheduler for Embedded Systems", WESE 12th Proceedings of the workshop on Embedded and Cyber-Physical Systems Education, ACM, New York, NY, USA, 2013. http://dx.doi.org/10.1145/2530544.2530553

[4] M. Hwang, P. Kim and D. Choi, "Least Slack Time Rate first: an Efficient Scheduling Algorithm for Pervasive Computing Environment", Journal of Universal Computer Science, Vol. 17, No. 6, pp. 912-925, 2011.

[5] M. Hwang, P. Kim and D. Choi, "Least Slack Time Rate first: New Scheduling Algorithm for Multi-Processor Environment", International Conference on Complex, Intelligent and Software Intensive Systems, pp. 806-811, 2010. http://dx.doi.org/10.1109/CISIS.2010.20

[6] M. Kaladevi, M.C. A., M. Phil and Dr. S. Sathiyabama, "A Comparative Study of Scheduling Algorithms for Real-Time Task", International Journal of Advances in Science and Technology, Vol. 1, No. 4, pp. 9-14, 2010.

[7] L. Niu, "Energy Efficient Scheduling Techniques for Real-Time Embedded Systems with QOS Guarantee", 16th International conference of Embedded and Real-Time Computing Systems and Applications (RTCSA), pp. 163-172, August 2010.

[8] L. Almeida, S. Fischmeister, M. Anad and I. Lee, "A Dynamic Scheduling Approach to Designing Flexible Safety-Critical Systems", Proceedings of the 7th ACM and IEEE International conference on Embedded Software (EMSOFT), pp. 67-74, New York, NY, USA, 2007. http://dx.doi.org/10.1145/1289927.1289942

[9] J. H. Anderson, V. Bud and U. C. Devi, "An EDF-based Scheduling Algorithm for Multiprocessor Soft Real-Time Systems", Proceedings of the 17th Euromicro Conference on Real-Time Systems (ECRTS), pp. 199-208, 2005. http://dx.doi.org/10.1109/ECRTS.2005.6

[10] L. A. Cortes, "Verification and Scheduling Techniques for Real-Time Embedded Systems", Department of Computer and Information Science, Dissertation No. 920, Linkoping University, Sweden, 2005.

[11] S. Baruah and J. Goossens, "Scheduling Real-Time Tasks: Algorithms and Complexity", pp. 1-35, 2003.

[12] A. Dudani, F. Mueller and Y. Zhu, "Energy-Conserving Feedback EDF Scheduling for Embedded Systems with Real-Time Constraints", Proceedings of the Joint conference on Languages, Compilers and Tools for Embedded Systems: Software and Compilers for Embedded Systems, Vol. 37, No. 7, pp. 213-222, New York, NY, USA, July 2002. http://dx.doi.org/10.1145/566225.513865

[13] N. C. Audsley, A. Burns, M. F. Richardson and A. J. Wellings, "Hard Real-Time Scheduling: The Deadline-Monotonic Approach", Proceedings of IEEE Workshop on Real-Time Operating Systems and Software, pp. 133-137, 1991.W. D. Doyle, "Magnetization reversal in films with biaxial anisotropy," in *1987 Proc. INTERMAG Conf.*, pp. 2.2-1–2.2-6.