Using Dynamic Moving Average in Real-Time Systems to Minimize Overhead and Response Time for Scheduling Periodic Tasks

Ahmed Alsheikhy

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

Summary
In real-time systems, scheduling algorithms are used in control situations where it is a crucial or a critical to complete a task successfully within a specific time interval. Many scheduling techniques consider scheduling tasks according to their Worst-Case Execution Time (WCET) or average execution time while neglecting a change in their probability distributions. In real-time applications such as multimedia, Using either WCET or the average value to schedule several tasks is impractical and inappropriate and could cause a catastrophic result. The previous studies show that the multimedia real-time applications such as Audio or Video statistically has a great variation in their execution times which means scheduling them according to the WCET or the average execution time is insufficient and unwanted results may occur. In this paper, a new effective and efficient dynamic method to schedule periodic real-time tasks is presented based on using a dynamic moving average approach. Dynamic moving average refers to a change in a probability distribution being used when a task is added or removed. The objective is to develop a method that guarantees the delivering of all tasks to meet their timing constraints and also to minimize the overhead occurring from context switching between different tasks. Furthermore, enhancing the response time minimization is desired. Our intensive experiments on a developed simulation performance evaluation indicate that the developed method is capable of handling all tasks to meet their deadline times, achieving around an average of 24% to 49% reduction in the overhead and the response time enhancing by average of 50%.

Key words:
Real-time applications, efficient dynamic scheduling algorithm, timing constraints, periodic tasks, probability distribution.

1. Introduction

Particularly, many real-time applications consist of precedence timing constrained stochastic tasks. Stochastic tasks are defined as tasks which their execution and communication times are random variables and follow certain probability distributions [1,2]. Ability to schedule completely and successfully all those constrained tasks is needed. However, it is not easy procedure since scheduling problems are considered NP-hard problems [2]. Image and Information processing along with the weather modeling are perfect examples for real-time applications where processing mainly depends on the amount of data being received with a great variation [1]. In addition, the communication time for different tasks could fluctuate the execution time according to the network capacity, routing methods and traffic issues [2]. Several scheduling approaches focus and aim only to schedule different tasks with deterministic execution and communication times [2]. In another word, they assume tasks or processes with fixed execution times, mostly is the WCET. However, relaying on that factor is impractical since many applications may have conditional instructions or operations which require different execution times for different inputs [2,3,4]. In addition, considering only the WCET or the average times as a factor to schedule tasks is inappropriate when dealing with several tasks which share randomness and uncertainty in their execution times [1,2,3,4]. A system under consideration performance is affected by scheduling methods since they determine the processors and resource utilizations. The objective of scheduling schemes is to map between several tasks or processes with a uniprocessor or multiprocessor environments to guarantee the satisfaction of their timing constrains [2,3,4]. Modern high performance processors such as Intel and AMD Athlon perform several operations at the same time. Performing those operations require a good scheduling method in order to keep a system stable and under control. When several application run and compete for resources on the uniprocessor or multi processors, there is a need to have an efficient, a sufficient and an effective approach that handled different tasks properly. Scheduling stochastic real-time tasks requires the known of the execution time in advanced which is very hard in real-time multimedia applications [1,4,5,6]. Several significant works on stochastic tasks scheduling have been performed. Next section explores the recent researches and studies. Nowadays, two categories of scheduling algorithms are found which are static and dynamic. Preemptive and Non preemptive techniques exist in each type. Preemptive method refers to blocking a current executed task by another task with a higher priority while non preemptive refers to the continuous of the execution procedure even there is another task with high priority in the ready queue.

Manuscript received April 5, 2017
Manuscript revised April 20, 2017

In many real-time systems, applications may be composed of a task or several tasks that are independent which need to be executed under very strict timing constraints [3,4,5,6,7]. They need to be dynamically scheduled according to their unpredictable execution times [6]. Schedulability analysis must be performed prior to a task execution process. It involves a significant sufficient information about the probability distribution being used and behavior of the execution [6,7]. Several tasks characteristics must be taken into account when develop a scheduling method which can be summarized as follows: 1. Arrival times "r": can be defined as the time when a task becomes available in the ready queue list, also known as the release time, 2. Execution times and 3. Deadline times "d". Several heuristic scheduling techniques neglect the importance of the execution times behavior and their uncertainty nature when developing methods to schedule periodic real-time tasks [7,8]. This neglection issue is the motivation key to investigate an algorithm that schedules stochastic tasks with a minimum overhead and minimum response time. The contribution in this paper is done by proposing and developing a very efficient and effective hybrid dynamic scheduling technique for periodic real-time tasks, works either on the uniprocessor or multiple processors systems, according to unpredictable execution time behaviour in order to minimize the overhead occurs from context switching and to improve the response time. Hybrid method indicates that it cooperates with the Earliest Deadline First (EDF) algorithm when and if needed. The proposed approach 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. Minimizing the overhead, enhancing the speed up of response time reduction by providing a good timely response reaction, delivering all tasks completely and successfully, maximizing the used CPU(s) utilizations and keeping the stability of a system under investigation are the main objectives for the proposed scheme. Table 1 illustrates the assumptions and characteristics for the proposed technique to perform properly. In the remainder of this paper, related work on scheduling schemes is presented 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 is the conclusion of the paper.

Table 1: The proposed method characteristics

Dynamic and Preemptive
Hybrid: cooperates with the EDF algorithm
All tasks are independent
The probability distribution is known or can be estimated
Applicable on any probability distribution being used
Works on all platforms: Uniprocessor and multiple processors
Appearing on different processors at a same time is strictly banned

2. Related Work

Recently, scheduling techniques for periodic real-time tasks have been studied and developed extensively due to increasing demands for real-time systems and applications. There has been several works purely on periodic tasks with uncertainty execution times and behaviors [2,3,4]. An efficient dynamic method to schedule periodic real-time tasks with variant execution times using dynamic average estimation is presented in [1]. A. Alsheikhy et al developed the algorithm based on the probability distribution being used and existed in the system under consideration. The developed approach in [1] delivers all tasks perfectly with no deadline miss occurs. However, high overhead from context switching and required computations for determining the selected tasks are existed. In this paper, the proposed scheme reduces the overhead by nearly 49% maximum and improves the speed up of the response time over 50% as observed from the intensive simulation experiments. In [1], each unfinished task is examined and checked every time unit to determine task or set of tasks must be chosen first based on their rate value. In addition, when multiple tasks share the same value for their rate, a task or set of tasks with the shortest deadline time is selected and then allocated to the CPU(s). The rate is computed according to the slack, deadline "d" and current time "t" values. The proposed method within this paper uses the same principle and idea but in different way. It determines a minimum value x for all uncompleted tasks by computing the difference between their deadline and the remaining execution times. The minimum value x is defined as the time slot assigned by the CPU(s) to every unfinished task in the ready queue list. The value for x varies dynamically from time to time which makes it very flexible. Using flexible dynamic value for x reduces the overhead and improves the response time as well. However, the minimum value x in [1] is fixed at all time which makes it static.

A model of scheduling stochastic parallel tasks in application on heterogeneous cluster systems is proposed in [3]. K. L. Xiaoyong et al discussed the stochastics scheduling attributes to deal with various random variables in order to show that the expected execution times is greater than or equal to the value of determined tasks. A Stochastic Dynamic Level Scheduling (SDLS) scheme is proposed. The proposed method combines the stochastic bottom levels with stochastic dynamic levels. Speed up and standard deviation were the performance parameters being tested and evaluated. The SDLS approach is applicable only on normal distribution while the proposed method within this paper deals with various probability distributions which makes it very flexible to be used and applied in many applications regardless their distributions. The SDLS scheme was compared with other three existing

heuristics scheduling algorithms which were SHEET, HEFT and Rob-HEFT. More information about the SDLS method is found in [3].

J. Li et al in [4] proposed a method to schedule parallel real-time tasks using a federated scheduling technique. Federated scheduling method is a scheme to schedule parallel tasks by allocating a dedicated cluster to a task or a set of tasks when its/their utilization ≥ 1 while execution other lower utilization tasks sequentially on a shared cluster. The average case of workload was used instead of the worst case load. Only stochastic tasks in soft real-time systems. A bounded expected tardiness criteria was used as the factor to estimate the boundary for stochastic tasks using federated scheduling methods. They mapped three federated schemes with different complexities for core allocation. Numerical evaluations for the proposed method was performed using randomly generated tasks sets. However, the proposed algorithm within this paper considers only hard stochastic real-time tasks where any deadline miss may cause a catastrophic or unwanted result. The authors in [4] calculated the expected tardiness in order to estimate the upper bound, BASIC and FAIR mapping approaches were used with a help from Linear Programming procedures. The WCET approach was not investigated in [4], however, the proposed method within this paper uses either the WCET or the average ones. Interested readers are referred to [4] for more information about the SDLS approach.

In [7], J. F. Kempf et al proposed a framework to estimate the expected performance of a given scheduling policy. A dynamic programming style procedure for automatically synthesizing an optimal scheduler for the expected time. An iterative computation of a stochastic time-to-go function was used. They presented an automatic derivation of optimal controller for non markovian continuous time processes. Initially, the estimation for the expected termination time under a given scheduler is developed according to the computation of the duration space. Furthermore, a local stochastic time-to-go and a global stochastic time-to-go values are computed. More information about the proposed framework can be found in [7].

K. Li et al in [8] proposed an energy-aware scheduling method for task execution cycles on heterogeneous computing systems with normal distribution only. A heuristic energy-aware stochastic tasks scheduling algorithm (ESTS) was developed by them by using linear programming approach. They claimed that their scheme achieved high scheduling performance for independent tasks with lower complexity. However, a trade-off between the scheduling length and energy consumption occurred.

3. The Proposed Algorithm

The proposed scheme in [1] is very efficient since it delivers all tasks successfully with no deadline miss occurs. However, it suffers from high computational demands according to the algorithm policy. In addition, high overhead from context switching takes place too since the approach checks each time unit to decide which task must be selected first from the tasks present in the ready queue. The proposed algorithm within this paper improves the mentioned scheme in [1] since it minimizes the overhead from context switching and also reduces the response time needed to complete a task successfully as observed from our experiments. The motivations to develop the proposed approach in this paper are summarized as follows:

- Developing dynamic and hybrid method to schedule periodic real-time tasks effectively and efficiently. Hybrid refers to cooperating with the EDF algorithm as stated earlier.
- Reducing the overhead from context switching and speeding up the required response time to finish any process completely and successfully.
- A method that works either on uniprocessor or multiprocessor environments.
- Maximizing CPUs and resources utilization while it keeps a system stable under several conditions or circumstances.
- Eliminating idling state “mode” to prevent any deadline miss that may occur.
- Feasible and realistic approach by ensuring that all processes meet their deadline times under any circumstance.
- Guaranteeing all tasks meet their timing constraints and all available resources are fully utilized are considered the main objectives for the proposed method.

In order for the proposed scheme to work effectively, several assumptions are made and taken into consideration which can be summarized as follows:

I. Preemptive approach which means any task can be blocked by another task with higher priority.

II. Task migration is allowed so any task finishes its execution on any available processor.

III. All tasks in the ready queue are available upon selection and they are independent.

IV. Any process is not allowed to appear on multiple processors at the same time.

IV. Combines with EDF algorithm when and if needed.

Algorithm: Dynamic Hybrid Scheduling Method

Input: Several periodic real-time tasks with their timing constraints such as deadline times, execution times, release times and period times.

Output: Scheduled tasks with minimum overhead from context switching and higher speed up than method in [1].

1. ***Initialization: Generate random numbers for the deadline times (d), arrival time (r), probability vector (p) and the execution time (c)***
2. ***Initially, remaining execution time (C') = C, completed tasks (com) = 0 and deadline miss (dm) = 0***
3. *Compute value x for all processes in the ready list, where* $x_i = d_i - C_i'$
4. *Find minimum value for x and assign it to Δx, which represents the time slot assigned by the CPUs to each executed task*
5. *The initial average value for* $C_i'0 = E[c] = \sum_{i=1}^{n} ci * pi$
6. ***While (ready queue != 0)***
7. *Compute* $slack_i$, *where* $slack_i = d_i - t - c_i'$, *i is the process index and t represents current time value*
8. *A rate or ratio* R_i *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*

9. *A task with smallest rate gets the highest priority and assigned first to the system resources; if more than tasks have the same rate; then the task or set of tasks with shortest deadline is selected first.*
10. *The selected task is executed for time equal to the value of Δx*
11. *For uncompleted tasks, the remaining execution time* C_i' *is determined as follows:*

if current (t) ≤ c_i, then

$C_i' = C_{i-1}' - \Delta x$

else

$C_i' = \frac{cir0 - \sum_{j=1}^{i} pj \cdot cj}{1 - pc} - \Delta x$, *where p_c refers to* the probability values of all processes with the execution time value "0".

12. ***If (C'(i) == 0), then***
13. *Remove* $C_i(t)$ *from ready queue and com = com + 1*
14. *Check if Δx is smaller than the smallest* C_i', *if so then, Δx =* C_i' *only if no deadline miss occurs.*
15. *else*
16. *continue proceeding with remaining tasks*
17. *If* ($c_i' > di$) *when process reaches its deadline, then*
18. *Deadline miss occurs and dm = dm + 1*
19. *If a new process arrives, insert it at its right place*
20. ***do***
21. *End*

Each periodic real-time tasks has timing constraints which are: release times "r", deadline times "d", execution times, period times "P'. The period time is defined as a time when each process is repeated. For the proposed scheme, P and d are the same. The achieved overhead reduction is nearly 50% as observed in the simulation experiments. Furthermore, the response time speed up increases around 50% for uniprocessor and more than 60% for multiple processor environments.

O(1) is the complexity of determining the remaining execution time as shown and proved in [21]. The probability vector P, which is associated with the tasks available in the ready list, is normalized automatically when a new process is added to it or an existing process is removed when it is successfully done. The remaining execution time "Cir" takes either positive or negative value according to changing in a type of distribution being used whereas the rate "Ri" takes only value ≥ 0. It is very clearly that the rate "R" becomes bigger as the process approaches its deadline time. The proposed approach can be applied on any real-time system where processing time varies from time to time such as multimedia systems where processing depends on amount of data which has great variations in voice and video.

4. Simulation Experiments and Numerical Evaluation

The developed simulation system was used to evaluate the proposed algorithm. Various conditions and circumstances were applied on the approach to prove its validity and strength. The intensive simulation experiments showed that the proposed method provided the desired results by:

1. Delivering all processes successfully completely with no deadline miss.
2. Reducing the overhead from context switching around 50% in average. However, the maximum obtained reduction in the overhead was nearly 64% when 10 CPUs were used with 45 tasks existed in 6 sets. So 45 tasks were randomly generated in each set.
3. Enhancing the speed up of the response time was around 35% to 42% in the uniprocessor environment while over than 50% was obtained for multiple processor environments.

Around 400,000 tasks with more than 200 sets were randomly generated and tested with an average of 10,000 times. Each set was composed of 10,000 to 20,000 tasks with random values for deadline and execution times, also each set of tasks was associated with probability vector "P" where ∑ Pi = 1. The maximum number of used CPUs "M" in the simulation was 10 and the developed simulation system works either in the uniprocessor or multiprocessor environments. The simulation is capable of telling the total number of processes that met their timing constraints successfully, the total number of overhead took place in the proposed algorithm, the response time taken to complete each set and the total number of tasks that failed

to complete their execution times before or at their deadline times.

In addition, a comparison between the overhead occurred and the response time needed for the algorithm in [1] and the proposed scheme within this paper was conducted and included in the simulation. The execution time (C), deadline time (d) and probability vectors 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 325 time units. The arrival times (r) was also generated randomly by the simulation under a constraint that r < c and d. Table 2 illustrates the characteristics of used device to test the proposed algorithm and shows its validation.

Table 2: characteristics and platform of used device

Platform Name	*System Type*	*CPU*	*Speed*	*RAM*
Windows 10 Pro	64 bit	Intel I5-3210M Core 2 Due	2.50 Ghz	4 GB

The following tables show the results of applying the proposed algorithm for both environments under several circumstances and/or conditions. The tables illustrate the total number of tasks that met their timing constraints "**TTC**", the total number of tasks that were unable to meet their deadline times "**TTF**", the total number of overhead happened from context switching between several processes on different CPUs "**TOV**" and the response time needed to deliver all tasks successfully "**RT**" in seconds. The second column in each table represents the total number of sets being generated and the total number of generated tasks in each set. All experiments were performed in normal distribution. However, several probability distributions can be applied on the proposed algorithm.

Example 1: Uniprocessor with the same arrival time (r = 0)

Table 3. Results of uniprocessor with the same arrival time

Number of Iterations	*Number of sets and tasks*	*TTC*	*TTF*	*TOV*	*RT "s"*
3500	4/30	179	0	153	69
5000	12/64	802	0	751	178
10,000	7/95	936	0	922	368
15,000	20/300	6103	0	5897	892
23,000	25/500	12596	0	12495	1368

The proposed method delivered all tasks completely and successfully without any deadline miss occurred. Example 2 illustrates the result of applying the scheme in the uniprocessor environment with different arrival times for several tasks. There is a chance that several processes may share the same arrival time. However, using different arrival times influence the needed response time to finish all tasks as observed as shown in table 4

Example 2: Uniprocessor with the different arrival times (r ≥ 0)

Table 4. Results of uniprocessor with different arrival time

Number of Iterations	*Number of sets and tasks*	*TTC*	*TTF*	*TOV*	*RT "s"*
2450	6/20	142	0	178	397
4500	10/90	1092	0	1390	531
15,000	8/110	921	0	858	785
30,000	15/350	5591	0	6034	2944
50,000	48/400	25692	0	25534	5428

As illustrated in table 4, having different arrival times impacts the response needed by increasing it which is very obvious since several processes start their cycles later. Nevertheless, the proposed algorithm handled all tasks perfectly since there was no deadline miss.

Example 3: different arrival times (r ≥ 0) with M = 5

Table 5. Results of using 5 CPUs

Number of Iterations	*Number of sets and tasks*	*TTC*	*TTF*	*TOV*	*RT "s"*
5000	11/50	1396	0	759	394
7000	15/210	5293	0	3298	639
10,000	20/200	7120	0	3989	542
28,000	20/482	15587	0	6119	921
35,000	70/690	70385	0	38596	3849

Table 5 illustrates that using several CPUs enhanced the speed up nearly 50% when comparing the same circumstances in the uniprocessor environment. The overhead occurred was significantly reduced too since the dynamic moving average mechanism adjusts the value of time slot assigned by the CPUs to each task.

Example 4: different arrival times (r ≥ 0) with M = 9

Table 6. Results of using 9 CPUs

Number of Iterations	*Number of sets and tasks*	*TTC*	*TTF*	*TOV*	*RT "s"*
20,000	9/30	4437	0	692	317
7000	21/33	5932	0	2894	478
14,000	35/120	8710	0	3911	499
54,000	50/566	49912	0	710	635
100,000	100/1000	140,03	0	33,029	1,381

Example 5: different arrival times (r ≥ 0) with M = 10

Table 7. Results of using 10 CPUs

Number of Iterations	*Number of sets and tasks*	*TTC*	*TTF*	*TOV*	*RT "s"*

10,000	15/20	2673	0	398	291
13000	20/35	4918	0	551	611
21,000	25/200	13254	0	468	498
65,000	40/250	42699	0	10332	1204
100,000	88/500	103451	0	32007	20510

Tables 6 and 7 clarify that the proposed approach was able to deliver all processes with different arrival times. Next tables demonstrate the comparison study conducted between the developed algorithm in [1] and the proposed scheme within this paper based on the TOV and RT for both methods. The conducted comparison was performed in multiple processor environments with 5 CPUs. Furthermore, the arrival time values were set to 0 which indicates that all processes were randomly generated at the same time. In tables 8 and 9, "TPA" represents The Proposed Approach whereas "DM" refers to the Developed Method in [1]. "NoI" refers to the Number Of Iterations and "NoST" refers to the Number Of Sets and Tasks in each set. Third column for both performance metrics "TOV and RT" in tables 7 and 8 which is represented by % refers to the improvement obtained when using the proposed method (TPA).

Table 8. Conducted comparison analysis for TOV

NoI	***NoST***	***TOV***		
		DM	***TPA***	***%***
10,000	10/20	782	419	46.4
25,000	15/50	1496	683	54.3
50,000	30/250	3599	1927	46.5
100,000	50/600	13,528	7012	48.2
200,000	100/800	54,982	30,419	44.7

Table 9. Conducted comparison analysis for RT

NoI	***NoST***	***RT***		
		DM	***TPA***	***%***
10,000	10/20	201	83	58.7
25,000	15/50	568	393	30.8
50,000	30/250	1249	538	56.9
100,000	50/600	7917	4833	39
200,000	100/800	15958	8992	43.7

Both previous tables (8 and 9) clearly show that the TPA outperforms the DM, which developed in [1], in terms of reduction in the TOV and the obtained speed up for the RT. Nearly 50% was achieved for both performance metrics for either uniprocessor or multiple processors environments. Several conditions were applied to determine the behaviors of both methods and investigate how they react under severe circumstances.

Table 10 lists some of the known distributions with a formula to determine the remaining execution time (Cr) values as illustrated in [21], more information can be found in [21]

Table 10. List of known probability distributions

Distribution Type	***Remaining Execution Time Equation***	***Explanation***
Uniform	$Cr = \frac{2-t}{2}$	t represents the current time
Exponential	$Cr = \frac{1}{\mu}$	μ is the service rate, $\frac{1}{\mu}$ *is the mean service rate*
Power tail	$Cr = 1 + \frac{t}{1+\alpha}$	α *is a positive constant*
Erlangian-3	$Cr = \frac{1}{\mu} * \left[\frac{3+2\mu t+\left(\frac{\mu t)^2}{2}\right)}{1+\mu t+\left(\frac{\mu t)^2}{2}\right)}\right]$	Has 3 identical exponential servers in cascade. One at a time

5. Conclusion and Future Work

This paper presented improved scheme to minimize the overhead occurs from context switching and enhance the speed for the response time needed to deliver all tasks successfully for periodic real-time tasks using dynamic moving average based on the approach developed in [1]. The proposed scheme guarantees that all processes in the ready queue meet their timing constraints while producing minimum overhead and response time. Using dynamic moving average method for uncertainty execution times to schedule real-time tasks gives the flexibility needed to achieve the obtained results.

The proposed approach can be applied on different probability distribution. However, this paper presented only discrete distribution. Furthermore, the proposed algorithm keeps all available CPUs in the system busy at all times to schedule more tasks. Furthermore, it keeps systems stable and provides a good timely response time as observed in the experiments that were performed and conducted. Several examples were given to demonstrate how the proposed scheme works. In addition, comparison study and analysis evaluation between the TPA and DM were performed and conducted. The results from the comparison analysis indicate that the TPA outperforms the DM with around 50% improvement for TOV and RT metrics.

For future work, investigating the impact of using the proposed approach on the produced power consumption during the evaluation procedures will be conducted. Also, developing a method to apply the proposed method for Internet of Things (IoT) applications where the execution time varies from time to time will be studied and performed.

References

[1] Alsheikhy A., Ammar R., Elfouly R., Alharthi M. and Alshegaifi A., "An Efficient Dynamic Scheduling Algorithm for Periodic Tasks in Real-Time Systems Using Dynamic

Average Estimation", 2016 IEEE Symposium on Computers and Communication (ISCC 2016), pp. 820-824, Italy, June 2016.

[2] Alsheikhy A., Elfouly R., Alharthi M., Ammar R. and Alshegaifi A., "An Effective Real-Time Dynamic Scheduling Approach for Periodic Tasks", International Journal of Computing, Communications and Instrumentation Engg. (IJCCIE), Vol. 3, Issue 2, pp. 279-383, May 2016.

[3] Li K., Tang X. and Veeravalli B., "Scheduling Precedence Constrained Stochastic Tasks on Heterogeneous Cluster Systems", IEEE Transactions on computers, Vol. 64, No. 1, pp. 191-204, January 2015.

[4] Li J., Agrawal K., Gill C. and Lu C., "Federated Scheduling for Stochastic Parallel Real-Time Tasks", Proceedings of IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2014), pp. 1-10, August 2014.

[5] Miller B., Vahid F. and Givargis T., "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.

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

[7] Kempf J. F., Bozga M. and Maler O., "As Soon as Probable: Optimal Scheduling Under Stochastic Uncertainty", Proceedings of International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS 2013), pp. 385-400, Rome, Italy, March 2013.

[8] Li K., Tang X. and Yin Q., "Energy-Aware Scheduling Algorithm for Task Execution Cycles with Normal Distribution on Heterogeneous Computing Systems", 41st International conference on parallel processing, pp. 40-47, 2012.

[9] Tasneem S., Zhang F., Lipsky L. and Thompson S., "Comparing Different Scheduling Schemes for M/G/1 Queue", 6th International Journal on Electrical and Computer Engineering (ICECE), pp. 746-749, Dhaka, Bangladesh, 2010.

[10] Abdelmaksoud E. Y., "Performance and Reliability-Driven Scheduling Approach for Efficient Execution of Parallelizable Stochastic Tasks in Heterogeneous Computing Systems", International Journal Open Problems in Computer Science and Mathematics, Vol. 3, No. 2, pp. 137-163, June 2010.

[11] Iqbal N. and Henkel J., "SETS: Stochastic Execution Time Scheduling for Multicore Systems by Joint State Space and Monte Carlo", Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD 2010), pp. 123-130, San Jose, CA, 2010.

[12] Tasneem S., Ammar R., Lipsky L. and Sholl H., "Improvement of Real-Time Job Completion Using Residual Time-Based (RTB) Scheduling", International Journal of Computers and their Applications, Vol. 17, No. 3, pp. 117-132, March 2010.

[13] Cong J. and Gururaj K., "Energy Efficient Multiprocessor Task Scheduling Under Input-Dependent Variation", Proceedings of IEEE C on Design, Automation and Test in Europe conference Exhibition, pp. 411-416, April 2009.

[14] Satish N., Ravindran K. and Keutzer K., "Scheduling Task Dependence Graphs with Variable Task Execution Times onto Heterogeneous Multiprocessors", Electrical Engineering and Computer Sciences, University of California at Berkley April 2008.

[15] Chen J. J., Yang C. Y., Lu H. I. and Kuo T. W., "Approximation Algorithms for Multiprocessor Energy-Efficient Scheduling of Periodic Real-Time Tasks with Uncertain Task Execution Time", Proceedings of IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2008), pp. 13-23, 2008.

[16] Beck J. C. and Wilson N., "Proactive Algorithms for Job Shop Scheduling with Probabilistic Durations", Journal of Artificial Intelligence Research 28, pp. 183-232, 2007.

[17] Kamthe A. and Lee S. Y., "Stochastic Approach to Schedule Multiple Divisible Tasks on a Heterogeneous Distributed Computing System", IEEE International Parallel and Distributed Processing Symposium on Heterogeneous Computing Workshop, pp. 1-11, March 2007.

[18] Tasneem S., Lipsky L., Ammar R. and Sholl H., "Using Residual Times to Meet Deadlines in M/G/C Queues", NCA, pp. 128-138, 2005.

[19] Tasneem S., Sholl H. and Ammar R., "Practical Methods for Deadline Scheduling using Process Residual Time", CAINE, pp. 175-180, 2005.

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

[21] Tasneem S., "Applications and Effects of Process Residual Time Information in Time Sensitive Dynamic Scheduling", Ph.D. Dissertation, University of Connecticut, USA, 2005.

[22] Tasneem S., Ammar R. and Sholl H., "A Methodology to Compute Task Remaining Execution Time", Proceedings of the International Symposium on Computers and Communications (ISCC), pp. 74-79, 2004.

[23] Dogan A. and Ozguner F., "Genetic Algorithm Based Scheduling of Meta-Tasks with Stochastic Execution Times in Heterogeneous Computing System", Proceedings of Cluster Computing, Vol. 7, No. 2, pp. 177-190, April 2004.

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

Ahmed Alsheikhy, received the B.S. and M.S. degrees in Electrical and Computer Engineering from King Abdulaziz University 2004 and 2010, respectively. In 2016, he received the Ph.D. degree in Computer Science and Engineering from University of Connecticut, USA. During 2004-2010, he stayed in Technical Vocational Training Corporation (TVTC), Saudi Arabia to train military soldiers about computer networks and technical support. His research interests are on Performance Evaluation, Artificial Intelligence, Satellite and Radar communication systems and Internet of Things.