Real-Time Control System

Atef Gharbi [1, 2]

[1]Faculty of Computing and Information Technology;
Northern Border University; KSA
[2]Université de Carthage, Institut National des Sciences Appliquées et de Technologie (INSAT),
LISI, Tunisia

Abstract
Tasks scheduling have been gaining attention in both industry and research. The scheduling that ensures independent task execution is critical in real-time systems. While task scheduling has gained a lot of attention in recent years, there have been few works that have been implemented into real-time architecture. The efficiency of the classical scheduling strategy in real-time systems, in particular, is still understudied. To reduce total waiting time, we apply three scheduling approaches in this paper: First In/First Out (FIFO), Shortest Execution Time (SET), and Shortest-Longest Execution Time (SLET). Experimental results have demonstrated the efficacy of the SLET in comparison with the others in most cases in a wide range of configurations.

Keywords: *Task, Real-Time System, Architecture, Scheduling.*

.1. Introduction

In general, the scheduling problem is a problem in which the aim is to properly assign available resources to tasks in order to maximize a specific objective, such as the makespan [1, 2], total completion time [3, 4], tardiness [5], overall lateness [6], total throughput time [7], and so on. In this paper, we are interested in real-time scheduling. We begin firstly with the definition of "Real-Time System" which can be defined as: "A real-time system is one in which the correctness of a result not only depends on the logical correctness of the calculation but also upon the time at which the result is made available" [8].

A real-time task is a task that has a critical deadline, i.e. one that must be executed within a certain amount of time [9, 10]. This concept reinforces the idea that time is one of the most critical elements of the system, and that tasks have timing constraints. These tasks usually include the control or reaction to events occurring in the external world in "real time". A hard real-time task is one that must respect the deadline because in case of failure to do so will result in disagreeable effects [11]. However, a soft real-time task is one in which respecting the deadline is looked-for but not necessary [12]. In this paper, we are interested more in soft real-time task. Besides, we consider that the tasks are pre-emptible then the scheduler is pre-emptive. In this case, if a task is considered more important or urgent, it may take control of the processor instead of another task, whose execution is stopped and resumed later.

The characteristics of the system on which scheduling algorithms are applied are generally used to classify them based on the following features:

The scheduling may be uniprocessor or multiprocessor. Uniprocessor scheduling is used when all tasks are done on a single processor [13]. If is not the case (i.e. there are many processors), we say "multiprocessor scheduling" which means multicore processor platform [14]. In this paper, we consider only uniprocessor scheduling.

There are two types of scheduling: static and dynamic scheduling. The static scheduling necessitates full details about the tasks to be scheduled (such as deadlines, priorities, times, etc.) [15, 16]. These details must be known a priori to propose a scheduling for the tasks before executing it.

The schedule is said to be dynamic if the feasibility can be evaluated at run time and updates in the configuration can be generated. Static schedules must be prepared offline at all times.

Dynamic schedules may be constructed either off-line or on-line [17]. The dynamic scheduling is considered off-line if the whole scheduling problem is known a priori but the execution is modified at run time [18]. The dynamic scheduling is considered on-line if the future is unpredictable or unspecified [19]. We are interested in this paper with off-line dynamic scheduling which has the advantage of determinism.

In this paper, we consider Real-Time Control System composed of intelligent robot ensuring the control of the production system in real-time. To do so, we propose a hierarchical control framework for real-time control. At a higher level, the main components are: the intelligent robot controlling the real-time system. Designing a generic architecture to incorporate an intelligent robot for controlling a production system in real time entails dealing with a few common problems such as how to choose the task to be executed and the scheduling strategy to be applied. The basic architecture can be defined in one of two ways: (1) Specialized modules can be created to ensure specific functions, or (2) A general procedure can be used to implement all of the processing functions. Based on our study, we choose the second solution to propose a generic architecture for real-time control system.

Manuscript received April 5, 2021
Manuscript revised April 20, 2021
https://doi.org/10.22937/IJCSNS.2021.21.4.3

Many tasks are assigned to the intelligent robot in order to be executed. These tasks must be scheduled locally on the robot. The intelligent robot may have a range of tasks from which to choose dynamically, but some may be impractical due to resource and time constraints. Therefore, we present three approaches for increasing feasibility: First In/First Out strategy (FIFO), Shortest Execution Time strategy (SET) and Shortest-Longest Execution Time strategy (SLET). All of them take into account the total waiting time. These scheduling approaches do not guarantee the generation of an optimal schedule, but they are capable of generating an acceptable schedule to minimize the waiting time for the overall tasks. We study these approaches on three datasets and we compare the results to determine the best solution to be applied.

The main contributions of this paper are: (1) a hierarchical control framework that combines model-based methods for dynamic scheduling of real-time tasks; (2) a set of three scheduling approaches: First In/First Out strategy (FIFO), Shortest Execution Time strategy (SET) and Shortest-Longest Execution Time strategy (SLET) to minimize the total waiting time.

The paper has been organised in the following way. The section 2 presents the conceptual model of real-time control system. The section 3 analyses the scheduling problem by introducing three different scheduling policies: FIFO, SET and SLET. We present and discuss the results and decide which one leads to optimal results.

2. Conceptual Model of Real-Time Control System

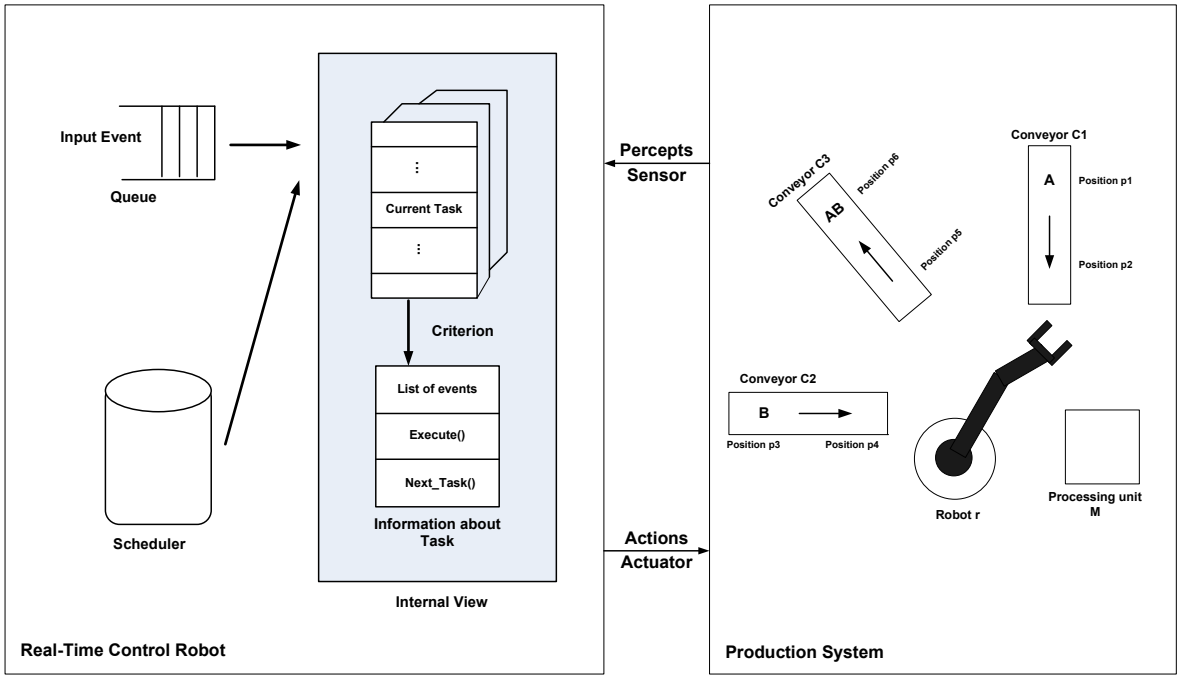

Figure 1. Real-Time Control System

A Real-Time Control System is composed of two main components which are: Real-Time Control Robot and the Production System (Figure 1). In this paper, we are interested more in the Real-Time Control Robot component that's why we define it in more details. Designing a modular architecture to incorporate an intelligent robot for controlling a production system in real time entails dealing with a few common issues. Some functions, such as how to choose the task to be performed and the scheduling strategy to be applied by the intelligent robot, must be implemented, making it difficult to design them to work with great flexibility, while other functions involving high computational load can be carried out in one of two ways: (1) Specialized modules can be created to ensure specific functions like robot motion, and perception function, for example. (2) A general procedure can be used to implement all of the processing functions. If the number of processing functions must be held to a minimum, the former method is more effective, while the latter solution provides more flexibility and requires more design effort to be generic. The basic architecture that has been used as a guideline in this study is based on the second solution. We develop an intelligent control robot as shown in figure 2. We propose according to the well-known UML language the concepts

of ***RealTimeSystem***, ***ControlRobot***, ***Behavior***, ***Scheduler***, ***Queue*** and ***Tasks***.

- ✓ ***RealTimeSystem***: is characterized by several robots that can control the system. The System is dynamic which permits to add (resp. remove) any Control Robot at any time with the method *addRobot()* (resp. *removeRobot()*).
- ✓ ***ControlRobot***: represents different behaviors that can be applied by a Control robot. The Control Robot behavior is flexible thanks to methods *addBehavior()* and *removeBehavior()* which facilitates to change the robot behavior without having the need to stop the whole system.
- ✓ ***Behavior***: represents the solution to be applied. The Behavior is composed of a list of tasks with precedence constraints to be executed.
- ✓ ***Scheduler***: The task scheduler is one of the Control Robot's most important components, since it determines which of the ready tasks should be executed. If no ready tasks are available at a given time, no task can be executed, and the Control Robot will remain idle until a task becomes available.

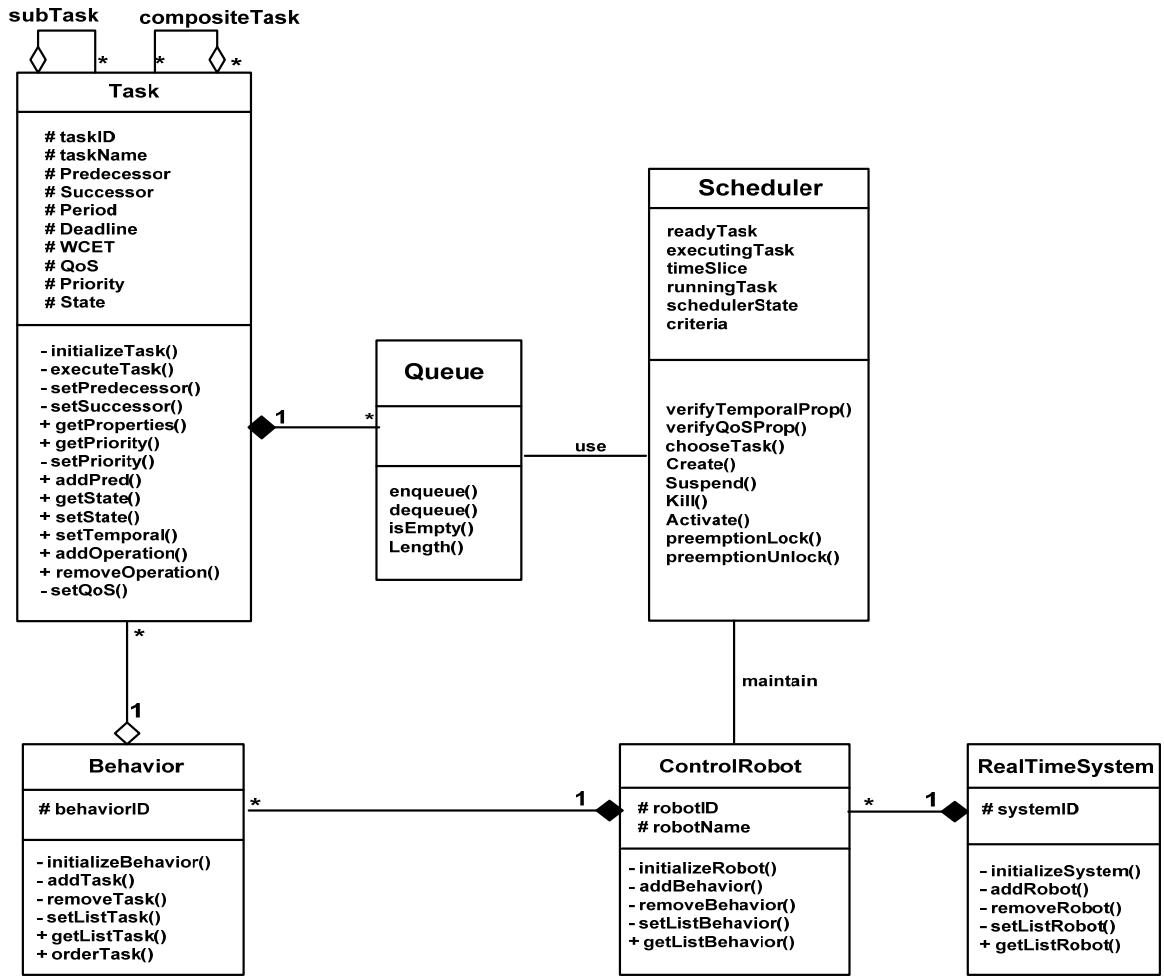

Figure 2. The general conception of a Real-Time Control System based on Tasks

As there are many tasks in competition with other tasks to be executed. The scheduler is expected to meet explicitly response-time constraints by enabling a scheduling policy that ensures timely completion of critical tasks. A scheduler related to a real-time Control Robot is characterized by:

- readyTask: a queue that holds a collection of tasks in ready state.
- runningTask: a queue that keeps track of the tasks that are currently running.

- timeSlice: the threshold of preempting a task (the quantity of time assigned to a task before its preemption).

It is common in the scheduling domain to model task state by specifying a limited collection of machine modes, such as "on," "off," and "stand-by." The scheduler associates a finite state machine used to describe the transitions to switch from one state to another. The task may be in one of the following possible states: Create, Wait, Run, Suspend or Terminate. Every task is in one state at any given time:

- ***Wait***: The scheduler puts a task in waiting state to be allocated in the future. Based on a set of parameters, the scheduler defines which ready task will be performed next (for example priority criterion i.e. the task having the highest priority will be assigned to the processor). Since it is not the highest priority, the scheduler switches the task state from Suspend to Wait (The link number 1 in the figure 3).
- ***Suspend***: The scheduler suspends a task (there are many reasons such that waiting for an event or a simple delay). The scheduler alters the task state from *Run* to *Suspend* because it does not have the resource (The link number 3 in the figure 3).
- ***Run***: The scheduler runs a task, so that its instructions can be executed. At any given time, only one task can be in this state, while all other tasks can be in different states at the same time. The scheduler updates the task state from *Suspend* to *Run* because it becomes unblocked and has the highest priority (The link number 2 in the figure 3). The scheduler changes the task state from *Wait* to *Run* because it becomes having the highest priority (The link number 6 in the figure 3).
- ***Terminate***: The scheduler terminates the task execution, the task allocator deletes it and releases the resources it was using. The scheduler alters the task state from *Run* to *Terminate* because it finishes its execution (The link number 4 in the figure 3). The scheduler updates the task state from *Suspend* to *Terminate* because it eliminates this task (The link number 4 in the figure 3). The scheduler changes the task state from *Wait* to *Terminate* because it eliminates this task (The link number 8 in the figure 3).

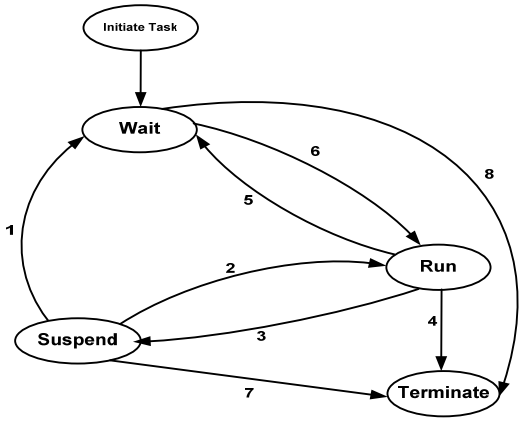

Figure 3. Diagram of Task states.

- ✓ ***Queue***: Several tasks can be ready or blocked at the same time. As a result, the system keeps a queue for blocked tasks and another for ready tasks. The latter is organized in a priority order, with the most important priority at the top of the list. When a task in the ready state is allocated a processor, it shifts from the ready state to the running state. This processor role is known as dispatching, and it is carried out by the dispatcher, which is a component of the scheduler.
- ✓ ***Task***: may be composed of several sub-tasks and each Task may have precedence constraints. Each task allow controls of devices by reading and interpreting data from sensors, before reacting and activating the appropriate actuators.

3. Scheduling Problem in a Real-Time Control System

The scheduling problem in multi-robot system includes:
- A set of intelligent robots (either Faulty Robots or Helping Robots)
- A collection of tasks that need to be carried out
- A set of constraints which determines the execution order of the tasks.

If the intelligent robot is free and has enough time (which means the intelligent robot studies its current list of tasks already scheduled to be executed and specifies if there is time for the new task to be done before its deadline), then it accepts to execute this task.

3.1. Scheduling Problem: Definition

The intelligent robot tries to find an idle time slot or window (time frame) in which the requested task can be carried out. The idle time slot is defined by the start time and deadline time for the requested task. In this idle time slot, some tasks have been already scheduled to be executed. It is possible that the idle time slot is filled with other tasks.

Figure 4 depicts the scheduling problem. In fact, the robot has three tasks to execute A, B and C. the task A (B, C respectively) has 10 (20, 15 respectively) time units to be executed. The task A is released at 0. The deadline for these 3 tasks is 100 time units. The robot tasks are scheduled to run in the first 45 time units of the time window. While carrying out the task A, the robot receives a task request at 5 time units and the deadline is 70.

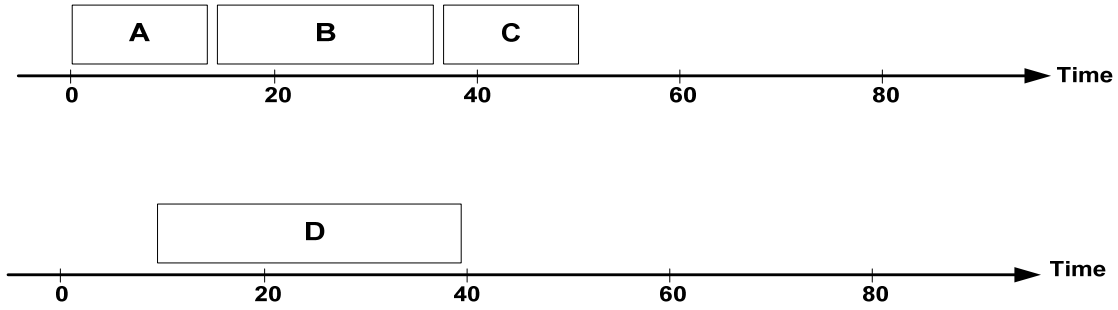

Figure 4. Scheduling problem

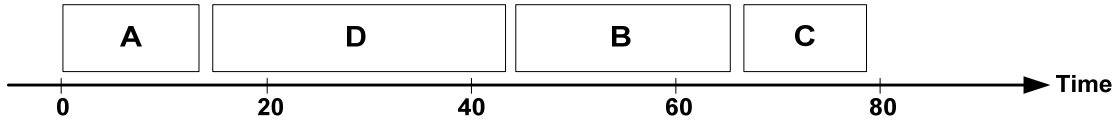

Figure 5. A completed scheduling

Since Task D has 30 time units and the time window is 70, it is possible to rearrange the tasks between (B, C) and D to find a solution to this problem (see Figure 5).

If the schedule exists, the robot should inform the other corresponding robot that it accepts to schedule within its own tasks. If there is no solution, then the robot sends a negative answer to the other part.

The intelligent robot can apply several policies such as:

- First In/First Out strategy (FIFO): the intelligent robot uses the First In/First Out criteria to organize requested tasks based on the time arrival;
- Shortest execution time strategy (SET): the intelligent robot selects the requested tasks having the smallest execution time;
- Shortest-Longest execution time strategy (SLET): the intelligent robot uses the same queue as above but it divides into two queues based on the median value: the first one is used for short requested tasks and the second one is used to long requested tasks. After that, the intelligent robot chooses firstly two requested tasks having the smallest execution time and secondly one requested task having the longest execution time.

The agent finds a solution in running time of O (n*m) where n is the number of tasks scheduled and m is the number of requested task to be scheduled. Indeed, every existing task is compared to the requested task which leads to a nested loop. In the worst case, it will be Max ($O(n^2)$, $O(m^2)$).

3.2. Formalization

A : Set of all available robot a ∈ A. Each robot a is indexed numerically but also may have a unique name associated with it.

T Set of all tasks *i T* that must be completed to achieve an objective. Each task *i* is indexed numerically but may have a unique name that describes the service needed to complete it.

For each task, we have

- ✓ ri = release date of task i
- ✓ di = due date of task i
- ✓ Ci = the completion time
- ✓ Fi = Ci - ri, the flowtime
- ✓ Li = Ci - di, lateness of task i
- ✓ Ti = max{0, Li}, tardiness of task i
- ✓ Ei = max{0, -Li}, earliness of task

$C_{max} = max_{i=1..n}\{Ci\}$ makespan

$L_{max} = max_{i=1..n}\{Li\}$ maximum lateness

$T_{max} = max_{i=1..n}\{Ti\}$ maximum tardiness

The helping robot has some important data to handle which are:

- $Queue^{Agent}$ represents a queue associated to a different kinds of requested tasks coming from faulty robots. Each requested task contains data about:
- ✓ Executing_time: amount of time to execute the requested task.
- ✓ Arrival_time: the time at which the requested task is arrived to the intelligent robot
- Free: Boolean data to indicate if the intelligent robot is available
- Available_time: data to indicate the amount of time that the intelligent robot is still free.
- N represents the number of requested tasks saved in the queue related to an Intelligent Robot.
- Task_time: the time at which the intelligent robot begins to handle the requested task
- WT: the waiting time for a requested task till it is treated by the intelligent robot
- AWT represents the average waiting time for the different requested tasks in a queue related to an Intelligent Robot.

For the sake of simplicity, we present here only the main steps of the algorithm applied to the different requested tasks related to an Intelligent Robot. Let Request_i be a requested task related to the $Queue^{Agent}$ such that $1 \leq i \leq N$. We assume that the Intelligent Robot computes the waiting time of each requested task Request_i denoted by WT_i. The waiting time is a measure of the total time that a requested task waits in a queue. It corresponds to the duration between the arrival time of the requested task (denoted by Arrival_time) and the initiation of its treatment by the Intelligent Robot (denoted by Task_time).

WT_i = Task_time – Arrival_time

We denote in addition by AWT the average waiting time of all the requested tasks belonging to $Queue^{Agent}$. It is equal to the sum of the different waiting times of different requested tasks WT_i divided by their number where $1 \leq i \leq N$.

$$AWT = \sum_{i=1}^{N} WT\_i / N$$

The intelligent robot treats the requested task periodically where the Intelligent Robot checks if it is free, selects a requested task according to a specific riterion, calculates the waiting time associated to this requested task and then deletes it from the corresponding queue as follows:

```
N ← 0
For each period Δ
    If (Free = true) then
        While Queue^Agent.length() > 0 do
            Request ← Choose_request()
            If (Available_time > Request.
        Executing_time) then
                Task_time ← currentTime()
                Queue^Agent .remove(Request);
                WT_i ← Task_time – Request.
            Arrival_time
                AWT ← AWT + WT_i
                N ← N + 1
                AWT ← AWT / N
        End if
        End while
    End if
End for
```

3.3. Simulation

In this paper, we consider and compare several alternatives to choose a requested task. We present a comparative study between well-known re-scheduling strategies (FIFO, SET, and SLET). We propose to evaluate the performance by applying these strategies so that we determine the best policy that the Intelligent Robot should take.

These results were performed using SimMRS (Simulator Multi-Robot System tool) which is a software tool that we have developed for simulation. To evaluate the different strategies in our experiment, we use our simulator: SimMRS that enables us to conduct experiments through generating random tasks. The optimization objective is to minimize the average waiting time over all tasks in the different queues.

Three different datasets were utilized through testing the proposed algorithm against three different re-scheduling algorithms: First In/First Out strategy (FIFO), Shortest Execution Time strategy (SET) and Shortest-Longest Execution Time strategy (SLET). Each dataset consists of randomly generated ten tasks as it is illustrated in Table 1.

Comparison Metrics

Three performance metrics are applied to the schedules generated by the three re-scheduling policies (FIFO, SET, and SLET).The comparison metrics to

evaluate performance are the Average Waiting Time (AWT) and the unscheduled tasks number (UTN).

- The average waiting time (AWT) is a measure of the total time that all requests wait in a queue. It is equal as predefined in the previous section:

$$AWT = \sum_{i=1}^{N} WT\_i / N$$

Where WT_i = Service_time – Arrival_time

- The unscheduled tasks number (UTN) represents the number of requested tasks that an intelligent robot cannot schedule.

Table 1: Datasets

Tasks	Dataset 1		Dataset 2		Dataset 3	
	Burst time	Arrival time	Burst time	Arrival time	Burst time	Arrival time
T1	30	0	170	0	35	0
T2	80	1	130	1	180	1
T3	120	2	110	2	70	2
T4	160	3	250	3	110	3
T5	200	4	30	4	10	4
T6	220	4	60	5	90	5
T7	180	4	220	3	30	6
T8	50	5	100	3	70	7
T9	70	3	50	4	20	3
T10	60	6	120	6	60	8

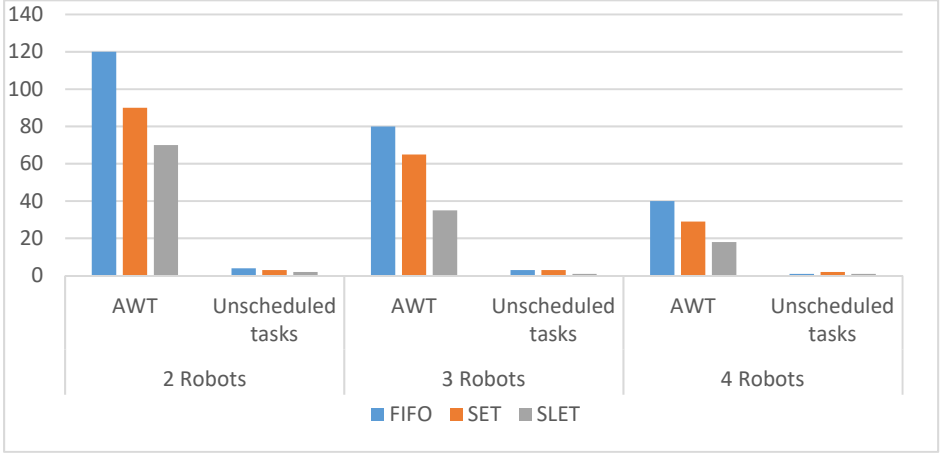

Figure 6. First Experimentation Results using Dataset1, on (2, 3, 4) Robots

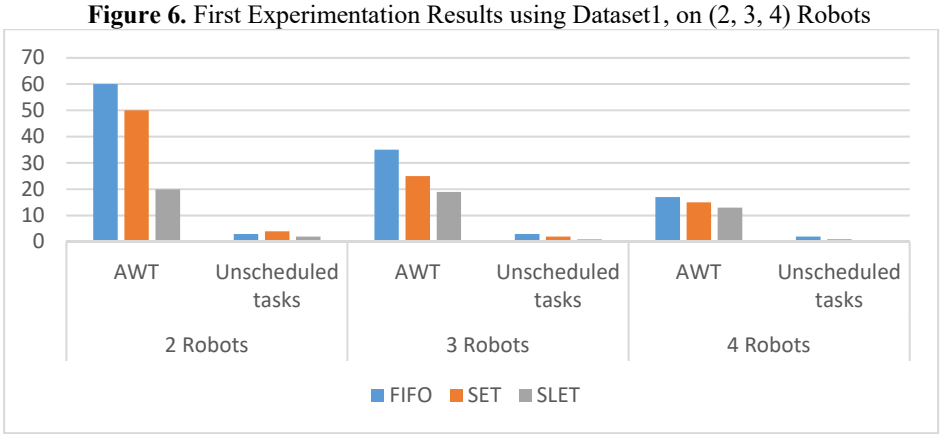

Figure 7. Second Experimentation Results using Dataset2, on (2, 3, 4) Robots

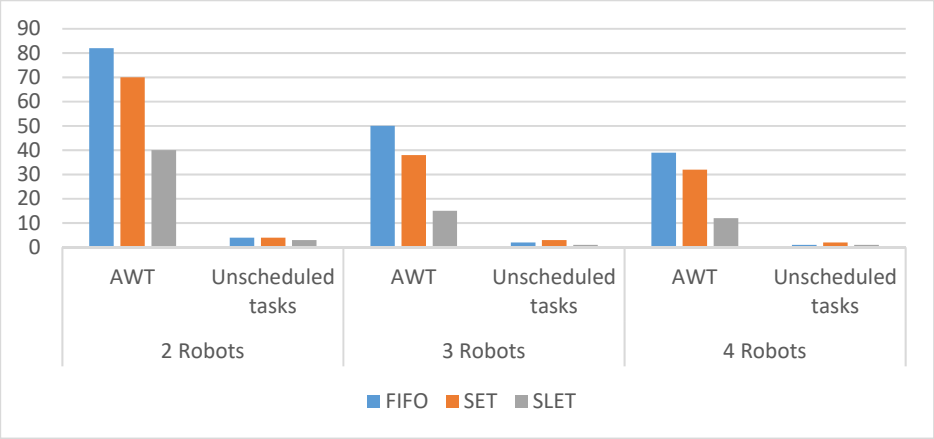

Figure 8. Third Experimentation Results using Dataset3, on (2, 3, 4) Robots

3.4. Interpretation

To give a closely look at the overall performance of the approaches, the figures 6, 7, and 8 present the Average Waiting Time (AWT) as well as the number of unscheduled tasks for each strategy. If we focus on the first criterion (AWT), as seen from the curves in Figures 6, 7, and 8, we conclude that the best solution to be applied by the Intelligent Robot is SLET approach. Whenever the SET policy is applied, the tasks having long burst time will suffer from starvation. Results have shown that scheduling based on FIFO policy is quite inadequate the tasks coming late also will suffer from starvation. In the SLET policy, we tried to reduce the average waiting time (measured in seconds) to ensure fairness in picking out tasks for execution based on the balance between the short tasks from the first queue and the long tasks from the second one. Thus, the SLET policy can be applied to provide better performance for the intelligent robot. This result may be expected because the FIFO or SET approach can lead to starvation. Nevertheless, the FIFO approach generates a huge values of AWT so it is considered as the worst approach. This degradation of AWT is due to that the Intelligent Robot gives priority to only requests coming first whereas the other queues are neglected which leads to heavy AWT. By considering all these interpretations, we recommend to apply the SLET policy.

If we consider the second criterion, we notice that the SLET policy provides the best results in comparison to FIFO and SET policies. Based on these two criterion, it is highly recommended to deploy SLET policy in scheduling as it presents optimal results. Most of the previous scheduling studies have focalized on only one criterion to compare.

4. Conclusion

This paper addresses the scheduling problem in real-time system. To do so, we present firstly the conceptual modelling of real-time system. Secondly, we study the scheduling through three approaches reducing the total waiting time: First In/First Out strategy (FIFO), Shortest Execution Time strategy (SET) and Shortest-Longest Execution Time strategy (SLET). Empirical results show that SLET approach outperformed the others in most cases. These scheduling approaches do not guarantee the generation of an optimal schedule, but they are capable of generating an acceptable schedule to reduce the waiting time for the overall tasks. In the future, we are planning to generate optimal scheduling in our solution.

We will also show how our methodology can be applied to real-world situations involving robot teams, and how feasibility analysis can be used to predict the solution's scalability to large robot teams. As future work, we can simulate the results on Xenomai which is a real-time embedded Linux.

Acknowledgments:

The authors gratefully acknowledge the approval and the support of this research study by the grant no-7494-CIT-2017-1-8-F- from the Deanship of Scientific Research at Northern Border University, Arar, K.S.A.

References

[1] Mohammad Mahdi Ahmadian, Mostafa Khatami, Amir Salehipour, T.C.E. Cheng, Four decades of research on the open-shop scheduling problem to minimize the makespan, European Journal of Operational Research, 2021, ISSN 0377-2217, https://doi.org/10.1016/j.ejor.2021.03.026.

[2] Golshan Madraki, Robert.P. Judd, Accelerating the calculation of makespan used in scheduling improvement heuristics, Computers & Operations Research, Volume 130, 2021, 105233, ISSN 0305-0548, https://doi.org/10.1016/j.cor.2021.105233.

[3] Rubing Chen, Jinjiang Yuan, C.T. Ng, T.C.E. Cheng, Single-machine hierarchical scheduling with release dates and preemption to minimize the total completion time and a regular criterion, European Journal of Operational Research, Volume 293, Issue 1, 2021, Pages 79-92, ISSN 0377-2217, https://doi.org/10.1016/j.ejor.2020.12.006.

[4] Long Wan, Jiajie Mei, Jiangze Du, Two-agent scheduling of unit processing time jobs to minimize total weighted completion time and total weighted number of tardy jobs, European Journal of Operational Research, Volume 290, Issue 1, 2021, Pages 26-35, ISSN 0377-2217, https://doi.org/10.1016/j.ejor.2020.07.064.

[5] Shijin Wang, Wenli Cui, Feng Chu, Jianbo Yu, Jatinder N.D. Gupta, Robust (min–max regret) single machine scheduling with interval processing times and total tardiness criterion, Computers & Industrial Engineering, Volume 149, 2020, 106838, ISSN 0360-8352, https://doi.org/10.1016/j.cie.2020.106838.

[6] Gur Mosheiov, Daniel Oron, A note on scheduling a rate modifying activity to minimize total late work, Computers & Industrial Engineering, Volume 154, 2021, 107138, ISSN 0360-8352, https://doi.org/10.1016/j.cie.2021.107138.

[7] E Mayer, J Raisch, Time-optimal scheduling for high throughput screening processes using cyclic discrete event models, Mathematics and Computers in Simulation, Volume 66, Issues 2–3, 2004, Pages 181-191, ISSN 0378-4754, https://doi.org/10.1016/j.matcom.2003.11.004.

[8] Gambier, Adrian. "Real-time control systems: a tutorial." 2004 5th Asian Control Conference (IEEE Cat. No. 04EX904). Vol. 2. IEEE, 2004.

[9] Kun Cao, Guo Xu, Junlong Zhou, Mingsong Chen, Tongquan Wei, Keqin Li, Lifetime-aware real-time task scheduling on fault-tolerant mixed-criticality embedded systems, Future Generation Computer Systems, Volume 100, 2019, Pages 165-175, ISSN 0167-739X, https://doi.org/10.1016/j.future.2019.05.022.

[10] Ehsan Ghadaksaz, Saeed Safari, Storage capacity for EDF–ASAP algorithm in energy-harvesting systems with periodic implicit deadline hard real-time tasks, Journal of Systems Architecture, Volume 89, 2018, Pages 10-17, ISSN 1383-7621, https://doi.org/10.1016/j.sysarc.2018.03.005.

[11] Yi-wen Zhang, Cheng Wang, Chang-long Lin, Energy-aware sporadic tasks scheduling with shared resources in hard real-time systems, Sustainable Computing: Informatics and Systems, Volume 15, 2017, Pages 52-62, ISSN 2210-5379, https://doi.org/10.1016/j.suscom.2017.06.002.

[12] Guowei Wu, Zichuan Xu, Temperature-aware task scheduling algorithm for soft real-time multi-core systems, Journal of Systems and Software, Volume 83, Issue 12, 2010, Pages 2579-2590, ISSN 0164-1212, https://doi.org/10.1016/j.jss.2010.08.017.

[13] Arnoldo Díaz-Ramírez, Pedro Mejía-Alvarez, Luis E. Leyva-del-Foyo, Comprehensive Comparison of Schedulability Tests for Uniprocessor Rate-Monotonic Scheduling, Journal of Applied Research and Technology, Volume 11, Issue 3, 2013, Pages 408-436, ISSN 1665-6423, https://doi.org/10.1016/S1665-6423(13)71551-7.

[14] Golnaz Taheri, Ahmad Khonsari, Reza Entezari-Maleki, Leonel Sousa, A hybrid algorithm for task scheduling on heterogeneous multiprocessor embedded systems, Applied Soft Computing, Volume 91, 2020, 106202, ISSN 1568-4946, https://doi.org/10.1016/j.asoc.2020.106202.

[15] Sandra Catalán, José R. Herrero, Enrique S. Quintana-Ortí, Rafael Rodríguez-Sánchez, Static scheduling of the LU factorization with look-ahead on asymmetric multicore processors, Parallel Computing, Volume 76, 2018, Pages 18-27, ISSN 0167-8191, https://doi.org/10.1016/j.parco.2018.04.006.

[16] Joel Matějka, Björn Forsberg, Michal Sojka, Přemysl Šůcha, Luca Benini, Andrea Marongiu, Zdeněk Hanzálek, Combining PREM compilation and static scheduling for high-performance and predictable MPSoC execution, Parallel Computing, Volume 85, 2019, Pages 27-44, ISSN 0167-8191, https://doi.org/10.1016/j.parco.2018.11.002.

[17] Libing Wang, Xin Hu, Yin Wang, Sujie Xu, Shijun Ma, Kexin Yang, Zhijun Liu, Weidong Wang, Dynamic job-shop scheduling in smart manufacturing using deep reinforcement learning, Computer Networks, Volume 190, 2021, 107969, ISSN 1389-1286, https://doi.org/10.1016/j.comnet.2021.107969.

[18] Danyu Bai, Lixin Tang, Open shop scheduling problem to minimize makespan with release dates, Applied Mathematical Modelling, Volume 37, Issue 4, 2013, Pages 2008-2015, ISSN 0307-904X, https://doi.org/10.1016/j.apm.2012.04.037.

[19] Adeline T.H. Ang, Appa Iyer Sivakumar, Chao Qi, Criteria selection and analysis for single machine dynamic on-line scheduling with multiple objectives and sequence-dependent setups, Computers & Industrial Engineering, Volume 56, Issue 4, 2009, Pages 1223-1231, ISSN 0360-8352, https://doi.org/10.1016/j.cie.2008.07.018.