Scheduling time-triggered tasks in multicore real-time systems: a reinforcement learning approach

Félicien Fiscus-Gay
Computer Science
Auckland University of Technology
Auckland, New Zealand
felicien.fgay@gmail.com

Abstract

Background: Previous research and/or rationale for performing the study.
Aims: Hypotheses/propositions to be tested, or goal of the study.
Method: Description of the type of study, treatments, number and nature of experimental units (people, teams, algorithms, programs, tasks etc.), experimental design, outcome being measured.
Results: Treatment outcome values, level of significance.
Conclusions: Limitations of the study, implications of the results, and further work

Index Terms

real-time system, scheduling, time-triggered tasks, DAG, multicore

I. Introduction

Real-time systems are utilized in various domains such as air traffic control, public transportation, and automated vehicles. Unlike non-real-time systems, tasks in real-time systems must be both functionally correct and meet strict (or flexible) execution time constraints, known as deadlines. Failure to meet these deadlines can lead to severe consequences. The critical nature of these systems necessitates designing the system architecture with a focus on time and incorporating fault tolerance to ensure high reliability.

One example of such architecture is the time-triggered architecture (TTA) [1] [2], which offers a fault-tolerant communication protocol and a precise timing system to synchronize different electronic control units. Developing and running tasks on these architectures require in-depth knowledge of the system and its architecture, which complicates code reusability and scalability when adding hardware resources or upgrading to a larger system.

To address these issues, the Automotive Open System Architecture (AUTOSAR[1]) was developed. AUTOSAR introduces layers of abstraction between hardware, firmware, and software, enhancing software reusability and hardware scalability across different systems while maintaining safety and security standards. It is now the most widely used architecture among car manufacturers, with notable core partners including BMW, Ford, and Toyota.

Scalability, in particular, plays a crucial role in modern real-time systems. Increasingly, real-time systems such as autonomous cars or computer vision systems are enhancing their computational resources by transitioning to multiprocessor systems. This shift from uniprocessor to multiprocessor systems addresses the growing complexity and computational demands of tasks executed on these systems, aiming to reduce both the execution time of these tasks and the required resources [3].

Hence, an increasing number of real-time systems are utilizing multi-core hardware to parallelize their tasks and convert sequential programs into parallelized ones using frameworks such as OpenMP [2]. Unfortunately, in most real-life scenarios, the number of available processors/cores is fewer than the number of tasks/subtasks that can be executed in parallel (i.e., independent tasks). This means that not all independent tasks can be executed simultaneously on the system, raising the question: which task should be executed first?

This question is particularly important in a real-time context because having the wrong execution order, or schedule, could lead to, at best, a slow system, and at worst, deadline misses, which can have fatal repercussions. In the case of a self-driving car system, for instance, a slight delay of 500 ms in detecting a pedestrian crossing the road can, in some cases, be enough to drive over the pedestrian or cause a car accident. Note that the resources of real-time systems are scarce and limited, which is why using as little processing power as possible while ensuring that tasks meet their deadlines is of crucial importance.

The extreme case of this scheduling problem arises when only one processor is available to execute tasks. This is known as task scheduling on a uniprocessor, and [4] provided two major priority policies: Rate Monotonic (RM) and Earliest Deadline

[1]https://www.autosar.org/
[2]OpenMP (2011) OpenMP Application Program Interface v3.1. http://www.openmp.org/mp-documents/OpenMP3.1.pdf

First (EDF) for scheduling periodic tasks. However, when considering multiple processors, the scheduling problem becomes much more complex, and different task models must be considered.

A prevalent task model is the time-triggered task model, which specifies tasks that execute periodically and is well-suited for time-triggered systems. Another type of task is the Logical Execution Time (LET) task. The LET paradigm is based on the time-triggered paradigm and was originally introduced by the Giotto real-time programming language [5] and later refined by [6] into the Hierarchical Timing Language (HTL). The main principle behind the LET paradigm is that each task's inputs and outputs are read and written in zero time, i.e., constant time.

The benefits of using LET are twofold. Firstly, the zero-time communication semantics greatly improve the predictability of the overall system and make I/O operations (i.e., memory access) on shared resources deterministic, which is crucial in real-time systems due to the highly negative impact that memory access contentions can have on the system [7]. Secondly, using LET in programming also provides a layer of abstraction that facilitates the direct translation from modeling to implementation, thus ensuring the implementation of timing requirements, enhancing code maintenance, and producing a less error-prone code base [8].

One drawback of LET is its implementation overhead, which increases the execution times of tasks due to the zero-time communication semantics [9]. Despite this drawback, the advantages of LET make it attractive for real-time systems [10], which is why the focus here will be on time-triggered and LET task scheduling on multi-core systems. Given that the problem of scheduling independent tasks is NP-hard[3] [11], no scalable optimal algorithm exists. Therefore, heuristics are used to partially solve the problem.

Consequently, machine learning will be considered here as it can better approximate the unattainable perfect solution while being scalable in terms of computing time after the training phase. In other words, the research questions are:

RQ1 What is the current state-of-the-Art in time-triggered task scheduling ?

- RQ1.1 What is the current state-of-the-Art for independent task scheduling ?
- RQ1.2 What is the current state-of-the-Art for task scheduling with precedence constraints ?

RQ2 Can machine learning be a better solution to schedule time-triggered task ?

- RQ2.1 Can a machine learning solution compare to state-of-the art heuristics for scheduling Directed Acyclic Graph tasks (tasks with precedence constraints)
- RQ2.2 Can a machine learning solution be comparable to ILP solutions while being more scalable ?

To achieve this, the background section will introduce the various technical terms, concepts, and fundamental algorithms. Following this, a systematic literature review will be conducted to address R1, and finally, the artifact and experimental design, results, and conclusion will be presented to answer R2.

These research questions which were explored using SLR (RQ1) and *method from Section IV*.

The solution we propose has the following features..

The primary contributions of this paper are:

II. Background

Task scheduling introduces several fundamental concepts.

A. Periodic task and schedule

Firstly, a periodic task $\tau_i(C_i, D_i, T_i)$ is characterized by its worst-case execution time (wcet) C_i, its deadline D_i, and its period T_i. This definition can be expanded by including an initial offset, which corresponds to the time of the task's first execution, and an activation offset, which is the time delay between the task being ready to execute (i.e., its execution period has begun) and the task actually starting to run. Secondly, a schedule S is a function that assigns a boolean value for each task τ and each time tick t, indicating whether the task τ is running at time t. Therefore, a scheduling algorithm is the method that, given a set of tasks, produces a schedule S for the task set.

This task model and schedule definition are widely adopted in the literature (see section III) and are the building blocks of all scheduling algorithms. The periodic task model, in particular, is used to define more complex tasks such as DAG tasks (see below) that will be used as input in the machine learning model (see section IV).

B. DAG task

A Directed Acyclic Graph (DAG) task is a task that models the multiple subtasks of an chain of tasks that have a precedence constraints. For example, when considering the task τ_1 that makes an aircraft keep its altitude, you usually have a number of subtasks to handle this task, namely : reading from the altitude sensor (τ_{11}), reading for the speed sensor (τ_{12}), computing the

[3]If a problem is NP-hard, it means that it is very unlikely to find a solution in polynomial time complexity, i.e., solutions are not scalable

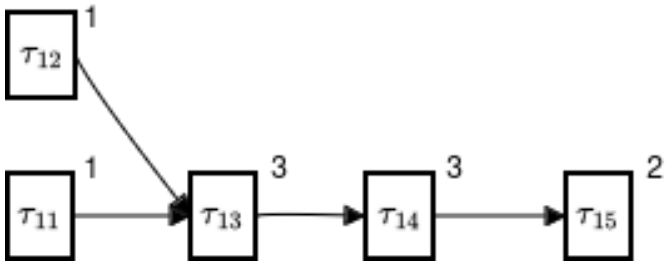

Fig. 1. DAG task τ_1. The nodes are the subtasks, the edges of the graph represent the precedence constraints between each subtasks and the worst-case execution time (wcet) of each subtask written as an exponent.

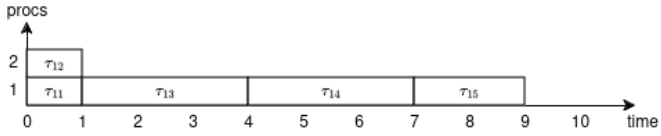

Fig. 2. Example of schedule for τ_1. The y axis represents the number of processors that are not idle. For the first two subtasks there are two processors active and for the rest there is only one active processor.

new speed for the aircraft to keep its altitude ($\tau_1 3$), computing the amount of thrust needed to achieve this new speed ($\tau_1 4$), and finally actuating the aircraft's jet engine ($\tau_1 5$). In this example, the DAG for τ_1 can be seen in Figure 1.

A DAG task τ_i also has a period T_i and a wcet C_i which is the sum of its subtasks wcets, and a deadline D_i. For instance, according to Figure 1, the wcet for τ_1 is 10 time units. You can also see how, for τ_1, the subtasks τ_{12} and τ_{11} can be parrallelized (i.e., executed in parallel) but the subtask τ_{13} needs to wait for both τ_{11} and τ_{12} to finish their execution before it can start running.

This concept will be the task model used in to conduct part of the systematic literature review (see Section III) and it also will be the task model used for designing the machine learning model (see Section IV).

C. Utilization factor

The utilization factor represents the percentage of processing time that a taskset $(\tau_1, \cdots, \tau_n)$ will utilize. Formally, it is defined as

$$U = \sum_{k=1}^{n} \frac{C_k}{T_k} \tag{1}$$

where U is the utilization factor. This concept is significant because, when evaluating a scheduling algorithm S, we desire S to effectively schedule tasksets that maximize the utilization factor U. Consequently, the higher the utilization factor bound for S, the more efficient the scheduling algorithm. Additionally, this concept is valuable in real-time systems where processing resources are often limited and expensive, making it crucial to maximize their usage.

This concept is also used either as a measurement when comparing two scheduling algorithms (see Section III), or used as a parameter to generate tasksets or DAG tasks with a fixed utilization (see Section IV).

D. Makespan

The makespan or end-to-end response time of a DAG task is the amount of time it takes for all the subtasks in the DAG task to finish executing when given a schedule. For instance, for the task τ_1 shown in Figure 1, the makespan of τ_1 for the schedule shown in Figure 2 is 9. Notice that in Figure 2, if the subtasks τ_{11} and τ_{12} were executed sequentially instead of in parrallel, the makespan would be one time unit longer, in this case 10 instead of 9.

This is a key measurement when dealing with DAG tasks (see Section III) and it will be the main efficacy criteria when comparing the machine learning model with state-of-the-art heuristics and ILP (see Section IV).

E. Acceptance ratio

When dealing with several independent DAG tasks or tasksets, the acceptance ratio is often used to measure the performance of a scheduling algorithm (see Section III). It consists of looking at a number of generated tasksets (or DAG tasks) and calculating the amount of schedulable (i.e., the schedule produced doesn't lead to a deadline miss) tasksets compared to the

total amount of taskets. The resulting percentage is the acceptance ratio and the closer it gets to 100% for a scheduling algorithm, the better the scheduling algorithm.

This concept is also used as a measurement, to assert the efficiency of scheduling algorithms when considering independent tasks (see Section III).

F. Optimality

A scheduling algorithm S is said to be optimal when the following condition is true: for every taskset Ω, if there exists a scheduling algorithm S' so that Ω is feasible by S', then Ω is also feasible by S. Where *feasible*, means that, using the schedule generated by S, all the tasks in the taskset will finish executing before their deadlines.

This concept is used in the literature, mainly for independent tasks scheduling (see Section III).

G. Approximation ratio

The approximation ratio is the comparison between the average number of processors required by a scheduling algorithm to make a random taskset feasible and the average number of processors needed by the theoretically optimal scheduling algorithm for the same taskset.

It is a way of measuring scheduling algorithms, especially when considering independent tasks (see Section III).

While the acceptance and approximation ratio are used to measure the performance of scheduling alrogithms for independent tasks, the makespan is only used for DAG tasks and tasksets representing chain of events.

III. Related Works

[**?**] is another good paper.

A. Systematic Literature Review process

The SLR process [**?**].

Scoping The following research questions were used to conduct this SLR.

- RQ1
- RQ2

Extant literature does not sufficiently answer this question (provide a summary of current secondary studies).

Planning *involved breaking down the RQs into individual search terms.* We identified the following search terms (concepts) which were combined in the following way. Some practice search runs were used to ensure that the search terms were correct and yielded correct results. Preliminary inclusion and exclusion criteria were ....

It was decided that search results would be recorded by the use of ... (provide details).

Searching

Searching was conducted on the following databases, and XYZ number of results were returned.

Screening and Eligibility Prune the list to identify the final list (max 20 papers).

B. Findings of the Literature Review

The works reviewed were compared on the following metrics.

- **Metric 1**: justification
- **Metric 2**: justification
- ... (add as many as needed)

A comparison of the works was carried out and the overall results are illustrated in Table I.

Provide a summary of what the results say.

Show how your work is novel.

TABLE I
SYSTEMATIC LITERATURE REVIEW RESULTS (GENERATE TABLES FROM WWW.TABLESGENERATOR.COM)

	Metric 1	Metric 2	Metric 3	Metric 4
[Work 1]				
[Work 2]				
[Work 3]				
[Work 4]				

IV. Research Methodology

This research could have been conducted using alternative methodologies. Provide a summary of the best fit methodologies, and their relative strengths and weaknesses (max 2-3 paras).

The "name of methodology" was chosen to conduct this research. Give details about how this methodology was adapted for your project (max 2-3 paras).

Include a clear plan with objectives and outcomes (gantt chart)

V. Contribution 1

Oh by the way, [**?**] did some great work.

Give an overall summary of the steps involved.

VI. Contribution 2

Give an overall summary of the steps involved.

VII. Experimental Results

Set up: what experiments/benchmarks were chosen?
Execution of results: how were the experiments conducted
Data: what was found to have happened?
Synthesis: what does the data mean?
Relevance: how does this work compare to others, and to what extent does it answer the RQs
Limitations:

VIII. Conclusions and Future Works

Acknowledgement

For referencing in LaTeX, check out: https://texblog.org/2014/04/22/using-google-scholar-to-download-bibtex-citations/

References

[1] H. Kopetz and G. Bauer, "The time-triggered architecture," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 112–126, 2003.

[2] H. Kopetz, "The time-triggered model of computation," in *Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No. 98CB36279)*. IEEE, 1998, pp. 168–177.

[3] C. Maiza, H. Rihani, J. M. Rivas, J. Goossens, S. Altmeyer, and R. I. Davis, "A survey of timing verification techniques for multi-core real-time systems," *ACM Computing Surveys (CSUR)*, vol. 52, no. 3, pp. 1–38, 2019.

[4] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46–61, 1973.

[5] T. A. Henzinger, B. Horowitz, and C. M. Kirsch, "Giotto: A time-triggered language for embedded programming," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 84–99, 2003.

[6] T. A. Henzinger, C. M. Kirsch, E. R. Marques, and A. Sokolova, "Distributed, modular htl," in *2009 30th IEEE Real-Time Systems Symposium*. IEEE, 2009, pp. 171–180.

[7] K. Nagalakshmi and N. Gomathi, "The impact of interference due to resource contention in multicore platform for safety-critical avionics systems," *Int. J. Res. Eng. Appl. Manage.*, vol. 2, no. 8, pp. 39–48, 2016.

[8] C. M. Kirsch and A. Sokolova, "The logical execution time paradigm," *Advances in Real-Time Systems*, pp. 103–120, 2012.

[9] A. Biondi and M. Di Natale, "Achieving predictable multicore execution of automotive applications using the let paradigm. in 2018 ieee real-time and embedded technology and applications symposium (rtas)," 2018.

[10] K.-B. Gemlau, L. Köhler, R. Ernst, and S. Quinton, "System-level logical execution time: Augmenting the logical execution time paradigm for distributed real-time automotive software," *ACM Transactions on Cyber-Physical Systems*, vol. 5, no. 2, pp. 1–27, 2021.

[11] J. Du and J. Y.-T. Leung, "Complexity of scheduling parallel task systems," *SIAM Journal on Discrete Mathematics*, vol. 2, no. 4, pp. 473–487, 1989.