Scheduling time-triggered tasks in multicore real-time systems: a machine 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 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 model is the Directed Acyclic Graph (DAG) task model which arises when a time-triggered task can be parallelized into subtasks which are the nodes of the graph. Those nodes have dependency constraints which are modeled by the directed edges

[1]https://www.autosar.org/

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

between the nodes. The DAG task model is used to model tasks that are parallelizable[5], fitting the ever-increasing multicore architectures found in today's real-time systems.

Given that the problem of scheduling independent tasks or dependency-constrained groups of jobs (i.e., DAGs) is NP-hard[3][6] [7], people have resorted to either heuristics to partially solve the problem, or the optimal but not scalable Integer Linear Programming (ILP) method.

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. The research questions are:

- RQ1 What is the current state-of-the-Art for DAG tasks scheduling ?
- RQ2 What machine learning techniques are used for DAG task scheduling ?
- RQ3 Can machine learning be a better solution to schedule DAG tasks ?
  - RQ3.1 Can a machine learning solution compare to state-of-the art heuristics for scheduling Directed Acyclic Graph tasks ?
  - RQ3.2 Can a machine learning solution compare to an ILP solution while being more scalable ?

To achieve this, the background section will introduce various technical terms, concepts, and fundamental algorithms. Following this, a systematic literature review will be conducted to address RQ1 and RQ2, and finally, the artifact and experimental design, results, and conclusion will be presented to answer RQ3.

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).

[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

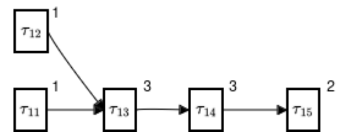

Fig. 1. DAG task τ_1. The nodes are the subtasks, the edges of the graph represent the precedence constraints between each subtasks and the worst-case execution time (wcet) of each subtask written as an exponent.

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 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 and considering their utilization bound(see Section III), or used as a parameter to

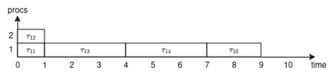

Fig. 2. Example of schedule for τ_1. The y axis represents the number of processors that are not idle. For the first two subtasks there are two processors active and for the rest there is only one active processor.

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. Capacity augmentation bound

Another measurement used when scheduling event-chains, or DAG, tasks is the capacity bounds, or capacity augmentation bound, which compares resource use to an theoretically optimal scheduling algorithm. It can also be used as a simple schedulability test. The matehmatical definition for a scheduling algorithm S of its capacity augmentation bound β is that, for any chain of events represented by a DAG G satisfying the following condition :

$$\beta \times U \leq m \land \beta \times len(G) \leq D, \tag{2}$$

the associated DAG is schedulable by S. Here, $len(G)$ is the length of the longest path, in terms of WCETs, in the graph G, D is the DAG task's deadline, m the number of processors and U is the utilization factor of the DAG task (with the WCET of the DAG being the sum of all tasks' WCET). As you can see, the lower β is, the better the scheduling algorithm.
This metric is used for DAG tasks or event-chains of tasks.

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. 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 metric, to assert the efficiency of scheduling algorithms when considering independent tasks (see Section III).

While the acceptance ratio, also called system schedulability, is used to measure the performance of scheduling alrogithms for independent tasks, the makespan and the capacity bound are only used for DAG tasks and tasksets representing chain of events.

H. RM and EDF scheduling

When designing a scheduling algorithm, the key decision involves determining which task should execute first when two or more independent tasks are ready to execute. This requires assigning each task a priority. [4] introduced two heuristics for this purpose: Rate Monotonic (RM) and Earliest Deadline First (EDF).

The RM algorithm is a fixed-priority scheduling algorithm, meaning that the priority of each task is known before execution begins. RM assigns the highest priority to tasks with the minimum execution rate, i.e., $\frac{C_k}{T_k}$, and is considered optimal for assigning fixed priorities to tasks. In contrast, EDF assigns priorities dynamically by selecting tasks based on which one has the earliest absolute deadline.

Figure 3 illustrates the difference between the two algorithms by scheduling the same two tasks, τ_1 and τ_2. τ_1 has a worst-case execution time of 0.5 time units and a period of 2 time units, while τ_2 has a worst-case execution time of 2 time units and a period of 3 time units. These are examples of implicit deadline tasks, where the relative deadline equals the end of their execution period.

Although EDF calculates each priority at runtime, it is optimal for uniprocessor scheduling and has a theoretical utilization bound of 1, which is the maximum possible for a feasible taskset on a single processor. RM, on the other hand, has a much lower utilization bound than EDF. While one might argue that RM introduces less runtime overhead and is therefore more practical, it has been shown that RM leads to more task preemptions (interrupting the execution of a task, as seen at times 2 and 4 for task τ_2 in Figure 3.a). This, combined with its lower utilization bound and non-optimality, makes EDF perform better than RM[8].

Although [4]'s work focused on uniprocessor systems, the proposed algorithms have also been applied to multi-processor scheduling. For example, Global EDF (GEDF) can be used on

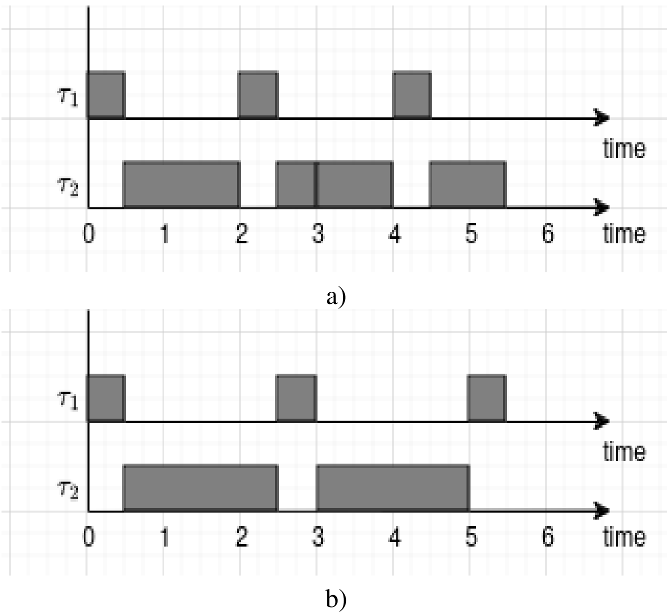

Fig. 3. Schedules of τ_1 and τ_2 using Rate Monotonic (a) and Earliest Deadline First (b) heuristics.

multi-core systems when allowing task migrations and Partitioned EDF (PEDF) is used when forbidding task migrations (The RM equivalents also exist).

III. Related Works

A. Systematic Literature Review process

This SLR aims at tackling RQ1 and RQ2. More precisely, the following research questions will be answered:

RQ1 What is the current state-of-the-Art for DAG tasks scheduling ?
RQ2 What machine learning techniques are used for DAG task scheduling ?

It will also be shown how the literature doesn't provide a complete answer to RQ3, hence the contributions of this paper.

From these research questions, several concepts have been isolated, namely, time-triggered tasks, the nature of the system (real-time multicore system), the scheduling of tasks, DAG tasks, and machine learning. The recording of the search results were done using the BibTeX LateX plugin combine with the google scholar "cite" feature.

Searching was conducted using the IEEE and ACM databases. According to the concepts identified above, the keyword chain used for searching was "("real-time" OR "real time") AND "system" AND ("time-triggered" OR "time triggered" OR "DAG" OR "Directed Acyclic Graph" OR "event chain" OR "event-chain") AND "task" AND ("scheduling" OR "scheduler" OR "schedule") AND ("multi-processors" OR "multi-cores" OR "multi processors" OR "multi cores" OR "multi-processor" OR "multi processor" OR "multi-core" OR "multi core")". EC1 EC2 EC3 1,259 -¿ 515 -¿ 403 -¿ 101 -¿ 19 –¿ 17 The search produced 1,259 results on the IEEE database which was reduced tp 515 papers when considering only articles published in the past 5 years.

Then the following exclusion criterias were used to filter out the rest of the articles, bringing the number of papers down to 19 (see Figure 4).

EC1 Not focusing on homogeneous multicores and hard RTS
"Heterogeneous" not in the title nor the abstract.
"mixed critical*" not in the title nor the abstract

EC2 Not focusing on scheduling
"scheduling" or "scheduler" or "schedule" in the title
"energy" not in the title
Focus on conference and journal papers

EC3 Not focusing on real-time systems, not proposing a scheduling algorithm, not using DAG or event-chain tasks.

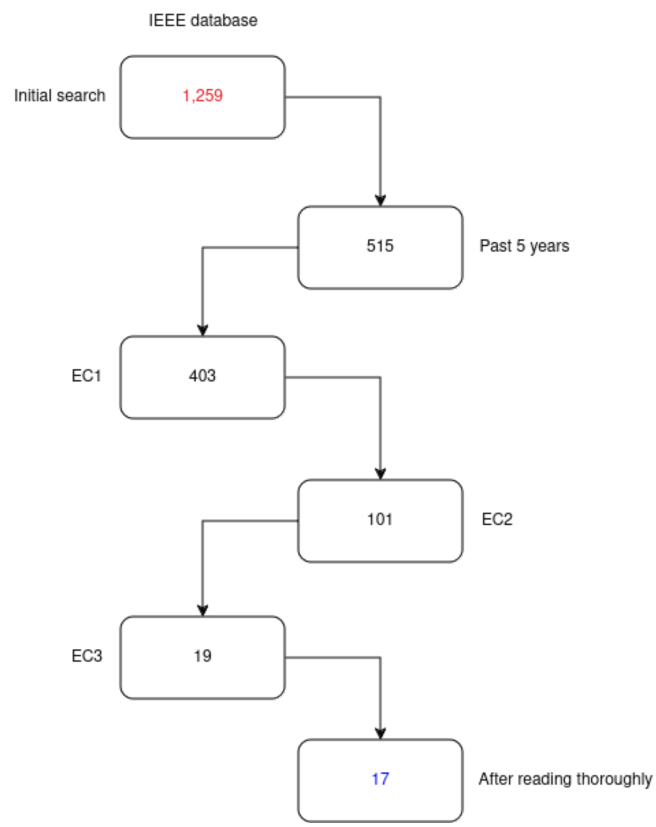

Fig. 4. SLR process diagram. EC stands for Exclusion Criteria, which are listed above.

B. Findings of the Literature Review

1) Non-Machine Learning techniques:

Guan et al. [9] use fluid scheduling to schedule multiple DAG tasks on a multicore system. Fluid scheduling has been used in previous work for independent time-triggered task scheduling[5][10] but very few consider DAG tasks. Fluid-scheduling is known for producing optimal scheduling algorithms. Their method decomposes a DAG task into several sequential segments in which the subtasks will execute according to the fluid scheduling model. Although their algorithm significantly outperforms existing algorithms, the main limitation that is common to all fluid-based scheduling algorithm is the runtime overhead induced by the fluid-scheduling model. Although authors in [9] briefly explain how to transform their scheduling algorithm to a non-fluid one for practical implementation, they do not evaluate the overhead caused by the frequent task migrations and preemptions. This overhead can lead to deadline misses on systems where the migration cost is high which is why it is important, especially with fluid-based algorithms, to consider this overhead[11]. Also, their algorithm only considers DAG task with implicit deadlines (D = T) which makes the response-time analysis simpler but to the cost of not covering all types of tasks that can execute on real-time systems.

As a follow up, Guan et al. [12] extend the fluid scheduling algorithm in [9] to constrained and arbitrary deadline tasks, especially focusing on DAG tasks with a deadline greater than their period, thus generalizing their previous fluid algorithm to all types of periodic tasks. Their main contributions are their new scheduling algorithm that performs better than existing methods in terms of acceptance ratio, and producing the first theoretical capacity bound for DAG tasks with deadlines greater than their periods. However, the authors still don't provide any evaluation on the amount of runtime overhead their scheduling algorithm implementation produces which generally lowers the actual acceptance ratio of the algorithm.

Instead of considering fluid-scheduling, a popular scheduling method is federated scheduling. Federated scheduling is based on the idea of assigning heavy tasks ($U > 1$) to multiple cores for the whole duration of the tasks' executions, and assigning light tasks ($U \leq 1$) to execute on cores that have not been assigned a heavy task. Although it is popular, it suffers from a resource wasting problem, especially when the difference between the critical path's length and the deadline is small, which many papers aim at solving [13] [14] [15] [16] [17] [18].

Jiang et al. [14], for instance, consider federated scheduling and GEDF and introduces a better metric called the util-tensity bound that extends the concept of capacity bound to have a better schedulability test. Based on this newly derived bound, the authors propose an extension to the classic federated algorithm, with very low tensity tasks being scheduled with GEDF, tasks with high-utilization and relatively high tensities are scheduled using the classic federated scheduling and low utilization tasks with relatively high tensities are scheduled using partitioned-EDF. Their algorithm, based on their newly derived bound, effectively improves the system schedulability of DAG tasks and reduces the resource wasting problem of federated scheduling. The main limitation of this paper is that they only consider GEDF for their util-tensity bound and also only consider implicit deadline DAG tasks which doesn't represent tasks that need to have a deadline lower than their period, for example.

This problem of resource wasting in federated scheduling is also tackled by Kobayashi and Azumi [17] where they propose a federated and bundled-based scheduling algorithm which enhances the schedulability of DAG tasks compared to existing federated scheduling algorithms. Their method consists of using federated scheduling for tasks with high critical path to deadline ratio and bundled scheduling for tasks with low critical path to deadline ratio. Unfortunately, this paper only looks at 3 DAG tasks to evaluate their algorithm which is a really small amount and is not representative of the different DAG tasks that can exist and leaves space for observational bias.

Jiang et al. [15] take another approach by proposing a virtually-federated scheduling algorithm that leverages the advantages of federated scheduling while improving the acceptance ratio for DAG tasks, outperforming existing algorithms. Their approach consists of adding a virtual layer of processors, on top of physical processors, and apply their federated-based scheduling algorithm on those virtual processors, thus enabling tasks to share a physical processor even though they are assigned to different virtual processors. The main drawback in [15] is that the authors only consider the heavy tasks (i.e., $U > 1$) and do not take the light tasks into account, meaning it doesn't consider tasks that only need one processor to execute.

To fix this limitation, Jiang et al. [16] extend their previous work[15] so that it considers both heavy and light tasks. The resulting virtually federated scheduling algorithm clearly outperforms any other federated-based scheduling algorithms in terms of acceptance ratio. However, they still only consider implicit or constrained deadline tasks and they don't provide any evaluation of the run time overhead their algorithm might induce, to compare with algorithms currently used in real-time systems, such as GEDF.

Guan et al. [13] consider arbitrary deadline tasks and especially DAG tasks that have a deadline that is greater than their period. They introduce a new federated scheduling algorithm that takes those type of tasks into account and compare it to existing global or federated scheduling approaches for arbitrary deadline tasks, significantly outperforming most of them in terms of acceptance ratio. Their approach consists of using this new proposed aglgorithm for heavy tasks that have a deadline bigger than their period, then using classic federated scheduling for the heavy tasks with a constrained deadline, and finally using EDF-First-Fit (EDF-FF) for the light tasks. Although the fluid-based method in [12] outperforms this new federated scheduling algorithm, the impracticality of fluid-based algorithm makes this algorithm the current best, in terms of acceptance ratio, for dealing with arbitrary deadlines. The main limitation of this work is that it doesn't tackle the resource wasting problem that classical federated scheduling, or their new algorithm, has or can potentially have, but only focuses on providing an algorithm for arbitrary deadline tasks.

For constrained deadline DAG tasks, He et al. [18] propose a federated-based scheduling algorithm that outperforms on average by more than 18% previous SOTA[16] in terms of acceptance ratio, making this work the current SOTA for constrained deadline multi-DAG scheduling. Their approach uses the notion of degree of parallelism, which they define rigorously, to improve the classic federated scheduling way of choosing the number of cores to assign each heavy tasks. They also propose a new response-time bound for constrained deadline DAG tasks based on this defined notion. Although their method clearly stands out, they don't consider intra-task scheduling at all when their motivation came from the notion of degree of parallelism being used but wrongly defined in previous intra-task scheduling work[19][20].

Federated scheduling isn't the only method used, Jiang et al. [21], for instance, propose a decomposition-based approach to schedule multi-DAG tasks as well as a metric for testing the schedulability of tasks. Their decomposition strategy proves to be the most efficient , according to the defined metric, and the scheduling algorithm derived from it shows promising results in terms of acceptance ratio. Their decomposition strategy basically works by first definin execution segments and then then assigning subtasks to those segments using the laxity[4] of those subtasks so that no segments are overloaded with workload. The main limitation of this work is that they only look at GEDF variants for priority assignment and do not evaluate their decomposition method using other scheduling heuristics for multi-DAGs. Most of the articles presented up to now tackle inter-task scheduling, not considering the intra-task execution schedule.

Indeed, intra-task scheduling[22][23] [24][25][26] [27] is often tackled as a separate problem due to the dependency constraints.

Xiao et al. [23], for instance, introduce a scheduling algorithm , 'MAS', that shortens the makespan of periodic DAG tasks compared to the classic EDF dynamic priorty scheduling technique. Their algorithm is based on a clustering approach, combined with a technique called task duplication, and evaluate their results on an actual simulation object for real-time scheduling. Unfortunately, their evaluation is only based on a single DAG task and they only compare their algorithm with EDF. Their algorithm also shows a scalability problem compared to other existing algorithm with comparable results.

A scalable way of scheduling sub-tasks of a DAG is looking at priority-list scheduling which He et al. [22] use to propose an algorithm that effectively outperforms other intra-task DAG scheduling algorithms in terms of makespan. Their priority assignment algorithm is based on the length of the paths passing through each vertex. The longer the maximum length, the higher the priority. This effectively takes advantage of the inner graph structure to optimize the intra-task execution order, which is something that hasn't been done before this work. Although the authors compare their results with existing scheduling heuristics, they do not consider the, mathematically optimal, ILP method to compare their makespan results with the mathematically minimum makespan.

This priority-list scheduling approach is also used by [19], extending their previous work[20] which develop a priority assignment based on the critical-path execution first (CPEF) concept, effectively outperforming He et al. [22] in terms of makespan and providing a federated-based multi-DAG scheduling algorithm, compatible with this new priority-list scheduling algorithm. Their multi-DAG scheduling algorithms also outpeforms the multi-DAG algorithm used in [22]. Their method uses the vertices in the critical path as producers of workload for the parallelizable vertices to consume. For multi-DAG they look at assigning processors to DAG tasks, like in federated scheduling, using a parallelism-aware workload distribution model that uses their Concurrent Producer Consumer (CPC) model to assign cores while minimizing the inter-task workload interference. One limitation of their work is that, although they consider constrained deadlines for the response-time analysis, they only use implicit deadline DAG tasks for evaluating the system schedulability of their multi-DAG scheduling algorithm.

Those intra-task scheduling papers only consider the theoretical makespan when task migration and preemption is instantanious, but they don't take the runtime overhead into account. One major factor in the runtime overhead is the communication delays between each subtasks of a DAG task, which are rarely considered. To that end, Shi et al. [24] propose an extension of the DAG task model to add execution groups that bind groups of subtasks to a single physical core, thus reducing inter-core communication which can cause major communication delays depending on the topolgy of the system. They also introduce a scheduling algorithm and compare the makespan of their approach to existing methods such as federated scheduling and critical path-based scheduling. Their method shows comparable results while minimizing the communication overheads. However, the evaluation is only done on 100 generated DAG tasks which is a small sample size compared to the other papers[19][22], exposing the experiment to obersvational biases. Also, although they introduce a way to extend their approach to schedule multiple DAGs, they do not provide any evaluation of that.

2) Machine Learning techniques:

Only a few papers consider machine learning to solve the scheduling problem.

Lee et al. [26], for instance, design a deep reinforcement learning (DRL) model called GoSu which takes a DAG task as input and outputs a priority list of the DAG's subtasks. The makespan resulting from this priority-list is then compared to the results in [19] and [22] and the DRL model proposed in [26] outperforms them by up to 3%. The model is comprised of a graph convolutional network layer to encode the graph structure information, and a sequential decoder layer based on the attention-mechanism, which produces a priority list of the vertices. The reinforcement learning uses the makespan as the reward to minimize and the REINFORCE algorithm is used to find the best policy. Although the time it takes for the model to run is measured, no comparison with the ILP method is done and there is no evaluation of the scalability of the model when

[4]Laxity is the gap between the total execution time of a task (potentially comprising the I/O delays) and its deadline.

increasing the amount of cores in the system or the amount of subtasks in a DAG task. But Lee et al. [26] isn't the only work considering DRL as a method for DAG intra-task scheduling.

Indeed, Zhao et al. [25] also use the DRL approach to tackle the intra-task scheduling problem and compare their results to the makespan obtained by solving the equivalent ILP problem. Their model achieves up to 75% of makespan reduction compared to ILP, that is, the makespan produced by the ILP method is 25% smaller than the makespan produced by their DRL model, which is a relatively good performance as the ILP approach gives the mathematically minimum makespan. The more important result, however, is that when you increase the subtasks in the DAG tasks, the ILP method explodes in terms of computing time when the DRL approach gives a result in a relativley short time, making it scalable, unlike ILP. The model uses a combination of a graph neural network with attention layers to better capture the structure and dependency information of the DAG task. The makespan is also used for the reward function but Soft Actor Critic algorithm is used for training the model. Unfortunately, the paper doesn't provide an evaluation when increasing the number of cores in the system and also doesn't compare their model to SOTA heuristics, which also aim to approximate the optimal solution given by the ILP method.

Guan et al. [27] focus more on optimizing communication delays by looking at the real-time simulation system FRTDS and using the I/O usage and ram allocation to construct a cost function, which is then used as a reward for their proposed DRL model to schedule DAG's subtasks. The cost is divided into a current cost, what we know, and the future cost which is predicted using the subtask's successors. The model performs better, in terms of makespan, than existing scheduling algorithms implemented on the FRTDS platform but because of the RL process accumulating the previous subtask's execution as experience to learn, the method uses a lot of memory which affects the speed of execution.

Table I gives a summary of the findings.

According to these findings, the current state-of-the-Art for DAG task scheduling (RQ1) seems to be a federated-based scheduling approach for inter-task scheduling and a global priority-list approach for intra-task scheduling [18][19].

In terms of machine learning techniques (RQ2), every article found uses deep reinforcement learning for scheduling DAG tasks. More specifically, the model proposed is often a combination of a kind of graph neural network and attention layers to produce a priority-list of the subtasks of the DAG task.

Multiple limitations were found for each paper. Notably, the type of task that is considered which mostly is constrained or implicit deadline tasks and arbitrary is rarely considered due to the complexity it adds to the scheduling analysis. Also, there hasn't been any use of supervise machine learning for real-time task scheduling and only one paper looked at comparing a machine learning solution to the optimal schedule provided by the ILP methodology.

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] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, "Proportionate progress: A notion of fairness in resource allocation," in *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, 1993, pp. 345–354.

[6] 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.

TABLE I
SLR SUMMARY TABLE

Ref.	Motivation	Contribution(s)	Limitation(s)	Methodology Summary
[9]	DAG tasks scheduling is getting more popular and fluid scheduling performs great theoretically	Provide a DAG-fluid scheduling algorithm that performs way better in terms of acceptance ratio then previous algorithms	Fluid scheduling is unpractical and introduces a lot of overhead and task migrations, also only for implicit deadlines tasks	fluid-based algorithm where it decomposes a DAG's subtasks into multiple sequential segments
[22]	DAGs are popular but no one looked at the intra-task execution order to leverage the graph structure	proposes a priority list scheduling algorithm for a single DAG task which performs better than SOTA in terms of makespan	no comparison with optimal priority assignment algorithms / optimal schedules.	uses the length (in terms of wcets) of each paths passing through the current vertex to assign the priority to the current vertex, the higher the length, the higher priority
[17]	Federated scheduling for DAG tasks is has proved efficient but for tasks where the difference between the critical path and the deadline is small, it can lead to over-allocating cores.	proposed a fedrated and bundled-based scheduling algorithm to avoid this problem and enhanced the schedulability of DAG tasks using their algorithms	They only compare their method with an example of a dag task set comprised of 3 dag tasks.	Uses federated scheduling for tasks with high critical path to deadline ratio and bundled scheduling for tasks with low critical path to deadline ratio.
[23]	DAG task scheduling is NP-hard so one can only approximate the optimal algorithm (when considering polynomial timed algorithms) and not a lot has been done on scheduling parrallel reccuring tasks	Introduces a scheduling algorithm 'MAS' that shortens the makespan of recurring DAG tasks compared to EDF	only compares EDF and MAS using one example of a DAG task for makespans and also only compares with EDF. Even though MAS shortens the makespan, it is less scalable than comparable algorithms.	The MAS algorithm integrates clustering, scheduling, and task duplication techniques and is evaluated using a TMS320C6678 simulation for measurements closely aligned with real-world results.
[14]	The capacity-bound measures performance and tests schedulability for DAG scheduling, but it may exclude schedulable tasks by using the same bound for normalized utilization and tensity.	Introduces a new bound called the util-tensity bound which proves to be a better schedulability test for GEDF with federated scheduling.	only looks at GEDF with federated scheduling and not other scheduling algorithms.	Uses GEDF for low-tensity tasks, federated scheduling for high-utilization, high-tensity tasks, and partitioned EDF for low-utilization, high-tensity tasks.
[21]	Decomposition-based scheduling can improve schedulability for DAG task scheduling but can also worsen it. It is, along with global scheduling, one of the main method to schedule DAG tasks.	Developed an efficient decomposition strategy and schedulability test. The resulting GEDF-based scheduling algorithm shows promising acceptance ratios.	Only looks at GEDF variants which is based on the EDF heuristics for priority assignments.	The decomposition works by first defining execution segments and then assigning subtasks to those segments based on their laxity so that there are no segments overloaded with workload.
[18]	The notion of degree of parallelism has been used for DAG task scheduling but lacks a clear definition in the research community.	Proposes a new response-time bound for DAG tasks as well as a new scheduling algorithm based on federated scheduling that outperforms the SOTA by more than 18% on average	They don't say which intra-task scheduling algorithm is used (just that it's work-conserving) and they don't consider intra-task scheduling.	They modify federated scheduling by optimizing core allocation for heavy tasks based on the degree of parallelism.
[24]	Many DAG intra-task scheduling algorithms overlook inter-core communication delays. In robotics and automotive applications, grouping subtasks on a single processor can reduce these delays by using the L1 cache.	Extend DAG to EG-DAG to bind subtasks to a single core, reducing communication delays. Their new scheduling algorithm shows similar performance to existing methods with lower overhead.	The evaluation has been done using only 100 DAG tasks which is quite low to cover all different types of DAG tasks. They propose a way to schedule multiple DAGs but do not offer evaluation results for that.	They use list scheduling with one list per execution group and use worst-fit heuristic to map the execution groups to the processors.
[13]	Federated scheduling works well for constrained deadlines but struggles with arbitrary deadline DAG tasks, especially with long WCET. Allowing job migration leads to pessimistic schedulability analysis.	Propose a new federated scheduling algorithm for arbitrary deadline DAG tasks with long WCETs. It outperforms other algorithms in acceptance ratio.	Doesn't tackle the problem of resource wasting when using federated scheduling or their new version of it.	The new federated algorithm addresses tasks with deadlines longer than periods and high densities, using standard federated scheduling for tasks with shorter deadlines and EDF-FF for low-density tasks.
[25]	The NP-hard nature of DAG multicore scheduling makes ILP solutions time-consuming, leading researchers to explore scalable heuristics.	Uses Deep Reinforcement Learning to learn an optimal scheduling policy for DAG tasks, comparing it to the ILP method.	Doesn't compare the machine learning method with SOTA heuristics, but only compares with ILP.	They use a Graph Neural Network with attention layers to capture structure and dependencies, maximizing the negative makespan as the reward function.
[28]	Current methods for allocating shared resources in multi-core real-time systems use static analysis or heuristics, which may not cover all scenarios, leading to higher WCETs and reduced schedulability.	They use Deep Reinforcement learning to propose a holistic scheduling and allocation framework and their model shows better schedulability than existing methods.	Only considers independent periodic tasks and also only considers even-EDF and even-RM when comparing schedulability performance.	The platform has an LLC architecture with a shared memory bus. The DRL model uses MLP and proximal policy optimization to generate time-triggered schedules and memory allocations.
[19]	A previous work done by Zhao et al. [20] introduced a fixed-priority scheduling algorithm for DAG intra-task scheduling which performed better than SOTA but didn't extend the method to multi-DAG scheduling.	"Extends the CPC model from Zhao et al. [20] to multi-DAG scheduling with a Parallelism-aware workload distribution algorithm, improving system schedulability and outperforming existing methods.	Considers constrained deadlines DAG tasks for the analysis but only consider implicit deadlines DAG tasks for the experiment evaluation.	Uses a critical path first model, prioritizing nodes on the critical path and allocating parallel execution time to subtasks. For multi-DAG scheduling, they use a federated-like approach based on the degree of parallelism of the DAG tasks.

[26]	Several heuristics for DAG intra-task scheduling have been used but no scalable optimal scheduling algorithm exists.	Propose a DRL-based model for computing intra-task priorities in DAG scheduling, outperforming SOTA by up to 3% in makespan.	No ILP method comparison for minimum makespan is made, and scalability of the GoSu DRL model with more cores/subtasks isn't shown.	The network uses a Graph Convolutional Network and an attention-based decoder to generate a priority list, trained with REINFORCE using negative makespan as the reward.
[16]	Virtual scheduling, using threads as virtual processors, has been considered in the past but never for DAG task scheduling. The similar federaded scheduling method suffers from resource wasting.	Use the concept of virtual processors to provide a virtually-federated scheduling algorithm which significantly outperforms other federated scheduling methods in terms of acceptance ratio.	Only considers implicit and constrained deadline DAG tasks and doesn't consider the running time overhead that the proposed method induces.	Introduce active and passive virtual processors (VPs) per core. The active VP executes high-priority tasks, while unused active VP time is treated as a passive VP for low or high-priority tasks.
[12]	Most DAG studies use implicit deadlines, with few focusing on arbitrary deadlines, especially when deadlines exceed the task period. Fluid scheduling shows promise but is only applied to implicit deadlines.	Propose a fluid scheduling algorithm for constrained and arbitrary deadline DAG tasks, introducing the first capacity bound for deadlines longer than periods. It outperforms SOTA in acceptance ratio (as of 2022).	As for every fluid scheduling based algorithm the issue of runtime overhead is not entirely considered as they don't evaluate this metric.	They first decompose each DAG task into segments of sequential tasks and then assign execution rates to each tasks or threads. Those two steps aim at producing a fluid schedule so that it appears as though each DAG task is continuously running on the cores.
[27]	Most scheduling algorithm that consider resource use consider it as constraints rather than considering them as part of the scheduling decision process.	Propose a DRL-based algorithm for task scheduling on the FRTDS simulation system, outperforming existing algorithms in makespan for single DAG tasks.	The reinforcement learning algorithm uses the previous tasks' execution as experience which implies a lot of memory usage, thus affecting the speed of execution.	It uses I/O and RAM allocation to create a cost function as the RL reward, combining current costs with future costs predicted from successor subtasks.
[15]	Federated scheduling has shown great potential for scheduling DAG tasks but suffers from a resource wasting problem which has been addressed but to a limited extent.	Propose a virtually-federated scheduling algorithm that reduces resource wastage while maintaining federated scheduling's benefits, and outperforms existing algorithms in acceptance ratio.	Only considers heavy tasks and constrained deadline DAG tasks.	Introduce Active and Passive VPs per core. Active VPs handle primary tasks and use excess capacity for Passive VPs. Allocation is based on deadlines, critical path length, and task usefulness.

[7] J. Ullman, "Np-complete scheduling problems," *Journal of Computer and System Sciences*, vol. 10, no. 3, pp. 384–393, 1975.

[8] G. C. Buttazzo, "Rate monotonic vs. edf: Judgment day," *Real-Time Systems*, vol. 29, pp. 5–26, 2005.

[9] F. Guan, J. Qiao, and Y. Han, "Dag-fluid: A real-time scheduling algorithm for dags," *IEEE Transactions on Computers*, vol. 70, no. 3, pp. 471–482, 2021.

[10] H. Cho, B. Ravindran, and E. D. Jensen, "An optimal real-time scheduling algorithm for multiprocessors," in *2006 27th IEEE International Real-Time Systems Symposium (RTSS'06)*. IEEE, 2006, pp. 101–110.

[11] A. Block and J. Anderson, "Accuracy versus migration overhead in real-time multiprocessor reweighting algorithms," in *12th International Conference on Parallel and Distributed Systems - (ICPADS'06)*, vol. 1, 2006, pp. 10 pp.–.

[12] F. Guan, L. Peng, and J. Qiao, "A fluid scheduling algorithm for dag tasks with constrained or arbitrary deadlines," *IEEE Transactions on Computers*, vol. 71, no. 8, pp. 1860–1873, 2022.

[13] ——, "A new federated scheduling algorithm for arbitrary-deadline dag tasks," *IEEE Transactions on Computers*, vol. 72, no. 8, pp. 2264–2277, 2023.

[14] X. Jiang, J. Sun, Y. Tang, and N. Guan, "Utilization-tensity bound for real-time dag tasks under global edf scheduling," *IEEE Transactions on Computers*, vol. 69, no. 1, pp. 39–50, 2020.

[15] X. Jiang, N. Guan, H. Liang, Y. Tang, L. Qiao, and Y. Wang, "Virtually-federated scheduling of parallel real-time tasks," in *2021 IEEE Real-Time Systems Symposium (RTSS)*, 2021, pp. 482–494.

[16] X. Jiang, H. Liang, N. Guan, Y. Tang, L. Qiao, and Y. Wang, "Scheduling parallel real-time tasks on virtual processors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 1, pp. 33–47, 2023.

[17] T. Kobayashi and T. Azumi, "Work-in-progress: Federated and bundled-based dag scheduling," in *2023 IEEE Real-Time Systems Symposium (RTSS)*, 2023, pp. 443–446.

[18] Q. He, N. Guan, M. Lv, and Z. Gu, "On the degree of parallelism in real-time scheduling of dag tasks," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2023, pp. 1–6.

[19] S. Zhao, X. Dai, and I. Bate, "Dag scheduling and analysis on multi-core systems by modelling parallelism and dependency," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 4019–4038, 2022.

[20] S. Zhao, X. Dai, I. Bate, A. Burns, and W. Chang, "Dag scheduling and analysis on multiprocessor systems: Exploitation of parallelism and dependency," in *2020 IEEE Real-Time Systems Symposium (RTSS)*, 2020, pp. 128–140.

[21] X. Jiang, N. Guan, X. Long, and H. Wan, "Decomposition-based real-time scheduling of parallel tasks on multicores platforms," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2319–2332, 2020.

[22] Q. He, x. jiang, N. Guan, and Z. Guo, "Intra-task priority assignment in real-time scheduling of dag tasks on multi-cores," *IEEE Transactions on Parallel and Distributed*

Systems, vol. 30, no. 10, pp. 2283–2295, 2019.

[23] S. Xiao, D. Li, and S. Wang, "Periodic task scheduling algorithm for homogeneous multi-core parallel processing system," in *2019 IEEE International Conference on Unmanned Systems (ICUS)*, 2019, pp. 710–713.

[24] J. Shi, M. Gtinzel, N. Ueter, G. v. der Bruggen, and J.-J. Chen, "Dag scheduling with execution groups," in *2024 IEEE 30th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2024, pp. 149–160.

[25] M. Zhao, L. Mo, J. Liu, J. Han, and D. Niu, "Gat-based deep reinforcement learning algorithm for real-time task scheduling on multicore platform," in *2024 36th Chinese Control and Decision Conference (CCDC)*, 2024, pp. 5674–5679.

[26] H. Lee, S. Cho, Y. Jang, J. Lee, and H. Woo, "A global dag task scheduler using deep reinforcement learning and graph convolution network," *IEEE Access*, vol. 9, pp. 158 548–158 561, 2021.

[27] Y. Guan, B. Zhang, and Z. Jin, "An frtds real-time simulation optimized task scheduling algorithm based on reinforcement learning," *IEEE Access*, vol. 8, pp. 155 797–155 810, 2020.

[28] Z. Xu, Y. Zhang, S. Zhao, G. Chen, H. Luo, and K. Huang, "Drl-based task scheduling and shared resource allocation for multi-core real-time systems," in *2023 IEEE 3rd International Conference on Intelligent Technology and Embedded Systems (ICITES)*, 2023, pp. 144–150.