Reference	Motivation	Contribution(s)	Limitation(s)	Methodology Summary
[3]	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
[5]	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
[12]	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.

[16]	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 combines techniques from clustering scheduling and task duplication algorithm and evaluates it on an actual simulation object (the TMS320C6678) so that the measurement are close to the real-life measurement you would get on the real system.
[7]	The use of multi-core systems can induce contentions because of shared memory / cache. This can lead to non-determinim and unpredictable behavior which violates the safety requirements of real-time systems, hence using LET tasks to fix those contentions but LET also suffers from additional execution time being added because of the implementation overhead.	Proposes a DAG LET tasks scheduling algorithm based that avoids contentions while reducing the running time overhead due to LET implementation	They are using a multiple-clusters, multi-core architecture to evaluate their scheduling algorithm but only consider one cluster.	Uses a minimum-laxity-first priority assignment for intra-tasks and Earliest Finish time (EFT) for assigning tasks to cores. Considers multi-rate dags and reducing the LET interval to decrease the makespan.

[10]	The capacity-bound is a bound used as a performance measurement but also as a schedulability test for DAG scheduling. However, it uses the same bound to bound both the normalized utilization and the tensity (critical path over the deadline) of a DAG task which can exclude DAG tasks that actually are schedulable but not according to this capacity-bound.	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.	The scheduling algorithm applied uses GEDF for very low tensity tasks, tasks with high utilization and relatively high tensities are scheduled using federated scheduling, and low utilization tasks with relatively high tensities are scheduled by partitioned EDF
[8]	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.	Develop a decomposition strategy as well as a metric / schedulability test and the decomposition strategy proves to be the most efficient one according to the defined metric. The scheduling algorithm derived from this decomposition strategy (using GEDF variants) shows promising results in terms of 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.

[6]	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.	Using the defined notion of degree of parallelism, they modify the federated scheduling approach by having a better way of choosing the number of cores for heavy tasks, which is based on the degree of parallelism of the heavy DAG task.
[15]	Although several DAG intra-task scheduling algorithm have been proposed in the literature, most of them ignore the communication delays, specifically the inter-core communications between the subtasks of a DAG tasks. In the Robotic / automotive industry, most communications can be done using the L1 cache of a processor, hence having a way to group subtasks into execution groups to execute on a single processor to remove inter-core communication delays.	Extend the DAG task model to EG-DAG (execution group dag) which binds groups of subtasks to a single physical core, thus reducing inter-core communication delays. They also introduce a scheduling algorithm and a wcrt for the algorithm while comparing the makespan to existing methods such as federated scheduling and critical-path based sub-tasks scheduling. Their method shows comparable performance while minimizing the communication overheads.	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.

[2]	Federated scheduling has shown great potential for constrained deadline tasks but for arbitrary deadline DAG tasks, especially those where the WCET is longer than the period, processor assignment is tricky and existing work have shown limitations by letting jobs migrate between the assigned processors, which produces a more pessimistic schedulability analysis.	Propose a new federated scheduling algorithm for arbitrary deadline DAG tasks with WCETs longer than their periods. Evaluate the proposed algorithm and comparing it to other scheduling algorithms, effectively outperforming most of them in terms of acceptance ratio	Doesn't tackle the problem of resource wasting when using federated scheduling or their new version of it.	The new federated algorithm is used when the deadlines are bigger than the periods, and the tasks have high densities (according to the classic federated scheduling algorithm). For the high densities tasks that have a deadline lower than their period, they use standard federated scheduling and for the low density tasks, they use EDF-FF.
[18]	The NP-hard aspect of DAG multi-core scheduling makes the optimal solutions (using ILP) time-consuming. Hence the researchers have looked at heuristic to have scalable solutions.	Uses Deep reinforcement learning to construct a model that attempts to learn the optimal (in terms of makespan) scheduling policy for DAG tasks and compares it to the mathematically optimal ILP method.	Doesn't compare the machine learning method with SOTA heuristics, but only compares with ILP.	They use a combination of Graph neural network with attention layers to better capture the structure and dependency information. They use the negative value of the makespan as the reward function to be maximized.

[17]	Current methods to allocate shared resources on multi-core real-time systems use either static analysis of tasks or heuristics which cannot represent all possible system usage scenarios, thus potentially producing higher WCETs and worse system 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 plaform model is a LLC architecture with a shared memory bus and the DRL model uses a dense network with the proximal policy optimization algorithm for training. The DRL model produces a time-triggered schedule table for each tasks' execution and each tasks' memory allocation.
[19]	A previous paper [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 Concurrent Producer and Consumer (CPC) model from [20] to multi-DAG task scheduling by proposing a new multi-DAG scheduling algorithm based on a Parallelism-aware workload distribution model which outperforms existing methods in terms of system schedulability.	Only considers constrained deadlines DAG tasks.	Uses a critical path first execution model by assigning the highest priorities to the node in the critical path and treating them as providers of parallel execution time for the 'consumers' which basically are the parralelizable subtasks for each section of the critical path. For multi-DAG scheduling, they use a method similar to federated scheduling but instead of heavy and light tasks, they look at what they call the degree of parallelism of the DAG tasks.

[13]	Several heuristics for DAG intra-task scheduling have been used but no scalable optimal scheduling algorithm exists.	Propose a Deep Reinforcement Learning based machine learning model that computes an intra-task priority list for single DAG task scheduling which outperforms SOTA by up to 3% in terms of makespan	No comparison is made with ILP methods that lead to the mathematically minimum makespan. Also, it doesn't show the scalability of the GoSu DRL model when increasing the amount of cores in the system.	The network is comprised of a Graph Convolutional network to encode the graph information as well as a sequential decoder based on the attention-mechanism to produce a priority list. The model is trained using REINFORCE with the negative makespan as a reward function.
[9]	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 the concept of an active Virtual processor and a passive virtual processor (VP) and assign one of each on each physical core. The active VP execute the high priority (equivalent of high density in federated scheduling) tasks and the unused processing time of the active VP is treated as a passive VP on which low priority (equivalent of low density in fedrated scheduling) or high priority task can execute.

[1]	Most studies on DAG use the implicity deadline but few are looking at DAGs with arbitrary deadline, especially in the case where the deadline is greater than the task's period. Fluid scheduling has showed promising results but has only been applied to DAG tasks with implicit deadlines.	Propose a fluid scheduling based algorithm for constrained and arbitrary deadlines DAG tasks. Introduce the first capacity bound for DAG with deadlines greater than their periods. Their algorithm performs better in terms of acceptance ratio than SOTA (at the time 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.
[4]	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 real-time simulation system called FRTDS. The proposed algorithm performs better than existing algorithm on the FRTDS platform in terms of makespan for single DAG task scheduling.	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 usage and ram allocation to construct a cost function which is used as a reward for the RL process. The current cost is combined with a future cost, predicted using the successors of the current subtasks.

[11]	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 efficiently tackles the resource wasting problem of federated scheduling while having the same advantages as federated scheduling. The algorithm performs better than existing algorithms in terms of acceptance ratio.	Only considers heavy tasks and constrained deadline DAG tasks.	Introduce two virtual processor per physical core, the Active VP and Passive VP that are complementary. The active VP has the priority in terms of processing capacity and for the processing capacity not used by an active VP is given to the corresponding passive VP. Next, the active VP are allocated to tasks using the difference between their deadline and the critical path's length as well as the minimal number of processor to schedule a task in federated scheduling. Then the passive VPs are allocated to tasks according to how useful they are for scheduling the specific task.

[14]	The LET paradigm is great at dealing with memory contentions and I/O determinism. But typical implementations of LET require local buffers for each core which write/copy data to/from global memory. This can be costly when dealing with huge amount of data, like sensor data in autonomous driving system. Direct Memory Access (DMA) is a possible solution to such a problem but LET hasn't yet been considered with the DMA protocol.	Propose a DMA-based protocol to handle LET communications on multicore systems that minimizes the read/write latency of each tasks. Also, they provide a scheduling algorithm for scheduling the communications using MILP to provide an optimal schedule which improved by up to 98% the communication delays compared to the classic Giotto approach of LET.	Doesn't consider the scalability of their method in terms of the number of tasks/data transfers to be scheduled. Also, they doesn't provide evaluation of known or existing DAG scheduling algorithm using their LET communication protocol compared to the same algorithm not using it.	For the protocol, for each core, an LET task is responsible for programming the DMA engine so that LET communications can happen. For the scheduling algorithm and data allocation, a Mixed-Ingeteger Linear Programming (MILP) problem is solved to minimize either the number of DMA data communications, or the maximum communication delay to period ratio of each tasks.

Table 1: SLR summary table

References

[1] Fei Guan, Long Peng, and Jiaqing Qiao. "A Fluid Scheduling Algorithm for DAG Tasks With Constrained or Arbitrary Deadlines". In: *IEEE Transactions on Computers* 71.8 (2022), pp. 1860–1873. DOI: 10.1109/TC.2021.3111512.

[2] Fei Guan, Long Peng, and Jiaqing Qiao. "A New Federated Scheduling Algorithm for Arbitrary-Deadline DAG Tasks". In: *IEEE Transactions on Computers* 72.8 (2023), pp. 2264–2277. DOI: 10.1109/TC.2023.3244632.

[3] Fei Guan, Jiaqing Qiao, and Yu Han. "DAG-Fluid: A Real-Time Scheduling Algorithm for DAGs". In: *IEEE Transactions on Computers* 70.3 (2021), pp. 471–482. DOI: 10.1109/TC.2020.2990282.

[4] Y. Guan, Bd. Zhang, and Z. Jin. "An FRTDS Real-Time Simulation Optimized Task Scheduling Algorithm Based on Reinforcement Learning". In: *IEEE Access* 8 (2020), pp. 155797–155810. DOI: 10.1109/ACCESS.2020.2997037.

[5] Qingqiang He et al. "Intra-Task Priority Assignment in Real-Time Scheduling of DAG Tasks on Multi-Cores". In: *IEEE Transactions on Parallel and Distributed Systems* 30.10 (2019), pp. 2283–2295. DOI: 10.1109/TPDS.2019.2910525.

[6] Qingqiang He et al. "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. DOI: 10.23919/DATE56975.2023.10137259.

[7] Shingo Igarashi et al. "Heuristic Contention-Free Scheduling Algorithm for Multi-core Processor using LET Model". In: *2020 IEEE/ACM 24th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*. 2020, pp. 1–10. DOI: 10.1109/DS-RT50469.2020.9213582.

[8] Xu Jiang et al. "Decomposition-Based Real-Time Scheduling of Parallel Tasks on Multicores Platforms". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39.10 (2020), pp. 2319–2332. DOI: 10.1109/TCAD.2019.2937820.

[9] Xu Jiang et al. "Scheduling Parallel Real-Time Tasks on Virtual Processors". In: *IEEE Transactions on Parallel and Distributed Systems* 34.1 (2023), pp. 33–47. DOI: 10.1109/TPDS.2022.3213024.

[10] Xu Jiang et al. "Utilization-Tensity Bound for Real-Time DAG Tasks under Global EDF Scheduling". In: *IEEE Transactions on Computers* 69.1 (2020), pp. 39–50. DOI: 10.1109/TC.2019.2936477.

[11] Xu Jiang et al. "Virtually-Federated Scheduling of Parallel Real-Time Tasks". In: *2021 IEEE Real-Time Systems Symposium (RTSS)*. 2021, pp. 482–494. DOI: 10.1109/RTSS52674.2021.00050.

[12] Tomoya Kobayashi and Takuya Azumi. "Work-in-Progress: Federated and Bundled-Based DAG Scheduling". In: *2023 IEEE Real-Time Systems Symposium (RTSS)*. 2023, pp. 443–446. DOI: 10.1109/RTSS59052.2023.00048.

[13] Hyunsung Lee et al. "A Global DAG Task Scheduler Using Deep Reinforcement Learning and Graph Convolution Network". In: *IEEE Access* 9 (2021), pp. 158548–158561. DOI: 10.1109/ACCESS.2021.3130407.

[14] Paolo Pazzaglia et al. "Optimal Memory Allocation and Scheduling for DMA Data Transfers under the LET Paradigm". In: *2021 58th ACM/IEEE Design Automation Conference (DAC)*. 2021, pp. 1171–1176. DOI: 10.1109/DAC18074.2021.9586200.

[15] Junjie Shi et al. "DAG Scheduling with Execution Groups". In: *2024 IEEE 30th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 2024, pp. 149–160. DOI: 10.1109/RTAS61025.2024.00020.

[16] Siyu Xiao, Dongguang Li, and Shiyao 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. DOI: 10.1109/ICUS48101.2019.8995985.

[17] Zijin Xu et al. "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. DOI: 10.1109/ICITES59818.2023.10356887.

[18] Mingjun Zhao et al. "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. DOI: 10.1109/CCDC62350.2024.10587381.

[19] Shuai Zhao, Xiaotian Dai, and Iain Bate. "DAG Scheduling and Analysis on Multi-Core Systems by Modelling Parallelism and Dependency". In: *IEEE Transactions on Parallel and Distributed Systems* 33.12 (2022), pp. 4019–4038. DOI: 10.1109/TPDS.2022.3177046.

[20] Shuai Zhao et al. "DAG Scheduling and Analysis on Multiprocessor Systems: Exploitation of Parallelism and Dependency". In: *2020 IEEE Real-Time Systems Symposium (RTSS)*. 2020, pp. 128–140. DOI: 10.1109/RTSS49844.2020.00022.