A Graph Attention Network Approach to Partitioned Scheduling in Real-Time Systems

Seunghoon Lee and Jinkyu Lee, *Senior Member, IEEE*

***Abstract*—Machine learning methods have been used to solve real-time scheduling problems but none has yet made an architecture that utilizes influences between real-time tasks as input features. This letter proposes a novel approach to partitioned scheduling in real-time systems using graph machine learning. We present a graph representation of real-time task sets that enable graph machine-learning schemes to capture the influence between real-time tasks. By using a graph attention network (GAT) with this method, our model successfully partitioned-schedule task sets that were previously deemed unschedulable by state-of-the-art partitioned scheduling algorithms. The GAT is used to establish relationships between nodes in the graph, which represent real-time tasks, and to learn how these relationships affect the schedulability of the system.**

***Index Terms*—Graph machine learning, real-time scheduling.**

I. Introduction

EFFICIENT usage of computing resources has been and is a crucial issue for embedded systems. Performing its functions and tasks well without wasting or exceeding its available resources is becoming more important as embedded systems, such as autonomous vehicles, and industrial robots use machine learning with GPGPUs. In many cases, these systems are time-sensitive and safety-critical; hence real-time task scheduling must be applied. Real-time tasks, which are tasks specific for real-time systems, have strict timing constraints and must complete their execution by a fixed deadline. We specifically address periodic tasks which are real-time tasks that are executed in a fixed and regular interval. Partitioned scheduling is one of the two representative methods to schedule tasks on real-time systems with multiple processors [1], [2]. The advantages of partitioned scheduling are the simplicity in terms of scheduling/implementation and the ease of developing tight schedulability analysis, both of which are well-known in the real-time systems area. In partitioned scheduling, tasks are grouped into the number of processors by using heuristics, such as first-fit, best-fit, and worst-fit. Then, the group of tasks is each scheduled into their assigned processors to secure safety-critical execution. However, the traditional heuristics are not optimal and there exist many feasible task sets (i.e., schedulable in certain methods of grouping tasks) but not schedulable by the task grouping method applied by existing heuristics [3].

In this letter, we develop a graph machine learning-based strategy that efficiently uses computing resources in task allocation. We present a novel graph representation scheme for real-time task sets that can be widely implemented on many graph-based machine-learning methods. We design the graph to be a portrayal of a single task set with multiple tasks. Each node of the graph represents the tasks and the directed edges connected to each node depict the influences that one task gives the other while being scheduled. This representation allows necessary features for real-time scheduling to be fully captured and utilized while training graph machine learning.

To verify our proposed method's effectiveness, we target partitioned scheduling heuristics and aim to schedule task sets previously considered unschedulable by inappropriate partitioning of existing studies. Using task sets, each deemed schedulable by the state-of-the-art method FBBFFD [4] (which utilizes the first-fit approach), best-fit, and worst-fit, we label tasks within the task set with the indexes of the processors assigned by the heuristics. These grouped and labeled task sets serve as training data for our neural network, enabling the neural network to learn its way of partitioning task sets more efficiently. During testing, when given any task set, the neural network produces its index as output, representing its partitioning decision. Our model successfully partitions and schedules task sets previously deemed unschedulable by well-known partitioned scheduling methods best-fit, worst-fit, and the state-of-the-art FBBFFD.

This letter makes the following contributions.

1) Designing a graph representation for real-time task sets.
2) Applying the design to graph machine learning, specifically for partitioned scheduling where each task affects the other during scheduling and demonstrating its effectiveness.

System Model: This letter deals with a set of periodic real-time tasks denoted as τ, each having two natural number characteristics: 1) T_i (the interval) and 2) C_i (the maximum time required to complete the task) [1]. We represent the parameters of a task $\tau_i \in \tau$ as $\tau_i(T_i, C_i)$. We target a real-time system; each task τ_i is released periodically every T_i, and its single job, starting at time t, must complete its work (which can take up to C_i) before its absolute deadline at $t + T_i$. If a single job of τ_i misses its deadline, the task τ_i is considered unschedulable. Our target system comprises m identical partitioned processors, and tasks are assigned and processed

Manuscript received 10 January 2024; revised 8 February 2024; accepted 10 March 2024. Date of publication 13 March 2024; date of current version 26 November 2024. This work was supported by the Samsung Electronics Company Ltd. under Grant IO201211-08084-01. This manuscript was recommended for publication by T. Azumi. *(Corresponding author: Jinkyu Lee.)*

The authors are with the Department of Computer Science and Engineering, Sungkyunkwan University, Suwon 16419, Republic of Korea (e-mail: seunghoon.l@skku.edu; jinkyu.lee@skku.edu).

Digital Object Identifier 10.1109/LES.2024.3376801

within each processor using fixed-priority scheduling (FPS). That is, each task has a preassigned task-level priority, and at any instant, FPS selects the job with the highest task priority for execution within the processor. Tasks with higher priorities than τ_k (noted as HI_k) cannot be interfered with during execution by tasks with lower priorities. Additionally, tasks with higher priority can interrupt and interfere with lower-priority tasks even while they are being executed on the processor. Since rate monotonic (RM), which assigns higher priorities to tasks with shorter T_i, is proven optimal among FPS [1], we assign task priorities according to RM in each partition.

II. Partitioned Scheduling With Graph Machine Learning

In this section, we will discuss the background of partitioned scheduling. We then will illustrate our partitioned scheduling method using graph representation and implementing state-of-the-art graph machine learning methods.

A. Partitioned Scheduling and Related Work

The primary goal of numerous prior studies on partitioned scheduling is to efficiently allocate a set of n tasks to m processors without any deadline miss of the n tasks.

In traditional partitioning algorithms, real-time tasks' utilization (u_i) is a key factor for applying bin-packing algorithms. The utilization of a single task is defined as $u_i = (C_i/T_i)$, where C_i is the worst-case execution time and T_i is the period of the task. For instance, in the best-fit allocation algorithm concerning FPS, tasks τ within the task set are ordered by priority. The highest priority task is allocated to the first processor, and its utilization becomes an indicator of how much of the processor's capacity is occupied. However, optimizing utilization does not always lead to the most efficient computing resource usage. For periodic real-time tasks, response time analysis is necessary to determine task feasibility for scheduling on processors.

The schedulability of τ_k is decided by the following lemma where tasks with higher priority than τ_k within the processor are noted as $\tau_i \in \mathsf{HI}_k$.

Lemma 1 [5]: A task set τ is schedulable by FPS on a single processor, if every $\tau_k \in \tau$ has R_k ($\leq T_k$) that satisfies

$$C_k + \sum_{\tau_i \in \mathsf{HI}_k} \left\lceil \frac{R_k}{T_i} \right\rceil \cdot C_i \leq R_k. \tag{1}$$

If τ_k has to finish at time R_k, tasks with higher priority must finish their execution by R_k. Thus, the number of times tasks with higher priorities than τ_k can be run during [0,R_k] are calculated,[1] and their execution time is multiplied. By adding $\lceil R_k/T_i \rceil \cdot C_i$ of all tasks with higher priority than τ_k, we can assure that τ_k can be run while not missing its deadline.

The state-of-the-art partitioned scheduling heuristic for real-time scheduling is FBBFFD [4]. The algorithm, which is based on the first-fit method, uses a sufficient condition for determining whether to assign a task to a processor to obtain polynomial-time complexity.

[1]This holds under the condition that the synchronous release of higher-priority tasks yields the critical instant, which was proven in [6].

When it comes to implementing machine learning to partitioned scheduling, there have been no prior studies specifically targeting the subject. Although there have been many studies on using machine learning to solve the bin-packing problem [7], [8], [9], due to Lemma 1, naively applying these methods results in inaccurate training and poor testing results.

B. Graph Representation of Real-Time Tasks

To achieve accurate results using machine learning effectively, it is crucial to provide precise information about the environment to the neural network. In partitioned scheduling, using only task parameters T_i and C_i (execution time and period) along with the utilization $u_i = C_i/T_i$ does not provide sufficient information for the neural network to learn. This limited input would make the network reliant solely on task information and the scheduling outcomes from allocation heuristics. To address this limitation and enable meaningful connections between tasks based on their attributes, we need to consider task associations. Tasks with higher priority can influence tasks with lower priorities during scheduling. We introduce the concept of *interference* to quantify the impact of a higher priority task τ_i on a lower priority task τ_k. This notion is akin to schedulability analysis for multiprocessor scheduling, as seen in works like [10]. By incorporating this interference term, we can better capture task associations and improve the neural network's learning capabilities.

According to Lemma 1, τ_k is schedulable if there exists R_k ($\leq T_k$) that satisfies (1). Since R_k depends on a set of tasks whose priority is higher than τ_k, it is depending on which tasks are partitioned in a processor. To make it independent of partitioning, we consider an upper bound of R_k, which is T_k. Applying T_k to R_k, (1) can be written as follows:

$$\begin{aligned} & C_k + \sum_{\tau_i \in \mathsf{HI}_k} \left\lceil \frac{T_k}{T_i} \right\rceil \cdot C_i \leq T_k \\ \iff & \sum_{\tau_i \in \mathsf{HI}_k} \left\lceil \frac{T_k}{T_i} \right\rceil \cdot C_i \leq T_k - C_k \\ \iff & \frac{\sum_{\tau_i \in \mathsf{HI}_k} \left\lceil \frac{T_k}{T_i} \right\rceil \cdot C_i}{T_k - C_k} \leq 1. \end{aligned} \tag{2}$$

Considering that *interference* in (2) has to be used as an implicit parameter depicting the influence of τ_i to τ_k, we reduce volatility which occurs according to changes in the value of T_k and T_i. Thus, for linearization, we remove the ceiling function and upper-bound the LHS of (2) as follows:

$$\begin{aligned} \text{The LHS of Eq. (2)} &= \frac{\sum_{\tau_i \in \mathsf{HI}_k} \left\lceil \frac{T_k}{T_i} \right\rceil \cdot C_i}{T_k - C_k} \\ &\leq \frac{\sum_{\tau_i \in \mathsf{HI}_k} \left(\frac{T_k}{T_i} + 1 \right) \cdot C_i}{T_k - C_k} \\ &= \frac{\sum_{\tau_i \in \mathsf{HI}_k} \frac{C_i}{T_i} \cdot T_k + \sum_{\tau_i \in \mathsf{HI}_k} C_i}{T_k - C_k} \\ &= \frac{T_k \cdot \sum_{\tau_i \in \mathsf{HI}_k} \frac{C_i}{T_i} + \sum_{\tau_i \in \mathsf{HI}_k} C_i}{T_k - C_k} \\ &= \frac{T_k}{T_k - C_k} \cdot \sum_{\tau_i \in \mathsf{HI}_k} \frac{C_i}{T_i} + \frac{1}{T_k - C_k} \cdot \sum_{\tau_i \in \mathsf{HI}_k} C_i. \end{aligned} \tag{3}$$

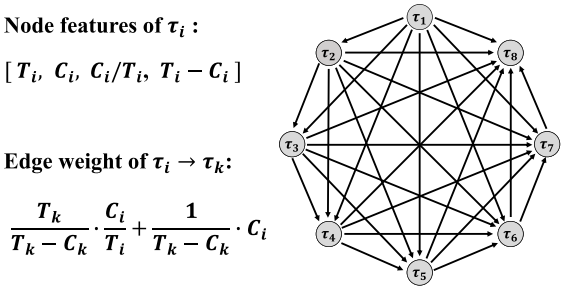

Fig. 1. Graph illustration of a sample task set with eight tasks for real-time scheduling.

Therefore, the directed edge $\tau_k \leftarrow \tau_i$, which is the *interference* of τ_i to τ_k, is given as follows:

$$\frac{T_k}{T_k - C_k} \cdot \frac{C_i}{T_i} + \frac{1}{T_k - C_k} \cdot C_i. \tag{4}$$

The nodes connected by directed edges of *interference* hold information on tasks in a task set. A single node in the graph depicts a single task τ_i, and all nodes contain four attributes that are commonly used as characteristics of a real-time task: 1) T_i; 2) C_i; 3) the utilization of a task ($u_i = T_i/C_i$); and 4) the slack ($T_i - C_i$). Note that our framework can also be expanded to cover the TCD model (in which the relative deadline D_i is not necessarily the same as the period T_i) as well by adding more parameters in each node.

C. Training and Testing Sample Regulation

To measure the performance of our graph representation, we must first structure and generate samples to be used to train the graph machine learning framework.

Training samples are composed of numerous task sets in which n number of tasks are partitioned and scheduled upon m number of processors by FBBFFD. The tasks each hold an index (0 <= index < m) that shows which processor the task has been allocated to by FBBFFD. Each node (task) holds four attributes as suggested in Section II-B; also, as we have calculated in Section II-B, the interference from a higher-priority task to a lower-priority task is illustrated as a directed edge. Overall, our training input design can be drawn as Fig. 1.

Testing samples are strictly screened to match the following conditions: 1) all task sets for testing are schedulable with FBBFFD when there is m number of processors and 2) all task sets are unschedulable by FBBFFD when there is $m-1$ number of processors. Much like the training samples, testing samples also hold information that structures as a graph: nodes, node attributes, edges, and edge weights. As the goal of this letter is to find new combinations of partitioning tasks to reduce computation power, the trained graph neural network targets partition schedule task sets with only $m - 1$ processors.

III. Evaluation

In this section, we evaluate the efficiency of our graph representation in partitioned scheduling.

A. Graph Attention Network Generation

We use the graph attention network (GAT) [11] as our target neural network for partitioned scheduling with our graph representation design. GAT employs an attention mechanism to represent nodes in graph-structured data, allowing each node to weigh its neighbors differently based on edge relevance. It works with various graph types, including weighted, unweighted, directed, and undirected graphs. GAT has shown exceptional performance in node classification tasks in diverse domains, such as social networks, neural science, and recommendation systems. The GAT in our framework consists of three layers, with each layer involving: 1) linear projection and regularization of nodes and edges; 2) calculation of edge attention; 3) neighborhood aggregation; and 4) residual and skip connections.[2]

B. Training and Testing Sample Generation

In our partitioned scheduling framework, we train GAT with two different sets for evaluation. One is a set consisting of 300 000 random task sets that are scheduled by FBBFFD where the index, which is given by FBBFFD, is used as a node classifying indicator in GAT. Another is a set much alike but scheduled and indexed by not only FBBFFD but also best-fit and worst-fit 100 000 sets each. Thus, in terms of how the neural network learns partitioned learning is classifying task sets by the number of processors according to the parameters given. Training sets are generated on these conditions: 1) $m =$ 8 processors; 2) $n \geq m + 1$ tasks per task set; 3) $\max(T_i) =$ 999; and 4) $\min(C_i) = 1$. Other than m, all other parameters are randomly generated within their boundaries. Then we form a graph for every randomly generated task set and train each graph attention network according to m.

For the testing sample, to test the GAT trained with FBBFFD indexed cases, we generate 100 000 task sets that hold these conditions: 1) deemed schedulable by FBBFFD with $m = 8$ processors; 2) deemed unschedulable by FBBFFD with $m = 7$ processors; 3) $n \geq m + 1$ tasks per task set; 4) $\max(T_i) = 999$; and 5) $\min(C_i) = 1$. Each testing set, which was deemed schedulable by FBBFFD with m processors, is put as input in the pretrained GAT according to m. GAT is made to classify these tasks into $m-1$ node classes, attempting to reduce the number of processors that are needed for the task set to be run based on FBBFFD. Then, the resulting $m - 1$ partitioned output is run on real-time analysis to check its schedulability and performance of graph representation embodied with GAT.

Furthermore, to comprehensively evaluate GAT's performance, we generate another testing sample trained with three heuristics: 1) FBBFFD; 2) best-fit; and 3) worst-fit. Similar to the previous scenario, we generate 100 000 task sets meeting specific conditions: 1) deemed schedulable by either FBBFFD, best-fit, or worst-fit with $m = 8$ processors; 2) deemed unschedulable by FBBFFD, best-fit and worst-fit with $m = 7$ processors; 3) $n \geq m + 1$ tasks per task set; 4) $\max(T_i) = 999$; and 5) $\min(C_i) = 1$. Thus, if GAT successfully schedules any task set in this testing sample with

[2]Code is available at https://github.com/SeungHoon00/Partitioned-Scheduling-with-GAT.

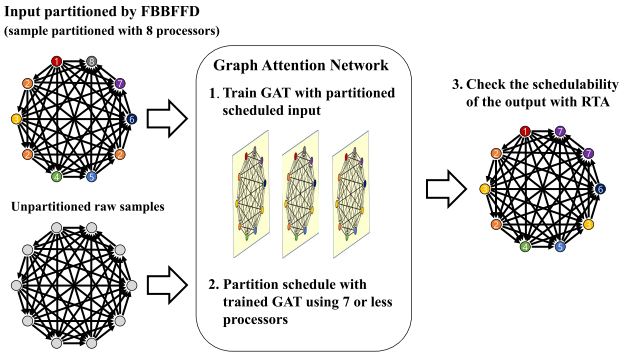

Fig. 2. Partitioned scheduling framework using graph representation and graph attention network.

$m = 7$ processors, it is a unique partitioned scheduling method generated only with GAT using our graph representation model.

C. Evaluation Results

The training and testing process is depicted in Fig. 2. The training input example in Fig. 2 contains ten tasks, partitioned and scheduled on eight processors using FBBFFD. To train the GAT, we use 300 000 random training samples of task sets partitioned and scheduled by eight processors with FBBFFD. Once GAT is trained, its weights are applied to partition the testing sets, such as in the case shown in Fig. 2, where sets are partitioned into seven processors. GAT generates an output by assigning each task in the task set an index of processors. Subsequently, the partitioned scheduling by the trained GAT is evaluated using RTA to check for schedulability. It is important to note that our framework relies on simple supervised learning with GAT and does not explicitly consider RTA during the partitioned scheduling process.

We conducted experiments training the graph attention network (GAT) on datasets labeled by FBBFFD. Notably, GAT effectively scheduled 132 tasks on $m = 7$ processors, a task set previously considered unsolvable by the FBBFFD heuristic. In further testing across different heuristic-labeled datasets, including FBBFFD, best-fit, and worst-fit, GAT identified 23 unique scheduling solutions, showcasing its ability to discover diverse solutions compared to traditional heuristics. Although these experimental results may seem modest—132 out of 100 000 and 23 out of 100 000—keep in mind that all 100 000 samples were deemed unschedulable by their respective trained heuristics. We must emphasize the cases where our GAT model achieved partitioned scheduling with fewer computing resources, reducing from 8 to 7 processors—a significant 12.5% reduction—which no other heuristic accomplished.

D. Case Study

In this section, we present how our model partitions and schedules task sets that were deemed unschedulable by other heuristics.

A task set $\{\tau_1(111, 58), \tau_2(129, 104), \tau_3(141, 96), \tau_4(265, 4), \tau_5(276, 210), \tau_6(490, 297), \tau_7(494, 75), \tau_8(829, 316), \tau_9(854, 445), \tau_{10}(899, 408)\}$, which is a task set consisted of ten tasks, each task τ being represented by (T, C), is deemed unschedulable onto seven processors by all of the partitioned scheduling heuristics. GAT however schedules this task set onto seven processors by partitioning tasks as the following: $[\tau_1]$ $[\tau_2, \tau_4]$ $[\tau_3, \tau_7]$ $[\tau_5]$ $[\tau_6]$ $[\tau_8]$ $[\tau_9, \tau_{10}]$, where each bracket depicts a single processor. This partitioning method acts similarly to the best-fit heuristic which however cannot schedule this task set due to the utilization bound $(n(2^{1/n} - 1.0))$, i.e., 0.828 for $n = 2$. Our GAT however schedules without being restricted by the utilization bound where the tasks scheduled onto the 7th processor have a utilization sum of 0.974 (= 445/854 + 408/899) which is far beyond the bound, 0.828.

Another task set deemed schedulable by GAT, $\{\tau_1(17, 13), \tau_2(286, 20), \tau_3(296, 265), \tau_4(298, 231), \tau_5(315, 64), \tau_6(325, 6)], \tau_7(570, 73), \tau_8(588, 173), \tau_9(658, 359), \tau_{10}(677, 369), \tau_{11}(840, 261), \tau_{12}(961, 556)\}$, also shows similar traits when partitioned scheduled. This task set is scheduled by GAT as follows: $[\tau_1]$ $[\tau_2, \tau_3]$ $[\tau_4]$ $[\tau_5, \tau_6, \tau_7]$ $[\tau_8, \tau_9]$ $[\tau_{10}]$ $[\tau_{11}, \tau_{12}]$. The utilization sums of the 2nd, 5th, and 7th processors are each 0.965, 0.840, and 0.9 which are also larger than the utilization bound, letting the GAT partition schedule this task set which was deemed unschedulable by existing heuristics.

IV. Conclusion

In this letter, we present a graph representation of real-time task sets that enables graph machine-learning schemes to capture the influence between tasks. We demonstrated that by using a graph attention network with our graph representation method for real-time systems, it successfully schedules task sets that were deemed unschedulable by the state-of-the-art partitioned scheduling algorithms. For future work, we aim to expand this graph representation to solve other real-time scheduling problems while enjoying the benefit of advances in graph machine learning.

References

[1] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Comput. Surveys*, vol. 43, no. 4, pp. 1–44, 2011.
[2] A. Burmyakov and B. Nikolić, "An exact comparison of global, partitioned, and semi-partitioned fixed-priority real-time multiprocessor schedulers," *J. Syst. Archit.*, vol. 121, pp. 1–13, Dec. 2021.
[3] G. Levin, S. Funk, C. Sadowski, I. Pye, and S. Brandt, "DP-FAIR: A simple model for understanding optimal multiprocessor scheduling," in *Proc. 22nd Euromicro Conf. Real-Time Syst.*, 2010, pp. 3–13.
[4] N. Fisher, S. Baruah, and T. P. Baker, "The partitioned scheduling of sporadic tasks according to static-priorities," in *Proc. 18th Euromicro Conf. Real-Time Syst. (ECRTS)*, 2006, pp. 1–10.
[5] N. Audsley, A. Burns, and A. Wellings, "Deadline monotonic scheduling theory and application," *Control Eng. Pract.*, vol. 1, no. 1, pp. 71–78, 1993.
[6] C. Liu and J. Layland, "Scheduling algorithms for multi-programming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, 1973.
[7] H. Hu, X. Zhang, X. Yan, L. Wang, and Y. Xu, "Solving a new 3D bin packing problem with deep reinforcement learning method," 2017, *arXiv:1708.05930*.
[8] F. Mao et al., "Small boxes big data: A deep learning approach to optimize variable sized bin packing," in *Proc. IEEE 3rd Int. Conf. Big Data Comput. Service Appl. (BigDataService)*, 2017, pp. 80–89.
[9] R. Verma et al., "A generalized reinforcement learning algorithm for online 3D bin-packing," 2020, *arXiv:2007.00463*.
[10] J. Lee, A. Easwaran, and I. Shin, "Maximizing contention-free executions in multiprocessor scheduling," in *Proc. IEEE Real-Time Technol. Appl. Symp.*, 2011, pp. 235–244.
[11] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," 2017, *arXiv:1710.10903*.