ML for RT: Priority Assignment Using Machine Learning

Seunghoon Lee*, Hyeongboo Baek†, Honguk Woo*, Kang G. Shin‡, Jinkyu Lee*§

*Department of Computer Science and Engineering, Sungkyunkwan University (SKKU), Republic of Korea
†Department of Computer Science and Engineering, Incheon National University (INU), Republic of Korea
‡Department of Electrical Engineering and Computer Science, The University of Michigan – Ann Arbor, U.S.A.
seunghoon.l@skku.edu; hbbaek@inu.ac.kr; hwoo@skku.edu; kgshin@umich.edu; jinkyu.lee@skku.edu

Abstract*—As machine learning (ML) has been proven effective in solving various problems, researchers in the real-time systems (RT) community have recently paid increasing attention to ML. While most of them focused on timing issues for ML applications (i.e., RT for ML), only a little has been done on the use of ML for solving fundamental RT problems. In this paper, we aim at utilizing ML to solve a fundamental RT problem of priority assignment for global fixed-priority preemptive (gFP) scheduling on a multiprocessor platform. This problem is known to be challenging in the case of a large number (n) of tasks in a task set because exhaustive testing of all priority assignments (as many as $n!$) is intractable and existing heuristics cannot find a schedulable priority assignment, even if exists, for a number of task sets. We systematically incorporate RT domain knowledge into ML and develop an ML framework tailored to the problem, called PAL. First, raising and addressing technical issues including neural architecture selection and training sample regulation, we enable PAL to infer a schedulable priority assignment of a set of n tasks, by training PAL with same-size (i.e., with n tasks) samples each of whose schedulable priority assignment has already been identified. Second, considering the exhaustive testing of all priority assignments of each task set with large n makes it intractable to provide training samples to PAL, we derive inductive properties that can generate training samples with large n from those with small n, through empirical observation of PAL and mathematical analysis of the target gFP schedulability test. Finally, utilizing the inductive properties and additional techniques, we propose how to systematically implement PAL whose training sample generation process not only yields unbiased samples but also is tractable even for large n. Our experimental results demonstrate PAL covers a number of additional task sets, each of which has never been proven schedulable by any existing approaches for gFP.

I. Introduction

Recent years have witnessed studies related to Machine Learning (ML) in real-time systems (RT). Starting with [1] that optimizes ML workloads in a time-sensitive multi-tasking environment, most of them have addressed timing issues for ML applications (classified as *RT for ML*), while a little has been done to solve fundamental RT problems (classified as *ML for RT*).

Recognizing this imbalance, this paper aims at utilizing ML to solve a fundamental, challenging RT research problem. Targeting global fixed-priority preemptive (gFP) scheduling that assigns a fixed priority to each task and schedules the m highest-priority tasks in each time slot (where m is the number of processors on the platform), we focus on the following priority assignment problem, associated with one of the high-performance state-of-the-art schedulability tests of gFP [2], [3], [4], [5]. Given a task set that is not deemed schedulable by the target schedulability test with any existing heuristic priority assignments on an m-processor platform, find at least one *schedulable priority assignment* for the task set that is deemed schedulable by the target schedulability test. The problem is challenging (and its solution is useful) for a task set with a large number (n) of tasks, because (a) there is no known optimal priority assignment for gFP on a multiprocessor platform (unlike the uniprocessor case [6]), (b) existing heuristic priority assignments cannot discover a schedulable priority assignment, if exists, for a number of task sets, and (c) it is intractable to test all priority assignments (as many as $n!$) with large n (e.g., $15! = 1,307,674,368,000$). Note that there exists a technique that allows to find a schedulable priority assignment without investigating all priority assignments [7], but it cannot be applied to high-performance schedulability tests for gFP, to be discussed in Section II-A.

To find a solution to this problem, we systematically incorporate RT domain knowledge into *supervised learning* [8] that maps an input to an output by training input-output pairs (called *samples*), and develop an ML framework specialized for the problem, called PAL (Priority Assignment Learning Framework). The first goal of PAL is to infer a schedulable priority assignment for a task set with n tasks on an m-processor platform, by training with a sufficiently large number of same-size (i.e., with n tasks on an m-processor platform) training samples. To this end, we need to address the following questions.

- Q1. How can we design sample structure (i.e., input and output), and which neural architecture is suitable for the sample structure?
- Q2. Among many schedulable priority assignments for a given task set, which assignment forms a favorable training sample for effective training/inference? What is the criterion for finding such an assignment?

To answer Q1, we design the sample structure in the most intuitive way: input as a sequence of task parameters, and output as a sequence of task indexes, the position of each of which indicates its priority, to be presented in Eq. (5). The sample structure needs a neural architecture to be capable of associating each task index in an output with its corresponding task parameters in its input. This is because, such capability is effective in training PAL with samples so as to infer a schedulable priority assignment, compared to a neural archi-

§Jinkyu Lee is the corresponding author.

tecture that interprets a sequence of task indexes in an output as enumeration of integer numbers. To this end, we adopt a neural architecture called *pointer networks* [9], which learn the conditional probability of each output element that corresponds to the position of an input element.

For Q2, since a task set tends to have multiple schedulable priority assignments, we usually have many options in generating training samples each of whose input is a given task set and output is its schedulable priority assignment. Among these options, we argue that it is effective to use a single "best" training sample for a given task set in inferring a schedulable priority assignment of other task sets. To select the "best" priority assignment, we adopt the notion of *system hazard* [10], which captures how sufficiently (or barely) a task set with a target priority assignment is schedulable. Since a task set with a priority assignment that yields a smaller system hazard is easier to maintain the schedulability of the task set in the case of increase of any execution time, decrease of any period or an addition of a new task, we define the *premier* priority assignment for a task set as the priority assignment of that task set which yields the smallest system hazard. Then, a sample with the premier priority assignment guides PAL toward a schedulable priority assignment of other task sets, by offering the features of a schedulable priority assignment that can be applied to the most number of task sets. By addressing Q1 and Q2, PAL can infer a schedulable priority assignment of a target task set, as long as we train PAL with samples, the size of each of which is the same as the target task set and its output is the premier priority assignment.

Despite its capability of inferring a schedulable priority assignment, PAL as of now cannot solve the priority assignment problem for a task set with a large number of tasks (n). That is, while we can generate a premier sample (whose output is a premier priority assignment) by exhaustively testing all priority assignments of a task set with small n, the same does not hold for a target task set with large n. This raises the following critical issue.

Q3. How can we design the sample-generation process that is tractable for large n?

To address Q3, we first define a notion of a *pseudo-premier* priority assignment for a task set, whose system hazard is smaller than all known heuristic priority assignments for the task set. This notion makes it tractable to determine whether a given priority assignment of a task set with large n is a pseudo-premier or not, as it only requires investigation of the heuristic priority assignments. By utilizing the tractability, we derive two types of inductive properties to generate candidates for pseudo-premier samples with large n, from pseudo-premier samples with small n: one from empirical observation for PAL and the other from mathematical analysis of the target schedulability test. Considering the two types of inductive properties are complementary to each other in terms of unbiased and tractable generation of samples, we propose how to systematically implement the sample-generation process by utilizing the two types of inductive properties and additional techniques. Once PAL is trained with the generated samples, PAL is capable of inferring a pseudo-premier (or at least schedulable) priority assignment for a target task set with given large n, thus solving the priority assignment problem.

Our ML framework is empirically shown to have two salient features. First, it can solve the target priority assignment problem effectively, as it can find a priority assignment schedulable by the target schedulability test for a number of task sets. Second, it covers a number of additional gFP-schedulable task sets, each of which has not been proven schedulable by any of existing approaches for gFP.

This paper makes the following contributions.

- The first attempt to utilize supervised learning for solving the priority assignment problem — an important fundamental problem in the RT research field;
- Design of the ML framework specialized for the priority assignment problem, and proposal of a methodology for incorporating RT domain knowledge in the ML framework development;
- Success in solving the target priority assignment problem by the proposed ML framework; and
- Discovery of a number of additional gFP-schedulable task sets, each of which has not been proven gFP-schedulable by any existing approaches.

The remainder of this paper is structured as follows. Section II describes our priority assignment problem with its target schedulability test. Section III develops the foundation for PAL, and Section IV derives inductive properties for PAL. Section V presents the implementation of PAL. Section VI demonstrates our experimental results, and finally, Section VII concludes the paper.

II. Background and Goal

We first state our target priority assignment problem for gFP, along with existing approaches that improve schedulability of gFP. We then summarize our target schedulability test for gFP, which is associated with the priority assignment problem.

A. Priority Assignment Problem and Related Work

In this paper, we consider the implicit-deadline sporadic task model [6], [11]. Each task τ_i in a task set τ is modeled by (T_i, C_i), where T_i is the minimum separation (or period) and C_i is the worst-case execution time. Each task τ_i invokes a series of jobs, each separated from its predecessor by at least T_i time units and supposed to finish its execution within T_i time units after its release. We assume the quantum length is one unit, so all task parameters (i.e., T_i and C_i) are integers. We consider a multiprocessor platform consisting of m identical processors.

We focus on gFP (**g**lobal **F**ixed-**P**riority preemptive) scheduling, which assigns a fixed priority to each task. In each time slot, gFP chooses and executes m jobs whose invoking tasks' priority is the highest. A task set τ is said to be *schedulable* by gFP with a given priority assignment, if there is no single job deadline miss for all possible (infinitely-many) job sequences when they are scheduled by gFP with the priority assignment. A schedulability test for gFP finds sufficient condition(s) for schedulability of a target task set with a given priority assignment. Then, the priority assignment problem associated with a schedulability test for gFP is to find

TABLE I. AVERAGE TIME TO TEST THE SCHEDULABILITY OF ALL PRIORITY ASSIGNMENTS FOR A TASK SET OF $6 \leq n \leq 11$ TASKS ON TWO PROCESSORS BY THE SCHEDULABILITY TEST [3]

n	6	7	8	9	10	11
Time (second)	0.004	0.028	0.236	2.32	31.2	379

at least one priority assignment for a task set on m processors, which is deemed schedulable by a target schedulability test for gFP (called *schedulable* priority assignment). The priority assignment problem is challenging for a task set with a large number (n) of tasks, because of the three reasons (a), (b) and (c) mentioned in Section I.

Regarding the reason (c) in Section I, we need to investigate time to test the schedulability of every priority assignment of a task set. Since there are $n!$ priority assignments for a task set with n tasks and it takes $O(n^2 \cdot \max_{\tau_i \in \tau} T_i)$ to apply a high-performance state-of-the-art schedulability test in [3] (also to be used as our target schedulability test), the time-complexity to check the schedulability of all priority assignments for a task set by the schedulability test is $O(n! \cdot n^2 \cdot \max_{\tau_i \in \tau} T_i)$, which is exponential in n. To confirm the intractability for large n, we measure the actual time under a PC, when the number of processors (m) is two and the number of tasks (n) is $6 \leq n \leq 11$; Table I records the average time for each n. As shown in the table, the ratio between the time for $(n-1)$ and that for n is no smaller than n; if we assume the ratio is exactly n (that is smaller than the actual ratio), the time for $n = 15$ is expected to be $12,416,040$ seconds = 144 days. Therefore, it is intractable to solve the priority assignment problem by exhaustive testing of all priority assignments of a task set with large n, although the criterion for large n depends on the available computing power.

To improve the schedulability of gFP in spite of the intractability of investigating all priority assignments, there have been two categories of study: (S1) develop low-performance OPA-compatible (optimal priority assignment) schedulability tests [12], [13] and apply them to Audsley's OPA [7], and (S2) develop high-performance OPA-incompatible schedulability tests [2], [3], [4], [5] and apply them to heuristic priority assignments [6], [14], [15]. Under the schedulability tests for (S1), a task's schedulability only depends on a set of its higher-priority tasks, not on the relative priorities among the higher-priority tasks. While this property enables finding of a schedulable priority assignment without investigating all priority assignments using Audsley's OPA (which investigates only $O(n^2)$ tasks), it prevents use of slack values, which are generated by relative priorities of the higher-priority tasks and used to reduce the interference to the task of interest. Therefore, the schedulability tests for (S1) themselves without OPA exhibit low schedulability performance.

On the other hand, schedulability tests for (S2) utilize the slack values of higher-priority tasks, which yield high performance in terms of schedulability, but cannot apply Audsley's OPA. Due to this inability, existing studies found the following "good" heuristic priority assignment policies: the smaller T_i, the higher priority (DMPO) [6]; the smaller $(T_i - C_i)$, the higher priority (D-CMPO) [14]; and the smaller $(T_i - k \cdot C_i)$, the higher priority, where $k = \frac{m-1+\sqrt{5 \cdot m^2 - 6 \cdot m + 1}}{2 \cdot m}$ (DkC) [15]. Note that there are some other heuristic priority assignment policies, but they are known to be dominated by DMPO, D-CMPO and DkC [16], [13].

Due to the tradeoff between low-performance schedulability tests applied to OPA and high-performance schedulability tests applied to heuristic priority assignments, no dominance between (S1) and (S2) exists. However, under a given priority assignment, every low-performance schedulability test for (S1) is dominated by its corresponding high-performance schedulability test for (S2) (e.g, DA [12] dominated by RTA [2], DA-LC [13] dominated by RTA-LC [3]). Hence, if we target a task set that is deemed schedulable by *neither* any state-of-the-art high-performance schedulability test with heuristic priority assignments (e.g., RTA-LC with DMPO, D-CMPO and DkC) *nor* any low-performance OPA-compatible schedulability test with OPA (e.g., DA-LC with OPA), this task set, with a "good" priority assignment, can be deemed schedulable by the state-of-the-art high-performance schedulability test. Therefore, this paper aims at solving the following problem with our ML framework to be developed in Sections III and IV.

Problem 1: Given a task set τ, for which the existing heuristic priority assignments (i.e., DMPO, D-CMPO, and DkC) cannot be deemed schedulable by the target schedulability test in [3][1] (i.e., Lemma 1 to be presented in Section II-B, called `RTA-LC`) on m processors, Find at least one priority assignment for τ, which is deemed schedulable by the test on m processors.

After developing the proposed ML framework, Section VI will evaluate its effectiveness in solving Problem 1 in comparison with existing approaches for gFP, including a hybrid approach of S1 and S2 in [17]. Note that (iii) takes advantage of both S1 and S2 in that it determines as many priorities as possible using an OPA-compatible schedulability test in S1 (i.e., DA-LC [13]), and then assigns the remaining priorities using an OPA-incompatible schedulability test in S2 (i.e., RTA-LC [3]) with backtracking (by limiting the number of trials for tractability).

There has been prior work on applying ML to solve real-time scheduling problems [18], [19], [20], [21], [22], [23], [24], [25]. Ae and Aibara [18] focused on shortening the scheduling time in a hard real-time environment using a processor made out of neural network. Chen and Huang [23] used competitive learning rules on Hopfield neural network to solve preemptive multitasking without task migrations, and Guo and Baruah [25] solved a real-time scheduling problem on a uniprocessor platform using a single layer RNN model. There have also been approaches to scheduling problems by building neural networks on the foundation of k-out-of-N rule [19], [20], [21]. Attempts have been made to apply neural network to a job-shop scheduling problem [22], [24]. Our work is different from all of these in that (i) our target problem is different from these prior studies, and (ii) we develop new approaches to the priority assignment problem, including the generation of large-size samples from small-size samples using properties of *both* neural architecture *and* schedulability analysis.

[1]While the priority assignment problem associated with any of the state-of-the-art high-performance schedulability tests for gFP in [2], [3], [4], [5] can be applied to our ML framework to be developed in Sections III, IV and V, we choose to apply the schedulability test in [3] (i.e., Lemma 1) for exploiting lower time-complexity but high schedulability performance.

B. Target Schedulability Test

Now we explain the target schedulability test in Problem 1. In a nutshell, the test calculates the response time of every task $\tau_i \in \tau$ (denoted by R_i), meaning that every job of τ_i finishes its execution within R_i time units from its release. If $R_i \leq T_i$ holds for every $\tau_i \in \tau$, the task set is deemed schedulable by the target test. To this end, the test calculates the notion of *interference*; let $I_{k \leftarrow i}(\ell)$ denote the cumulative interval length where a job of τ_k cannot execute due to execution of jobs of τ_i ($\neq \tau_k$) in an interval of length ℓ that starts with the job of τ_k's release. Considering a job cannot execute in a time slot only when other m higher-priority jobs execute in the slot, a job of τ_k finishes its execution within R_k time units from its release if the following holds under gFP [2]:

$$R_k = C_k + \left\lfloor \frac{\sum_{\tau_i \in \mathsf{HI}_k} I_{k \leftarrow i}(R_k)}{m} \right\rfloor, \tag{1}$$

where HI_k denotes a set of tasks in τ, each of which has a higher priority than τ_k.

The remaining issue is to upper-bound the summation term of $I_{k \leftarrow i}(R_k)$ in Eq. (1) for every job of τ_k. Under gFP, a higher-priority task τ_i can interfere with a lower-priority task τ_k during any job execution of τ_i. Therefore, we need to calculate the maximum execution of jobs of τ_i in an interval of length ℓ, which is calculated by [2]

$$W_i(\ell) = N_i(\ell) \cdot C_i + \min\Big(C_i, \ell + T_i - C_i - S_i - N_i(\ell) \cdot T_i\Big), \tag{2}$$

where $N_i(\ell) = \left\lfloor \frac{\ell + T_i - C_i - S_i}{T_i} \right\rfloor$, and S_i denotes the slack value of τ_i, meaning that every job of τ_i finishes its execution no later than S_i time units ahead of its deadline; once R_i is calculated, S_i is set to $(T_i - R_i)$. Here we explain the job execution/release pattern for $W_i(\ell)$. The first job of τ_i executes as late as possible, and the following jobs execute as early as possible; the interval of interest starts when the first job starts its execution. With the job execution/release pattern, $N_i(\ell)$ calculates the number of jobs of τ_i whose deadline is within the interval of interest of length ℓ; such jobs contribute $N_i(\ell) \cdot C_i$ to $W_i(\ell)$, and the last job whose deadline is after the interval of interest (if exists) contributes the second term of the RHS (right-hand-side) of Eq. (2) to $W_i(\ell)$.

Then, Bertogna and Cirinei [2] judge the schedulability of τ by calculating R_k for $\tau_k \in \tau$ as follows. From the highest-priority task to the lowest-priority task, we perform the following process with setting $S_i = 0$ for every task τ_i. For each τ_k, starting from $R_k = C_k$, we calculate the RHS of Eq. (1) by replacing $I_{k \leftarrow i}(R_k)$ to $W_i(R_k)$ for every $\tau_i \in \mathsf{HI}_k$. If the value is larger than R_k, we set R_k to the value, and repeat the calculation until R_k is larger than T_k (deemed unschedulable) or R_k is converges (deemed schedulable). If the task is deemed unschedulable, we conclude that τ is unschedulable; otherwise, we update $S_k = T_k - R_k$, and proceed with the next task. If every task is deemed schedulable, we conclude that τ is schedulable.

Based on [2], Guan *et al.* [3] reduces the amount of interference by deriving the following condition for a critical instant [3]: under gFP, a critical instant of a job of a task occurs only when there are at most $(m-1)$ higher-priority tasks, each of which has a carry-in job in the interval of the job of interest.[2]

For a task τ_i that does not have any carry-in job in an interval of interest of length ℓ, we need to calculate the maximum execution of jobs of τ_i in the interval, which is calculated by [3]

$$W_i'(\ell) = N_i'(\ell) \cdot C_i + \min\Big(C_i, \ell - N_i'(\ell) \cdot T_i\Big), \tag{3}$$

where $N_i'(\ell) = \left\lfloor \frac{\ell}{T_i} \right\rfloor$. The job execution/release pattern for $W_i'(\ell)$ is explained as follows: every job of τ_i executes as early as possible, and the interval of interest starts when the first job starts its execution. $W_i'(\ell)$ and $N_i'(\ell)$ correspond to $W_i(\ell)$ and $N_i(\ell)$, respectively.

Then, the sum of interference can be upper-bounded as follows [3].[3]

$$\sum_{\tau_i \in \mathsf{HI}_k} I_{k \leftarrow i}(\ell) \leq \sum_{\tau_i \in \mathsf{HI}_k} [W_i'(\ell)]^{\ell - C_k + 1} + \sum_{\tau_i \in \mathsf{HI}_k \mid \text{largest } m-1} [W_i(\ell)]^{\ell - C_k + 1} - [W_i'(\ell)]]^{\ell - C_k + 1}, \tag{4}$$

where $[A]^B$ denotes $\min\{A, B\}$. In the RHS of Eq. (4), we add $W_i'(\ell)$ for every $\tau_i \in \mathsf{HI}_k$ and add the difference between $W_i(\ell)$ and $W_i'(\ell)$ for $(m-1)$ tasks in HI_k that have the $(m-1)$ largest differences. Then, with any combination of up to $(m-1)$ tasks that have a carry-in job, the sum of interference of τ_k is upper-bounded by the RHS of Eq. (4). Note that the reason for upper-bounding $W_i(\ell)$ and $W_i'(\ell)$ as $(\ell - C_k + 1)$ is proven in [2]. Using the interference upper-bound, the schedulability test in [3] can be stated in the following lemma.

Lemma 1 (From [3], called RTA-LC): A task set τ is schedulable by gFP with a priority assignment $\mathcal{P}$, if $R_k \leq T_k$ holds for all $\tau_k \in \tau$, where R_k is calculated by finding R_k that satisfies Eq. (1) with the summation term replaced by the RHS of Eq. (4). Note that it is the same as [2] how to find R_k that satisfies Eq. (1) and how to reclaim the slack value S_k.

The lemma holds by Eqs. (1) and (4); the full proof is shown in [3]. The time-complexity for Lemma 1 is $O(n^2 \cdot \max_{\tau_i \in \tau} T_i)$ [2], where n is the number of tasks in each task set, because each of n tasks τ_i performs a summation of up to $(n-1)$ tasks up to $(T_i - C_i + 1)$ times.

III. PAL: Priority Assignment Learning Framework

Utilizing *supervised learning* [8] that maps an input to an output by training input–output pairs (called *samples*), we develop PAL, a priority assignment learning framework. The first goal of PAL is to infer the premier (or at least schedulable) priority assignment for a task set with n tasks on an m-processor platform, assuming we secure a sufficiently large number of same-size (i.e., with n tasks on an m-processor platform) samples each of whose premier priority assignment has already been identified. To this end, we first propose our sample structure design and neural architecture selection for

[2] A job is said to be *carry-in* in an interval if the job is released before the interval and has remaining execution at the beginning of the interval.

[3] In [3], there is a minor optimization of $W_i(\ell)$, which may deduct one unit of execution from $W_i(\ell)$.

supervised learning tailored to our target priority assignment problem (i.e., Problem 1). We then develop a strategy to regulate training samples for effective training/inference, by carefully selecting a criterion that finds the premier priority assignment for a given task set.

A. Sample Structure Design and Neural Architecture Selection

To design the supervised learning framework for our priority assignment problem (i.e., Problem 1), we first need to design the sample's input/output structure and select a neural architecture. We design the sample structure in the most intuitive way: input as a sequence of task parameters, and output as a sequence of task indexes, each of whose position represents its priority. For example, the following sample implies as follows: there are three tasks $\tau_1(T_1 = 5, C_1 = 1)$, $\tau_2(T_2 = 3, C_1 = 2)$ and $\tau_3(T_3 = 10, C_3 = 5)$, and the 1^{st}, 2^{nd} and 3^{rd} priority tasks are τ_2, τ_1 and τ_3, respectively.

$$\texttt{Input}[[5,1][3,2][10,5]] \rightarrow \texttt{Output}[2,1,3] \tag{5}$$

Then, we have to select a neural architecture suitable for supporting the sample structure of our priority assignment problem. Such a neural architecture should be capable of interpreting each output element as an order of input elements. To make an output embody the meaning of orders of tasks, we need to find a way to combine an input element (i.e., each task of a given task set) and an output element into a pair, which can be achieved using attention-based encoder-decoder architectures [26]. This has been widely used in neural language translation, where each input element represents a specific word which, in turn, has an exact equivalent throughout the output elements.

Among many neural architectures derived from encoder-decoder architectures that employ the attention mechanism, we select *pointer networks* [9] as a base neural architecture for solving our priority assignment problem. Pointer networks determine vectors by recurrently computing the hidden states and trained values from the model accordingly. These vectors, after going through a softmax function [27], are used directly as pointers to indicate the input elements. This characteristic not only makes it successfully associate an input element $[T_i, C_i]$ with its output element that is the corresponding order of the input element in the task set, but also enables each output element to have a *distinct* order; the latter is a necessary condition for every feasible priority assignment. In addition, pointer networks exhibit a capability in inferring an output for a sample without including the same-size samples in the training set (although the output accuracy is low), which can be exploited to generate large-size training samples for our priority assignment; see Section IV for a detailed explanation.

B. Training Sample Regulation

Once the sample structure and neural architecture are decided, the next step is to determine which samples to use for training. Suppose we test all $n!$ priority assignments for a set of n tasks, and identify s schedulable priority assignments. For the task set, there are various choices to generate samples in terms of how many samples (from 1 to s) and which schedulability priority assignments to be used. These choices are very important, as they determine the effectiveness of training samples in inferring a schedulable priority assignment for other task sets. As a naive choice, we may generate s samples for the task set using all of s schedulable priority assignments, as shown in the following example.

Example 1: Consider a set τ of tasks $\tau_1(T_1 = 5, C_1 = 2)$, $\tau_2(5, 2)$ and $\tau_3(10, 1)$ to be scheduled on a platform with $m = 2$ processors. According to Lemma 1, all $3! = 6$ priority assignments are schedulable. Suppose we use all priority assignments (as output) for the task set (as input), generating $3! = 6$ samples. Then, the samples cannot provide any guidelines for finding a schedulable priority assignment for other task sets. For example, consider another task set τ' by doubling C_i of $\tau_i \in \tau$, i.e., $\tau' = \{\tau'_1(5, 4), \tau'_2(5, 4), \tau'_3(10, 2)\}$. According to Lemma 1, only two priority assignments in which τ'_3's priority is the lowest are schedulable. No one can infer the two schedulable priority assignments for τ' from the six samples for τ.

Example 1 shows the importance of selecting training samples for a given task set, and necessitates identification of the "best" sample for a task set, which is capable of preserving the schedulability of the task set even if its task parameters are changed towards an infeasible task set (e.g., increase of any C_i, decrease of any T_i, or an addition of a new task). To select the "best" priority assignment that guides other tasks toward a schedulable priority assignment, we adopt (and adapt) the concept of *system hazard* [10] for a task set τ with a priority assignment $\mathcal{P}$ on m processors as follows.

$$\Theta(\tau, \mathcal{P}, m) = \max_{\tau_i \in \tau} R_i/T_i, \tag{6}$$

where R_i is calculated by Lemma 1 for τ with $\mathcal{P}$ on m processors. If there exists at least one task $\tau_i \in \tau$ whose R_i larger than T_i, $\Theta(\tau, \mathcal{P}, m) > 1.0$ holds. Therefore, $\Theta(\tau, \mathcal{P}, m) \leq 1.0$ implies τ with $\mathcal{P}$ is deemed schedulable on m processors by Lemma 1, while $\Theta(\tau, \mathcal{P}, m) > 1.0$ implies τ with $\mathcal{P}$ is not.

Since the system hazard $\Theta(\tau, \mathcal{P}, m)$ is the maximum ratio between the response time (R_i) and the period (T_i) for every $\tau_i \in \tau$, a task set with a priority assignment that yields a smaller system hazard is easier to preserve schedulability even when any C_i increases, any T_i decreases, or a new task is added. For example, consider a task set τ with two priority assignments $\mathcal{P}$ and $\mathcal{P}'$, which yield $\Theta(\tau, \mathcal{P}, m) = 1.0$ and $\Theta(\tau, \mathcal{P}', m) = 0.5$. Since the task set with $\mathcal{P}$ is barely schedulable, increase of any C_i or decrease of any T_i tends to make the task set with $\mathcal{P}$ unschedulable. On the other hand, $\mathcal{P}'$ is resilient to such task parameter changes, because the task set with $\mathcal{P}'$ is still schedulable even if every response time is doubled. As shown in the example, a priority assignment with a smaller system hazard can offer clues of a schedulable priority assignment that can be applied to more (similar) task sets.

Using the concept of $\Theta(\tau, \mathcal{P}, m)$, we define the "best" priority assignment as follows.

Definition 1: A priority assignment $\mathcal{P}$ for a task τ on an m-processor platform is said to be *premier*, if $\mathcal{P}$ yields a system hazard $\Theta(\tau, \mathcal{P}, m)$, which is no larger than 1.0 and the smallest among all the priority assignments for τ on an m-processor platform.

Then, we use a single sample for a given task set (i.e., a given input), whose output is the premier priority assignment. The following example shows the change of the situation in Example 1 if we use such a single sample.

Example 2: Recall τ and τ' in Example 1. For τ, the premier priority assignment $\mathcal{P}^*$ is $\texttt{Output}[1,2,3]$ (or $\texttt{Output}[2,1,3]$), which yields $\Theta(\tau, \mathcal{P}^*, 2) = 0.4$; all other priority assignments $\mathcal{P}$ yield $\Theta(\tau, \mathcal{P}, 2) = 0.6$. Then, the sample that maps τ to $\mathcal{P}^*$ can guide other task sets into a schedulable priority assignment. For example, the premier priority assignment $\mathcal{P}^*$ for τ makes τ' schedulable (i.e., $\Theta(\tau', \mathcal{P}^*, 2) = 1.0$), while all other priority assignments $\mathcal{P}$ for τ make τ' unschedulable (i.e., $\Theta(\tau', \mathcal{P}, 2) > 1.0$).

Using the concept of the premier priority assignment, we can transform Problem 1 to the following problem which exhibits a more suitable form for exploiting ML framework.

Problem 2: Given a task set τ on m processors, Find the premier priority assignment for τ on m processors.

We can solve Problem 1 by solving Problem 2, as stated in the following lemma.

Lemma 2: Suppose that we solve Problem 2 for a task set τ for which the existing heuristic priority assignments cannot be deemed schedulable by Lemma 1 on m processors. Then, the solution of Problem 2 is also a solution of Problem 1.

Proof: Suppose that the solution of Problem 2 is not a solution of Problem 1. Since $\Theta(\tau, \mathcal{P}, m) \leq 1.0$ holds for a premier priority assignment $\mathcal{P}$ by Definition 1, the premier priority assignment as a solution of Problem 2 is deemed schedulable by Lemma 1, implying that the assignment is also a solution of Problem 1. The supposition contradicts, thus proving the lemma. ■

Once PAL is trained with premier samples with n tasks on an m-processor platform using the sample structure design, neural architecture selection and training sample regulation, PAL becomes ready to infer the premier (or at least a schedulable) priority assignment for a target task set with n tasks on an m-processor platform, which allows to solve Problem 1 according to Lemma 2. Note that PAL is not always capable of solving Problem 1 (or Problem 2) for *all* task sets (even though premier samples are provided for training), as PAL tries to *infer* the premier priority assignment of each task set. However, it is useful/meaningful for PAL to solve Problem 1 even for a few task sets, if the task sets are not deemed schedulable by any prior work.

Then, the next issue is how to secure a number of premier samples with large n, to be addressed in Sections IV and V.

IV. Derivation of Inductive Properties for PAL

In this section, we first introduce a remaining challenge for PAL, i.e., generation of premier samples for large n. To address the challenge, we derive inductive properties that can generate large-size samples from small-size samples, from empirical observation of PAL and mathematical analysis of the target schedulability test in Lemma 1.

A. Challenge for PAL

In spite of its capability of inferring a schedulable priority assignment, PAL has not yet addressed a critical issue: how to generate samples each of whose output is the premier priority assignment? While it is tractable to find the premier priority assignment for a task set with small n by calculating the system hazard of all priority assignments for the task set by Lemma 1, the same cannot be said to hold for a task set with large n. According to Table I, we need 2.32×10000 seconds = 6.44 hours to test all priority assignments for 10,000 task sets with $n = 9$. Considering some task sets have no schedulable priority assignment (and therefore no premier priority assignments), it takes about up to 1 day to generate 10,000 task sets each of whose output is the premier priority assignment. Although the time depends on computing power, every computing platform eventually reaches some n^* that makes it intractable to test all priority assignments. Therefore, we face the following challenges for large n to solve the priority assignment problem with PAL: (i) it is intractable to generate premier samples by exhaustively testing all priority assignments, and (ii) it is also intractable to determine whether a priority assignment is premier or not (even if we have some candidates for the premier priority assignment). To address (i) and (ii), we assess the time-complexity of calculating the system hazard of every and one priority assignment of a task set, respectively.

Observation 1: For a task set with n tasks, it takes $O(n! \cdot n^2 \cdot \max_{\tau_i \in \tau} T_i)$ to calculate the system hazard of all priority assignments by Lemma 1, while it takes only $O(n^2 \cdot \max_{\tau_i \in \tau} T_i)$ to calculate the system hazard of a heuristic priority assignment (e.g., DMPO, D-CMPO or DkC) by the lemma.

To utilize Observation 1 and consider our target problem (i.e., Problem 1) is to find a schedulable priority assignment for a task set which is not deemed schedulable by existing heuristic priority assignments, we define the notion of a pseudo-premier priority assignment as follows.

Definition 2: A priority assignment $\mathcal{P}$ for a task τ on an m-processor platform is said to be *pseudo-premier*, if $\mathcal{P}$ yields a system hazard $\Theta(\tau, \mathcal{P}, m)$, which is no larger than 1.0 and smaller than all the system hazards associated with the existing heuristic priority assignments (i.e., DMPO, D-CMPO, and DkC) for τ on an m-processor platform.

We claim that a pseudo-premier priority assignment is capable of forming a sample for effective training. That is, by having a smaller system hazard, a pseudo-premier priority assignment is better than all existing heuristic priority assignments in terms of how accurately each assignment guides PAL towards a schedulable priority assignment of other task sets. Similar to premier samples, PAL gets ready to infer a pseudo-premier priority assignment for a target task set, by training it with pseudo-premier samples (whose output is a pseudo-premier priority assignment) the set of which has the same size as the target task set. Then, the target problem for PAL to solve can be expressed in conjunction with the notion of a pseudo-premier priority assignment as follows.

Problem 3: Given a task set τ on m processors, Find a pseudo-premier priority assignment for τ on m processors.

Similar to Problem 2, we claim that solving Problem 3

implies solving Problem 1 as follows.

Lemma 3: Suppose that we solve Problem 3 for a task set τ for which the existing heuristic priority assignments cannot be deemed schedulable by Lemma 1 on m processors. Then, the solution of Problem 3 is also a solution of Problem 1.

Proof: Suppose that the solution of Problem 3 is not a solution of Problem 1. Since $\Theta(\tau, \mathcal{P}, m) \leq 1.0$ holds for a pseudo-premier priority assignment $\mathcal{P}$ by Definition 1, the pseudo-premier priority assignment as a solution of Problem 3 is deemed schedulable by Lemma 1, meaning that the assignment is also a solution of Problem 1. The supposition contradicts, thus proving the lemma. ■

Using the notion of a (pseudo)premier priority assignment, in Sections IV-B and IV-C we will derive two types of inductive properties from empirical observation of PAL and from mathematical analysis of Lemma 1, both of which can generate (pseudo)premier samples with large n, from (pseudo)premier samples with small n.

B. Derivation of Inductive Properties from Lemma 1

We analyze our target schedulability test (i.e., Lemma 1) and derive two inductive properties from the schedulability test, one of which is stated in the following lemma.

Lemma 4: The following inductive property makes it possible to generate a (pseudo)premier sample with (m=x, n=y+1)[4] from a (pseudo)premier sample with (m=x, n=y).

I1. Suppose we know the premier priority assignment $\mathcal{P}$ for $\tau = \{\tau_i\}_{1 \leq i \leq y}$ on x processors. We construct τ^+ by adding a new task τ_{y+1} to τ, and $\mathcal{P}^+$ by adding τ_{y+1} as the lowest priority to $\mathcal{P}$. If we set C_{y+1} and T_{y+1} such that R_{y+1}/T_{y+1} is no larger than $\Theta(\tau, \mathcal{P}, x)$, then $\mathcal{P}^+$ is the premier priority assignment for τ^+ on x processors. Note that it may be impossible to set such C_{y+1} and T_{y+1}.
Also, the same holds for a pseudo-premier sample.

Proof: We prove the lemma by contradiction. Suppose that I1 is true except that $\mathcal{P}^+$ is not the premier priority assignment for τ^+ on x processors.

Since the priority of τ_{y+1} is the lowest, R_i under τ^+ with $\mathcal{P}^+$ is the same as R_i under τ with $\mathcal{P}$, for all $1 \leq i \leq y$. Therefore, $\Theta(\tau^+, \mathcal{P}^+, x) = \max\left(\Theta(\tau, \mathcal{P}, x), R_{y+1}/T_{y+1}\right)$ holds; by the condition of $R_{y+1}/T_{y+1} \leq \Theta(\tau, \mathcal{P}, x)$ in the supposition, $\Theta(\tau^+, \mathcal{P}^+, x) = \Theta(\tau, \mathcal{P}, x)$ holds. Since $\mathcal{P}^+$ is not the premier priority assignment for τ^+ on x processors, there exists a premier priority assignment $\mathcal{P}^{+\prime}$ ($\neq \mathcal{P}^+$) for τ^+ on x processors, which satisfies $\Theta(\tau^+, \mathcal{P}^{+\prime}, x) < \Theta(\tau^+, \mathcal{P}^+, x) = \Theta(\tau, \mathcal{P}, x)$.

We construct $\mathcal{P}'$ for τ, by removing the priority of τ_{y+1} from $\mathcal{P}^{+\prime}$. Once we remove a task from a task set without changing every other task's relative priority and the number of processors, the response time of every other task decreases or remains unchanged, and hence $\Theta(\tau, \mathcal{P}', x) \leq \Theta(\tau^+, \mathcal{P}^{+\prime}, x)$ holds. Therefore, $\Theta(\tau, \mathcal{P}', x) \leq \Theta(\tau^+, \mathcal{P}^{+\prime}, x) < \Theta(\tau^+, \mathcal{P}^+, x) = \Theta(\tau, \mathcal{P}, x)$ holds, which means $\mathcal{P}$ is not the premier priority assignment for τ on x processors; this contradicts the supposition.

The proof of I1 for a pseudo-premier sample is similar to that for the premier sample. In this case, we use the fact that if a heuristic priority assignment (i.e., DMPO, D-CMPO and DkC) yields its system hazard for τ equal to Θ, it cannot yield the system hazard for τ^+ less than Θ. ■

One may wonder how to set C_{y+1} and T_{y+1} such that R_{y+1}/T_{y+1} is no larger than $\Theta(\tau, \mathcal{P}, x)$ in I1. One simple way is to randomly generate T_{y+1}, find the largest C_{y+1} using binary search, and randomly set C_{y+1} within the largest value. In this case, we need to test at most $O(\log(T_{y+1}))$ task sets by Lemma 1. Note that it is possible for some (pseudo)premier samples to have no feasible pair of C_{y+1} and T_{y+1}; in this case, we cannot use I1 for the samples; therefore, we try other (pseudo)premier samples to derive larger-size (pseudo)premier samples.

We now introduce another inductive property derived from mathematical analysis of Lemma 1 as follows.

Lemma 5: The following inductive property makes it possible to generate the premier sample with (m=x+1, n=y+1) from the premier sample with (m=x, n=y).

I2. Suppose that we know the premier priority assignment $\mathcal{P}$ for $\tau = \{\tau_i\}_{1 \leq i \leq y}$ on x processors. We construct τ^+ by adding a new task τ_{y+1}, and $\mathcal{P}^+$ by adding τ_{y+1} as the highest priority to $\mathcal{P}$. If we set C_{y+1} to be no smaller than $\max_{1 \leq i \leq y}(T_i - C_i + 1)$, and T_{y+1} to $C_{y+1}/\Theta(\tau, \mathcal{P}, x)$, then $\mathcal{P}^+$ is the premier priority assignment for τ^+ on $(x+1)$ processors.

Proof: We prove the lemma by contradiction. Suppose that I2 is true except that $\mathcal{P}^+$ is not the premier priority assignment for τ^+ on $(x+1)$ processors.

Since the priority of τ_{y+1} is the highest and $C_{y+1} \geq \max_{1 \leq i \leq y}(T_i - C_i + 1)$ holds, τ_{y+1} always contributes the maximum interference (i.e., $R_k - C_k + 1$) to all other tasks τ_k (for $1 \leq k \leq y$) in the calculation of the RHS of Eq. (4). For τ, Eq. (1)'s convergence for τ_k implies the numerator sum in Eq. (1) should be within $\left[(R_k - C_k) \cdot x, (R_k - C_k + 1) \cdot x\right)$. For τ^+, $I_{k \leftarrow y+1}(R_k) = R_k - C_k + 1$ is added to the numerator sum in Eq. (1), and then the numerator sum becomes within $\left[(R_k - C_k) \cdot (x+1) + 1, (R_k - C_k + 1) \cdot (x+1)\right)$. After dividing the sum by $x+1$ and applying the floor function according to Eq. (1), we get $R_k - C_k$, yielding the convergence of Eq. (1) for τ^+. Hence, R_k under τ^+ with $\mathcal{P}^+$ on $(x+1)$ processors is equal to R_k under τ with $\mathcal{P}$ on x processors for all $1 \leq k \leq y$. Therefore, $\Theta(\tau^+, \mathcal{P}^+, x+1) = \max\left(\Theta(\tau, \mathcal{P}, x), R_{y+1}/T_{y+1}\right)$ holds. Considering the highest-priority task's response time is the same as its worst-case execution time, R_{y+1}/T_{y+1} is equal to C_{y+1}/T_{y+1}, which is the same as $\Theta(\tau, \mathcal{P}, x)$ in the supposition; hence, $\Theta(\tau^+, \mathcal{P}^+, x+1) = \Theta(\tau, \mathcal{P}, x) = C_{y+1}/T_{y+1}$ holds.

Since R_i cannot be smaller than C_i for every task τ_i, any priority assignment $\mathcal{P}^*$ for τ^+ on $(x+1)$ processors cannot yield $\Theta(\tau^+, \mathcal{P}^*, x+1)$ smaller than C_{y+1}/T_{y+1}. Since C_{y+1}/T_{y+1} is equal to $\Theta(\tau^+, \mathcal{P}^+, x+1)$, this implies that $\mathcal{P}^+$

[4]From now on, we will let a sample with (m, n) denote a sample with n tasks on m processors; the same applies to a priority assignment with (m, n) as well as a task set with (m, n).

is the premier priority assignment for τ^+ on $(x+1)$ processors, which contradicts the supposition. ■

Although applying I2 to a pseudo-premier sample for τ on x processors does not always yield a pseudo-premier sample for τ^+ on $(x+1)$ processors, the resulting priority assignment for τ^+ on $(x+1)$ processors is guaranteed to yield the minimum system hazard among all priority assignments for τ^+ on $(x+1)$ processors (which is C_{y+1}/T_{y+1} in the proof). This means that I2 yields a pseudo-premier sample for τ^+ on $(x+1)$ processors unless one of the heuristic priority assignments for τ^+ on $(x+1)$ processors yields the minimum system hazard. Therefore, I2 can be used to generate a pseudo-premier sample with larger m, from that with smaller m, as follows: applying I2 to a pseudo-premier sample, checking whether the resulting priority assignment is pseudo-premier or not, and using the assignment if it is pseudo-premier.

C. Derivation of Inductive Properties from PAL

We derive two other inductive properties from empirical observation of PAL, which can generate large-size samples from small-size ones. The first inductive property is given as follows.

Observation 2: The following inductive property makes it possible to generate (pseudo)premier samples with large n from (pseudo)premier samples with small n, which was observed in our experiments.

- I3. If PAL is trained with a sufficiently large number of (pseudo)premier samples with (m=x, n≤y), it is possible to infer a (pseudo)premier priority assignment for a task set with (m=x, n=y+α), with a low probability, where $\alpha \geq 1$.
- I3′. If we add a small number of (pseudo)premier samples with (m=x, n=y+α) to the training set for I3, the probability of inferring a (pseudo)premier priority assignment for a task set with (m=x, n=y+α) increases dramatically.

The inductive property of I3 comes from the benefit and limitation of our base neural architecture (i.e., pointer networks). While it is *possible* for the neural architecture to infer an output of a sample without including the same-size samples in the training set, the probability for the result to be desirable is low. Thus, it is not only effective but also mandatory to derive and use I1 in securing a number of (pseudo)premier samples (and therefore in solving the target priority assignment problem by our ML framework) by utilizing I3′.

We present example experimental results for I3 and I3′ (our implementation settings will be presented in Section V). First, once we train PAL with a set of 500,000 training samples, which consists of 500,000/13 pseudo-premier samples for each pair of (m=4, 5≤n≤17), PAL succeeds in inferring 8 pseudo-premier priority assignments out of 1,000 tested task sets with (m=4, n=18). Second, we directly derive 100 pseudo-premier samples of (m=4, n=18) from samples with (m=4, n=17) using I1, add the 100 samples to the training set which now consists of 500,100 samples, and re-train PAL with them. As a result, PAL infers 148 pseudo-premier priority assignments out of the same 1,000 tested task sets with (m=4, n=18), which is a significant improvement over the inference without same-size training samples generated by I1. Since it takes 168 seconds to try 1,000 tested task set, the expected time to generate $10,000$ pseudo-premier samples in the former and that in the latter are 210,000 seconds (≈ 58.3 hours) and 11,351 seconds (≈ 3.2 hours), respectively.

In addition to I3, we observe another inductive property from the experimental results of PAL.

Observation 3: The following inductive property makes it possible to generate (pseudo)premier samples with large m from (pseudo)premier samples with small m, which was observed in our experiments.

- I4. If PAL is trained with a sufficiently large number of (pseudo)premier samples with (m=x, n≤y), it is possible to infer a (pseudo)premier priority assignment for a task set with (m=x+β, n=y), with a low probability, where $\beta \geq 1$.
- I4′. If we add a small number of (pseudo)premier samples with (m=x+β, n=y) to the training set for I4, the probability of inferring a (pseudo)premier priority assignment for a task set with (m=x+β, n=y) increases dramatically.

Unlike I3, the inductive property of I4 comes not only from the base neural architecture, but also from the target schedulability test. Suppose that we have a task set τ and its pseudo-premier priority assignment $\mathcal{P}$ on a platform with x processors. Let R_i and R'_i denote the response time of $\tau_i \in \tau$ with $\mathcal{P}$ on x processors and $(x+\beta)$ processors, respectively. In Eq. (1), we can observe the difference between R_i and R'_i according to the change of the denominator from x to $(x+\beta)$. Considering the fact that the change applies to all tasks, there exist many task sets in each of which the task having the largest R_i/T_i with $\mathcal{P}$ on x processors is the same as the task having the largest R'_i/T_i with $\mathcal{P}$ on $(x+\beta)$ processors. In such task sets, $\mathcal{P}$ tends to be a pseudo-premier priority assignment for τ even on $(x+\beta)$ processors. Therefore, it is possible for features of pseudo-premier samples with (m=x, n=y) to guide PAL towards a pseudo-premier assignment of other task sets with (m=x+β, n=y).

V. Implementation of PAL

In this section, we describe how to systematically implement PAL in detail. We first present the neural architecture settings then the task set generation process, which can be used not only for generating training samples, but also for evaluating PAL in Section VI. The sample generation process addresses the following two important issues:

- G1. How to generate unbiased pseudo-premier samples, and
- G2. How to reduce time for the sample-generation process and make the process tractable for large n.

Then, the training process will be described after the sample generation process.

A. Setting Neural Architecture

To implement PAL, we use two PCs equipped with an Intel Zeon Silver 4216 processor, a 32GB RAM, and an Titan RTX

GPGPU with 24GB memory. Our baseline neural architecture of pointer networks [9] is set up as follows: encoder/decoder built with LSTM [28] cells with 512 hidden units trained with learning rate of 0.001 and batch size of 512 (empirically chosen), which is optimized by Adam optimizer [29]. Sparse categorical cross-entropy was used as a loss function. The input data consists of numerical sequences of (T_i,C_i) and the order of tasks as shown in Eq. (5). The number of training samples for each (m, n) is empirically chosen as 500,000, to be detailed in Section V-D.

B. Generating Target Task Sets

Since we aim to solve Problem 1, the qualification of target task sets are (C1) not to be schedulable by Lemma 1 with one of the heuristic priority assignments (i.e., DMPO, D-CMPO, DkC), and (C2) to be schedulable by gFP with some priority assignment. The qualification is used for the entire implementation of PAL: generating pseudo-premier samples whose task sets satisfy the qualification, training PAL with them, and inferring a pseudo-premier priority assignment for each target task set that satisfies the qualification. Therefore, we need to generate a large number of task sets that satisfy the qualification.

To generate a target task set, we adapt a popular set generation method [30], [12], [31], and consider 10 distributions for C_i/T_i to generate a variety of task sets: binomial distribution with constants 0.1, 0.3, 0.5, 0.7 and 0.9, and exponential distribution with constants 0.1, 0.3, 0.5, 0.7 and 0.9. Each task set for given (m, n) is generated as follows. First, we randomly choose one of the 10 distributions. Second, we generate n tasks by repeating the following procedure n times for each τ_i: to generate the period (T_i) in [10,1000] according to the log-uniform distribution [32], to generate C_i/T_i according to the chosen distribution, and to calculate C_i as a multiplication between the generated T_i and C_i/T_i. Third, we discard the task set if (C1) or (C2) is violated. Since there is no exact known condition for (C2), we apply a necessary test in [17], called the C-RTA test. We also discard the task set if it is schedulable by OPA with DA-LC [13], implying that a schedulable priority assignment is already identified by OPA-compatible version of Lemma 1. If the generated task set is not discarded in the third step then we use it else we repeat the entire process until the newly-generated task set passes the third step.

C. Securing Unbiased Samples with Reasonable Time

We generate pseudo-premier samples for the following pairs of (m, n): (m=2, $5 \leq n \leq 15$), (m=4, $5 \leq n \leq 20$), and (m=6, $7 \leq n \leq 25$), where m is the number of processors and n is the number of tasks in each task set. To generate a pseudo-premier sample with each (m, n), whether n is larger than n^* or not is important; n^* denotes the largest n that makes it tractable to generate a given number of pseudo-premier samples by exhaustively testing all priority assignments of each task set by Lemma 1, and n^* is set to 8 considering our computing platform and the maximum number of samples with each (m, n) to be generated. Due to the intractability, the case of $n > n^*$ necessitates utilizing the inductive properties of PAL to generate a large-size pseudo-premier sample. One may consider the following two approaches: (i) directly deriving a target-size pseudo-premier sample from an existing smaller-size pseudo-premier sample (i.e., applying I1 and/or I2 in Section IV-B), and (ii) training PAL with smaller-size samples only, trying to infer a priority assignment for a task set generated as in Section V-B, and using the sample only if the assignment is pseudo-premier (i.e., applying I3 and/or I4 in Section IV-C). Although (i) is capable of immediately generating each pseudo-premier sample without relying PAL's inference process, the task sets in the generated samples are severely biased in terms of task utilization (i.e., favorable to G2, not to G1). This because, I1 and I2, respectively, add a task with low- and high-utilization to the task set of an existing pseudo-premier sample. On the other hand, since (ii) tries to infer a pseudo-premier priority assignment for task sets randomly generated by Section V-B, (ii) can generate a variety of pseudo-premier samples in terms of task utilization. However, it takes a long time to generate a given number of samples (i.e., favorable to G1, not to G2); for example, if we apply I3 only, it is expected to take 58.2 hours to generate 10,000 pseudo-premier samples with (4, 18) as mentioned in Section IV-C.

To make the sample-generation process to achieve G1 and G2 together, we systematically utilize the inductive properties of PAL and additional techniques. For $m = 2$, we generate 100,000, $100{,}000/2$, $100{,}000/3$ and $100{,}000/4$ pseudo-premier samples (that are also premier) by exhaustively testing all priority assignments of target task sets by Lemma 1, for $n = 5$, 6, 7 and 8, respectively. We will explain later why the number of generated task sets varies with n; in short, training for $n = 5$ uses 100,000 task sets with $n = 5$, while training for $n = 6$ uses $100{,}000/2$ task sets with $n = 5$ and $100{,}000/2$ task sets with $n = 6$. We then apply the data augmentation technique with $\rho = 5$ (to be explained), securing $100{,}000 \times 5$, $100{,}000/2 \times 5$, $100{,}000/3 \times 5$ and $100{,}000/4 \times 5$ pseudo-premier samples for $n = 5$, 6, 7 and 8, respectively. We then utilize I1 in Section IV-B and I3′ in Section IV-C to generate pseudo-premier samples for each $n = 9, 10, 11, \ldots$ in a sequential manner as follows. First, we select $N^{\mathsf{unbiased}} = 500,000$ generated unbiased pseudo-premier samples, which consist of $500{,}000/(n-5)$ samples with (2, 5), those with (2, 6), ..., and those with (2, $n-1$); also, we generate $N^{\mathsf{biased}} = 100$ biased pseudo-premier samples with (2, n), by applying I1 to 100 unbiased pseudo-premier samples with (2, $n-1$). Second, we apply I3′ as follows: train PAL with a set of the 500,000 unbiased pseudo-premier samples and the 100 biased ones, and repeat to infer a pseudo-premier priority assignment for a task set with (2, n) generated by Section V-B until we have $100{,}000/(n-4)$ pseudo-premier samples with (2, n). Finally, we apply the data augmentation technique with $\rho = 5$, securing $100{,}000/(n-4) \times 5$ pseudo-premier samples with (2, n). Note that although the generation of pseudo-premier samples with $n > n^* = 8$ requires at most 125,000 pseudo-premier samples for every $5 \leq n \leq 8$, the number of generated samples for $n = 5$, 6 and 7 is larger than 125,000; the rest of the samples will be used for training process for $n = 5$, 6 and 7 itself, to be detailed in Section V-D.

The sample-generation process for $m = 4$ is the same as that for $m = 2$. For $m = 6$, the difference is the smallest valid n for the sample generation, which is $m + 1 = 7$. Note that a set of n ($\leq m$) tasks is trivially schedulable on m processors, making any priority assignment meaningless; therefore, a

sample with (m, n) is valid only if $n \geq m + 1$. Therefore, we generate 100,000 and 100,000/2 pseudo-premier samples by exhaustively testing all priority assignments by Lemma 1 for $n = 7$ and $n = 8$, respectively; the remaining process is similar to that for $m = 2$.

A data augmentation technique is used in the above sample-generation process in order to reduce the time to generate a given number of pseudo-premier samples, and it operates as follows. For a given generated pseudo-premier samples, we shuffle the order of tasks and the corresponding priority assignment together, yielding a new pseudo-premier sample, which is logically the same as the original one. For example, if Eq. (5) is a pseudo-premier sample, we may generate the following duplicated sample: $\texttt{Input}[[3,2][5,1][10,5]] \rightarrow \texttt{Output}[1,2,3]$ by changing the position of the first and second elements in both input and output. Note that despite the logical equivalence, PAL can interpret the duplicated sample as different due to their different input/output order. Let ρ denote how many samples we can secure from an original pseudo-premier sample; for PAL, we empirically choose $\rho = 5$, securing five samples from every original sample.

Our implementation for the sample generation can achieve G1 and G2 at the same time as follows. First, when we apply I3′ for generating samples with (m, n), we add $N^{\text{biased}} = 100$ biased samples with (m, n) derived from I1. While the biased same-size samples accelerate the probability of inferring a pseudo-premier priority assignment for a set of n tasks (thus achieving G2), the number of such samples is far smaller than the number of unbiased different-size samples (i.e., $N^{\text{biased}} = 100 << N^{\text{unbiased}} = 500{,}000$), thus helping achieve G1. Second, when we generate samples with (m, n), we reuse the generated samples with (m, n') for every $n' < n$. This implementation helps the sample generation process to be scalable to large n in that the number of samples to be generated by I3′ with I1 decreases as n increases (achieving G2). For example, in case of $m = 4$, the number is $100{,}000/5 = 20{,}000$ for $n = 9$, while the number is $100{,}000/16 = 6{,}250$ for $n = 20$. Third, by applying the data augmentation technique with ρ, we reduce the time for generating samples to $1/\rho$, which helps achieve G2. We would like to stress that our implementation of the sample generation is designed to overcome an inherent disadvantage of supervised learning, needing a number of unbiased samples even for large n. Note that $N^{\text{biased}} = 100$ and $N^{\text{unbiased}} = 500{,}000$ are empirically set for better schedulability performance of PAL. For example, if we increase N^{biased}, it makes training sets more biased, yielding less schedulability performance of PAL for large n eventually.

D. Training Generated Samples

To infer a pseudo-premier priority assignment for a target task set with n tasks on m processors, we train PAL with a set of $N^{\text{unbiased}} = 500{,}000$ unbiased pseudo-premier samples, which varies with (m, n) as follows. If $m = 2$ or $m = 4$, for every $5 \leq y \leq n$, we select $500{,}000/(n-4)$ samples with (m, y) among the unbiased pseudo-premier samples with (m, y) generated as described in Section V-C. For example, the set for (4, 5) consists of 500,000 unbiased pseudo-premier samples with (4, 5) only, while the set for (4, 20) consists of $500{,}000/16 = 31{,}250$ unbiased pseudo-premier samples with (4, y) for every $5 \leq y \leq 20$. When it comes to $m = 6$, the set consists of $500{,}000/(n-6)$ unbiased pseudo-premier samples with (6, y) for every $7 \leq y \leq n$. In fact, the sample-generation process in Section V-C was already designed to provide the required number of unbiased pseudo-premier samples to the training process for each (m, n).

E. Discussion

Suppose we have completed the sample-generation process for all pairs of (m, n) mentioned at the beginning of Section V-C, but later we need to find a pseudo-premier priority assignment for a task set with (x, y) which does not belong to (m, n) pairs of the generated samples. We describe how to utilize the proposed inductive properties and sample-generation process for each of the following three cases: (i) samples with (x, y') for $y' < y$ have already been generated; (ii) case (i) does not hold, but samples with ($x-1$, y') for $y' \leq y$ have already been generated; and (iii) neither (i) nor (ii) holds.

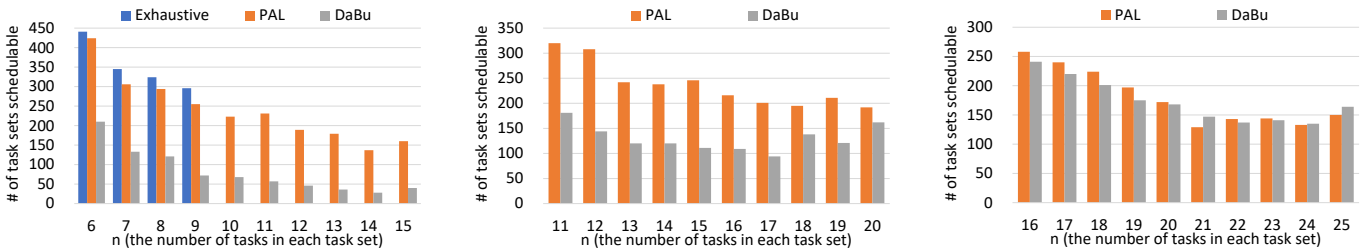
For (i), we continue generating samples for up to (x, y) as described in Section V-C. For example, if (x, y) = (2, 17), we first generate $500{,}000/12$ unbiased pseudo-premier samples with (2, 16) by applying I3′ with I1 and the data augmentation technique, and then generate $500{,}000/13$ unbiased pseudo-premier samples with (2, 17) in the same way. Second, we train PAL with a set of 500,000 samples, which consist of $500{,}000/13$ unbiased pseudo-premier samples with (2, y') for every $5 \leq y' \leq 17$. Finally, we let PAL infer a priority assignment for the target task set with (2, 17).

As to (ii), it is inefficient to generate all the samples with (x, y') for every $5 \leq y' \leq y$ in a sequentially manner. Instead, we utilize I2 in Section IV-B and I4′ in Section IV-C as explained below for (x, y) = (3, 15). First, we select $N^{\text{unbiased}} = 500{,}000$ generated unbiased pseudo-premier samples, which consist of $500{,}000/11$ samples with (2, y') for every $5 \leq y' \leq 15$; we also generate $N^{\text{biased}} = 100$ biased pseudo-premier samples with (3, 15), by applying I2 to 100 unbiased pseudo-premier samples with (2, 14). Second, we apply I4′ as follows: train PAL with a set of the 500,000 unbiased pseudo-premier samples and the 100 biased ones, and repeat the inference of a pseudo-premier priority assignment for a task set with (3, 15) generated as in Section V-B until we have a sufficiently large number of pseudo-premier samples with (3, 15); we thereafter apply the data augmentation technique. Finally, we train PAL with the generated samples with (3, 15) only, and let PAL infer a priority assignment for the target task set with (3, 15).

To resolve the case (iii), we need to exploit the strategy for (i) and that for (ii) together. For example, if (x, y) = (3, 20), we generate samples with (2, 16), those with (2, 17), ..., and those with (2, 20) in a sequential manner by applying the strategy for (i). Then, using the strategy for (ii), we generate samples with (3, 20), train PAL with them, and infer a priority assignment for the target task set with (3, 20).

VI. Evaluation

We generate 1,000 task sets as described in Section V-B for the following pairs of (m, n): (m=2, $6 \leq n \leq 15$), (m=4, $11 \leq n \leq 20$), and (m=6, $16 \leq n \leq 25$). Note that task sets for training and those for evaluation are separated without any



(a) $m = 2$ with $6 \leq n \leq 15$ (b) $m = 4$ with $11 \leq n \leq 20$ (c) $m = 6$ with $16 \leq n \leq 25$

Fig. 1. The number of task sets whose schedulable priority assignment subject to Lemma 1 is identified by Exhaustive, PAL and DaBu

overlap. We count the number of task sets proven schedulable by the following techniques among the 1,000 generated task sets for each (m, n): (i) the priority assignment by PAL with Lemma 1, (ii) a limited backtracking technique to find a schedulable priority assignment with Lemma 1 [17] (denoted by DaBu), and (iii) the three heuristic priority assignments with the schedulability test in [5] that is known to dominate other OPA-incompatible tests in [2], [3], [4] including Lemma 1 (denoted by ZLL). We would like to remind that all tested task sets are schedulable by neither the three heuristic priority assignments with Lemma 1 nor OPA with DA-LC [13], according to Section V-B; therefore, (ii) and (iii) are the only existing studies that may deem each tested task set gFP-schedulable.

Since (ii) is the only existing study that finds a schedulable priority assignment subject to Lemma 1 for the tested task sets, we first compare (i) with (ii) in Section VI-A, which shows the effectiveness of PAL in solving our target priority assignment problem. We next analyze how much the solution helps cover additional task sets each of which has not been proven schedulable by any existing approaches for gFP, i.e., (ii) and (iii), in Section VI-B. Finally, we discuss the time required for sample generation, model training and inference of priority assignment by PAL in Section VI-C.

A. Evaluation of Solving the Priority Assignment Problem

In this subsection, we evaluate how effectively PAL solves Problem 1—finding a priority assignment deemed schedulable by Lemma 1. To this end, we compare the number of task sets deemed schedulable by PAL with that by DaBu, which is the only existing study that may find a priority assignment schedulable by Lemma 1 for our tested task sets. We also count the number of task sets each of which has at least one priority assignment schedulable by Lemma 1 among 1,000 tested task sets for each pair of (m, n). This is possible by exhaustively testing all priority assignments of each task set by Lemma 1; we denote the number as Exhaustive. Due to time-intractibility of investigating all priority assignments of the tested task sets with large n, we can have the results of Exhaustive only for $n \leq 9$.

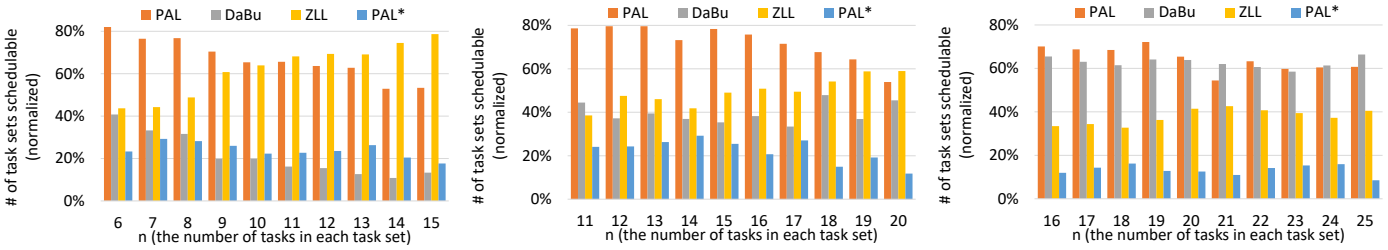
If we focus on the ratio between the number of task sets deemed schedulable by PAL and that by Exhaustive for each pair of (m, n), the ratio is 424/441=96.1%, 306/345=88.7%, 294/324=90.7%, and 255/296=86.1%, respectively for $n = 6, 7, 8$ and 9 with $m = 2$ as shown in Fig. 1(a). That is, PAL can identify a schedulable priority assignment for most (i.e., 86.1%–96.1%) of task sets with $n \leq 9$, which have at least one priority assignment schedulable by Lemma 1 but are schedulable by none of the three heuristic priority assignments with Lemma 1. Also, if we consider that Exhaustive tends to decrease as n increases, we expect a similar trend holds for $10 \leq n \leq 15$ with $m = 2$.

While we analyze PAL's performance compared to the absolute standard (i.e., Exhaustive) only for $m = 2$, we can discuss PAL's performance by comparing PAL and DaBu as shown in Fig. 1. For $m = 2$ and $m = 4$, PAL exhibits far better schedulability performance than DaBu; the ratio between the number of task sets schedulable by PAL and that by DaBu is between 201.9% (424/210 for $n = 6$) and 497.2% (179/36 for $n = 13$) for $m = 2$, and between 118.5% (192/162 for $n = 20$) and 221.6% (246/111 for $n = 15$) for $m = 4$. On the other hand, PAL is comparable to DaBu for $m = 6$ in that the ratio is between 87.8% (129/147 for $n = 21$) and 112.6% (197/175 for $n = 19$). Note that PAL can improve the schedulability of gFP even in case that the number of task set schedulable by PAL is smaller than that by DaBu, as long as PAL can cover some additional gFP-schedulable task sets which are not deemed schedulable by other existing studies including DaBu; this will be discussed in Section VI-B.

In summary, the evaluation results demonstrate that our ML framework succeeded in solving the target priority assignment problem as PAL finds a schedulable priority assignment for most of task sets covered by Exhaustive and exhibits schedulability performance superior or at least comparable to DaBu, which is the only existing study capable of yielding a schedulable priority assignment subject to Lemma 1 for the tested task sets.

B. Evaluation of Schedulability Improvement of gFP

We now evaluate the effectiveness of PAL in covering additional gFP-schedulable task sets, which are deemed schedulable by none of the existing studies. Fig. 2 shows the number of task sets schedulable by PAL, DaBu, ZLL and PAL*, respectively, normalized by the number of task sets schedulable by at least one of them, for each pair of (m, n), where the number of task sets schedulable by PAL* means the number of task sets schedulable by PAL, but not by both of DaBu and ZLL. The effectiveness can be demonstrated by the bar for PAL*, as the bar represents the ratio *between* the number of task sets schedulable by PAL only *and* that schedulable by at least one of PAL, DaBu and ZLL. As shown in Fig. 2, the bar for PAL* is 17.7%–29.3% for $m = 2$, 11.8%–29.2% for $m = 4$, and 8.5%–16.2% for $m = 6$. The evaluation results confirm that PAL can be a good alternative if a task set is proven schedulable by none of existing studies for gFP.



(a) $m = 2$ with $6 \leq n \leq 15$ (b) $m = 4$ with $11 \leq n \leq 20$ (c) $m = 6$ with $16 \leq n \leq 25$

Fig. 2. The number of task sets schedulable by PAL, DaBu, ZLL and PAL*, normalized by the number of task sets schedulable by at least one of them

Next, we show that PAL outperforms DaBu and ZLL under many settings including large-size task sets, of which it is time-intractible to test all possible priority assignments. As shown in Fig. 2, the bar for PAL is the largest for $6 \leq n \leq 10$ with $m = 2$,[5] $11 \leq n \leq 19$ with $m = 4$, and $16 \leq n \leq 20$ & $22 \leq n \leq 23$ with $m = 6$. Unlike the bar for PAL*, the relative schedulability performance of PAL compared to the maximum of DaBu and ZLL tends to decrease as n increases. Note that the figure shows DaBu and ZLL cannot always outperform each other, which has never been presented before.

In summary, PAL can improve gFP-schedulability by covering additional gFP-schedulable task sets, which have been deemed schedulable by none of existing studies. Also, the schedulability performance of PAL is superior to that of existing studies under many settings.

C. Time Taken to Implement and Use PAL

The time taken to implement PAL can be divided into three parts: (i) generation of all necessary samples, (ii) training PAL with the samples for every pair of (m, n) separately, and (iii) priority assignment inference of each tested task set by PAL. For (i), it takes around two weeks to secure the required number of task sets for every pair of (m, n). For example, it takes 1,868, 12,238 and 20,864 seconds to secure 500,000/6, 500,000/11, 500,000/16 samples with (m=4, n=10), (m=4, n=15) and (m=4, n=20), respectively. For (ii), it takes a few minutes to train PAL with 500,000 samples with given (m, n), e.g., 525, 608 and 610 seconds, respectively for (4, 10), (4, 15) and (4, 20). For (iii), it takes a few seconds for PAL to output the resulting priority assignment of a target task set. For example, it takes 0.59, 1.72 and 2.78 seconds to make the resulting priority assignment of a task set, respectively for (4, 10), (4, 15) and (4, 20). The times for (i), (ii) and (iii) are mostly dependent of n. Note that it takes 0.08 and 0.45 second to check whether a task set is gFP-schedulable by DaBu and ZLL, respectively for (4, 20).

Therefore, once we complete (i) and (ii) in around two weeks, we can infer a priority assignment for a task set within a few seconds. We would like to stress that we need to perform (i) and (ii) *only once* regardless of the number of task sets tested by PAL to infer their priority assignments. It should be noted that the times for (i), (ii) and (iii) highly depend on the underlying computing system. The time can be reduced significantly if we use a cluster of high-performance servers. For example, considering that (i) and (ii) can be parallelized with some overhead to divide computation workloads (i.e., partitions of task sets) and combine results, the times for (i) and (ii) can be reduced to a few hours, if we use a cluster consisting of 100 PCs each of which is the same as the one we used.

[5]The schedulability performance of ZLL is much higher than that of PAL under $m = 2$ with $14 \leq n \leq 15$. We conjecture that it is because the schedulability test of ZLL can deem a task set schedulable even though the task set cannot be deemed schedulable by Lemma 1 with any priority assignment. For example, out of 225, 177, 187 and 220 task sets schedulable by ZLL for $m = 2$ with $n = 6, 7, 8$ and 9, respectively, 84, 74, 73 and 90 task sets cannot have any priority assignment schedulable by Lemma 1, and this trend is expected to stand out for $m = 2$ with $14 \leq n \leq 15$.

VII. Conclusion and Limitation

In this paper, we have developed the ML framework PAL, specialized for the priority assignment problem for gFP. Our implementation of PAL succeeded in finding a schedulable priority assignment for a number of task sets, each of which has not yet been proven schedulable by any existing approach for gFP. We expect the incorporation of RT domain knowledge in the ML framework will help solve other challenging problems in the RT research field.

Thanks to the proposed novel concepts and techniques tailored to solve the priority assignment problem using supervised learning, PAL is found more effective than prior work for task sets with some range of large n, whose schedulable priority assignment cannot be found via exhaustive search of all priority assignments. However, we could not completely resolve the scalability issue as (i) PAL needs to construct the pseudo-premier samples incrementally and (ii) in general, it becomes more difficult to guess a schedulable priority assignment for a set with n tasks as n increases. Unlike existing non-ML techniques, (ii) in turn affects the time to generate samples for PAL.

There are two directions to extend this paper: one for accommodating a more general task model, and the other for improving schedulability performance. First, we consider extending this framework to the constrained-deadline task model in which each task's relative deadline is no larger than its period (or the minimum separation). This is more challenging because a constrained-deadline task needs three parameters to express while an implicit-deadline one needs only two. We can develop ways to adapt PAL to be effective for more task parameters. Second, we may investigate other criteria that select the "best" priority (than the system hazard), which are effective to implicit-deadline task sets and/or constrained-deadline task sets. For example, it would be interesting to apply the scaling factor, by which all task execution times can be multiplied without compromising the task set's schedulability.

ACKNOWLEDGEMENT

We thank all reviewers and the shepherd for their valuable comments and suggestions.

This research was supported by the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (NRF-2019R1A2B5B02001794, NRF-2021R1A2B5B02001758, NRF-2020M3C1C2A01080819). The work reported in this paper was also supported in part by the US Office of Naval Research under Grant No. N00014-18-1-2141. Jinkyu Lee is the corresponding author.

REFERENCES

[1] H. Zhou, S. Bateni, and C. Liu, "S^{3}DNN: Supervised streaming and scheduling for gpu-accelerated real-time DNN workloads," in *Proceedings of IEEE Real-Time Technology and Applications Symposium (RTAS)*, 2018, pp. 190–201.

[2] M. Bertogna and M. Cirinei, "Response-time analysis for globally scheduled symmetric multiprocessor platforms," in *Proceedings of IEEE Real-Time Systems Symposium (RTSS)*, 2007, pp. 149–158.

[3] N. Guan, M. Sitgge, W. Yi, and G. Yu, "New response time bounds for fixed priority multiprocessor scheduling," in *Proceedings of IEEE Real-Time Systems Symposium (RTSS)*, 2009, pp. 387–397.

[4] N. Guan, M. Han, C. Gu, Q. Deng, and W. Yi, "Bounding carry-in interference to improve fixed-priority global multiprocessor scheduling analysis," in *Proceedings of IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2015, pp. 11–20.

[5] Q. Zhou, G. Li, and J. Li, "Improved carry-in workload estimation for global multiprocessor scheduling," *IEEE Transactions on Parallel Distributed Systems*, vol. 28, no. 9, pp. 2527–2538, 2017.

[6] C. Liu and J. Layland, "Scheduling algorithms for multi-programming in a hard-real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[7] N. Audsley, "Optimal priority assignment and feasibility of static priority tasks with arbitrary start times," Department of Computer Science, University of York, Tech. Rep. YCS164, 1991.

[8] S. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Prentice Hall, 2002.

[9] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Proceedings of International Conference on Neural Information Processing Systems (NIPS)*, 2015, pp. 2692–2700.

[10] D.-T. Peng and K. G. Shin, "A new performance measure for scheduling independent real-time tasks," *Journal of Parallel and Distributed Computing*, vol. 19, no. 1, pp. 11–26, 1993.

[11] A. Mok, "Fundamental design problems of distributed systems for the hard-real-time environment," Ph.D. dissertation, Massachusetts Institute of Technology, 1983.

[12] M. Bertogna, M. Cirinei, and G. Lipari, "Schedulability analysis of global scheduling algorithms on multiprocessor platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, pp. 553–566, 2009.

[13] R. Davis and A. Burns, "Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems," *Real-Time Systems*, vol. 47, pp. 1–40, 2011.

[14] M. Bertogna, "Real-time scheduling for multiprocessor platforms," Ph.D. dissertation, Scuola Superiore Sant'Anna, Pisa, 2007.

[15] B. Andersson and J. Jonsson, "Fixed-priority preemptive multiprocessor scheduling: to partition or not to partition," in *Proceedings of IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2000, pp. 337–346.

[16] R. Davis and A. Burns, "Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems," in *Proceedings of IEEE Real-Time Systems Symposium (RTSS)*, 2009, pp. 398–409.

[17] ——, "On optimal priority assignment for response time analysis of global fixed priority pre-emptive scheduling in multiprocessor hard real-time systems," Department of Computer Science, University of York, Tech. Rep. YCS-2009-451, 2010.

[18] T. Ae and R. Aibara, "Programmable real-time scheduler using a neurocomputer," *Real-Time Systems*, vol. 1, no. 4, pp. 351–363, 1990.

[19] C. Cardeira and Z. Mammeri, "Neural networks for multiprocessor real-time scheduling," in *Proceedings of Euromicro Conference on Real-Time Systems (ECRTS)*, 1994, pp. 59–64.

[20] ——, "Preemptive and non-preemptive real-time scheduling based on neural networks," in *Proceedings of International Federation of Automatic Control (IFAC)*, vol. 28, no. 22, 1995, pp. 67–72.

[21] ——, "Neural network versus max-flow algorithms for multiprocessor real-time scheduling," in *Proceedings of Euromicro Conference on Real-Time Systems (ECRTS)*, 1996, pp. 175–180.

[22] C.-S. Zhang, P.-F. Yan, and T. Chang, "Solving job-shop scheduling problem with priority using neural network," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, 1991, pp. 1361–1366.

[23] R.-M. Chen and Y.-M. Huang, "Competitive neural network to solve scheduling problems," *Neurocomputing*, vol. 37, no. 1–4, 2001.

[24] S. Yang, D. Wang, T. Chai, and G. Kendall, "An improved constraint satisfaction adaptive neural network for job-shop scheduling," *Journal of Scheduling*, vol. 13, pp. 17–38, 2010.

[25] Z. Guo and S. K. Baruah, "A neurodynamic approach for real-time scheduling via maximizing piecewise linear utility," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 27, no. 2, pp. 238–248, 2016.

[26] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *Proceedings of International Conference on Learning Representations (ICLR)*, 2015, pp. 1–15.

[27] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," *arXiv preprint arXiv:1611.01144*, 2016.

[28] N. Srivastava, E. Mansimov, and R. Salakhudinov, "Unsupervised learning of video representations using LSTMs," in *Proceedings of International Conference on Machine Learning (ICML)*, 2015, pp. 843–852.

[29] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proceedings of International Conference on Learning Representations (ICLR)*, 2015, pp. 1–15.

[30] T. P. Baker, "Comparison of empirical success rates of global vs. paritioned fixed-priority and EDF scheduling for hand real time," Dept. of Computer Science, Florida State University, Tallahasee, Tech. Rep. Technical Report TR-050601, 2005.

[31] J. Lee, A. Easwaran, and I. Shin, "Maximizing contention-free executions in multiprocessor scheduling," in *Proceedings of IEEE Real-Time Technology and Applications Symposium (RTAS)*, 2011, pp. 235–244.

[32] P. Emberson, R. Stafford, and R. I. Davis, "Techniques for the synthesis of multiprocessor tasksets," in *International Workshop on Analysis Tools and Methogologies for Embedded and Real-Time Systems*, 2010, pp. 6–11.