AI-Based Scheduling for Adaptive Time-Triggered Networks

Carlos Lua, Daniel Onwuchekwa, Roman Obermaisser
Department of Embedded Systems
University of Siegen
Siegen, Germany
carlos.luamorales@uni-siegen.de

***Abstract*—Time-triggered systems are ideal for safety-critical systems due to the inherent determinism and better fault tolerance. However, the current trend of adaptation in time-triggered systems is typically limited to switching between a small number of precomputed schedules. Artificial n e ural n e tworks (A NNs) have the potential to overcome this limitation. Adaptation in time-triggered systems requires switching to a new schedule, which satisfies c o rrectness c o nstraints a n d m e ets t h e t i ming requirements. Computing such a schedule is time consuming, thus in general, not feasible at runtime. ANNs provide the opportunity for learning schedules and thus inferring new schedules at runtime with short delays. However, ANNs have not been sufficiently e x ploited for optimising time-triggered scheduling. In this paper, an ANN is implemented to learn schedules to provide adaptation for time-triggered systems while ensuring that collision and precedence constraints are met. In our evaluation, the AI-based scheduler is compared with conventional scheduling algorithms such as list scheduling and genetic algorithm in terms of makespan and computation time. The results show the AI-based scheduler's potential when increasing the scheduling problem's complexity.**

***Keywords*—Time-Triggered systems, scheduling, adaptation, machine learning**

I. Introduction

In safety-critical applications, essential properties of time-triggered systems must be preserved. These exhibited properties include avoidance of resource contention without dynamic resource arbitration, implicit synchronization, guaranteeing of timing constraints, implicit flow c ontrol, a nd f ault containment [1]. Scheduling in time-triggered systems is traditionally carried out offline. The scheduling strategies for time-triggered systems include mathematical techniques, heuristics, and neighbourhood search. Adaptation in time-triggered systems is motivated by the need to provide energy efficient operation, fault recovery, and adaptation to changing environmental conditions.

Time-triggered systems have the potential to implement fault recovery by switching to configurations t h at d o n o t u s e failed resources. These can be done by modifying the allocation of services to failed resources, substituting failed services, or switching to degraded service modes. Other run time changes that can require the reconfiguration o f t h e t i me-triggered network include energy management and user-defined operating modes. These run time changes are referred to in this work as context events.

This work has been supported by the research project FRACTAL in part by the EC under grant number 877056 and the BMBF under grant number 16MEE015K.

Existing solutions to providing adaptation services for time-triggered systems often require the use of the meta-scheduling approach [1], [2], in which schedules are precomputed offline for a variety of context events. The computed schedules resulting from the context events are stored within the embedded device in which schedule changes are made when triggered by new scenarios. Nevertheless, there is an exponential growth in the number of schedules with increasing context events, resulting in a state-space explosion problem. Since embedded devices are memory constrained, an alternative approach to resolving the applicability of a large set of context events within a time-triggered system is required. Hence, a more efficient scheduler is needed to compute new schedules on the fly while avoiding collisions and satisfying precedence constraints of the time-triggered network. For example, list scheduling techniques [3] are fast, but the results are typically inferior concerning makespans compared to metaheuristic algorithms such as Genetic Algorithm [4]. Meteheuristics provide better results in terms of makespan, but their computational time is too high for execution at runtime. The goal is thus to provide timeliness (short makespans) in combination with short runtime for scheduling.

An AI-based scheduler for a time-triggered network is developed utilising deep reinforcement learning in [5]. At each hop, an agent trained by a developed scheduling scheme takes the system state as input and generates an action that directs the timing and path of the time-triggered frame. However, not all actions in the scheduler can be converted to a valid schedule, hence the need for a control gate to filter invalid actions. We differ from this approach by ensuring that only valid schedules are generated in our AI-based scheduling approach for time-triggered networks.

In this paper, we address a safe and adaptive artificial intelligence (AI) quasi-static scheduling approach for time-triggered scheduling problems. The priority of jobs is predicted using an AI-based model trained offline. With recent technological advancements towards deploying AI accelerators in hardware, AI models can now easily be deployed in hardware. At runtime, adaptation can be triggered upon the occurrence of a context event such as the failure of the processing element. The adaptation is attained by accurately predicting the job priority, which is then used to perform an online computation of a new schedule.

GEFÖRDERT VOM Bundesministerium für Bildung und Forschung

The contribution of this paper is as follows

- An AI-based scheduling method for time-triggered networks.
- A design methodology for creating a specific scheduling dataset that can be used to train a schedule inference model for adaptation in time-triggered systems.
- Experimental insight into the quality of makespans and computation time of AI-based scheduling for time-triggered network applications compared to heuristics such as list scheduling.

The outcome of this work can be leveraged by time-triggered applications built upon hardware equipped with AI-accelerators such as in the Xilinx Versal ACAP devices [6]. The remainder of this work is organised as follows. Section II discusses the related work. The system model is presented in Section III. The architecture of the AI-based scheduler is presented in Section IV. The experimental evaluation of the proposed model is presented in Section V and the results are presented and discussed in Section VI. The conclusion is discussed in Section VII.

II. Related work

The classification of the processing sequence of jobs using a hybrid deep neural network is proposed in [7] for a job-shop scheduling problem. A convolution two-dimensional transformation is used to convert the irregular scheduling data into regular multidimensional data, after which a deep learning framework is used to solve the subproblems. The focus of this work was on job shop scheduling in contrast to our time-triggered scheduling problem. We explore further strategies to model and provide dynamic reconfiguration for time-triggered systems.

The application of machine learning to scheduling problems can be grouped into two categories [7]. The first category has explored machine learning to optimise heuristic scheduling techniques, and these contributions can be seen in [8], [9], [10]. In the second category, machine learning frameworks are utilized to realize a learning-based model. Shahrabi et al. [11] propose a scheduling method based on variable neighbourhood search for dynamic job shop scheduling problems with random arrival of jobs and machine breakdowns. They used reinforcement learning with the Q-learning algorithm to obtain the variable neighbourhood search parameters at the rescheduling point. A simple multi-resource cluster scheduler based on deep reinforcement learning, DeepRM, is designed in [12]. DeepRM is utilized in a non-preemptive dynamic job arrival setting. Nasiri et al. [13] presented an online scheduling solution to an open shop problem. The proposed approach consists of discrete event simulation, multi-layer perceptron, radial basis function, and data envelopment analysis for optimizing the composite dispatching rule in an open shop scheduling problem.

A survey on scheduling for time-triggered systems is carried out in [14]. In regard to time-triggered scheduling, Vlk et al. [15] utilized formalism based on Integer Linear Programming (ILP) to demonstrate the applicability of schedules resulting from the enhancement made to an IEEE 802.1Qbv switch. Oliver et al. [16] presented an approach to synthesize the communication schedules for Time-Sensitive Networking (TSN) based on IEEE 802.1Qbv requirements in which SMT/OMT (Satisfiability Modulo Theories/Optimization Modulo Theories) solvers are used. Mahfouzi et al. [17] presented a solution for both scheduling and routing problems for a networked cyber-physical system based on the TSN standard. In their work, an SMT formulation for simultaneous routing and scheduling is presented, and the use of heuristics to improve synthesis efficiency based on route subsets and time slices. The application of machine learning techniques to solve time-triggered scheduling problems has not been widely studied.

In a recent work [5], deep reinforcement learning is applied to the scheduling of time-triggered Ethernet, where a scheduling agent is first trained offline and later deployed for the online scheduling of time-triggered flows. The applied technique uses a directed graph to represent the network topology and resource information. In contrast to this approach, we propose a strategy that provides freedom of collision and fulfils precedence constraints for the time-triggered system by predicting only the task's priorities using an ANN model and subsequently computing the corresponding schedule, which is correct by construction.

III. System Model

Scheduling time-triggered systems is typically done at design time, i.e., static schedule computation. The problem description of the time-triggered system is presented in [14]. The system consists of a set of periodic tasks $\mathcal{T} = \{\tau_1, \tau_2, ..., \tau_n\}$, where a periodic task τ_i is a countably infinite set of iterations called jobs. The kth iteration of task τ_i is represented as τ_i^k. The jobs of the same task are characterised with the same inter-arrival period T_i, processing time p_i, and deadline d_i.

We have presented the time-triggered scheduling problem for a program $\mathcal{P}$ as an application model and a platform model. The application model is a direct acyclic graph (DAG), called a task graph $G = (V, E, w, c)$. V is a set of tasks for a program $\mathcal{P}$, and $\mathcal{E}$ is set of edges which show the communication between the tasks. The computation cost w is a positive weight associated with a given node $n \in V$. Jobs from the same task do not necessarily need to have equal computation costs. The communication cost c is a nonnegative weight associated with members of the edge E. An example of the application model consisting of 4 tasks is shown in Figure 1. Task τ_1,τ_2,τ_3 τ_4 consist of a total number of jobs $n = 5$, $n = 2$, $n = 2$, $n = 2$, $n = 2$, respectively. The computation costs are shown on the side of each job, and the communication costs are shown on each edge.

The platform model is an undirected acyclic graph used to model the physical hardware resources of the program $\mathcal{P}$. It models available hardware resources such as cores, memories, and the interconnection pattern of routers. The platform model is illustrated using Figure 2. The example consists of 4 end system and 9 schedulable switches.

IV. AI-Based Scheduler

We present an AI-based scheduler architecture consisting of three main building blocks, namely a scenario generator,

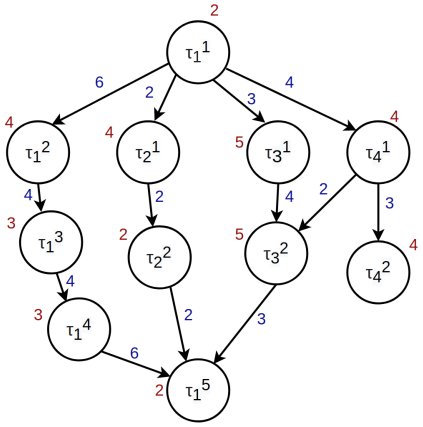

Figure 1. Example of an application model.

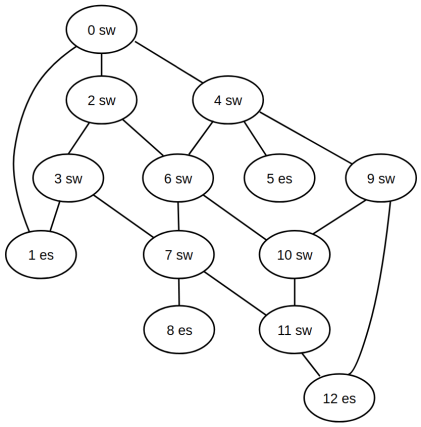

Figure 2. Example of a platform model.

scheduler block, and block for machine learning algorithms, illustrated using Figure 3. The scenario generator is responsible for generating diverse application models, which are then fed into the scheduler to obtain schedules associated with each given platform and application model. The priorities are extracted from the schedules and used together with the corresponding application and platform model outputs from the scenario generator to create a dataset. This dataset is then used to train a machine learning model.

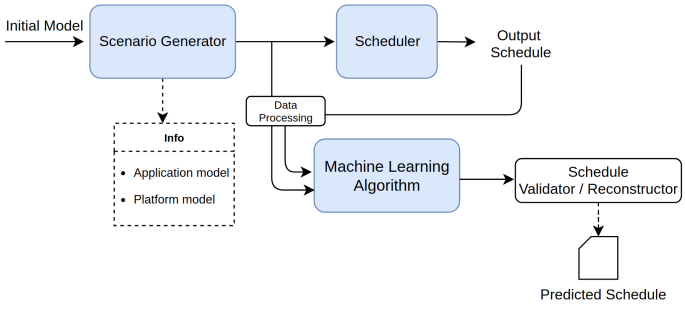

Figure 3. Block diagram of the AI-Based scheduler.

A. Scenario Generator

The scenario generator as shown in Figure 3 is composed of the Stanford Network Analysis Platform (SNAP). SNAP is a general-purpose network analysis and graph mining library inspired by the work in [18]. SNAP is used in this work to generate multiple scenarios. It makes use of a network theory (or graph theory) based approach for developing functions that can be used in the analysis and manipulation of large networks. The library takes the inputs from an initial model specifying parameters such as the number of jobs, number of platform model participants (switches and cores), and the in-degree and out-degree of each job. The scenario generator uses the SNAP functions to generate multiple scenarios of application and platform models.

B. Scheduler Block

The scheduler block shown in Figure 3 takes the output platform and application models from the scenario generator and finds a feasible schedule for each example. The scheduling problem is a well-defined optimisation problem that can be tackled with different approaches such as mathematical techniques, scheduling heuristics, metaheuristics and neighbourhood search. In this work, the genetic algorithm (GA), a metaheuristics method, is used to compute the schedules for each example. Nevertheless, other scheduling approaches such as list scheduling, Ant Colony Optimisation (ACO) can be utilised in the scheduler block.

In this work, the chromosomes of the GA are set to optimise the processor allocation and the job order. The objective function evaluates the makespan to find the best feasible schedule. The output schedule is obtained from the GA, but only the job priority is fed into the machine learning algorithm.

C. Machine Learning Algorithm Block

Due to safety considerations, we focused on learning only the priorities of the jobs. Relying on a machine learning algorithm to directly predict schedules for a safety-critical application is not certifiable. Machine learning models are generalised approximation models, and time-triggered applications require an exact schedule in the event of online schedule adaptation. Therefore, the priorities are predicted, and a schedule verification algorithm is enabled to ensure that only correct schedules are generated from the prediction.

There is no restriction in the type of machine learning algorithm to be deployed in this block (Machine Learning Algorithm in Figure 3). However, in this work a feed forward artificial neural network (ANN) is used to learn the tasks' priorities. The ANN learns the scheduler which is subsequently used to predict job priorities, after which the schedule reconstruction algorithm shown in algorithm 1 is deployed to obtain a schedule for the application. The algorithm works as follows. Initially, all jobs without parents are stored in a vector. The job to be allocated is chosen from the previous vector based on the predicted priorities. After selecting the job, the next step is to assign it to a specific processor. The processor is selected based on the earliest start time, which requires the processor's status and the Data Ready Time (DTR) determined by the messages.

After repeating this process with each of the processors, the lowest value determines the best option. Once the processor is selected, it then checks for message collisions which depends on the configuration of the platform model.

Algorithm 1: Job Allocation

Input: $jobs, msgs, ES$
Output: schedule, makespan
Function `Allocation`$(jobs, msgs, ES)$**:**
  Append jobs without parent nodes in *jobs_ready*
  for *each* N *in* $jobs$ **do**
    Select from *jobs_ready* the job with the highest *weight*
    for *each* m *in* $end\_systems$ **do**
      if $parents[j]$ *is* 0 **then**
        Append $endTime[m]$ to vector $start$
      else
        Append $max(endTime[m], DRT(m, j, msgs, jobs))$ to vector $start$
      end if
    end for
    $st\_time[j] = min(start)$
    $end\_time[j] = st\_time[j] + ex\_time[j]$
    $end\_sys[j] = ES[st\_time.idx(min(start))]$
    if $parents[j]$ **then**
      for *each* p *in* $parents[j]$ **do**
        Append $inj\_time + comm\_cost(p, j)$ in vector *arrrival*
      end for
      $st\_time[j] = max(max(arrival), min(start))$
      $end\_time[j] = st\_time[j] + ex\_time[j]$
      $endTime[idx.min(start)] = end\_time[j]$
    else
      $endTime[idx.min(start)] = end\_time[j]$
    end if
    Remove j from vector $jobs\_ready$
    Append $children[j]$ to vector $jobs\_ready$
  end for
  Append makespan
  return schedule, makespan

V. Experimental Setup

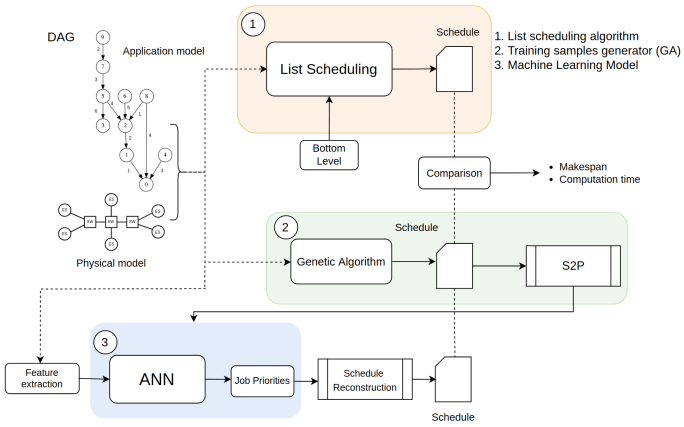

Figure 4. Experiment setup diagram.

The experiment conducted in this work is illustrated using the setup in Figure 4. The objective of the experiment is to evaluate the performance of the AI-based scheduler by comparing the makespan and computation time to both a list scheduler and GA-based scheduler. The evaluations carried out in this work are done on three main building blocks shown in the figure. All three blocks accept an application and platform model described in section III as input, and output the corresponding schedule. The makespan and computation time of the output schedules are then evaluated.

A. List Scheduling Block

The first block represents a *list scheduling algorithm* [19, p. 108] that uses the bottom level of each job as a measure for the priority, which, together with the precedence constraints identified in our system model, produces a sorted list. Each job on the list is then allocated to a specific end system that allows their earliest start time. A comparison between 8 different priority schemes, namely, computation-top-level, computation-bottom-level, bottom-level + criticalComm, bottom-level + maxComm, CP-maxComm, CP_top-level, CP_bottom-level_top-level, and bottom-level is carried out in [20]. The results show that the bottom level approach has better results in terms of shorter makespans. For this reason, the bottom-level is used in this work for comparison.

B. Genetic Algorithm Block

The second block is the GA-based scheduler [19, p. 170], and in our application the objective function is the makespan and the chromosones are the job priorities and end system allocation. The GA-scheduler is implemented using the GAlib library [21] which contains three variations of the genetic algorithm: simple, steady-state, and incremental. The variations differ in how they create new individuals and replace the old ones during the course of an evolution. The variation used in this work is the steady-state which allows to specify how much of the population should be replaced during each generation. The GA configurations utilised in this work are as follows.

- Number of generations ($ngen$): 3000
- Population size ($popsz$): 800
- Probability of mutation ($pmut$): 0.4
- Probability of crossover ($pcross$): 0.001
- Percentage of the population to be replaced : 0.25

The selection of parameters was adopted based on the scheme reported in [22]. The parameter selection is based on a large population, motivated by high search space efficiency and a slow convergence rate. Very low probabilities of crossover and mutation normally prevent a random search by the GA. The parameters $popsz$, $ngen$, $pmut$, $pcross$ were first initialised with 100, 1000, 0.01 and 0.4, respectively. After which, the parameter tuning approach in [23] was used to generate schedules repeatedly, and parameters resulting in schedules with the shortest average makespan are then selected.

The GA-algorithm utilises the application model and platform model to compute the schedules which are evaluated based on the makespan to optimise the job priorities and the end system allocation.

C. ANN Block

The third block is an ANN-based scheduler, which is based on a model used to predict the job priorities. The ANN is used to learn the GA-based scheduler. The SNAP library described in the scenario generator component of Figure 3 generates 20,000 samples of different application models and a fixed platform model. A total of one hundred jobs having varying worst-case execution time is configured for the application model. The dataset used to train the ANN scheduler consists of input features extracted from the application model, and priorities obtained from the GA schedules as output feature.

1) Input Features: The input features of the ANN are summarized in Table I. The features contain the job attributes and computations that attempt to capture the relationship between jobs, such as execution time, messages sent, messages received, top-level, and bottom-level values.

TABLE I
SCHEDULE FEATURES.

Feature	Description	Formula
$f_{1,j}$	Job id / Number of jobs	j/n
$f_{2,j}$	Job execution time / Total execution time	e_j/e_{Total}
$f_{3,j}$	Max. size - msgs sent / Maximum size	$ms_{j,out}/ms_{max}$
$f_{4,j}$	Max. size - msgs received / Maximum size	$ms_{j,in}/ms_{max}$
$f_{5,j}$	Number of msgs sent / Total number of msgs	$m_{j,out}/m_{Total}$
$f_{6,j}$	Number of msgs received / Total number of msgs	$m_{j,in}/m_{Total}$
$f_{7,j}$	Top level value / Maximum top level value	tl_j/tl_{max}
$f_{8,j}$	Bottom level value / Maximum button level value	bl_j/bl_{max}

2) Output Features: The GA-scheduler is first used to compute the schedules that correspond to each input application model, after which the output features are obtained by applying the schedule-to-priority (S2P) algorithm. Algorithm 2 shows the S2P algorithm. The S2P algorithm converts the priorities from the GA schedule into a multi-binary format that is used to train the ANN scheduler. This format re-orders the priorities in a way that compares the priority of every job with each other on a one-to-one basis.

Algorithm 2: S2P algorithm

```
Input: size
Output: binary_labels
Function S2P(size):
    for s in range(size) do
        for each i in num_jobs − 1 do
            for each j in range(i + 1, num_jobs) do
                if position[i] > position[j] : then
                    Append '1' to vector label_vector
                else
                    Append '0' to vector label_vector
                end if
            end for
        end for
        Append label_vector to binary_labels
    end for
    return binary_labels
```

3) ANN Model: The ANN model consists on three layers:

- Input Layer: The number of nodes in this layer is determined by the number of jobs and features extracted. In our experiment, the number of extracted features per job is equal to 8, the number of input nodes for 10, 40, 70, and 100 job datasets is 80, 320, 560, and 800, respectively.
- Hidden Layer: The amount of nodes in the hidden layer varies in accordance with the number of the input nodes. In our experiment it ranges from 100 to 1000 nodes.
- Output Layer: The number of nodes is determined by the formula:

$$n * (n-1)/2,$$

where n is the total number of jobs. The formula comes from the S2P algorithm explained in the previous block.

The ReLU (Rectifier Linear Unit) activation function is used after the input layer and the Sigmoid function is used before the output nodes. The loss is computed from all the labels as a whole since we are dealing with multi-label classification.

The Binary Cross-Entropy Loss (BCE) function is usually employed for binary labels, but since our data is not balanced, we used a modified version of the BCE called $'weighted\_binary\_crossentropy'$ [24]. This function adjusts the BCE by adding a weighting. The weights are determined dynamically for every batch by identifying how many positive and negative classes are present and modifying them accordingly. After tuning, the learning rate of the ANN is set to 0.001, and the number of learning epochs is set to 300.

Four different datasets are generated for this experiment, and each dataset comprises 20,000 scheduling problems. The difference between each dataset is in the number of jobs. Data sets 1 - 4 consist of 10, 40, 70, and 100 jobs, respectively. The dataset was split into training and testing data in a ratio of 80:20, respectively. An additional split was made to create validation data consisting of 10% of the training samples. The neural networks's training parameters were chosen based on the binary accuracy of the predictions, keeping in mind that low accuracy values do not necessarily indicate an invalid schedule, as the predictions can still lead to alternative solutions with different makespans. The schedule reconstruction implemented in algorithm 1 ensures the validation of the schedules by only comparing the priority of the jobs that are ready to be allocated, fulfilling the precedence constraints.

VI. Results and Discussion

We conducted three classes of experiments for our evaluation. The three classes include experiments using the list scheduler, ANN-based scheduler, and GA-based scheduler. For each class, the 4000 scheduling problems from the testing data consisting of 10, 40, 70, and 100 jobs are evaluated. The average of the makespans of the 4000 scheduling problems are shown in Figure 5. It can be seen that the schedules computed by the GA-based approach have better makespans compared to the list scheduler and the ANN model. We show a plot of the makespans against the number of jobs for each class of the experiment in Figure 5. It can be observed that the list scheduler have lower makespans than the ANN-based approach for a job size of 10 and 40. However, as the number of jobs increases, the ANN produces makespans lower than the list scheduler, as seen in the case of 70 and 100 jobs. In real time-triggered systems, we can expect a higher number of jobs, which will

increase the gap of the makespans between the list scheduling and the ANN-based approach.

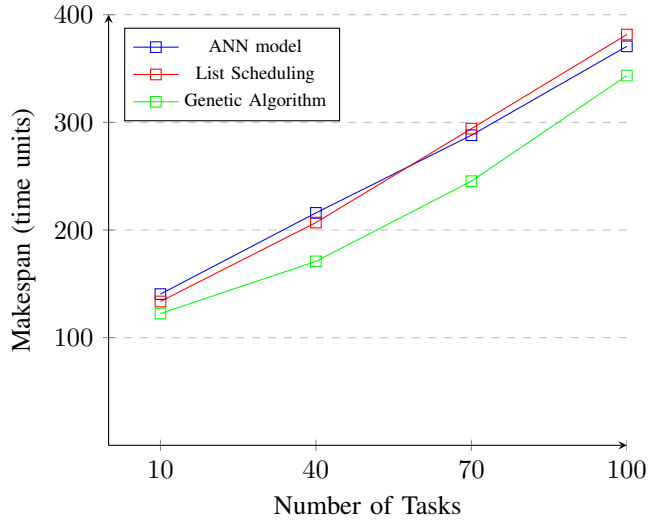

Figure 5. Makespan comparison.

The implication of the contribution is the applicability of the ANN-based approach for dynamic reconfiguration of safety criticality application in response to context events. A GA-based scheduler is not desirable for real-time systems with stringent computational time requirements. In Table 6, we show the computational times for each of the evaluations carried out. The computation time of the GA-based approach is much more than that of the list scheduler and the ANN-based scheduler. It can be seen that for a set of 100 jobs, it takes 4500s to compute compared to 1.47s and 1.105s for the ANN-based model and List scheduler, respectively.

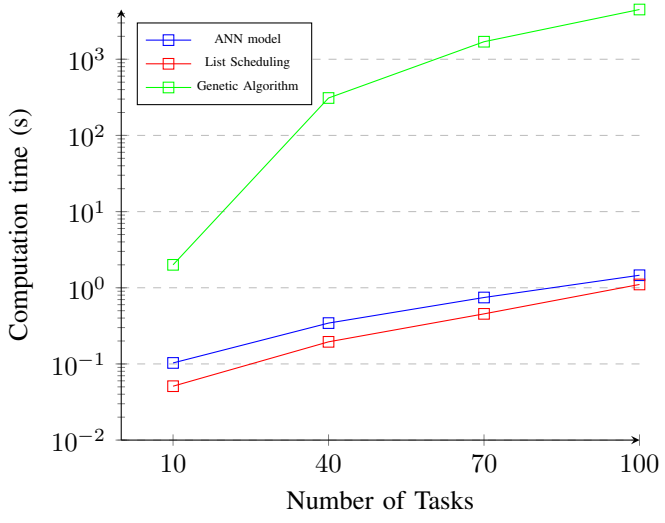

Figure 6. Computation time comparison.

The experiments are conducted using the University of Siegen's OMNI computing cluster [25], with 439 regular compute nodes operated with Linux operating system. Each node comprises two AMD EPYC 7452 CPU processors, with 32 cores per EPYC CPU and a 2.35-3.35 GHz CPU frequency.

In Figure 6, we show a plot of the computational time for each job. It shows that the GA-based approach is poor in terms of computation time. However, the List scheduler outperforms the ANN-based model. The experiment's outcome in comparing the ANN-based approach to the List scheduler shows that for providing adaptation services for applications, there is a trade-off between getting an optimal makespan and computation time when using the ANN-based approach.

Due to the recent trend in incorporating AI-accelerators to modern hardware devices such as the Versal ACAP [6], the opportunity to apply AI-based approaches for real-time applications is now plausible. Consequently, the proposed ANN-based scheduling techniques can be applied for the dynamic reconfiguration of time-triggered systems.

VII. Conclusion

This work investigates the potential of AI-based scheduling for real-time applications. It formulates a method to utilise AI-based scheduling schemes to generate schedules that are only correct by construct. The AI-based scheduling approach predicts job priorities which are subsequently used to generate correct schedules. The proposed techniques can provide adaptation services for time-triggered based applications. In this work, the proposed AI-based scheduling technique is compared with the List scheduling and genetic algorithm based scheduler. The results show the potential of the AI-based scheduler in producing better makespans than the list scheduler and GA-based approach as the number of jobs increases. Nevertheless, the AI-based scheduler still falls short of computational time compared to the list scheduler. However, it allows applications to utilise the trade-off between an optimal makespan and the computation time requirement.

References

[1] R. Obermaisser, H. Ahmadian, A. Maleki, Y. Bebawy, A. Lenz, and B. Sorkhpour, "Adaptive time-triggered multi-core architecture," *Designs*, vol. 3, no. 1, p. 7, 2019.

[2] B. Sorkhpour, A. Murshed, and R. Obermaisser, "Meta-scheduling techniques for energy-efficient robust and adaptive time-triggered systems," in *2017 IEEE 4th International Conference on Knowledge-Based Engineering and Innovation (KBEI)*. IEEE, 2017, pp. 0143–0150.

[3] T. Yang and A. Gerasoulis, "List scheduling with and without communication delays," *Parallel Computing*, vol. 19, no. 12, pp. 1321–1344, 1993.

[4] E. S. Hou, N. Ansari, and H. Ren, "A genetic algorithm for multiprocessor scheduling," *IEEE Transactions on Parallel and Distributed systems*, vol. 5, no. 2, pp. 113–120, 1994.

[5] C. Zhong, H. Jia, H. Wan, and X. Zhao, "Drls: A deep reinforcement learning based scheduler for time-triggered ethernet," in *2021 International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2021, pp. 1–11.

[6] K. Vissers, "Versal: The xilinx adaptive compute acceleration platform (acap)," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2019, pp. 83–83.

[7] Z. Zang, W. Wang, Y. Song, L. Lu, W. Li, Y. Wang, and Y. Zhao, "Hybrid deep neural network scheduler for job-shop problem based on convolution two-dimensional transformation," *Computational intelligence and neuroscience*, vol. 2019, 2019.

[8] T. R. Ramanan, R. Sridharan, K. S. Shashikant, and A. N. Haq, "An artificial neural network based heuristic for flow shop scheduling problems," *Journal of Intelligent Manufacturing*, vol. 22, no. 2, pp. 279–288, 2011.

[9] M. A. Adibi and J. Shahrabi, "A clustering-based modified variable neighborhood search algorithm for a dynamic job shop scheduling problem," *The International Journal of Advanced Manufacturing Technology*, vol. 70, no. 9-12, pp. 1955–1961, 2014.

[10] A. Maroosi, R. C. Muniyandi, E. Sundararajan, and A. M. Zin, "A parallel membrane inspired harmony search for optimization problems: A case study based on a flexible job shop scheduling problem," *Applied Soft Computing*, vol. 49, pp. 120–136, 2016.

[11] J. Shahrabi, M. A. Adibi, and M. Mahootchi, "A reinforcement learning approach to parameter estimation in dynamic job shop scheduling," *Computers & Industrial Engineering*, vol. 110, pp. 75–82, 2017.

[12] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proceedings of the 15th ACM workshop on hot topics in networks*, 2016, pp. 50–56.

[13] M. M. Nasiri, R. Yazdanparast, and F. Jolai, "A simulation optimisation approach for real-time scheduling in an open shop environment using a composite dispatching rule," *International journal of computer integrated manufacturing*, vol. 30, no. 12, pp. 1239–1252, 2017.

[14] A. Minaeva and Z. Hanzálek, "Survey on periodic scheduling for time-triggered hard real-time systems," *ACM Computing Surveys (CSUR)*, vol. 54, no. 1, pp. 1–32, 2021.

[15] M. Vlk, Z. Hanzálek, K. Brejchová, S. Tang, S. Bhattacharjee, and S. Fu, "Enhancing schedulability and throughput of time-triggered traffic in ieee 802.1 qbv time-sensitive networks," *IEEE Transactions on Communications*, vol. 68, no. 11, pp. 7023–7038, 2020.

[16] R. S. Oliver, S. S. Craciunas, and W. Steiner, "Ieee 802.1 qbv gate control list synthesis using array theory encoding," in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2018, pp. 13–24.

[17] R. Mahfouzi, A. Aminifar, S. Samii, A. Rezine, P. Eles, and Z. Peng, "Stability-aware integrated routing and scheduling for control applications in ethernet networks," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 682–687.

[18] J. Leskovec and R. Sosič, "Snap: A general-purpose network analysis and graph-mining library," *ACM Trans. Intell. Syst. Technol.*, vol. 8, no. 1, Jul. 2016. [Online]. Available: https://doi.org/10.1145/2898361

[19] O. Sinnen, *Task scheduling for parallel systems*. John Wiley & Sons, 2007, vol. 60.

[20] O. Sinnen and L. Sousa, "Comparison of contention aware list scheduling heuristics for cluster computing," in *Proceedings International Conference on Parallel Processing Workshops*, 2001, pp. 382–387.

[21] M. Wall and A. Galib, "A c++ library of genetic algorithm components," *Boston: Mechanical Engineering Department Massachusetts Institute of Technology*, 1996.

[22] P. Muoka, D. Onwuchekwa, and R. Obermaisser, "Adaptive scheduling for time-triggered network-on-chip-based multi-core architecture using genetic algorithm," *Electronics*, vol. 11, no. 1, p. 49, 2022.

[23] A. Hassanat, K. Almohammadi, E. Alkafaween, E. Abunawas, A. Hammouri, and V. Prasath, "Choosing mutation and crossover ratios for genetic algorithms—a review with a new dynamic approach," *Information*, vol. 10, no. 12, p. 390, 2019.

[24] M. R. Rezaei-Dastjerdehei, A. Mijani, and E. Fatemizadeh, "Addressing imbalance in multi-label classification using weighted cross entropy loss function," in *2020 27th National and 5th International Iranian Conference on Biomedical Engineering (ICBME)*. IEEE, 2020, pp. 333–338.

[25] Universität Siegen, https://cluster.uni-siegen.de.