The Third International Workshop on Dynamic Scheduling Problems
Adam Mickiewicz University, Poznań, July 5th – 6th, 2021
DOI: 10.14708/isbn.978-83-951298-6-5p57-62

Dual-criticality scheduling on non-preemptive, dynamic processors using RL agents

Nourhan Sakr*
American University in Cairo, Egypt

Youssef Hussein
American University in Cairo, Egypt

Karim Farid
American University in Cairo, Egypt

Keywords: mixed-criticality scheduling, varying-speed processors, reinforcement learning

1 Introduction and related work

Real-time embedded systems have stringent non-functional requirements on cost, weight, and energy that give rise to the study of *mixed-criticality* (MC) systems, where functionalities of different *criticality levels* are consolidated into a shared hardware platform (Barhorst et al. [1], Burns and Davis [6]). The literature discusses the schedulability of MC systems under various conditions and objectives (Baruah et al. [3], Gu et al. [7], Baruah et al. [4]). In this work, we study a dual-criticality, non-preemptive system with a varying-speed uniprocessor.

This varying-speed processor makes MC scheduling dynamic: In real-time systems, it cannot be predetermined if, when or for how long the processor would *degrade*, i.e. its speed would drop. Under speed stochasticity, it is critical to guarantee the running of high (HI) criticality jobs by their deadlines, sometimes even at the cost of not running low (LO) criticality jobs at all, when operating under degradation.

Scheduling MC systems non-preemptively (even without degradation) is $\mathcal{NP}$-hard (Lenstra et al. [9]). Baruah and Guo [5] model an LP to preemptively schedule dual-criticality jobs on a varying-speed processor. Agarwal and Baruah [2] further discuss the online nature of the problem and its intractability. We agree with the authors that MC scheduling is inherently an online problem, as it better depicts real-time scheduling and the dynamic nature of this problem. Therefore, we devise deep reinforcement learning (deep RL or DRL) to tackle the problem presented by Baruah and Guo [5], under both the offline and online setting.

*Speaker, e-mail: n.sakr@columbia.edu

2 Problem formulation

We model the system functionalities by a set of independent jobs J, each job j is defined by its release date r_j, deadline d_j, processing time p_j (representing worst-case execution time) and criticality level $\chi_j \in \{\text{LO}; \text{HI}\}$, describing a dual-criticality system of low and high criticality jobs. A speed-v processor runs a job j in $\frac{p_j}{v}$ time units.

The system assumes two modes of operation: *normal* ($v = 1$) and *degradation* mode ($v < 1$). We assume a self-monitoring system that immediately knows when a degradation occurs during runtime. The degradation speed v is observed then. An MC instance is a set of MC jobs J that are schedulable on a varying-speed processor. According to Lemma 1 in Baruah and Guo [5], J is schedulable if an earliest deadline first (EDF) policy schedules all jobs on a speed-1 processor and all HI jobs on a speed-$v_{\min}$ processor. That is, there is a degradation bound $v_{\min}$, below which J would be no longer schedulable. Finally, a *correct schedule* runs all jobs by their deadlines as long as $v = 1$ and guarantees all HI jobs to meet their deadlines regardless of the speed.

3 Model and evaluation

Our environment is modeled as a Markov decision process (MDP). At each timestep t, a reinforcement learning (RL) agent interacts with an environment by receiving an observation o_t from a state space S and taking a corresponding action a_t from a given action space A. This action is (later) rewarded or penalized (i.e. negative reward) using a reward function $r(a_t, o_t)$. The agent's goal is to maximize the reward in an *episode*.

Episode, states and actions. We consider one episode to be a series of decisions taken until all jobs in J either run (and complete by the deadline) or expire (cannot meet the deadline). At a given time t, an observation o_t is a buffer of jobs $B_t \subseteq J$ and the processor speed v_t. Each job in the buffer is represented by its static tuple (r_j, d_j, p_j, χ_j), and three status parameters that indicate whether the job was released, expired (i.e. missed its deadline) or was scheduled to run. Given o_t, the action function produces a selection for the index of the job to be scheduled at t. We next outline three environments that help us stage the transition into our target online problem.

Offline environment. In an offline setting, all decisions are taken at $t = 0$ when jobs are known but degradation cannot be observed. Jobs can be scheduled at anytime $t \in [\min_j\{r_j\}, \max_j\{d_j - p_j\}]$. We, therefore, set $B_t = J$ and $v_t = 1, \forall t$, thereby assuming normal operation. The the RL agent schedules jobs sequentially, so all job binary status parameters are updated before every new decision. This sequential process also ensures non-preemption, i.e. when a job j is selected for t, the next decision is made for time $t + p_j$. During this time window, some jobs may expire before selection.

Varying buffer (VB). In an online setting, not all jobs are known to the agent a priori. This constraint introduces a modeling issue since the buffer size should be fixed for each observation, yet the number of observable jobs changes at each time step. We create the VB environment as a transitional stage, where we fix the size of the buffer B_t at n. If $|J| > n$, we add the jobs with least laxities to the buffer. (The laxity of a job, defined as $l_j = d_j - p_j$, gives early warnings on expiry. When $l_j \leq t$, this means that even if the agent schedules job j at t, the time would not be sufficient for it to complete before d_j. Thus, the agent is penalized for choosing such jobs.) Otherwise, we add all jobs plus a number of dummy jobs that would bring the buffer size to n. To avoid scheduling dummy jobs, we label them as expired and set their criticality to LO.

Online environment. This environment is closest to mimicking real-time scheduling. Our agent is now able to observe the processor speed, so we relax the constraint on $v_t = 1$. We build B_t similar to that of the VB environment but impose two additional restrictions: To be added to B_t, a job must have been released ($r_j \leq t$) and have enough time to complete ($l_j \geq t$). As such, all dynamic features of our system are captured.

To the best of our knowledge, we are the first to use RL agents for scheduling MC systems and, hence, have no previous RL-based benchmarks. We remedy this limitation by using the staging process above for testing and validation. Although, we are ultimately interested in modeling the online problem (see Sect. 1), we start with an offline environment in order to leverage available offline benchmarks. Once validated, our offline environment became the baseline for the VB environment and subsequently the VB became the benchmark for the online environment. In this abstract, we highlight the reward function and results of the online environment only.

Online reward function. We use the *number of completed jobs* as the main metric of comparison. A secondary goal is to complete all HI jobs in J. The online agent receives a reward equivalent to the number of executed jobs at the end of each episode. This design achieves the highest (over episode) average reward, emphasizing the priority of running HI jobs. Otherwise, the agent is penalized for selecting jobs that have expired or have run before. We test other reward designs, such as applying rewards or penalties instantaneously rather than at the end of an episode, or scaling the reward by v (for higher rewards in case of degradation), but they were all deemed less successful.

Algorithm choice. In general, DRL allows more unstructured input to the agent with larger data. We tested RL algorithms from RLLib and Stable Baselines, where Ape-X DQN (Horgan et al. [8]) produced the best results in scheduling jobs and learning EDF behavior.

4 Our results

Simulation. We generate various instances of J and processor speeds. The details of data generation are masked from this abstract but rely heavily on a modification of Baruah and Guo [5]. It is worth noting that each set J is verified for schedulability before being fed to the RL agent. Our hardware limitations cap the size of our instances, $|J|$ at 30. We also study the settings needed for the warm-up period, number of episodes (steps) and run 500 iterations per experiment.

Evaluation criteria. We evaluate environments based on the general reward evaluation, which assesses the agent behavior against the mean reward over a series of episodes, i.e. the average number of executed jobs per instance. Note that the maximum reward that the agent can receive in any episode i is $|J_i|$, assuming all jobs complete without expiring. We additionally conduct a degradation analysis, which is essential to understanding the sensitivity of the learned policy to the degradation speed.

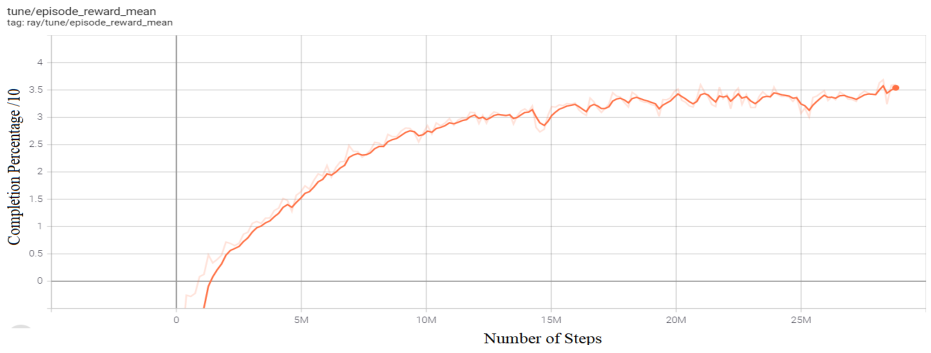

Figure 1. General reward evaluation results of the online environment.

Preliminary results. After its success in the offline environment, Ape-X was capable of correctly scheduling around 30-40% of the provided instances when introduced to the online environment (see Fig. 1). The limitations come from two sources: a problem-specific challenge and a hardware challenge. The online environment is non-clairvoyant and imposes unpredictable degradation in the performance. We believe that this can be improved by augmenting the RL agent with a prediction module that can make forecasts on the expected processor speeds. On the hardware side, we are limited by resources (14 CPUs working in parallel and one empowered GPU), which did not allow us to run large scale examples. The results produced in Fig. 1 are based on instances that are 30 jobs each and a buffer size set at $n = 10$. We believe that obtaining more resources that can run larger examples will yield better training and testing results, as the agent has more data to learn. However, these preliminary results show a promising approach that is worth discussing.

Degradation analysis. When the processor degrades, it is expected that the agent prioritizes completing all HI jobs. The model should learn to sacrifice LO jobs to free resources for the HI ones. We measure the sensitivity of the agent to degradation speeds: In a total of 10,000 job scenarios(episodes), we label the scenarios where the agent completes all HI jobs as successful. These scenarios are fed to the RL agent, and the agent's performance is assessed under varying speeds $v \in [0.1, 1]$. Fig. 2 shows that online environments perform slightly better than the other two environments and that lower speeds yield decreasing performances until it plateaus at almost $15 - 20\%$. This phenomenon is likely attributed to the ability of the RL agent to now observe the processor's speed and dynamically adapt to any disruptions exposing the system.

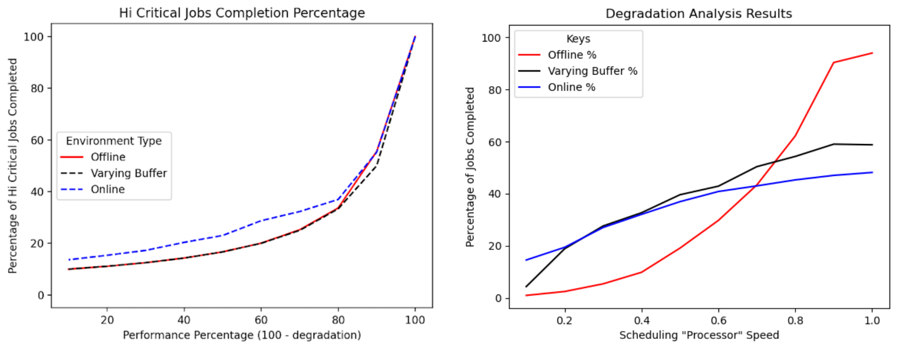

Figure 2. Degradation analysis results for comparing the three environments

5 Future research

For future work, we wish to improve our performance using larger examples and contrast our results against the OCBP benchmark proposed by Agarwal and Baruah [2]. We also wish to extend our work to other variants of the problem, such as preemptive scheduling, multiprocessor systems with resource sharing, or integral data generation. Furthermore, we plan to study other evaluation metrics, conduct an error analysis on unsuccessful schedules, as well as assess the benefit of augmenting a forecasting module for predicting speeds a priori.

References

[1] J. Barhorst, T. Belote, P. Binns, J. Hoffman, J. Paunicka, P. Sarathy, J. Scoredos, P. Stanfill, D. Stuart, R. Urzi, A research agenda for mixed-criticality systems, white paper, 2009.

[2] K. Agarwal, S. Baruah, Intractability issues in mixed-criticality scheduling, *30th EuroMicro Conference on Real-Time Systems*, 2018, article no. 11, 11:1–11:21, doi: 10.4230/LIPIcs.ECRTS.2018.11.

[3] S. Baruah, V. Bonifaci, G. D'Angelo, H-H. Li, A. Marchetti-Spaccamela, N. Megow, L. Stougie, Scheduling real-time mixed-criticality jobs, *IEEE Transactions on Computers*, **61** (2011), 1140–1152, doi: 10.1109/TC.2011.142.

[4] S. Baruah, A. Easwaran, Z. Guo, Mixed-criticality scheduling to minimize makespan, *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, 2016, article no. 7, 7:1–7:13, doi: 10.4230/LIPIcs.FSTTCS.2016.7.

[5] S. Baruah, Z. Guo, Mixed-criticality scheduling upon varying speed processors, *34th Real-Time Systems Symposium*, 2013, 68–77, doi: 10.1109/RTSS.2013.15.

[6] A. Burns, R. Davis, Mixed criticality systems-a review, Department of Computer Science, University of York, Tech. Rep. 172, 2016.

[7] C. Gu, N. Guan, Q. Deng, W. Yi, Improving OCBP-based scheduling for mixed-criticality sporadic task systems, *19th International Conference on Embedded and Real-Time Computing Systems and Applications*, 2013, 247–256, doi: 10.1109/RTCSA.2013.6732225.

[8] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. van Hasselt, D. Silver, Distributed prioritized experience replay, arXiv preprint arXiv:1803.00933, 2018.

[9] J. K. Lenstra, A. H. G. Rinnooy Kan, P. Brucker, Complexity of machine scheduling problems, *Annals of Discrete Mathematics*, **1** (1977), 343–362, doi: 10.1016/S0167-5060(08)70743-X.