Project Report - Designing HPC job schedulers using Reinforcement Learning

Report by:	Mentored by:	Supervised by:
Ishan Sharma	Mr. Anirban Ghose	Prof. Soumyajit Dey

1 Introduction

The project aims to develop intelligent schedulers for Embedded systems in automobiles. We consider a heterogeneous Advanced Driver-Assistance System **ADAS** platform comprising heterogeneous ECUs consisting both CPU and GPGPU cores and a set of object detection pipelines executing periodically to process sensor data leveraging state of the art DNNs and CNNs. There exists a natural requirement for real time guarantees for executing these object detection pipelines. The existing ADAS scheduling algorithms are not equipped to handle tasks with time varying dynamic requirements dictated by different driving contexts. Instead of investigating every possible mapping decision, an intelligent task scheduler needs to be designed which will predict task-device mapping decisions.

2 Previous accomplishments

The past work on the project has resulted in the development of a Discrete Event Driven Simulation Engine which acts as the environment for the reinforcement learning agent. Deep Q learning algorithm with duelling and prioritised experience replay was used to train the inference engine after prudent design of reward assignment and state extraction methodologies.

3 Literature Survey

We now aim to develop a framework for distributed training of the Reinforcement learning agent. The general architecture consists of one or more parameter (centralised) server(s) and many worker nodes. We must decide the correct methodology of gradient updates: synchronous gradient descent, where the worker nodes pull the latest model parameters from the parameter server, acquire lock on the parameters and send the updated weights for one batch of data. The parameter server updates the weights when all workers are done with their chunk of work. (Chen *et al.*, 2016) adds backup workers for to mitigate worker(s) becoming unresponsive. On the other hand, in asynchronous gradient descent the workers can not acquire lock on the parameters which leads to the possibility of parameters becoming stale in a worker node, but downpour SGD (Dean *et al.*, 2012) found relaxing consistency requirements in practice to be remarkably effective. While the above described ways can be generally applied to any deep learning problem, (Ong *et al.*, 2015) outlines detailed gradient update algorithms for parameter and worker nodes. All the aforementioned works do not mention the architecture of the replay memory, which is essential to modern RL algorithms. (Nair *et al.*, 2015) introduce Gorila and (Horgan

et al., 2018) present Ape-X architecture to address this issue with centralised replay memories accessible to both paramter and worker nodes, differing in the way replay memory is updated.

4 Problems to be solved

There exist N different jobs in the system. Each job can have one of k possible periods. Thus, there are k^N different period configurations. First thing to note here is that the search space is exponential in the number of jobs. Each period configuration runs for S simulations. This entire process runs for E epochs. Therefore, the entire computation is $\mathcal{O}(k^N \times S \times E)$. Second thing to note, the search space shifts only a bit between successive period configurations. The current implementation of the RL scheduler (on one worker) relies on the learning accumulated in the previous period configurations for learning the next one. This reliance gradually increases towards the end of training. **The first problem** is how should this reliance among different period configurations to speed up the training be handled in a distributed system? **The second problem** is how should exponential decay be handled in exploration-exploitation trade-off? One of following two approaches can be followed to solve the second problem after establishing its optimality:

- Distribute a proportion of S simulation runs of one period configuration among all workers at once and accumulate the results upon completion.
- Allot all S simulation runs of one period configuration to one worker.

At last, we propose the following architecture (Figure 1) for RL scheduling agent's training in the distributed environment. The centralised parameter server(s) receive gradient updates from the learners and the learners synchronise their respective parameters from the parameter server(s). The replay memory is decoupled from the agent, learner, or the parameter server. The agent interacts with the Discrete Event Driven Simulation Engine (environment), with actions as mapping decisions and states as resource and application DAG status. The agent ultimately stores transition tuple in the replay memory, where the learners sample these tuples from.

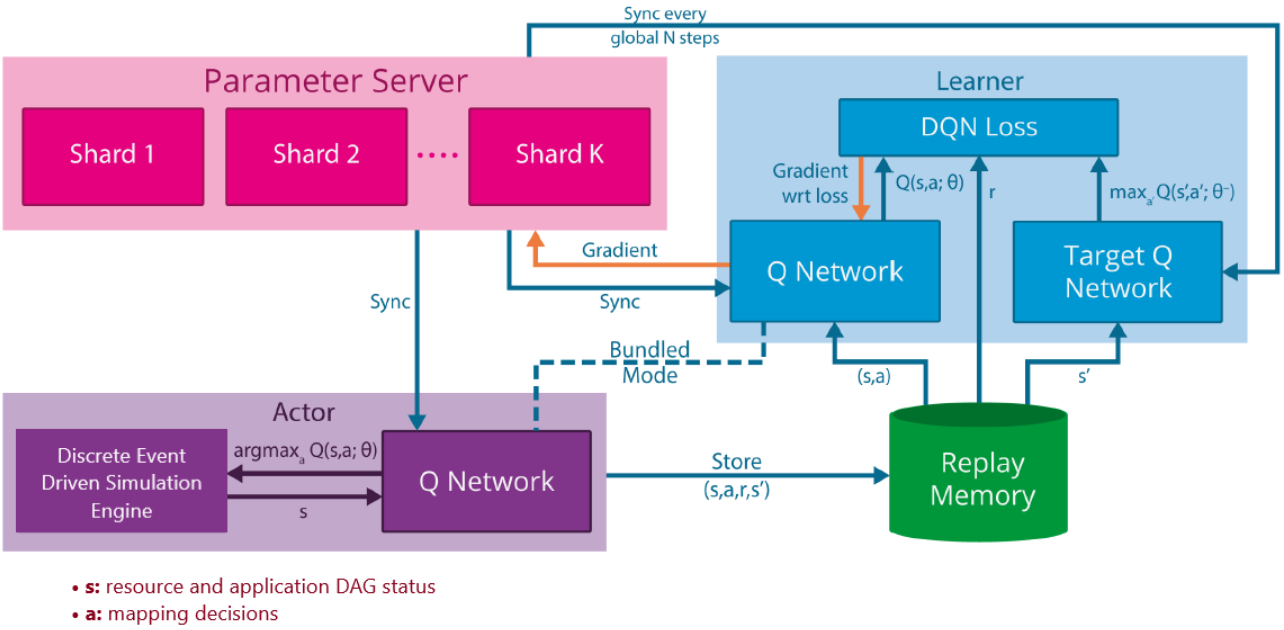

Figure 1: RL Scheduling System Overview

References

J. Chen, X. Pan, R. Monga, S. Bengio and R. Jozefowicz, *arXiv preprint arXiv:1604.00981*, 2016.

J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker *et al.*, 2012.

H. Y. Ong, K. Chavez and A. Hong, *arXiv preprint arXiv:1508.04186*, 2015.

A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. De Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen *et al.*, *arXiv preprint arXiv:1507.04296*, 2015.

D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. Van Hasselt and D. Silver, *arXiv preprint arXiv:1803.00933*, 2018.