Extended Abstract: Using Parallelized Containers for Reinforcement Learning on Large Computer Clusters

Anirudh Nair [1], **Zifan Xu**[1], **Gauraang Dhamankar**[1], **Xuesu Xiao**[1]
[1]University of Texas at Austin
ani.nair@utexas.edu, zfxu@utexas.edu, gauraang@cs.utexas.edu, xiao@cs.utexas.edu

Introduction

Reinforcement Learning (RL) robotics projects require massive amounts of computing power for both training and experimentation. Moreover, it is likely that large-scale computing clusters may not have the required software to run experiments for certain projects, e.g. Robot Operating System (ROS) (Quigley et al. 2009). To remedy this, containers, such as Docker (Merkel 2014) and Singularity (Kurtzer, Sochat, and Bauer 2017), have been used to package all the necessary software needed to run experiments for RL. In addition, the use of a container improves reproducability since the container can be built into an image file and shared very easily.

While these containers can be built from scratch based on each case, the code to create the container, run experiments, and collect data is quite general. There are guides on how to build containers; however, there is no clear guide that exists for creating and implementing these containers for large scale RL robotics tasks on computer clusters. In this paper, we present our Parallelized Containers for Reinforcement Learning (PCRL) framework which provides a general foundation to carry out large scale RL robotics tasks. Moreover, we present an example of this framework being applied to a mobile robot navigation task.

Our PCRL framework methodology is specific to the actor-critic RL algorithms as well as the HTCondor scheduler and Singularity containers; however, this framework can be easily extended to use other RL algorithms, container methods, and schedulers.

Methodology

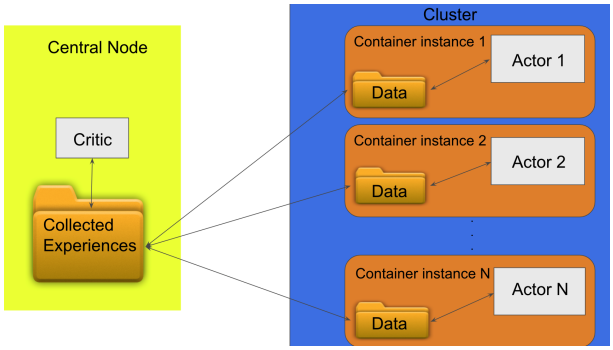
Building the Container

PCRL depends on a pre-built container for the learning task of interest, which works as a containerized black box with specifically designed input and output interfaces and can be further parallelized on large computer clusters. Singularity is becoming increasingly common for High Performance Computing (HPC). Moreover, many computer clusters only support Singularity due to its superior safety features compared to other container solutions, such as Docker (Merkel 2014). Therefore, we choose Singularity and present how to create a Singularity container with the required software, e.g. ROS, as opposed to other container methods.

The Actor-Critic Framework

Actor-Critic (AC) framework is one of the most popular RL framework, which is adopted by many state-of-the-art algorithms, such as TD3 (Fujimoto, van Hoof, and Meger 2018) and SAC (Haarnoja et al. 2018). The PCRL framework uses the AC framework by making the critic and all the actors as instances of a Singularity container: one for each actors. All container instances are created based on the same Singularity container image, with specific parameterization from a central node, i.e. the critic. When the container instances are submitted to the cluster, each compute node runs the actor algorithm inside the instance of the Singularity image specific to that node. To simultaneously share the experience generated by the actor and the policy updated by the critic, a buffer folder B on the host folder, which maintains the policy π_p for the actors, is mounted to the buffer folder B^* on each of the actor nodes. Figure 1 shows the structure of such a file system. As it describes in algorithm 1, the container instance in a actor node rollouts the experiences based on the latest policy and write experiences into B^*. The critic then collects the experiences from B at a constant frequency, updates the policy, and places the new policy into B for the actors in the containers to use, as in algorithm 2. Both the actor containers and the critic on the host node run in parallel.

Figure 1: Depiction of the movement of data between the container and the host node



Algorithm 1 Running Actors with Containers

Require: Container with Actor network as well as a buffer folder $\mathcal{B}^*$, Buffer folder on the host node $\mathcal{B}$ containing policy π_p.

1: $\mathcal{B}$ is bound to $\mathcal{B}^*$.
2: Submit a job consisting of the Actor container
3: **while** Training not ended **do**
4: // In container
5: Run the policy from the mounted folder.
6: Save experiences into the B^* folder.
7: **end while**

Algorithm 2 Running Critic on the Host Node

Require: Number of updates, Critic Network, Buffer folder on the host node $\mathcal{B}$ containing policy π_p

1: **for** each update step **do**
2: //On host node
3: Gather experiences from B
4: Update π_p with the experiences in B
5: Place updated π_p into B
6: **end for**

HTCondor (Tannenbaum et al. 2001)

Our PCRL framework is primarily specific to the HTCondor scheduler for large computing clusters. HTCondor takes in submission files, which contains commands and keywords to direct the queuing of jobs. In this submission file, Condor finds everything it needs to know about the job, such as the name of the executable to run, the initial working directory, and command-line arguments to the program. HTCondor then locates a machine in the cluster, packages up the job, and ships it off to be run by the machine. A Singularity universe job instantiates a Singularity container from a Singularity image, with parameterization specified for that actor. HTCondor manages the running of that container as an HTCondor job.

Example

We use the Adaptive Planner Parameter Learning from Reinforcement (APPLR) (Xu et al. 2020) as an example[1] of using PCRL. In APPLR, the RL agent learns a parameter policy, which adjusts the parameters of a robot navigation system to optimize navigation performance in obstacle-occupied environments.

Using PCRL, APPLR is trained on 300 Gazebo worlds. Singularity containers were used over Docker containers because our Mastodon computing cluster did not have Docker installed due to a couple of reasons. Firstly, for all practical purposes, Docker grants superuser privileges, something that is not recommended in multi-user settings like a computing cluster. Secondly, the Mastodon cluster uses the HTCondor scheduler, and users submit jobs with CPU, memory, and time requirements. The Docker command is just an API that talks to the Docker daemon, so the resource requests and actual usages do not match. Singularity, on the other hand, runs container processes without a daemon, so all jobs are just run as child processes.

Since we were using the Clearpath Jackal ground robot for all of our experiments, our Singularity container contained not only ROS Melodic but also all the packages required for the Jackal robot, such as jackal simulator and jackal desktop. Singularity builds containers from definition files: .def files that contain all the packages and installations that are required for the experiments. Once the container is built, we wrote a shell script to bind a folder on the host node to a folder in the container, so the experiences of the actors are saved into the folder on the host node and not lost. We then wrote a script to automatically create and submit the HTCondor submission jobs based on the number of actors we needed. Code for the .def file and script to generate and submit the submission files are provided [1].

Conclusions

In this abstract, we present a framework for using containers on large computing clusters for RL training and testing. More specifically, we detail the use of Singularity containers and HTCondor scheduler on large computing clusters to carry out training and testing of Actor-Critic algorithms. Our PCRL framework provides a way to make distributed training and testing easier using containers and holds three major benefits: (1) a generic parallelized RL training framework that is compatible with different RL algorithms; (2) can easily be applied to different robotic projects with unsupported software stack in the cluster; (3) inherit the reproducibility and distributability of the container.

References

Fujimoto, S.; van Hoof, H.; and Meger, D. 2018. Addressing Function Approximation Error in Actor-Critic Methods.

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor.

Kurtzer, G.; Sochat, V.; and Bauer, M. 2017. Singularity: Scientific containers for mobility of compute. *PLoS ONE* 12(5): e0177459.

Merkel, D. 2014. Docker: lightweight linux containers for consistent development and deployment. *Linux journal* 2014(239): 2.

Quigley, M.; Conley, K.; Gerkey, B.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; and Ng, A. Y. 2009. ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, volume 3, 5. Kobe, Japan.

Tannenbaum, T.; Wright, D.; Miller, K.; and Livny, M. 2001. Condor – A Distributed Job Scheduler. In Sterling, T., ed., *Beowulf Cluster Computing with Linux*. MIT Press.

Xu, Z.; Dhamankar, G.; Nair, A.; Xiao, X.; Warnell, G.; Liu, B.; Wang, Z.; and Stone, P. 2020. Applr: Adaptive planner parameter learning from reinforcement. *arXiv preprint arXiv:2011.00397* .

[1] PCRL Implementation for APPLR: https://github.com/dgauraang/APPLR-1