Decentralized Distributed PPO

Erik Wijmans[1,2*] **Abhishek Kadian**[2] **Ari Morcos**[2] **Stefan Lee**[1,3]
Irfan Essa[1] **Devi Parikh**[2,3] **Manolis Savva**[2,4] **Dhruv Batra**[1,2]
[1]Georgia Institute of Technology [2]Facebook AI Research
[3]Oregon State University [4]Simon Fraser University

Abstract

We present Decentralized Distributed Proximal Policy Optimization (DD-PPO), a method for distributed reinforcement learning in resource-intensive simulated environments. DD-PPO is distributed (uses multiple machines), decentralized (lacks a centralized server), and synchronous (no computation is ever 'stale'), making it conceptually simple and easy to implement. In our experiments on training virtual robots to navigate in Habitat-Sim (Savva et al., 2019), DD-PPO exhibits *near-linear scaling* – achieving a speedup of 107x on 128 GPUs over a serial implementation. We leverage this scaling to train an agent for *2.5 Billion* steps of experience (the equivalent of 80 years of human experience) – over *6 months of GPU-time* training in under 3 days of wall-clock time with 64 GPUs. This massive-scale training not only sets the state of art on Habitat Autonomous Navigation Challenge 2019, but essentially 'solves' the task – *near-perfect* autonomous navigation in an *unseen* environment *without* access to a map, directly from an `RGB-D` camera and a GPS+Compass sensor.

1 Introduction

Recent advances in deep reinforcement learning (RL) have given rise to systems that can outperform human experts at variety of games (Silver et al., 2017; Tian et al., 2019). These advances, even more-so than those from supervised learning, rely on significant numbers of training samples, making them impractical without large-scale, distributed parallelization. Thus, scaling RL via multi-node distribution is of importance to AI – that is the focus of this work.

Several works have proposed systems for distributed RL (Silver et al., 2017; Tian et al., 2019). These works utilize two core components: 1) workers that collect experience ('rollout workers'), and 2) a parameter server that optimizes the model. The rollout workers are then distributed across, potentially, thousands of CPUs. However, synchronizing thousands of workers introduces significant overhead (the parameter server must wait for the *slowest* worker). To combat this, they wait for only a few rollout workers, and then *asynchronously* optimize the model.

However, this paradigm – of a single parameter server and thousands of (typically CPU) workers – appears to be fundamentally incompatible with the needs of modern computer vision and robotics communities. Over the last few years, a large number of works have proposed training virtual robots (or *'embodied agents'*) in rich 3D simulators before transferring the learned skills to reality (Beattie et al., 2016; Das et al., 2018). Unlike Gym or Atari, 3D simulators require GPU acceleration, and, consequently, the number of workers is greatly limited ($2^{5 \text{ to } 8}$ vs. $2^{12 \text{ to } 15}$). The desired agents operate from high dimensional inputs (pixels) and, consequentially, use deep networks (ResNet50) that strain the parameter server. Thus, there is a need to develop a distributed architecture.

Contributions. We propose a simple, synchronous, distributed RL method that scales well. We call this method Decentralized Distributed Proximal Policy Optimization (DD-PPO) as it is decentralized (has no parameter server), distributed (runs across many different machines), and we use it to scale Proximal Policy Optimization (Schulman et al., 2017).

*Work done while an intern at Facebook AI Research. Correspondence to `etw@gatech.edu`.

Workshop on Systems for ML at NeurIPS 2019

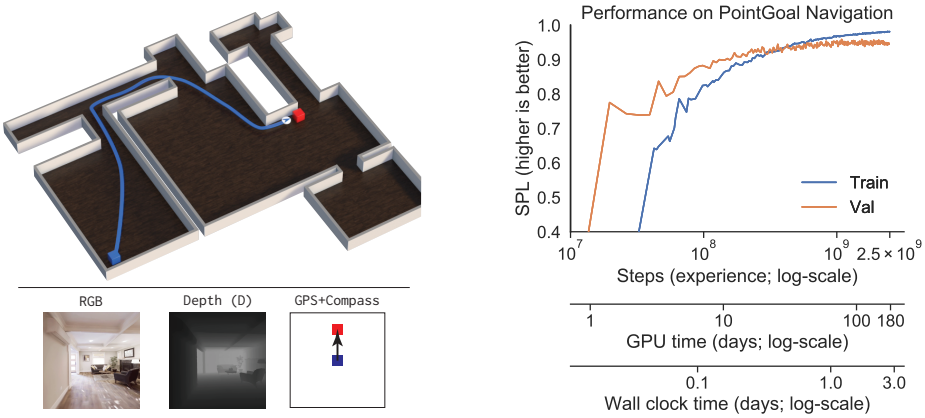

Figure 1: Left: In PointGoal Navigation, an agent must navigate from a random starting location (blue) to a target location (red) specified relative to the agent ("Go 5m north, 10m east of you") in a previously *unseen* environment *without* access to a map. Right: Performance (SPL; higher is better) of an agent equipped with `RGB-D` and GPS+Compass sensors on the Habitat Challenge 2019 (Savva et al., 2019) train & val sets. Using DD-PPO, we train agents for over *180 days* of GPU-time in under 3 days of wall-clock time with 64 GPUs, achieving state-of-art results and 'solving' the task.

In DD-PPO, each worker alternates between collecting experience in a resource-intensive and GPU accelerated simulated environment and optimizing the model. This distribution is *synchronous* – there is an explicit communication stage where workers synchronize their updates to the model (the gradients). To avoid delays due to stragglers, we propose a *preemption threshold* where the experience collection of stragglers is forced to end early once a pre-specified percentage of the other workers finish collecting experience. All workers then begin optimizing the model.

We characterize the scaling of DD-PPO by the steps of experience per second with N workers relative to 1 worker. We consider two different workloads, 1) simulation time is roughly equivalent for all environments, and 2) simulation time can vary dramatically due to large differences in environment complexity. Under both workloads, we find that DD-PPO scales *near-linearly*. While we only examined our method with PPO, other on-policy RL algorithms can easily be used and we believe the method is general enough to be adapted to *off*-policy RL algorithms.

We utilize DD-PPO to train agents for PointGoal Navigation (Anderson et al., 2018) (`PointGoalNav`) (Fig. 1). We achieve state-of-the-art on both `RGB-D` and `RGB` tracks in the Habitat Challenge 2019, and 'solve' the task of `PointGoalNav` for agents with GPS+Compass. These agents 1) almost always reach the goal (failing on 1/1000 val episodes on average), and 2) reach it *nearly as efficiently as possible* – nearly matching (within 3% of) the performance of a *shortest-path oracle*! It is worth stressing how uncompromising that comparison is – in a *new* environment, an agent navigating without a map traverses a path nearly matching the shortest path on the map. This means there is no scope for mistakes of any kind – no wrong turn at a crossroad, no back-tracking from a dead-end, no exploration or deviation of any kind from the shortest-path.

2 Decentralized Distributed Proximal Policy Optimization

In reinforcement learning, the dominant paradigm for distribution is asynchronous (see Fig. 2). Asynchronous distribution is notoriously difficult – even minor errors can result in opaque crashes – and the parameter server and rollout workers necessitate separate programs.

In supervised learning, however, synchronous distributed training via data parallelism (Hillis & Steele Jr, 1986) dominates. As a general abstraction, this method implements the following: at step k, worker n has a copy of the parameters, θ_n^k, calculates the gradient, $\partial\theta_n^k$, and updates θ via

$$\theta_n^{k+1} = \texttt{ParamUpdate}\Big(\theta_n^k, \texttt{AllReduce}\big(\partial\theta_1^k, \ldots, \partial\theta_N^k\big)\Big) = \texttt{ParamUpdate}\Big(\theta_n^k, \frac{1}{N}\sum_{i=1}^{N}\partial\theta_i^k\Big), \quad (1)$$

where `ParamUpdate` is any first-order optimization technique (*e.g.* gradient descent) and `AllReduce` performs a reduction (*e.g.* mean) over all copies of a variable and returns the result to all workers.

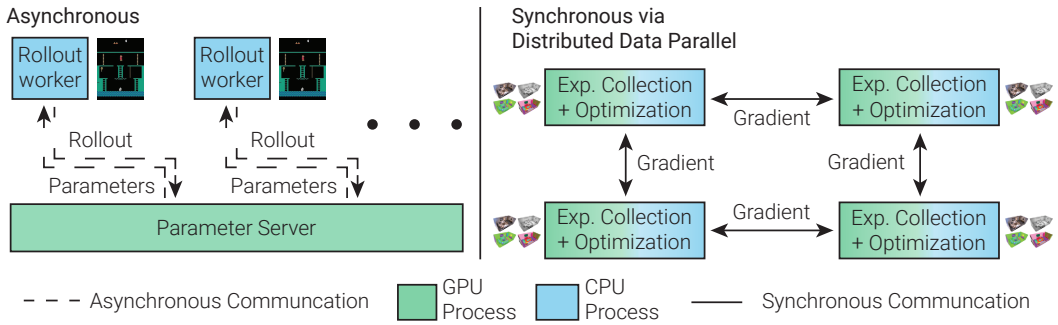

Figure 2: Comparison of asynchronous distribution (left) and synchronous distribution via distributed data parallelism (right) for RL. Left: rollout workers collect experience and asynchronously send it to the parameter-server. Right: a worker alternates between collecting experience, synchronizing gradients, and optimization. We find this highly effective in resource-intensive environments.

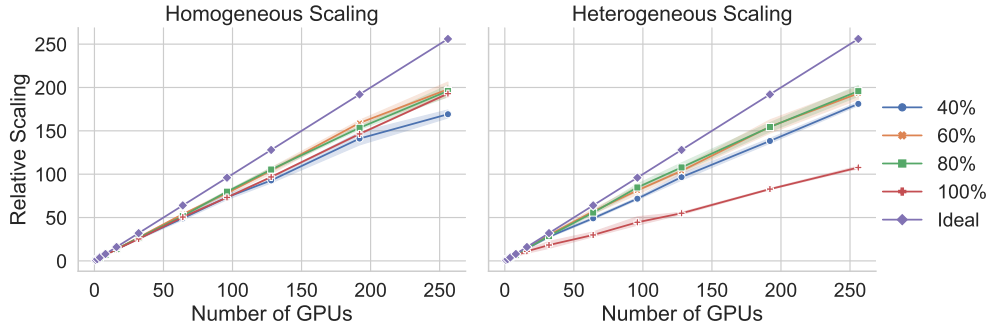

Figure 3: Scaling performance (in steps of experience per second relative to 1 GPU) of DD-PPO for various preemption threshold, $p\%$, values. Shading represents a 95% confidence interval.

Distributed DataParallel scales very well (near-linear scaling up to 32,000 GPUs), and is reasonably simple to implement (all workers synchronously running identical code).

We adapt this to on-policy RL as follows: At step k, a worker n has a copy of the parameters θ_n^k; it gathers experience (rollout) using $\pi_{\theta_n^k}$, calculates the parameter-gradients ∇_θ via any policy-gradient method (*e.g.* PPO), synchronizes these gradients with other workers, and updates the model.

A key challenge to using this method in RL is variability in experience collection run-time. In supervised learning, all gradient computations take approximately the same time. In RL, some resource-intensive environments can take significantly longer to simulate. This introduces significant synchronization overhead as every worker must wait for the slowest to finish collecting experience. To combat this, we introduce a preemption threshold where the rollout collection stage of these stragglers is preempted (forced to end early) once some percentage, $p\%$, (we find 60% to work well) of the other workers are finished collecting their rollout; thereby dramatically improving scaling. We weigh all worker's contributions to the loss equally and limit the minimum number of steps before preemption to one-fourth the maximum to ensure all environments contribute to learning.

While we only examined our method with PPO, other on-policy RL algorithms can easily be used and we believe the method can be adapted to *off*-policy RL algorithms. Off-policy RL algorithms also alternate between experience collection and optimization, but differ in how experience is collected/used and the parameter update rule. Our adaptations simply add synchronization to the optimization stage and a preemption to the experience collection stage.

3 Benchmarking: How does DD-PPO scale?

In this section, we examine how DD-PPO scales under two different workload regimes – homogeneous (every environment takes approximately the same amount of time to simulate) and heterogeneous (different environments can take orders of magnitude more/less time to simulate). We examine

the number of steps of experience per second with N workers relative to 1 worker. We compare different values of the preemption threshold $p\%$. We benchmark training our ResNet50 `PointGoalNav` agent with `Depth` on a cluster with Nvidia V100 GPUs and NCCL2.4.7 with Infiniband interconnect.

Homogeneous. To create a homogeneous workload, we train on scenes from the Gibson dataset, which require very similar times to simulate agent steps. As shown in Fig. 3 (left), DD-PPO exhibits *near-linear scaling* (linear = ideal) for preemption thresholds larger than 50%, achieving a 196x speed up with 256 GPUs relative to 1 GPU and an 7.3x speed up with 8 GPUs relative to 1.

Heterogeneous. To create a heterogeneous workload, we train on scenes from both Gibson and Matterport3D. Unlike Gibson, MP3D scenes vary significantly in complexity and time to simulate – the largest contains 8GB of data while the smallest is only 135MB. DD-PPO scales poorly at a preemption threshold of 100% (no preemption) due to the substantial straggler effect (one rollout taking substantially longer than the others); see Fig. 3 (right). However, with a preemption threshold of 80% or 60%, we achieve *near-identical* scaling to the homogeneous workload! We found no degradation in performance of models trained with any of these values for the preemption threshold despite learning in large scenes occurring at a lower frequency.

4 Related Work

Synchronous Distributed RL. Our closest related work is that of Stooke & Abbeel (2018). They also propose to use distributed data parallelism to scale reinforcement learning but for Atari games, and do not find it very effective. We hypothesize that this is due to the number of environments per worker that is achievable with resource-light simulated environments like Atari and OpenAI Gym – 32 or 64 rollout workers per worker is common. This distributed technique relies on a single worker collecting experience from multiple environments in a *synchronous* method that steps all environments in lock-step. This introduces significant synchronization costs as every step in the rollout must be synchronized across as many as 64 processes – on average, it takes approximate the same amount of time to step 8 pong environments as it does 1, but it takes 10 times longer to step 64. In contrast, we evaluate this technique in resource-intensive environments, where it is common to only have 2 or 4 environments per worker, and find this technique to be effective.

Straggler Effect Mitigation. In supervised learning, the straggler effect is commonly caused by heterogeneous hardware or hardware failures. Chen et al. (2016) propose a pool of b "back-up" workers (there are $N + b$ workers total) and perform the parameter update once N workers finish. In comparison, their method a) requires a parameter server, and b) discards all work done by the stragglers. Chen et al. (2018) propose to dynamically adjust the batch size of each worker such that all workers perform their forward and backward pass in the same amount of time. Our method aims to reduce variance in *experience collection* times. While DD-PPO does dynamically adjust a worker's batch size, this is a necessary side-effect of on-policy RL.

References

P. Anderson, A. Chang, D. S. Chaplot, A. Dosovitskiy, S. Gupta, V. Koltun, J. Kosecka, J. Malik, R. Mottaghi, M. Savva, et al. On evaluation of embodied navigation agents. *arXiv:1807.06757*, 2018.

C. Beattie, J. Z. Leibo, D. Teplyashin, T. Ward, M. Wainwright, H. Küttler, A. Lefrancq, S. Green, V. Valdés, A. Sadik, J. Schrittwieser, K. Anderson, S. York, M. Cant, A. Cain, A. Bolton, S. Gaffney, H. King, D. Hassabis, S. Legg, and S. Petersen. Deepmind lab. *arXiv*, 2016.

C. Chen, Q. Weng, W. Wang, B. Li, and B. Li. Fast distributed deep learning via worker-adaptive batch sizing. *arXiv:1806.02508*, 2018.

J. Chen, X. Pan, R. Monga, S. Bengio, and R. Jozefowicz. Revisiting distributed synchronous sgd. *arXiv:1604.00981*, 2016.

A. Das, S. Datta, G. Gkioxari, S. Lee, D. Parikh, and D. Batra. Embodied Question Answering. In *CVPR*, 2018.

W. D. Hillis and G. L. Steele Jr. Data parallel algorithms. *ACM*, 29(12):1170–1183, 1986.

M. Savva, A. Kadian, O. Maksymets, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, V. Koltun, J. Malik, D. Parikh, and D. Batra. Habitat: A Platform for Embodied AI Research. *ICCV*, 2019.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv:1707.06347*, 2017.

D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

A. Stooke and P. Abbeel. Accelerated methods for deep reinforcement learning. *arXiv:1803.02811*, 2018.

Y. Tian, J. Ma, Q. Gong, S. Sengupta, Z. Chen, J. Pinkerton, and C. L. Zitnick. Elf opengo: An analysis and open reimplementation of alphazero. *arXiv:1902.04522*, 2019.