SRL: Scaling Distributed Reinforcement Learning to Over Ten Thousand Cores

Wei Fu [1 2 *] **Zhiyu Mei** [1 2 *] **Guangju Wang** [2] **Huanchen Zhang** [1 2] **Yi Wu** [1 2]

Abstract

The ever-growing complexity of reinforcement learning (RL) tasks demands a distributed system to train intelligent agents by efficiently producing and processing a massive amount of data. In this paper, we propose a comprehensive computational abstraction for RL training tasks and introduce a scalable, efficient, and extensive RL system called ReaLly Scalable RL (SRL), featuring a novel architecture that separates three major computation components in RL training. Our evaluation demonstrates that SRL outperforms a popular open-source RL system RLlib (Liang et al., 2017) in training throughput. Moreover, to assess the learning performance of SRL, we conduct a benchmark on a large scale cluster with 32 Nvidia A100 GPUs, 64 Nvidia RTX 3090 GPUs and more than 12k CPU cores, reproducing the results of industrial production system from OpenAI, Rapid (Berner et al., 2019) in the hide-and-seek environment (Baker et al., 2019). The results show that SRL is capable of achieving up to 5 times training speedup compared to published results in Baker et al. (2019).

1. Introduction

Reinforcement Learning (RL) is a prominent subfield of machine learning that trains intelligent decision-making agents to maximize a cumulative reward signal, which has been a popular paradigm leading to a lot of AI breakthroughs (Silver et al., 2016; Berner et al., 2019; Vinyals et al., 2019). As RL tasks getting more and more complex, training a strong neural network policy requires millions to billions of trajectories. Simply generating such a massive amount of data sequentially would take hundreds or even thousands of years. Therefore, building a system that can parallelize the data collection process and perform efficient RL training over the massive trajectories becomes a fundamental requirement for applying RL to real-world applications.

Numerous open-source libraries or frameworks are available to facilitate efficient RL training (Liang et al., 2017; Espeholt et al., 2019; Hoffman et al., 2020). However, we have identified several limitations in these systems that hinder their abilities to train RL agents efficiently in various scenarios. First, existing open-source systems have made unnecessary assumptions about computing resources, such as types and physical locations of hardware accelerators, making their architectures only efficient in corresponding circumstances. Second, their implementations lack support for multi-node multi-GPU training, which can hinder efficient RL training in complex tasks that require a sophisticated policy and a large batch size. Third, prior works primarily focus on small- to medium-scales, resulting in simplistic implementation details with inadequate considerations for performance optimization.

To address the aforementioned challenges, we present a novel abstraction on the dataflows of RL training, which effectively unifies training tasks in diverse circumstances into a simple framework. At a high level, we introduce the notion of "*workers*" to represent all computational and data management components, each of which hosts distinct "*task handlers*" such as environments or RL algorithms. Workers are interconnected by "*streams*" and supported by "*services*". Based on such a framework, we propose a scalable, efficient, and extensive RL system, which we have named SRL (ReaLly Scalable RL), that can be highly efficient in a wide range of scenarios, ranging from local machines to large, customized clusters featuring heterogeneous computation resources. SRL encompasses three primary types of workers: *actor workers*, *policy workers* and *trainer workers*, corresponding to three major computational components in RL training tasks. To minimize communication overhead, data transfer between workers is facilitated by *inference streams* and *sample streams*, which can utilize either network sockets or local shared memory. After each training step, updated parameters are pushed to the parameter database and periodically broadcasted to policy workers by the *parameter service*.

[*]Equal contribution [1]Institute for Interdisciplinary Information Sciences, Tsinghua University, China [2]Shanghai Qi Zhi Institute, China. Correspondence to: Zhiyu Mei <meizy20@mails.tsinghua.edu.cn>, Wei Fu <fuwth17@gmail.com>, Yi Wu <jxwuyi@gmail.com>.

Work presented at the ES-FoMo Workshop at ICML 2023.

The architecture of SRL promotes complete decoupling of workers, thereby allowing for elastic scheduling and finding an optimal configuration on a customized cluster with heterogeneous hardware resources. This flexible approach enables the allocation of specific resources, such as CPUs or GPUs with varying computation powers, based on the requirements of each task handler.

We assessed the effectiveness of SRL on a diverse collection of RL environments. First, through comparison with a popular open-source RL system RLlib (Liang et al., 2017), we demonstrated that SRL can achieve significantly higher training throughput and resource efficiency in a distributed setting. Second, we conducted a benchmark of the system's learning performance in terms of both sample efficiency and wall-clock time. To the best of our knowledge, SRL is the first academic distributed RL system capable of reproducing the results of OpenAI's industrial production system, *Rapid* (Berner et al., 2019), on the hide-and-seek environment (HnS) (Baker et al., 2019), and even obtaining up to a 3$\sim$5x training speedup.

2. System Architecture

2.1. High-Level Design of SRL

To address the limitations of previous designs, we propose a more general computation abstraction of RL training tasks. SRL is composed of multiple interconnected "workers" that host distinct "task handlers", such as environments and RL algorithms. These workers are connected via data "streams" and supported by background "services". We illustrate the functionality of each component of SRL in Fig. 1. Specifically, SRL incorporates three core types of workers: *actor worker*, *policy worker*, and *trainer worker*, which are responsible for the three pivotal workloads in RL training tasks, i.e. environment simulation, policy inference and training. Actor workers simulate environments — they produce rewards and next-step observations based on received actions. Policy workers perform batched forward propagation of the neural policy to generate actions given observations. Trainer workers run stochastic gradient descent steps given training samples to update the policy. Actor workers post inference requests containing observations to policy workers, and policy workers respond by the generated actions. We abstract this client-server communication pattern as *inference stream*. In parallel with environment simulation, actor workers accumulate observation-action-reward tuples in the local buffer and periodically sends them to trainer workers. We abstract this push-pull communication pattern as *sample stream*. After each training step, the updated parameter is pushed to a *parameter server*, which handles pull requests from policy workers for parameter synchronization. All workers in SRL can be independently scheduled and distributed across multiple machines with heterogeneous resources, connected by most efficient interfaces available (See Fig. 2).

In Sec. 2.2, we will provide a detailed description of each system component and explain how SRL attains scalability, efficiency, and flexibility as an integrated system.

2.2. System Components

2.2.1. ACTOR WORKER AND POLICY WORKER

Actor workers host environments to handle the execution of black-box environment programs. Environment simulation is usually self-contained within each actor, making it straightforward to be massively parallelized given sufficient computation resources. On the other side, policy workers host RL agents (or policies) and provide batched inference service for actor workers. They can effectively utilize GPU devices for accelerated neural policy forward propagation.

Environment simulation is usually divided into episodes. In the beginning of each episode, an actor worker resets its environment and gets the first observation. Then before every environment step, each actor worker sends out the observation of the last step (or the initial reset) and requests an action from policy workers to continue to the next step.

Policy workers flush received inference requests from multiple actor workers, compute forward passes on policy models with batched observations, and respond them with output actions. To keep policy models up-to-date, policy workers also need to pull parameters from parameter servers once in a while. Data transmission, parameter synchronization, and neural network inference are handled in three different threads.

Our implementation of policy worker also supports a local CPU mode, which we call *inline inference*. In this case, inference stream module will ensure direct data transmission between an actor and its associated local policy worker with proper batching and without using the network.

2.2.2. TRAINER WORKERS

Trainer workers are responsible for computing gradient descent iterations. They receives samples from actor workers. Each trainer worker is embedded with a buffer to store samples waiting for being fetched into GPU. Before every iteration of gradient descent, a trainer worker aggregates a batch of samples from data buffer and load them into its GPU device. Jobs on CPU and GPU are separated into two threads.

In SRL, we support multi-GPU training across nodes. By our multi-trainer design, each trainer worker is assigned to exactly one GPU device, which means one trainer worker is a minimal unit for training computations.

Note that policy models in RL are usually small in size and

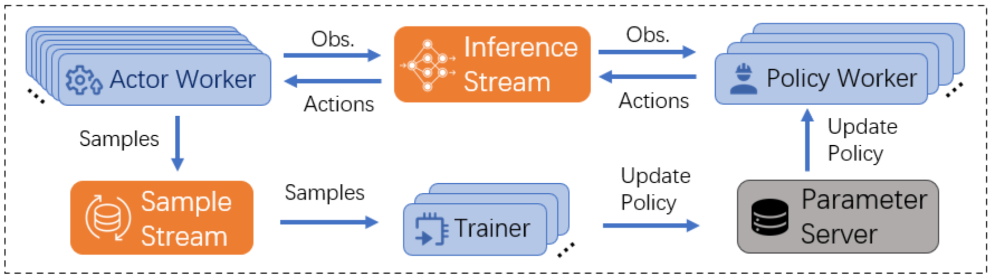


Figure 1: Core components of SRL. Blue arrows shows dataflow between workers and data streams. Blue boxes represent workers, which are responsible for computational workloads. Orange boxes represent data streams, represent communication between workers. Grey box represent storage, which is parameter server in our context.

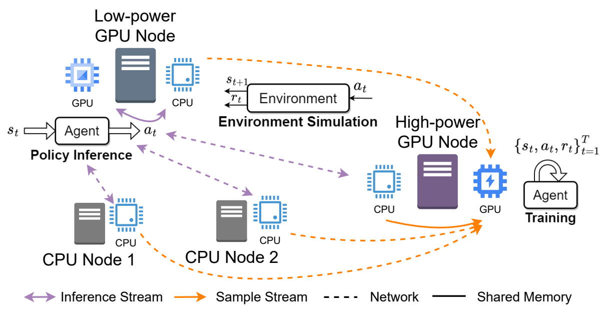


Figure 2: Distributed workload of SRL that can fully utilize all heterogeneous computation resources.

capable of fitting in a single GPU, so model parallelization is not required in most applications. Hence, we adopt single-program multi-data (SPMD) paradigm for our multi-trainer design. For a large batch of samples, we evenly distribute samples to multiple trainers, every one of which holds a copy of the same policy model. Each trainer computes gradient using their own copy, and synchronize the gradients to update the final policy model at the end of every training iteration. Trainer workers use PyTorch DistributedDataParallel (DDP) (Li et al., 2020) as the backend that communicate trainers and synchronize the gradients.

Additionally, there are circumstances where a single trainer worker may not be able to fully utilize the computing power of a GPU. To prevent unnecessary waste of computing powers, we allow other workers (e.g. policy workers) to share one GPU with a trainer worker.

2.2.3. Sample streams and Inference streams

In SRL, we identify two primitive types of data transmissions between workers. One is exchanging observations and actions between actor workers and policy workers. The other is sending samples from actor workers to trainer workers. Corresponding to the two types of data transmissions, we develop inference streams and sample streams, which have different data transmission patterns. Inference streams need to be duplex because actor workers send inference requests and policy workers are required to reply. Meanwhile, sample streams are simplex. Only actor workers send training samples to trainer workers, and trainer workers do not reply.

For network transfer, we implement inference stream as a pair of request-replay sockets and sample stream as a pair of push-pull sockets. For local shared-memory transfer, we instantiate inference stream as a block of pinned shared memory, i.e., each client is assigned to exactly one slot for read and write, and sample stream as a well-designed shared-memory FIFO queue. Different data streams establish independent and perhaps overlapped communications between groups of workers to ensure data from different policies never contaminate each other.

3. Experiments

This section presents an evaluation of the SRL by assessing its training throughput and learning performance. We employ Proximal Policy Optimization (PPO) (Schulman et al., 2017) as our primary choice of the RL algorithm. Experiments are evaluated in a large-scale cluster with following settings[1]: 4 nodes with 64 physical CPU cores and 8 Nvidia A100 GPUs + 64 nodes with 128 physical CPU cores and and an Nvidia 3090 GPU. Each node in the cluster had 512GB DRAM and are connected to each other by 100 Gbps intra-cluster network. Storage for the cluster was facilitated through NFS and parameter service, available on all nodes.

3.1. Training Throughput

We compared performance of SRL with RLlib (Liang et al., 2017) in distributed settings. The metric we use for evalua-

[1]All physical cores have hyperthreading enabled and count as 2 logical cores. In this section, if not emphasized, the term “CPU cores” will be referring to logical CPU cores.

Table 1: Training throughput of SRL and RLlib with 8 A100 GPU trainers. # CPU Cores (peak): CPU cores used for training sample generation when trainers reaches peak performance.

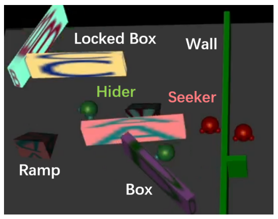
	Atari	Deepmind Lab	gFootball	SMAC
SRL(FPS)	643916	741396	89132	17053
# CPU Cores (Peak)	800	1600	3200	1280
SRL(FPS)	150288	65768	18988	4943
# CPU Cores	96	160	700	200
RLlib(FPS)	65585	34274	5802	2713
# CPU Cores (Peak)	96	160	700	200

tion is training environment frames per second (FPS), which refers to the number of environment frames consumed by all trainer workers per second.

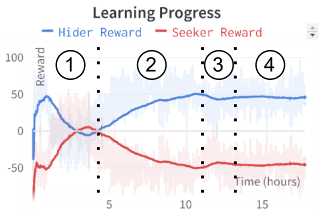
We run experiments on a set of academic environments, including Atari 2600 games (game *Pong*), Google Research Football (scenario *11_vs_11*), StarCraft Multi-Agent Challenge (map 27m_vs_30m), and Deepmind Lab (scenario *watermaze*), each of which possesses distinct characteristics in terms of observation type, speed, memory, etc. In Atari and DMLab environments, we adopt a traditional 4-frameskip setting, meaning that number of environment frames is 4 times actual training sample steps. We utilized 8 Nvidia A100 GPUs as trainer devices on the same machine with actors across different machines. For RLlib, we gradually increased the number of CPU cores until the trainers were fully occupied, and adding more CPU cores would not increase the overall throughput. We then record the number of CPU cores used and run the same configuration for SRL. Further, we also evaluated the highest FPS number that SRL is capable of reaching with 8 trainer workers. The results, presented in Table 1, demonstrate that, compared to RLlib, SRL achieved 6.3x to 21.6x higher maximal performance, and 1.4x to 3.3x performance when using the same numbers of CPU cores. We remark that trainer workers of SRL can effectively utilize training samples generated by much more CPU cores compared to RLlib's single-endpoint multi-threaded trainer. Moreover, actor workers in SRL are capable of generating more training samples with GPU-accelerated inference and environment rings. As a result, the overall performance of SRL dominates RLlib.

3.2. Learning Performance

While quantifying the training throughput of SRL has yielded valuable insights into its efficiency and scalability, it is equally crucial to evaluate its ability to develop intelligent agents (i.e., learning performance) in a realistic and challenging environment. In this context, the hide-and-seek (HnS) environment (Baker et al., 2019) emerges as an appealing choice. Due to the complexity of this task, Baker et al. (2019) employed the OpenAI Rapid system (Berner et al., 2019) for training, which makes it permissive for the research community to reproduce the results.



(a) HnS environment.



(b) Reward over time in HnS.

Figure 3: (a) A snapshot of HnS. (b) Rewards in HnS. Agent behavior evolves over four stages: running and chasing, box lock, ramp use, and ramp lock.

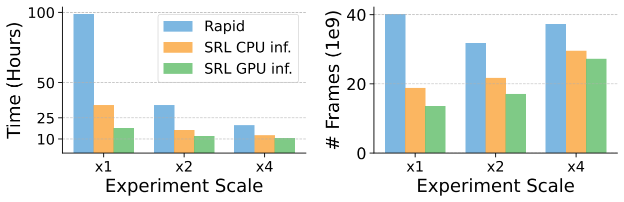


Figure 4: Time/data required to reach stage four in HnS.

We conduct experiments in the distributed setting using both inline CPU inference (denoted as *CPU Inf.*) and remote GPU inference (denoted as *GPU Inf.*). In our experiment, we present the training time and data volume required to achieve the ramp lock stage in HnS, as shown in Figure 4. Our results reveal that SRL is up to 3x faster than the Rapid system with the same architecture (*CPU Inf.*), while *GPU Inf.*can achieve up to 5x acceleration with further reduced time and environment interactions. We attribute the improvement in training efficiency to two reasons. First, our system design is more efficient and has a higher FPS than Rapid. Second, our flexible system design and fine-grained optimizations ensure that the efficiency of the RL algorithm is less affected by various system-related factors like network latency and out-of-date data, which leads to improved sample efficiency even with the same RL algorithm.

4. Conclusion

This paper presents a novel perspective on the dataflows of RL training, and proposes a scalable, efficient and extensive RL system named SRL. In our experiments, results show that SRL outperforms RLLib (Liang et al., 2017) in a distributed setup. We also show that SRL is efficient by comparing learning performance to OpenAI *Rapid* (Berner et al., 2019) in Hide-and-seek environment (Baker et al., 2019).

References

Baker, B., Kanitscheider, I., Markov, T. M., Wu, Y., Powell, G., McGrew, B., and Mordatch, I. Emergent tool use from multi-agent autocurricula. *CoRR*, abs/1909.07528, 2019. URL http://arxiv.org/abs/1909.07528.

Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., de Oliveira Pinto, H. P., Raiman, J., Salimans, T., Schlatter, J., Schneider, J., Sidor, S., Sutskever, I., Tang, J., Wolski, F., and Zhang, S. Dota 2 with large scale deep reinforcement learning. *CoRR*, abs/1912.06680, 2019. URL http://arxiv.org/abs/1912.06680.

Espeholt, L., Marinier, R., Stanczyk, P., Wang, K., and Michalski, M. SEED RL: scalable and efficient deep-rl with accelerated central inference. *CoRR*, abs/1910.06591, 2019. URL http://arxiv.org/abs/1910.06591.

Hoffman, M., Shahriari, B., Aslanides, J., Barth-Maron, G., Behbahani, F., Norman, T., Abdolmaleki, A., Cassirer, A., Yang, F., Baumli, K., Henderson, S., Novikov, A., Colmenarejo, S. G., Cabi, S., Gülçehre, Ç., Paine, T. L., Cowie, A., Wang, Z., Piot, B., and de Freitas, N. Acme: A research framework for distributed reinforcement learning. *CoRR*, abs/2006.00979, 2020. URL https://arxiv.org/abs/2006.00979.

Li, S., Zhao, Y., Varma, R., Salpekar, O., Noordhuis, P., Li, T., Paszke, A., Smith, J., Vaughan, B., Damania, P., and Chintala, S. Pytorch distributed: Experiences on accelerating data parallel training. *CoRR*, abs/2006.15704, 2020. URL https://arxiv.org/abs/2006.15704.

Liang, E., Liaw, R., Nishihara, R., Moritz, P., Fox, R., Gonzalez, J., Goldberg, K., and Stoica, I. Ray rllib: A composable and scalable reinforcement learning library. *CoRR*, abs/1712.09381, 2017. URL http://arxiv.org/abs/1712.09381.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, jan 2016. ISSN 0028-0836. doi: 10.1038/nature16961.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., Vezhnevets, A. S., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Paine, T. L., Gulcehre, C., Wang, Z., Pfaff, T., Wu, Y., Ring, R., Yogatama, D., Wünsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Kavukcuoglu, K., Hassabis, D., Apps, C., and Silver, D. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782): 350–354, Nov 2019. ISSN 1476-4687. doi: 10.1038/s41586-019-1724-z. URL https://doi.org/10.1038/s41586-019-1724-z.