MSRL: Distributed Reinforcement Learning with Dataflow Fragments

AUTHORS: H. ZHU, B. ZHAO, G. CHEN, W. CHEN, Y. CHEN, L. SHI, Y. YANG, P. PIETZUCH, L. CHEN

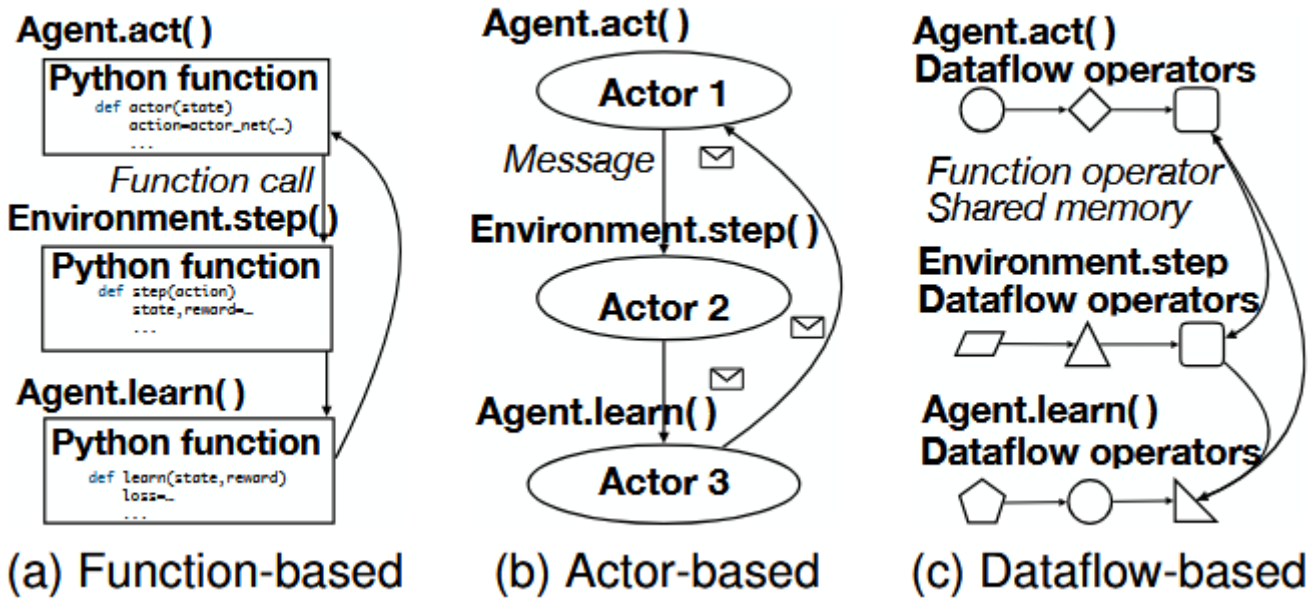
PRESENTED BY OGNEN PENDAROVSKI

Motivation: RL is powerful but resource demanding

- Reinforcement learning approaches for AI have yielded impressive results over the past decade: in board games, video games, robotics, and LLMs.
- These successes have required immense computational resources:
  - AlphaStar – 384 TPUv3 chips, 150 CPUs with 28 cores each;
  - OpenAI Five – 1536 optimizer GPUs, 1440 rollout GPUs, 172800 rollout CPUs;
  - RLHF and RLVR – high VRAM requirements even for small models, rollouts expensive due to high inference costs.
- To solve more such problems, scalable, highly parallel systems are needed.

RL distributed system classes

- Existing RL distributed system frameworks fall into three categories: function-based, actor-based, and dataflow-based approaches.



RL distributed system classes

- Function-based:
  - The RL algorithms are represented by functions/methods, called in the training loop and directly executed by workers;
  - Typical examples are Acme, SEED-RL, with Rlgraph allowing greater abstraction.
- Actor-based:
  - Treat algorithms as actors deployed on workers, distributed using a scheduler;
  - Examples include Ray, RLlib, MALib.
- Dataflow-based:
  - Describes algorithms as computation graphs of operators, with distribution strategies being fixed;
  - Examples are Podracer, RLlib Flow, WarpDrive.
- Hardware acceleration available only for neural network inference and training.

Motivation: new type of system

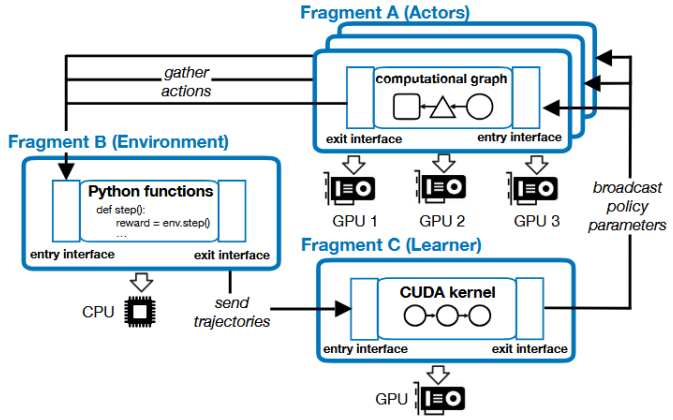
- Disentangling the specific algorithm (and its implementation) from the distribution strategy, allowing for separate optimization of both.
- Abstracting away the device type and software implementation of each subset of code being executed, allowing specialized optimization for each subtask, on top of variable distribution strategy.

MindSporeRL: fragmented dataflow paradigm

- Introduces fragmented dataflow paradigm.
- Each fragment is a specific dataflow graph built out of predefined primitives and can be independently executed on an abstracted worker, with tailored software implementations.
- All the fragments are then connected in a fragmented dataflow graph (FDG), with each entry and exit interfaces on each node.

MindSporeRL: fragment flexibility

- An example of a useful abstraction of fragment hardware and software architecture:
  - Actor fragment: computation graph of policy neural network, parallelized across several GPUs, with specific accelerated inference engine;
  - Environment fragment: parallelized Python code for environment simulation, run on multiple CPU cores;
  - Learner fragment: CUDA kernel performing calculations on received rollout rewards, updates network parameters.



MindSporeRL: distribution policies

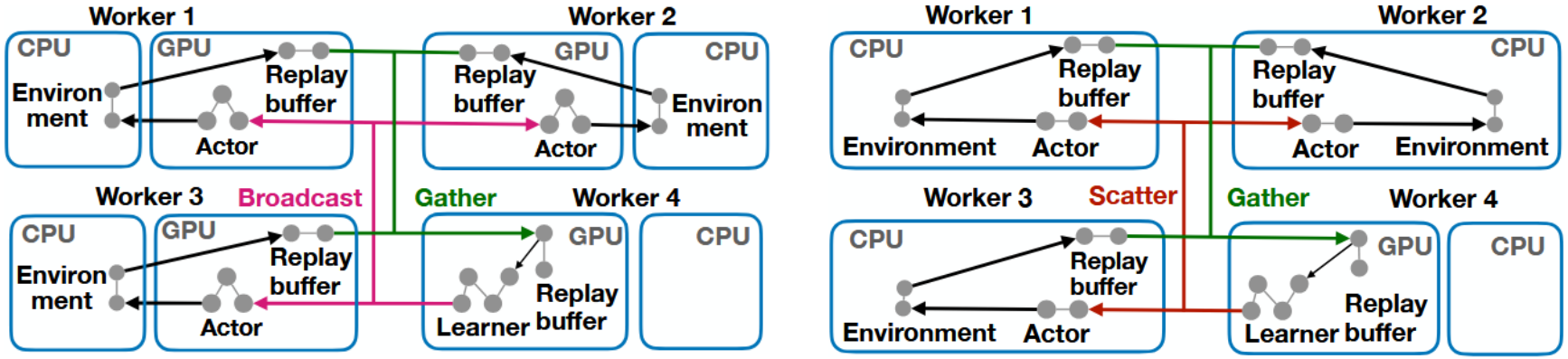
- The distribution policy allocates the fragments amongst the worker systems.
- The choice of policy is strongly dependent on the specific RL algorithm, the architecture of the neural network, as well as the hardware parameters of the cluster, including the number of chips, their individual performance, and interconnect bandwidth.

MindSporeRL: example distribution policies

- While the framework itself provides the ability to customize distribution policies using their templates, the paper presents several classes as examples:
  - DP-SingleLearnerCoarse
  - DP-SingleLearnerFine
  - DP-MultiLearner

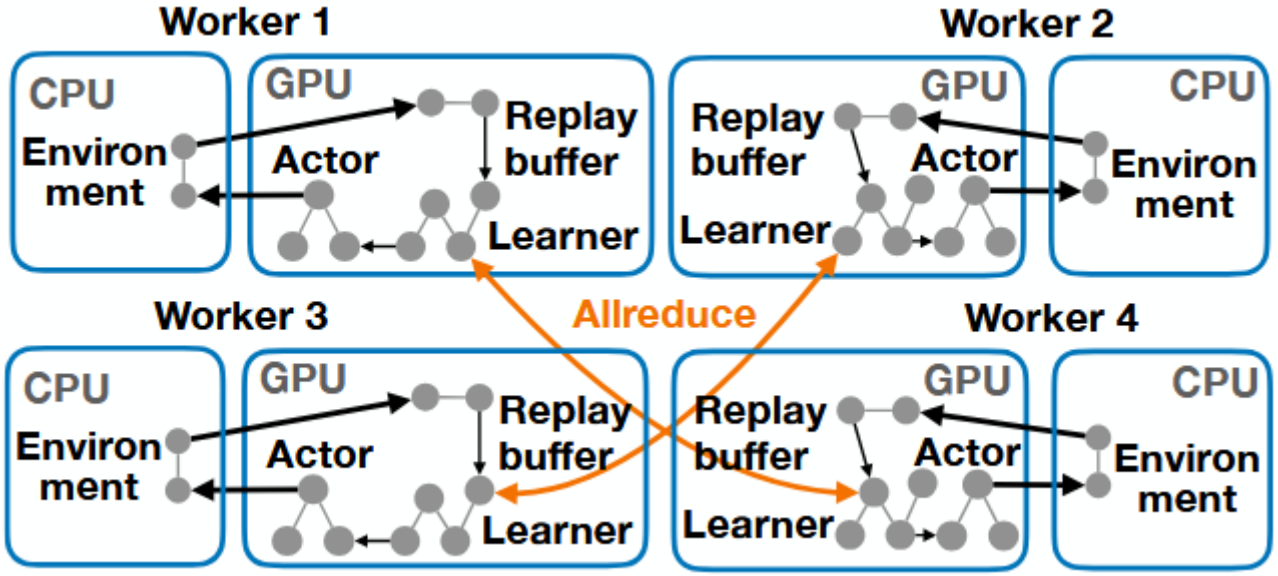
Single learner distribution policies

- DP-SingleLearnerCoarse – distributes multiple replicas of the actor and environment, while using a single learner.
  - Most suitable for complex environments but simpler neural networks.
  - The actors and the learner are all GPU accelerated, while the environment replicas run on CPUs.
- DP-SingleLearnerFine – like its coarse variant, only combines the actor and environment on the same CPU.
  - Not necessary to transmit policy parameters, better for large neural networks.



Multi learner distribution policy

- DP-MultiLearner performs learning with multiple workers.
- Each learner is co-located with an actor on a GPU, with the environment on the CPU of the same worker.
- It is a good choice when the generated training data is too great for single learner systems, or for decentralized training.



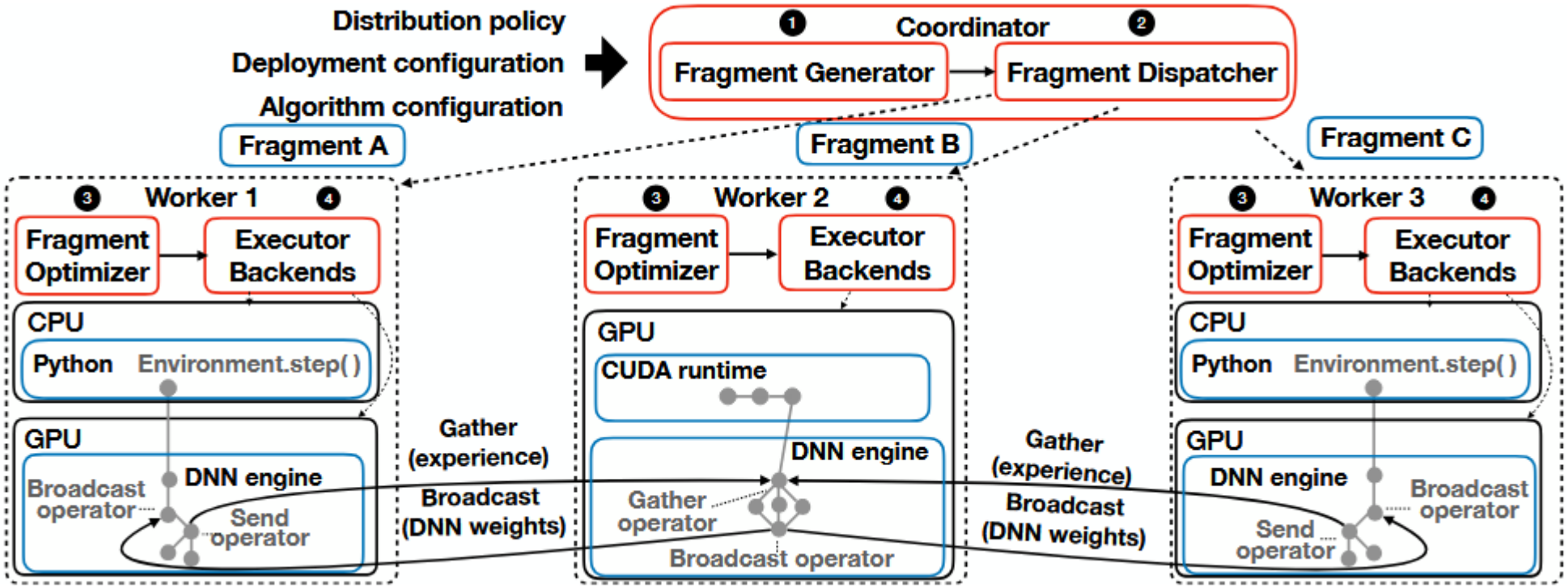
MindSporeRL: architecture

- The system architecture of MSRL is a coordinator/worker design.
- Given a distribution policy, the coordinator generates the fragmented dataflow graph and assigns the fragments to the workers.
- The worker devices all have at least one execution backend. The received fragments are optimized and sent to the backends.

MindSporeRL: architecture

- The coordinator consists of two components:
  - FDG generator: splits the RL algorithm according to the distribution policy, finds the boundaries and generates interfaces between fragments;
  - Fragment dispatcher: activates execution backends in each worker, sets up distributed communication between the fragment interfaces, and ultimately dispatches the fragments.
- The worker, similarly, has two parts:
  - Fragment optimizer: optimizes received fragments for the available backend by directly manipulating the AST;
  - Executor backends: performs the graph encoded computations on the target worker device.

MindSporeRL: architecture

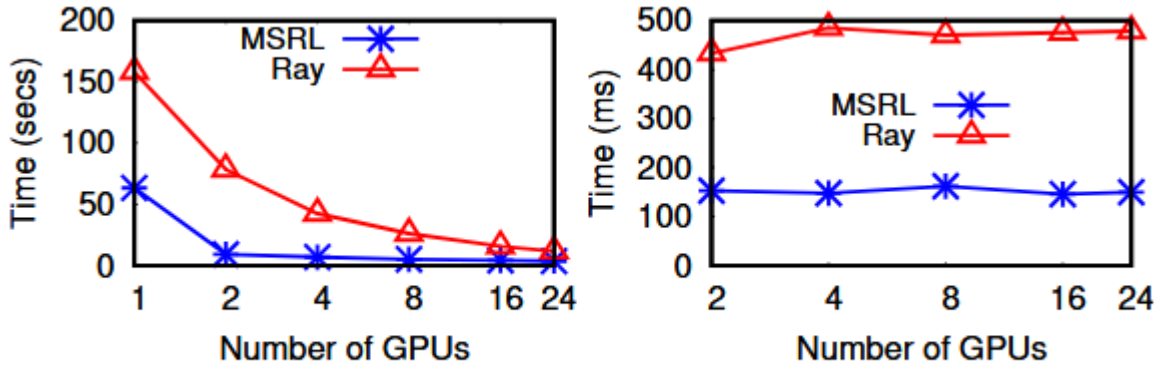


Empirical evaluation

- For evaluation purposes, the framework was used with the MindSpore DNN framework.
- The baseline comparison frameworks were Rllib (Ray) as a distributed systems competitor, and WarpDrive as a single GPU competitor.
- The measurements were performed on three RL algorithms:
  - Proximal policy optimization (PPO)
  - Multi agent PPO (MAPPO)
  - Asynchronous advantage actor-critic (A3C)

Evaluation against Ray

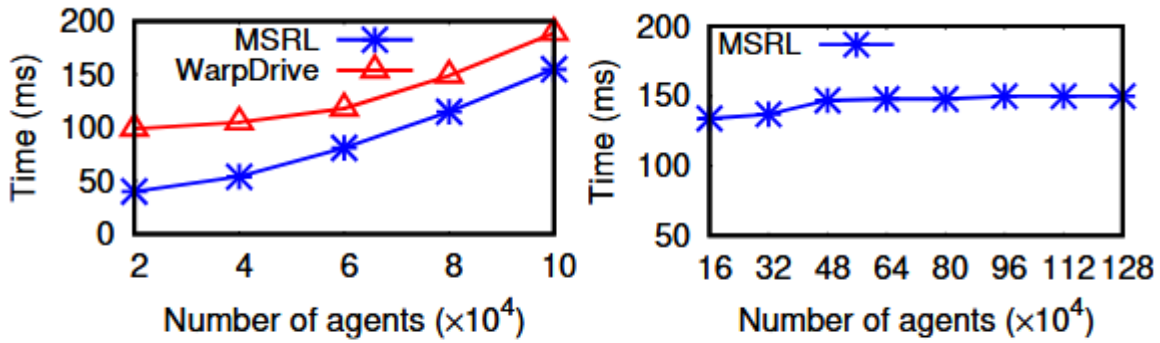
- For PPO, 320 environments were distributed between each actor, with a single learner for training.
- For A3C, multiple actors interacted with their environment and locally computed gradients which were learned asynchronously by one learner.



(a) Episode time vs. GPUs (PPO) (b) Episode time vs. GPUs (A3C)

Evaluation against WarpDrive

- The DP-GPUOnly distribution policy was used on MSRL, which combines the entire training loop into one fragment that is then copied to many devices.
- The comparison was done against WarpDrive.

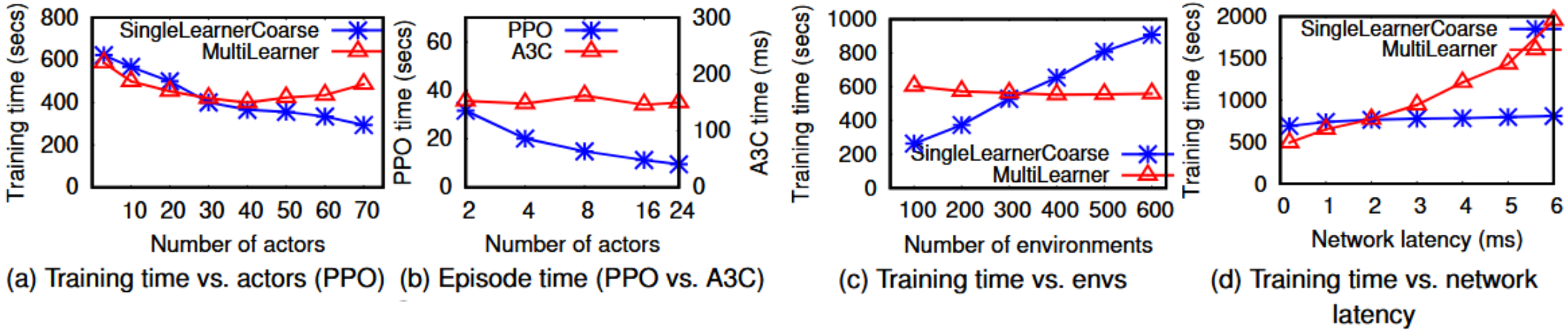


(a) Episode time vs. agents (1 GPU)

(b) Episode time vs. agents (n GPUs)

Evaluation with different distribution policies

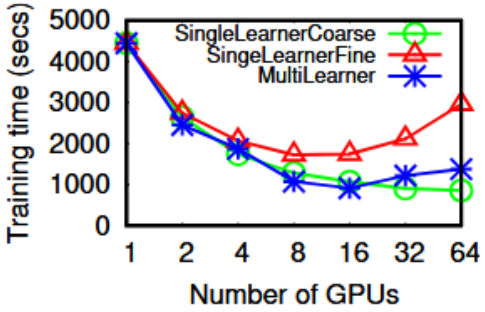
- Different policies differ in performance (a fundamental advantage in theory).
- DP-SingleLearnerCoarse was compared with DP-MultiLearner.



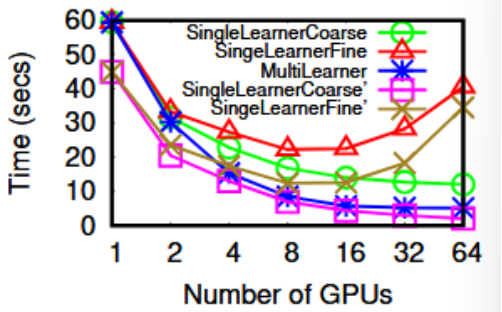
(a) Training time vs. actors (PPO) (b) Episode time (PPO vs. A3C) (c) Training time vs. envs (d) Training time vs. network latency

Evaluating hardware and software scalability

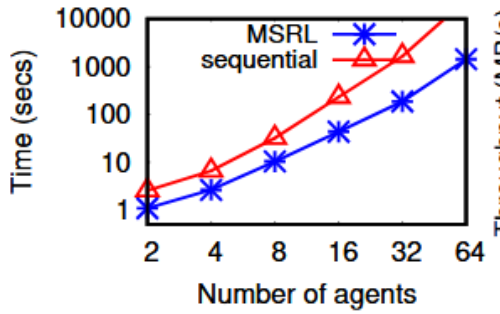
- The final set of comparisons is based on increasing the number of GPUs, agents, and episodes, and comparing the training time, episode time, throughput, and total reward value.



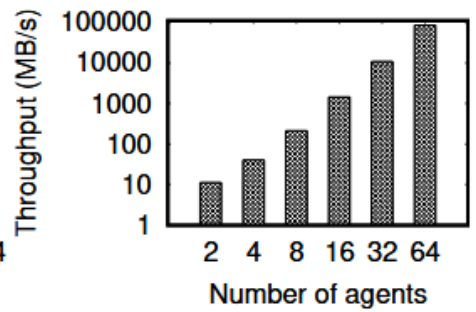
(a) Training time vs. GPUs



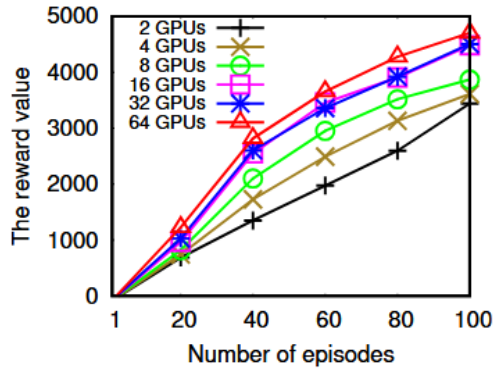
(b) Episode time vs. GPUs



(a) Training time per episode



(b) Training throughput



Personal opinion – strengths

- The primary advantage of the MSRL framework is the clever abstraction of fragments in conjunction with the programmable coordinator, allowing a decoupling between the selected RL algorithm and the way it is distributed in a cluster.
- The modularity afforded by this approach is an important component, as it theoretically allows arbitrary optimization of the software running on specific worker devices.

Personal opinion – weaknesses

- Multi-GPU scaling properties examined for up to 64 GPUs, a small number compared to the clusters used in the most successful RL projects of the past decade.
- The choice of distribution policy adds great complexity – this is to be expected, but the framework could use stronger defaults, ideally through an automated approach.
- Developer support seems to have stalled – no activity on the main GitHub page for nearly two years.

Summary

- Motivation: unnecessary coupling between algorithm logic and distribution strategy in existing frameworks.
- Potential solution: MSRL – uses new abstraction called fragmented dataflow graph, working in parallel with customizable distribution policies.
- Empirical results: better performance and modularity than previous systems.
- Potential drawbacks: unexplored at larger scales, lack of automatic distribution policies, and most crucially, seeming abandonment of the project.

References

- MSRL: Distributed Reinforcement Learning with Dataflow Fragments by H. Zhu et al.
- RLgraph: Modular Computation Graphs for Deep Reinforcement Learning by M. Schaarschmidt et al.
- Understanding and Alleviating Memory Consumption in RLHF for LLMs by J. Zhou et al.
- https://github.com/mindspore-lab/mindrl