

The City College of New York

CSC 59867-E: Senior Project II

AI Agents for Decision Making in the Real World

Scaling Multi-Agent AI problems with Distributed Learning

Saptarashmi Bandyopadhyay

By Saptarashmi Bandyopadhyay
Email: sbandyopadhyay@ccny.cuny.edu
Assistant Professor of Computer Science
City College of New York and Graduate Center at the City University of New York

October 21, 2026 CSC 59867 - E

Scaling Up: Distributed Processing for Multi-Agent AI

Saptarashmi Bandyopadhyay

Recall: Single-Agent to Cooperative Multi-Agent RL

- In Single-Agent RL, we want our agent to learn the optimal policy that takes a state *s* and tells us an action to take *a*

$$\pi^*(s) \rightarrow a$$

- In Multi-Agent RL, we extend this idea to multiple agents, often resulting in learning a joint action-value function

$$Q_{tot}(s, \mathbf{a})$$

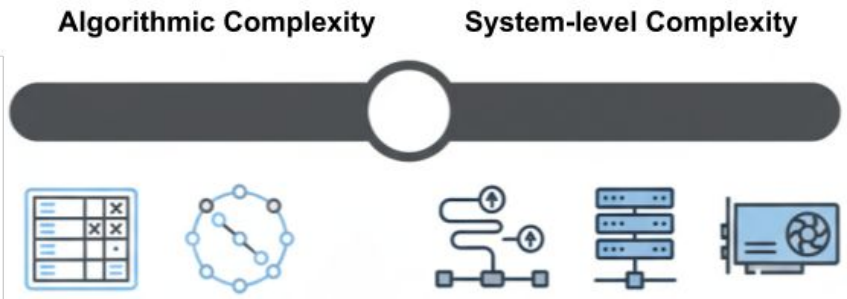
that gives us multiple actions, one for each agent

- Algorithms like VDN and QMIX learn to optimize this function, teaching the agents how to coordinate in the process. What happens when we need more than one machine to do this?

Sautarashmi Bandyopadhyay

From Algorithmic Complexity to System Complexity

- In MARL, the problems we usually deal with are *algorithmic*
  - Coordination, Credit Assignment, etc.
- At scale, we have *system-level* challenges
  - Computation, Memory, Bandwidth, etc.
- We need to consider not only *what* we compute, but *how*
- This is how we transition from a pure AI problem to AI + distributed systems problem



Sayantarashmi Bandyopadhyay

Three Problems in Distributed MARL

1. **Non-stationarity:** The environment is constantly changing as all other agents update their policy, resulting in a moving-target problem
2. **Credit Assignment:** With so much happening, how do you assign credit or blame to a single agent's action?
3. **Curse of Dimensionality:** The joint state-action space grows exponentially with the number of agents:

$$(|S \times A|^N)$$

this is computationally intractable!

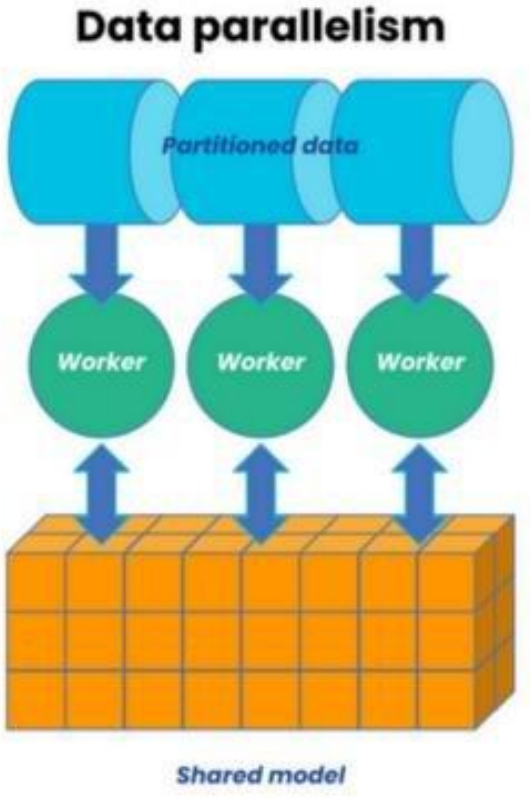
Saptarashmi Bandyopadhyay

Architectural Patterns for Distributed Learning

Saptarashmi Bandyopadhyay

Gradient Aggregation

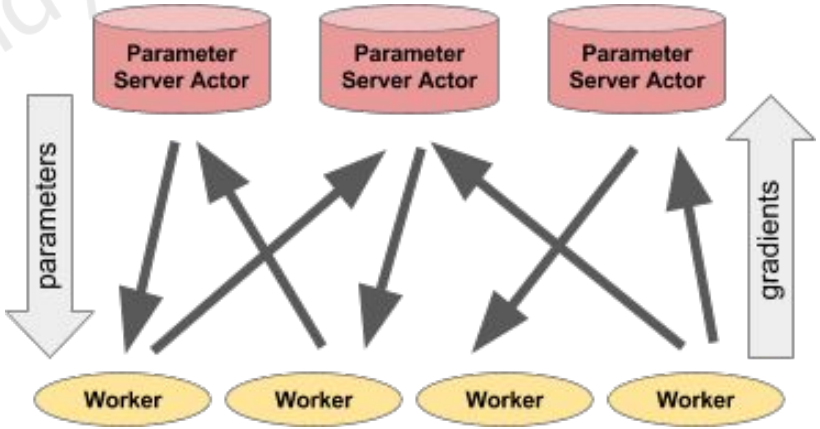
- In data-parallel training, we have a copy of the model on each worker node (e.g. GPU)
- Each worker computers gradients on its local batch of data (e.g. agent experiences)
- These gradients need to be aggregated across all the workers before the next update
- This step is the primary bottleneck in distributed computing



Saptarashmi Bandyop

Architecture 1: Parameter Server

- **Characteristics:** Centralized, Client-Server
- **Server Nodes:** Maintain authoritative global copy of the parameters
- **Worker Nodes:** Pull the latest parameters from the server, compute gradients on local data, and push gradients back to server
- The server must then aggregate the gradients and update the global model



Saptarashmi Bandyopadhyay

Parameter Server: Consistency Models

- **Synchronous Training:** The server wait for *all* the gradients before updating the model
  - **Pro:** Guarantees consistency, statistically equivalent to single-machine training
  - **Con:** Performance is constrained by :"stragglers")
- **Asynchronous Training:** Update occurs when *any* gradient is received
  - **Pro:** High throughput, robust to stragglers and failures
  - **Con:** Workers compute gradients using outdated parameters, which harms model convergence

Saptarashmi Bandyopadhyay

The Main Drawback of Centralized Architectures

- The centralized server needs to handle the communication with all N workers
- *Fan-out* of weights and *Fan-in* of gradients centers *all* network traffic on the server node
- As N grows the network bandwidth becomes the bottleneck
- This fundamentally limits how large a parameter server can scale

Saptarashmi Bandyopadhyay

Questions?

Saptarashmi Bandyopadhyay