Paper Review - Ray: A Distributed Framework for Emerging AI Applications

Sai Venkat Malreddy

29 september 2024

1 Idea and Contribution

The next generation of AI applications requires both high performance and flexibility, particularly in reinforcement learning (RL) where latency must be at the millisecond level while handling millions of tasks per second. These applications also demand support for fine-grained, heterogeneous computations. Existing frameworks, designed primarily for handling big data in supervised learning, are ill-suited for RL's unique needs. In response, the authors propose Ray, a low-latency, distributed execution framework for RL and machine learning (ML) applications. Ray is designed to address these challenges by providing a flexible computation model and dynamic execution, supporting fine-grained tasks. It unifies actor-based and task-parallel abstractions atop a dynamic task execution engine to handle the diverse workloads of training, simulation, and serving. For scalability and fault tolerance, Ray stores control state in a sharded metadata store while keeping other system components stateless, further enhanced by a bottom-up distributed scheduling strategy. This allows Ray to efficiently handle the demanding requirements of emerging AI workloads. The Ray paper has three main contributions: a generic system designed to train, simulate, and server RL models, the design and architecture of that system, and a programming model used to write workloads that run on the system.

2 Three Positive Comments

- **Decoupling Control and Logic**
  Ray separates the storage of control plane data (Global Control Store) from the logic implementation (schedulers). This separation enables the storage and computation layers to scale independently, which is crucial for meeting scalability objectives.
- **Heterogeneity-Aware Resource Utilization**
  Ray's heterogeneity-aware architecture allows users to optimize resource utilization by specifying requirements at the task or actor level, enabling efficient use of asymmetric computing environments and enhancing performance, cost-effectiveness, and scalability in diverse workloads
- **Simplifies the development experience**
  Ray's unified interface for task-parallel and actor-based programming simplifies the development experience by allowing developers to implement diverse workflows within a single framework. This design reduces the complexity of switching between different systems, enabling users to focus on building applications efficiently while enhancing productivity through straightforward and cohesive API interactions

3 Three Negative Comments

- **Challenges in Specialized Optimizations**
  Ray's general-purpose design may complicates the implementation of specialized optimizations for specific workloads.

- **Limitations Due to Lack of Support for Distributed Objects**
  The absence of support for distributed objects restricts Ray's ability to efficiently manage complex data structures across multiple nodes, necessitating developers to create custom solutions that can increase complexity and development time.

- **Overhead from Lineage Tracking**
  While lineage tracking aids in fault tolerance, it introduces overhead that may impact performance, particularly in high-throughput applications

4 Questions unanswered about the paper

- **Security**
  https://www.vicarius.io/vsociety/posts/securing-your-ray-ai-framework-understanding-vulnerabilities-and-best-practices

- **Lack of Justification for Methodology**
  The global scheduler computes the average task execution and the average transfer bandwidth using simple exponential averaging :

  The paper does not provide sufficient justification for the choice of exponential averaging as the method for calculating average task execution times and transfer bandwidth for scheduling by global scheduler. Understanding the reasoning behind this choice—such as its advantages over other averaging techniques