Ray

A Distributed Framework for Emerging AI Applications

R. Nishihara, P. Moritz, et al
October 17, 2018

University of California, Berkeley

Presented by: Devin Taylor
Table of contents

1. Introduction
   Problem Statement
   Background
   Related Work

2. Methodology
   Overview
   Programming model
   Architecture

3. Analysis
   Results
   Critical Analysis

4. Conclusion
Introduction
Problem Statement

Need for a computation framework that supports heterogeneous and dynamic computation graphs, while handling millions of tasks per second with millisecond-level latencies.
• High-performance, distributed execution framework for Python
• Key features include:
  • Heterogeneous, concurrent computations
  • Dynamic task graphs
  • High-throughput and low-latency scheduling
  • Transparent fault tolerance
  • Task-parallel and actor programming models
  • Horizontally scalable
• Applications:
  • Reinforcement learning
  • Hyperparameter tuning
  • Distributed training
Related Work

- CIEL\(^1\), Dask\(^2\)
  - Supports dynamic task graphs
  - Centralized scheduling architecture
  - No actor abstraction

- MapReduce\(^3\)
  - Implement BSP execution model
  - No actor abstraction
  - Centralized scheduling architecture

- TensorFlow Fold\(^4\), MXNet\(^5\)
  - Cannot modify DAG in response to task progress, task completion times, or faults
Methodology
Goal

• Implement a distributed framework suitable for modern AI applications

Requirements

• Flexibility - Functionality, duration, resource types
• Performance - scheduling
• Ease of development
Methodology - Programming model

- Remote functions return futures - `get()`, `wait()`
- Can specify resource allocation for remote functions at run time
- Supports nested remote functions
- Actor abstraction - Stateful edge to computation graph (data and control)

**Figure 1:** Nested remote functions
Methodology - Architecture

- Application layer
  - Driver - executes user program
  - Worker - executes remote functions
  - Actor - executes methods it exposes

- System layer
  - Global Control Store (GCS)
  - Bottom-up distributed scheduler
  - In-memory distributed object store - Apache Arrow

Figure 2: Architecture overview
Architecture - Global Control Store (GCS)

- Stores all metadata and state information
- Supports pub-sub infrastructure for internal communication
- Enables system to be stateless - enabling easy horizontal scalability
- Scaling achieved through sharding
Architecture - Bottom-up distributed scheduler

- Global scheduler with per-node local schedulers
- Tasks submitted to node’s local scheduler first
- Conditions under which global scheduler is invoked:
  - Overloaded
  - Cannot satisfy task requirements
  - Task inputs remote

**Figure 3:** Bottom-up distributed scheduler
Architecture - Overview

Figure 4: Overview of task execution

Figure 5: Overview of result retrieval
Analysis
Results - System

**Figure 6:** End-to-end scalability

- Linear
- 1.8M tasks per second

**Figure 7:** Object store performance

- Peak throughput > 15 GB/s
- Peak IOPS 18K
- 56 μs per operation
Results - RL Application

Figure 8: ES implementation

- Evolution Strategies (ES)
  Humanoid-v1 task
- Scaled to 8192 cores vs 1024
- 3.7 minutes vs 10 minutes

Figure 9: PPO application

- Proximal Policy Optimization (PPO)
- Ability to specify resource requirements
Critical Analysis

- Fault tolerance - potentially redundant due to statistical properties of most AI algorithms
- Specifying resource requirements - not always correctly understood
- Replication of GCS - single point of failure so requirement for fault tolerance
Conclusion
Conclusion

- Dynamic task graphs, GCS, bottom-up distributed scheduler, and actor programming model make Ray unique contribution.
- Scalability and performance make Ray useful for modern AI applications.
- Minor criticism around redundant architecture implementations.

Jeffrey Dean and Sanjay Ghemawat. 
**Mapreduce: simplified data processing on large clusters.**

Moshe Looks, Marcello Herreshoff, DeLesley Hutchins, and Peter Norvig.
**Deep learning with dynamic computation graphs.**

Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang.
**Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems.**