Ray: A Distributed Framework for Emerging AI Applications

Philipp Moritz,* Robert Nishihara,* Stephanie Wang, Alexey Tumanov, Richard Liaw,
Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, Ion Stoica
University of California, Berkeley

Abstract

The next generation of AI applications will continuously interact with the environment and learn from these interactions. These applications impose new and demanding systems requirements, both in terms of performance and flexibility. In this paper, we consider these requirements and present Ray—a distributed system to address them. Ray implements a unified interface that can express both task-parallel and actor-based computations, supported by a single dynamic execution engine. To meet the performance requirements, Ray employs a distributed scheduler and a distributed and fault-tolerant store to manage the system's control state. In our experiments, we demonstrate scaling beyond 1.8 million tasks per second and better performance than existing specialized systems for several challenging reinforcement learning applications.

1 Introduction

Over the past two decades, many organizations have been collecting—and aiming to exploit—ever-growing quantities of data. This has led to the development of a plethora of frameworks for distributed data analysis, including batch [20, 64, 28], streaming [15, 39, 31], and graph [34, 35, 24] processing systems. The success of these frameworks has made it possible for organizations to analyze large data sets as a core part of their business or scientific strategy, and has ushered in the age of "Big Data."

More recently, the scope of data-focused applications has expanded to encompass more complex artificial intelligence (AI) or machine learning (ML) techniques [30]. The paradigm case is that of *supervised learning*, where data points are accompanied by labels, and where the workhorse technology for mapping data points to labels is provided by deep neural networks. The complexity of these deep networks has led to another flurry of frameworks that focus on the training of deep neural networks and their use in prediction. These frameworks often leverage specialized hardware (e.g., GPUs and TPUs), with the goal of reducing training time in a batch setting. Examples include TensorFlow [7], MXNet [18], and PyTorch [46].

The promise of AI is, however, far broader than classical supervised learning. Emerging AI applications must increasingly operate in dynamic environments, react to changes in the environment, and take sequences of actions to accomplish long-term goals [8, 43]. They must aim not only to exploit the data gathered, but also to explore the space of possible actions. These broader requirements are naturally framed within the paradigm of *reinforcement learning* (RL). RL deals with learning to operate continuously within an uncertain environment based on delayed and limited feedback [56]. RL-based systems have already yielded remarkable results, such as Google's AlphaGo beating a human world champion [54], and are beginning to find their way into dialogue systems, UAVs [42], and robotic manipulation [25, 60].

The central goal of an RL application is to learn a policy—a mapping from the state of the environment to a choice of action—that yields effective performance over time, e.g., winning a game or piloting a drone. Finding effective policies in large-scale applications requires three main capabilities. First, RL methods often rely on *simulation* to evaluate policies. Simulations make it possible to explore many different choices of action sequences and to learn about the long-term consequences of those choices. Second, like their supervised learning counterparts, RL algorithms need to perform *distributed training* to improve the policy based on data generated through simulations or interactions with the physical environment. Third, policies are intended to provide solutions to control problems, and thus it is necessary to *serve* the policy in interactive closed-loop and open-loop control scenarios.

These characteristics drive new systems requirements: a system for RL must support *fine-grained* computations (e.g., rendering actions in milliseconds when interacting with the real world, and performing vast numbers of sim-

*equal contribution