# Comparative Evaluation of Distributed Machine Learning Platforms


1st Nawab khan
*dep.Computer Science. (BUITEMS.)*
*BUITEMS QUETTA (BUITEMS.)*
Quetta, Pakistan
Nawabyarmal123@gmail.com

2nd Syed Amjad shah
*dep.Computer Science.(BUITEMS.)*
*BUITEMS QUETTA (BUITEMS.)*
Quetta, Pakistan
amjadkhanaa528@gmail.com

3rd Abdul Siddique
*dep.Computer Science. (BUITEMS.)*
*BUITEMS QUETTA (BUITEMS.)*
Quetta, Pakistan
Sadiqfgians@gmail.com



***Abstract*—The rapid development of data-driven technologies requires advanced systems to process and analyze large amounts of data. Distributed computing systems, which distribute computing tasks across multiple computers, have become the core of machine learning. In this paper, we evaluate four distributed computing platforms: Apache Spark MLlib, PyTorch Distributed (DDP), Ray, and Distributed TensorFlow. This review is based on performance criteria such as resource management, resource participation, and advancements in machine learning. The tests evaluate the ability of each platform to perform a variety of tasks, remain stable during downtime, and provide a comfortable experience for users.After using liner regression as baseline model on various data size the results show that Ray Distributed does a great job of optimizing task scheduling , dynamic resource sharing and recovery and pyTorch provides the better recovery support for model .**



***Index Terms*—Keywords: Data-driven technologies, Distributed computing systems, Machine learning, Resource management Fault tolerance, Scalability, Linear regression, Baseline model ,Dataset, task scheduling, dynamic resource sharing, recovery**


## I. Introduction

The rapid evolution of machine learning has spanned industries, such as healthcare, finance, and transportation [3]. However, training models on large data sets requires a common set of methods. Distributed computing systems address this need by efficiently dividing tasks among groups of systems [4].

This study examines the four largest platforms: Apache Spark MLlib, PyTorch Distributed (DDP), Ray, and Distributed TensorFlow. The goal is to evaluate their performance in managing resources, handling failures, and supporting machine learning applications, and to provide plans for using them in real-world situations. Using liner regression as baseline model on various data sizes like data in original size and then increase the size for scalability performance .we will use California Housing dataset which will compare the performance of other complex machine learning algorithms .

## II. Problem Statement

The complexity of machine learning requires scalable platforms that can handle large datasets and workloads. Distributed computing systems meet this need by using the combined resources of multiple processors. However, selecting the most appropriate platform requires understanding many aspects such as design, error handling, and user access [1] [2]. This study focuses on the following fundamental questions: What are the advantages and disadvantages of Apache Spark MLlib, PyTorch Distributed (DDP), Ray, and Distributed TensorFlow in terms of resource management, out-of-bounds performance, and technological change? .

.

### A. *Linear Regression*

Linear regression is a machine learning algorithm used to predict a continuous output variable (dependent variable) based on one or more input variables (independent variables). It assumes a linear relationship between the variables.

The equation (with an independent variable x) is:

$$y = mx + c \tag{1}$$

- $y$: Predicted value (dependent variable)
- $x$: Input value (independent variable)
- $m$: Slope of the line (indicating the rate of change of $y$ with $x$)
- $c$: Intercept (value of $y$ when $x = 0$)

For more then one variables, the equation is extended to:

$$y = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b \tag{2}$$

Here $w_1, w_2, \ldots, w_n$ are weights (coefficients), and $b$ is the bias (intercept) [5].

### B. *California Housing dataset*

The California Housing dataset is a regression dataset from the 1990 U.S. Census with 20,640 samples of California districts. It includes features like median income, house age, and population, with the target variable being median house value. Widely used for machine learning tasks, it's included in Scikit-learn for benchmarking regression models [6].

## III. Literature review

With the growing demands of machine learning tasks, distributed computing systems have become essential for handling large datasets and complex models. This review explores four key platforms—Apache Spark MLlib, PyTorch Distributed (DDP), Ray, and Distributed TensorFlow—focusing on their performance in resource management, scalability, and integration for large-scale machine learning applications. The aim is to assess which platform provides the most efficient solutions for modern machine learning workloads.

1) **Apache Spark MLlib**
   Spark is a considerable framework for processing and distributed data storage. The MapReduce programming model adequately handles batch processing tasks. Studies show Spark's ability to manage large data sets while providing fault tolerance through a distributed file system [7] [8].
2) **PyTorch Distributed (DDP)**
   PyTorch Distributed (DDP) automates containerized application management, providing convenient resources and optimization capabilities. It is increasingly used to implement machine learning models in the cloud and on-premise [9] [10].
3) **Ray**
   Ray offers Python-based workflows parallel computing [11]. It is perfect for computing and data analysis because it is simple to scale from a single system to a decentralized group [12] [13].
4) **Distributed TensorFlow** Distributed TensorFlow focuses on real-time data stream processing and supports low latency and high throughput requirements [14]. It is more commonly used in machine learning pipelines that require instantaneous data [14]

.

## IV. Methodology

### A. Scope Definition

The methodology defines a structured approach to evaluate the performance and feasibility of distributed machine learning platforms under specific conditions. This study focuses on analyzing the scalability, resource utilization, resource participation, fault tolerance, and ease of use of four selected platforms. Each of these parameters is critical for assessing the overall efficiency and practical implementation of distributed machine learning in real-world scenarios.

**The methodology involves:**

1) **Dataset Selection**: Identifying data sets of varying sizes and complexity to simulate realistic machine learning workloads and assess scalability.
2) **Model Selection**: Implementing a linear regression model as a benchmark due to its computational simplicity and consistent resource requirements across platforms.
3) **Experimental Setup**: Deploying distributed machine learning platforms in a controlled environment with identical computational resources to ensure fairness in comparison.
4) **Parameter Measurement**:
   - **Scalability**: Analyze system performance as the size of the data set or the number of nodes increases.
   - **Resource Utilization**: Monitoring CPU, memory, and network bandwidth usage during model training.
   - **Resource Participation**: Measurement of the contribution of individual nodes in the distributed system.
   - **Fault Tolerance**: Simulating failures and observing the recovery mechanisms of the platform.
   - **Ease of Use**: Evaluating the simplicity of setup, deployment, and debugging based on user experience.

### B. Objective Setting

- The objective of this study is to systematically evaluate and compare the performance, scalability, and usability of four distributed machine learning platforms under controlled experimental conditions.
- By achieving these objectives, this study aims to provide actionable insights into the feasibility and practicality of deploying distributed machine learning platforms in real-world scenarios.

### C. Setup and Environment

we will create a control simple and affordable environment:

1) For parallel computing: we will use Anaconda in a local environment that includes all necessary packages,libraries, and Editor to work with and if some are needed, we will install.
2) For distributed computing: we will use Google colab with T4 free GPU offered by google.

## Discussion and Results

TABLE I
Comparison Platform Features (Parallel-Computing)

| Platform | Scalability | Resource Uitilization | Ease of Use | Training Time |
|---|---|---|---|---|
| Ray | Excellent | Excellent | Good | Very Fast |
| TensorFlow | High | Good | Very High | Slow |
| PyTorch DDP | Very High | High | Moderate | Moderate |
| Spark MLlib | Good | Very High | High | Fast |

TABLE II
Comparison Platform Features (Distributed-Computing)

| Platform | Scalability | Resource Uitilization | Ease of Use | Training Time |
|---|---|---|---|---|
| Ray | Excellent | Excellent | Good | Very Fast |
| TensorFlow | High | High | Good | Moderate |
| PyTorch DDP | Very High | Excellent | Moderate | Fast |
| Spark MLlib | High | High | Moderate | Fast |

TABLE III
COMMUNICATION OVERHEAD

| Platform | Computing Environment | Communication Overhead |
|---|---|---|
| Ray | Parallel/Distributed | Low/Low |
| TensorFlow | Parallel/Distributed | VeryHigh/High |
| PyTorch DDP | Parallel/Distributed | High/Good |
| Spark MLlib | Parallel/Distributed | High/High |

TABLE IV
FAULT TOLERANCE COMPARISON ACROSS PLATFORMS

| Platform | Environment Platform | Fault Recovery | Time Impact (%) |
|---|---|---|---|
| Ray | Parallel/Distributed | Yes | 2% |
| TensorFlow | Parallel/Distributed | Yes | 9% |
| PyTorch DDP | Parallel/Distributed | Yes | 4% |
| Spark MLlib | Parallel/Distributed | Yes | 6% |

## RESOURCE MANAGEMENT

- Ray demonstrated excellent resource optimization, effectively balancing workloads across clusters on both small and large dataset.
- TensorFlow has a lot of preset options and parameters, but its training speed model is slow.
- PyTorch Distributed provide strong recovery mechanisms to ensure minimal data loss in the event of a failure.
- Spark MLlib showed outstanding efficiency in handling GPU-intensive tasks but required significant configuration.

Ray excelled in scalability, resource utilization, and training time, with minimal communication overhead and a fault recovery impact of only 2%. TensorFlow offered high ease of use but suffered from slow training speed and a 9% time impact during fault recovery even on small dataset. PyTorch DDP balanced scalability and reliability, with moderate training time and a 4% recovery impact. Spark MLlib showcased efficiency in GPU-intensive tasks but required significant configuration, with a fault recovery impact of 6%. Overall, Ray emerged as the most versatile platform, while TensorFlow and PyTorch DDP excelled in specific use cases.

## CONCLUSION

This research aimed to compare four major platforms—Ray, TensorFlow, PyTorch DDP, and Spark MLlib—for parallel and distributed machine learning/computing. Ray demonstrated superior scalability, resource utilization, and training time efficiency, making it the most versatile choice for distributed computing tasks. TensorFlow, while easy to use, showed slower training speeds and a higher fault recovery impact. PyTorch DDP offered strong scalability and fault tolerance, with moderate training time, making it a solid choice for more complex, large-scale tasks. Spark MLlib excelled in GPU-intensive tasks but required significant configuration and showed higher communication overhead.

## FUTURE WORK

Future work will explore the integration of these platforms into real-world, large-scale machine learning applications to evaluate their performance in diverse environments. Additionally, further research will focus on optimizing fault tolerance and reducing communication overhead, particularly for TensorFlow and Spark MLlib, to enhance their effectiveness in distributed machine learning tasks.

## REFERENCES


[1] T. Kraska, A. Talwalkar, J. C. Duchi, R. Griffith, M. J. Franklin, and M. I. Jordan, "MLbase: A distributed machine-learning system," in *Proceedings of CIDR*, vol. 1, pp. 2-1, Jan. 2013.

[2] S. Alqahtani and M. Demirbas, "Performance analysis and comparison of distributed machine learning systems," *arXiv preprint arXiv:1909.02061*, 2019.

[3] A. Galakatos, A. Crotty, and T. Kraska, "Distributed machine learning," 2018.

[4] V. Shah and S. Shukla, "Data distribution into distributed systems, integration, and advancing machine learning," *Revista Espanola de Documentacion Cientifica*, vol. 11, no. 1, pp. 83-99, 2017.

[5] X. Su, X. Yan, and C.-L. Tsai, "Linear regression," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 4, no. 3, pp. 275-294, 2012.

[6] A. Chen, "Deep learning in real estate prediction: An empirical study on California house prices," 2024.

[7] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107-113, 2008.

[8] T. White, *Hadoop: The Definitive Guide*, O'Reilly Media, 2015.

[9] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, Omega, and Kubernetes," *ACM Queue*, vol. 14, no. 1, pp. 70-80, 2016.

[10] K. Hightower, B. Burns, and J. Beda, *Kubernetes: Up and Running*, O'Reilly Media, 2017.

[11] A. Sergeev and M. D. Balso, "Horovod: Fast and easy distributed deep learning in TensorFlow," arXiv, 2018, [Online]. Available: https://arxiv.org/abs/1802.05799.

[12] M. Abadi, P. Barham, J. Chen, et al., "TensorFlow: A system for large-scale machine learning," *USENIX OSDI*, 2016.

[13] M. Rocklin, "Dask: Parallel computation with blocked algorithms and task scheduling," in *Proc. of the 14th Python in Science Conf.*, 2015.

[14] O. Gottesman, et al., "Scaling data analytics with Dask in Python," *Journal of Computational Science*, vol. 42, pp. 12-23, 2020.


github:https://github.com/sadiqfgians/newMLcoding