DS5110/CS5501: Big Data Systems Project Final Report
Comparing Data Systems Performance on Handwritten Digit Recognition

Elisabeth Waldron, Chaitali Harge, Lingzhen Zhu, Fadumo Hussein
University of Virginia
{psa7rm, afj8am, qrb2gn, fmh7pv}@virginia.edu

Abstract

This project aims to evaluate the performance of distributed computing systems, specifically Ray and Dask, in implementing the K-nearest neighbors (KNN) clustering and the Convolutional neural network (CNN) classification model for the Scikit-learn handwritten digit recognition. Leveraging the AWS S3 storage system, we delve into the effectiveness of using online storage compared to loading data from local file storage. We focus on assessing these distribution frameworks' performance and determining the most efficient system based on factors such as execution time, scalability, and I/O usage for computing large-scale machine learning applications.

1 Introduction

In the rapidly evolving field of machine learning, the capability to process large-scale datasets efficiently is critical. This paper explores the performance of distributed computing frameworks, specifically Ray and Dask, in the context of implementing machine learning algorithms such as KNN and CNN for digit recognition using the handwritten digit dataset (Figure 1) from Scikit-learn library. The study is particularly focused on evaluating these frameworks under the computation-intensive demands of training and inference, emphasizing factors like execution time, scalability, and I/O usage.

Background: Traditional approaches to machine learning often struggle with scalability and efficiency, especially as data volumes and model complexity increase. Distributed computing offers a solution by parallelizing tasks across multiple computing resources. Ray and Dask are two prominent distributed computing frameworks that facilitate this scalability but differ in their architectural approaches and handling of computational tasks.

Objectives: The primary objective of this research is to benchmark Ray and Dask in the context of KNN clustering and CNN classification, two prevalent algorithms for pattern recognition and image classification. By leveraging the AWS S3 storage system, this study also compares the performance impact of accessing data from online storage versus local file storage systems.

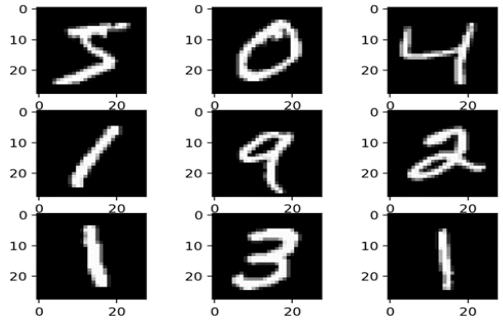

Figure 1: Example of digit dataset

Significance: The significance of this research lies in its potential to provide insights into the optimal configurations and conditions under which these distributed systems perform best. This could guide future applications of machine learning in industrial settings, where efficient data processing is crucial.

The structure of this report is organized as follows: We begin by providing an overview of the project objectives. Subsequently, we will discuss the methods and theoretical framework of our project. After that, we will delve into the detailed design of our experiments and the results of our analysis, highlighting the performance of the distributed computing system. Last, we will conclude with discussions of our results and our findings from this project.

2 Methodology

The methodology involves setting up a distributed computing environment on AWS EC2 T3-large instances, utilizing S3 as the primary data storage medium. We will systematically

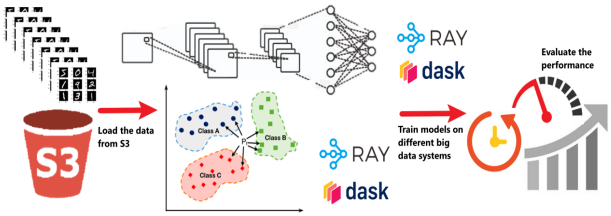

Figure 2: Project methodology overview

measure and compare the performance of Ray and Dask when performing KNN and CNN tasks, focusing on execution time, scalability, and efficiency of data retrieval and processing. The overall design of our project is shown in Figure 2.

2.1 Amazon S3

Amazon S3 (Simple Storage Service) is a robust and scalable object storage solution provided by Amazon Web Services (AWS). It stores data as objects within containers known as "buckets" and is designed to offer 99.99% durability and 99.99% availability of objects. S3 ensures high security with detailed access controls and supports automatic data replication across multiple locations. It's highly scalable, allowing for storage and retrieval of any amount of data at any time, making it suitable for a wide range of applications, from small files to large-scale enterprise data. S3 also includes features for data management such as lifecycle policies and versioning, facilitating efficient data handling and cost management.

Using Amazon S3 as input storage for our project makes sense for several reasons. Firstly, S3's scalability and high durability ensure that our large datasets are stored securely and are readily accessible, which is crucial given the probable discontinuation of HDD options on EC2 instances. Furthermore, S3's compatibility with data processing frameworks like Dask and Ray allows for efficient handling and processing of data directly from S3, avoiding potential performance bottlenecks associated with transferring large datasets to local storage on EC2. This approach not only optimizes data accessibility and processing speed but also leverages the robust and cost-effective storage solution that S3 provides, making it a smart choice for us to managing project data efficiently.

Here are the procedures for setting up Amazon S3: a) Access S3 in AWS b) Upload data file c) Access IAM d) Look for EC2 default role e) Access instances' action f) Select Modify IAM role in the security setting g) Add the default EC2 IAM role and then press update

AWS S3 is used as the primary data storage solution in our experiments to evaluate the impact of remote storage on the performance of distributed machine learning tasks. Data stored in S3 buckets is accessed directly by the distributed systems running on AWS EC2 instances.

2.2 DASK

Dask is a flexible parallel computing library designed to scale up from single machines to large clusters. It orchestrates the execution of large-scale computations with a dynamic task scheduling system that can handle a variety of workload types including arrays, dataframes, and machine learning tasks. Dask's architecture allows it to integrate closely with Python's existing data ecosystem, which includes libraries such as NumPy, pandas, and Scikit-learn.

For our project, Dask is employed to distribute tasks related to KNN and CNN processing. We compare the performance of Dask when configured with 2, 4, and 6 workers (as each node equipped with 2 CPU cores). This enables us to assess how Dask scales with increasing computational resources and to determine the optimal worker configuration for balancing load and minimizing computation time.

2.3 Ray

Ray is an open-source framework that makes it easier to build and scale distributed applications. Ray provides a simple, flexible API for parallel and distributed computing, making it ideal for high-performance machine learning applications. Ray excels in scenarios requiring real-time execution and fine-grained task management. Its core components include a dynamic task graph execution framework, which supports automatic parallelization and scheduling of complex computation graphs.

Similar to our Dask setup, Ray's role is to facilitate distributed training of CNN models and efficient execution of KNN clustering with 2, 3, and 4 workers. The varied worker setups help in understanding Ray's performance characteristics, particularly how effectively it manages more granular scaling and resource allocation in comparison to Dask.

2.4 KNN

KNN is a simple yet effective machine-learning algorithm used for both classification and regression. For image classification, KNN works by comparing the new image against stored images and predicting the label based on the majority label of the nearest neighbors.

The KNN model is applied to the digit recognition task, where the distance between feature vectors (representing images) is calculated to find the closest matches (neighbors). We utilize a feature extraction step to convert images into a suitable format for KNN processing, focusing on the distance calculations and neighbor aggregation, which are critical for the performance and scalability of the algorithm in a distributed setting.

In our project, this algorithm is applied to segment large-scale digit data into distinct clusters, facilitating deeper analysis and insights into handwritten digit recognition.

2.5 CNN

CNNs are a category of deep neural networks that are particularly powerful for tasks like image classification. A typical CNN architecture consists of several layers: convolutional layers, pooling layers, and fully connected layers. Each layer applies different filters and captures various features at different levels of abstraction. The model is trained using backpropagation with a softmax layer at the output to handle multi-class classification of digit images. CNNs have demonstrated remarkable efficacy in handling multi-class classification tasks, particularly in scenarios involving image and sequential data, by effectively capturing hierarchical patterns and spatial dependencies within the input data.

Transitioning to a CNN model could offer significant advantages when comparing framework performance in AWS, particularly due to its potential requirement for GPUs. CNNs are renowned for their effectiveness in handling tasks such as image recognition and classification, primarily because they excel at capturing spatial hierarchies and patterns within data. However, this enhanced capability comes with computational demands, often necessitating the use of specialized hardware like GPUs for efficient training and inference.

In this study, a sequential model is implemented to classify handwritten digits, leveraging their capability to extract features from raw pixel data effectively. The structure of our model is listed in Table 1.

Table 1: CNN Model Structure

Layer	**Description**
Input	8 x 8 grayscale image
Conv2D	32 filters of size 3 x 3, ReLU activation
MaxPooling2D	2 x 2 max pooling
Flatten	Flatten the output of the previous layer
Dense	128 units, ReLU activation
Dense	10 units, Softmax activation

3 Result

After successfully connecting to the S3 data bucket and performing data operations such as listing and downloading objects, we have our digits dataset loaded into our workspace, and we split the data into the training set (1437 images) and the testing set (360 images) in an 80:20 ratio. Then, we performed KNN and CNN models on both the Dask and the Ray under different numbers of workers. Finally, we monitor the performance via the dashboard of Dask and Ray, as well as using the magic command in Python to record the wall time, which here represents the total runtime of the model.

3.1 ML Models Result

KNN Model

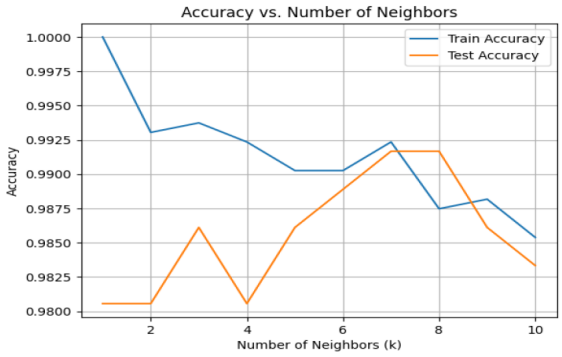

Figure 3: Test vs Train K-means model

We evaluated the KNN classifier's performance with various k values ranging from 1 to 10. Our results consistently showed high levels of accuracy, with both training and test accuracies close to 99%. The detailed classification reports revealed that the precision, recall, and F1-score across different classes were remarkably high, generally exceeding 0.98, which indicates excellent reliability and effectiveness of our model in recognizing and differentiating between the different digits.

For k=1, the model achieved perfect training accuracy, but this was likely due to overfitting, as it memorizes the training data. As kk increased, we observed a slight decline in overfitting, evidenced by more stable test accuracies and improved generalization across unseen data. The best generalization performance was observed with kk values between 5 and 8, where the test accuracy peaked at 99%. The detailed results are shown on the Table 2 and Figure 3.

These results demonstrate the capability of the k-NN classifier in handling complex pattern recognition tasks like handwritten digit classification. The high accuracy and excellent class-wise metrics suggest that k-NN, with an appropriately chosen k, is highly effective for this type of problem.

Table 2: Classification Metrics for Different k Values

k	**Train Acc.**	**Test Acc.**	**Precision**	**Recall**	**F1-Score**
1	1.00	0.98	0.98	0.98	0.98
2	0.99	0.98	0.98	0.98	0.98
3	0.99	0.98	0.98	0.98	0.98
4	0.99	0.98	0.98	0.98	0.98
5	0.99	0.99	0.99	0.99	0.99
6	0.99	0.99	0.99	0.99	0.99
7	0.99	0.99	0.99	0.99	0.99
8	0.99	0.99	0.99	0.99	0.99
9	0.99	0.98	0.98	0.98	0.98
10	0.98	0.98	0.98	0.98	0.98

CNN Model

For our CNN model, training was executed over ten epochs,

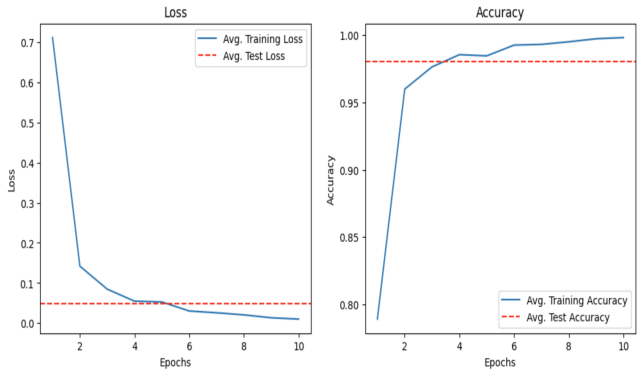

Figure 4: Loss and Accuracy for CNN model

with each epoch involving a complete pass through the entire training dataset. The models were evaluated based on their loss and accuracy metrics on both training and validation sets.

The training phase demonstrated rapid convergence of the loss function, with the initial sharp decrease stabilizing as the epochs progressed. Specifically, the average training loss plummeted from about 0.7 to approximately 0.02 by the end of the training period, while the test loss hovered slightly higher, indicating a strong fit to the training data without significant overfitting.

In terms of accuracy, our models performed exceedingly well. The average training accuracy consistently increased, reaching near-perfect scores by the 10th epoch. Trainer 4 achieved perfect training accuracy, illustrating the potential of our CNN architecture when coupled with optimal training conditions. The test accuracies also reflected high effectiveness, with values such as 0.9861 for Trainer 2 being among the highlights.

These outcomes validate the capability of CNNs to accurately recognize and classify handwritten digits, showcasing their robustness against variations in handwriting styles. The consistency across different training initializations further underscores the stability of our approach. The detailed results are shown on the Figure 4.

3.2 Performance Comparison

The performance of Dask and Ray, along with the effectiveness of the KNN and CNN models, are evaluated based on execution time, scalability, and I/O usage. This comprehensive evaluation will help determine the most efficient system for handling large-scale machine learning applications, particularly in cloud-based environments.

Dask System

In our investigation of parallel computing frameworks for machine learning tasks, Dask demonstrated a robust capability to enhance computational efficiency, particularly when scaled across multiple workers. For instance, our implementation of

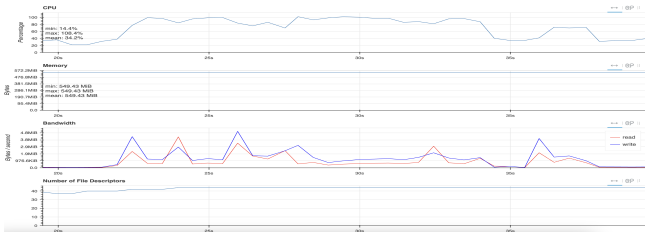

Figure 5: Dashboard for 2 Dask Workers

Dask reduced the execution time for the K-Nearest Neighbors (KNN) classification from 3.78 seconds with two workers to 2.69 seconds with six workers. This trend was similarly observed with the ResNet18 model, where Dask decreased the execution time from 15.6 seconds with two workers to 13.1 seconds with six workers, showcasing its scalability and efficiency in handling computationally intensive tasks.

Comparatively, Ray, while maintaining consistent performance metrics, did not achieve the same level of efficiency for complex models such as ResNet18, where its execution times were notably higher, affirming Dask's superior performance in this scenario.

However, the integration of Dask with Amazon S3 introduced significant overhead, complicating the parallel processing and resulting in operational instability, such as frequent worker crashes. These issues were exacerbated by the inherently small size of the MNIST dataset, leading to a cost-performance ratio that sometimes negated the benefits of Dask's accelerated computing capabilities. Furthermore, though there was a reduction in run time through dask there is not a significant improvement with increased number of workers and threads. See figure 5 for cpu performance specifically for 2 dask workers (these results were similar for 4 and 6 workers as well) (Figure 5).

Thus, while Dask substantially reduced processing times and demonstrated potential for large-scale data processing, the practical deployment challenges and associated costs must be carefully managed to fully realize its benefits in real-world applications.

Ray System

Our experiments involved running the KNN and ResNet (a variant of CNN) models across different numbers of Ray workers. We systematically recorded the time taken to process the same dataset under varying worker counts (2, 3, and 4 workers), which allowed us to observe the impact of parallel processing capabilities provided by Ray. An overview of dashboard for Ray running CNN model is shown on Figure 6.

The KNN model showed relatively consistent times across different worker counts, with times slightly increasing from 3.19 seconds with three workers to 3.42 seconds with four workers. This slight increase could be attributed to the overhead involved in managing a larger number of workers, as KNN computations are less computationally intensive com-

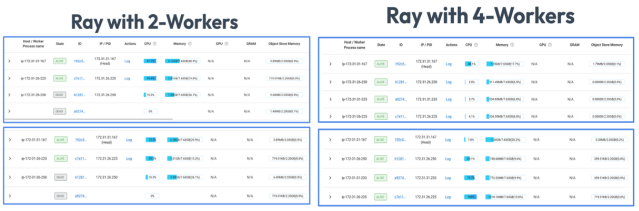

Figure 6: Dashboard for Ray Running CNN

pared to deep learning models.

The CNN model demonstrated significant variability in execution times, which decreased markedly as the number of workers increased. Specifically, the model runtime decreased from 30.6 seconds with two workers to an impressive 13.1 seconds with four workers, highlighting Ray's effectiveness in handling more computationally intensive tasks through parallel processing.

The scalability of Ray was evident, particularly with the CNN model, where the increase in workers dramatically reduced computation times. This result underscores the potential benefits of using Ray for complex machine learning tasks that are resource-intensive and time-sensitive.

Moreover, the resource utilization metrics (CPU and memory usage) provided insights into the efficiency of resource management across different worker setups. With an increase in workers, there was a more balanced and effective distribution of workload, leading to better utilization of computational resources.

4 Conclusion

In conclusion, our comparative analysis of Dask and Ray underscores their distinct performance characteristics in handling large-scale machine learning tasks. In Table 3, Dask demonstrates notable improvements in computational efficiency, exemplified by the substantial reduction in execution times. For instance, the execution time for KNN classification decreased from 3.78 seconds with 2 workers to 2.69 seconds with 6 workers, while the ResNet18 model execution time decreased from 15.6 seconds to 13.1 seconds, showcasing Dask's scalability and efficiency. Conversely, Ray showcases consistency in performance metrics but exhibits lower efficiency compared to Dask, particularly with complex models like ResNet18. Moreover, challenges associated with Dask's integration with Amazon S3, such as operational instability and frequent worker crashes, highlight the importance of addressing deployment hurdles for both systems. Our comparative analysis provides actionable insights for selecting the optimal data system, considering specific task requirements and deployment considerations, ultimately enhancing the efficiency and effectiveness of large-scale machine learning applications.

Table 3: Performance Comparison of Dask and Ray

	Dask			Ray		
	2	4	6	2	3	4
KNN for Clas.	3.78	2.73	2.69	3.25	3.19	3.42
ResNet18	15.6	13.2	13.1	30.6	25.7	13.1

5 Metadata

The presentation of the project can be found at:

https://www.youtube.com/watch?v=2hHoUZpmwNY

The code/data of the project can be found at:

https://github.com/psa7rm/DS-5110---Big-Data

[3] [1] [2]

References

[1] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[2] ChatGPT. https://chat.openai.com/share/9c9bc28d-6837-4271-9ca1-f1684b477083. 2024.

[3] Tanvi Penumudy. A beginner's guide to knn and mnist handwritten digits recognition using knn from scratch. JAN 2021.