A Comparative Analysis of Ray and Dask: Scaling Frameworks for Big Data Analysis and Machine Learning in Python

Athanasios Varis
ECE
NTUA
Athens, Greece
el19606@mail.ntua.gr

Georgios Vlachopoulos
ECE
NTUA
Athens, Greece
el19926@mail.ntua.gr

Ioannis Nikiforidis
ECE
NTUA
Athens, Greece
el18123@mail.ntua.gr

***Abstract*—As the volume and complexity of data continue to surge, the demand for efficient and scalable frameworks for big data analysis and machine learning has become crucial. This paper provides a comparative analysis of Ray and Dask, two Python-based frameworks designed to address the challenges of distributed computing in large-scale data and machine learning applications. The study delves into the features and performance characteristics of both Ray and Dask, exploring their capabilities in handling distributed computing tasks, parallelizing computations, and seamlessly scaling across clusters. We examine their suitability for various use cases. Through benchmarking experiments and real-world use cases, this paper aims to provide insights into the strengths and limitations of Ray and Dask, aiding practitioners and researchers in selecting the most appropriate framework based on their specific requirements. The findings contribute to a deeper understanding of the trade-offs involved in choosing between these two scaling frameworks, ultimately guiding decision-making processes in the context of big data analysis and machine learning tasks in Python.**

***Index Terms*—python, scaling frameworks, ray, dask, big data analysis, machine learning**

I. System and Software Description

A. *Virtual Machines [1], [2]*

In this project we will be using Virtual Machines provided by Okeanos-Knossos to conduct our experiments. Virtual Machines provide flexibility, efficiency and scalability.

- **Isolation:** Virtual machines provide a level of isolation between different applications and services running on the same physical hardware. This isolation enhances security by minimizing the risk of one application affecting or compromising another.
- **Consistent Environments:** VMs enable the creation of consistent and reproducible environments across different stages of a project's life-cycle. This helps in avoiding compatibility issues between development, testing, and production environments, ensuring a smoother deployment process.
- **Resource Efficiency:** VMs allow for efficient resource utilization by enabling multiple virtualized instances to run on a single physical server.
- **Scalability:** Virtual machines facilitate scalability by allowing you to dynamically adjust resources allocated to each VM based on changing project requirements. This adaptability ensures that your project can scale up or down efficiently to handle varying workloads.
- **Cost Savings:** Using virtual machines can lead to cost savings by reducing the need for physical hardware and associated maintenance costs. It allows for better utilization of existing hardware and provides a more cost-effective solution.

B. Python [3], [4]

Python has become a powerhouse in the fields of big data analysis and machine learning, owing to its versatility, extensive libraries, and a vibrant community of developers. Here are a few key aspects of big data analysis and machine learning in Python:

- **Extensive Libraries:** Python boasts a rich ecosystem of libraries specifically designed for big data analytics and machine learning. Libraries such as NumPy, pandas, and scikit-learn provide robust tools for data manipulation, analysis, and machine learning model development.
- **Scalability with Dask and Ray:** Frameworks like Dask and Ray empower Python developers to scale their applications seamlessly. These tools enable parallel and distributed computing, allowing users to process large datasets and execute machine learning tasks efficiently across clusters of machines.
- **Machine Learning Frameworks:** Python is home to some of the most widely used machine learning frameworks, including TensorFlow and PyTorch. These frameworks facilitate the development, training, and deployment of complex machine learning models, covering a broad spectrum of applications from computer vision to natural language processing.
- **Data Visualization:** Python excels in data visualization, with libraries like Matplotlib, Seaborn, and Plotly offering powerful tools for creating insightful visualizations. Effective data visualization is crucial for understanding complex patterns and trends in big datasets.
- **Community Support:** Python's large and active community plays a vital role in the development and adoption of tools for big data analysis and machine learning. This community-driven approach results in frequent updates, extensive documentation, and a wealth of tutorials and resources.
- **Interoperability:** Python integrates well with other technologies and languages, making it a preferred choice for building end-to-end data pipelines. It can seamlessly connect with databases, cloud services, and other programming languages, allowing for smooth transitions between different stages of data processing and analysis.
- **Education and Accessibility:** Python's syntax is clear and concise, making it an accessible language for beginners. This has contributed to its popularity in educational settings and has led to a growing pool of data scientists and engineers with proficiency in Python.

In summary, Python's versatility, extensive library support, scalability through frameworks like Dask and Ray, and a thriving community make it a go-to language for big data analysis and machine learning applications. Its ecosystem empowers developers and data scientists to tackle complex problems, analyze vast datasets, and build sophisticated machine learning models with ease.

C. Ray [5], [6]

Ray is a distributed computing framework for Python that is designed to make it easy to scale and parallelize applications. Ray provides a simple API for parallel and distributed computing, allowing developers to write scalable applications without needing to explicitly manage the complexities of distributed systems. Key features of Ray include:

- **Task Parallelism:** Ray allows you to parallelize Python functions, turning them into distributed tasks that can run in parallel across a cluster of machines.
- **Distributed Data Processing:** Ray provides abstractions for working with distributed data, making it easier to scale data processing tasks.
- **Distributed Memory:** Ray supports distributed memory, allowing you to share data between tasks running on different machines.
- **Actor Model:** Ray extends the actor model, providing a way to create distributed stateful objects that can encapsulate both data and computation.
- **Ray Libraries:** Ray includes libraries that leverage its capabilities for specific use cases, such as RLlib for reinforcement learning and Tune for hyperparameter tuning.

- **Dynamic Task Graphs:** Ray dynamically constructs task dependency graphs, enabling efficient scheduling of tasks and resource management.

Ray is commonly used in the fields of machine learning, distributed computing, and reinforcement learning. It provides a flexible and scalable solution for parallel and distributed computing tasks, making it easier for developers to build applications that can scale across large clusters of machines.

D. Dask [7], [8]

Dask is another distributed computing framework for Python, but it focuses on parallel computing and task scheduling. It's particularly well-suited for parallelizing operations on large data-sets that don't fit into memory. Key features of Dask include:

- **Dynamic Task Scheduling:** Dask breaks down large computations into smaller tasks and dynamically schedules them to run in parallel. This allows for the efficient use of resources and can handle computations that are larger than the available memory.
- **Parallel Data Structures:** Dask provides parallel versions of familiar data structures in Python, such as arrays (Dask Arrays), dataframes (Dask DataFrames), and bags (Dask Bags). These structures can scale computations beyond the limits of a single machine.
- **Integration with Existing Libraries:** Dask integrates seamlessly with popular Python libraries such as NumPy, pandas, and scikit-learn, allowing users to parallelize and distribute their existing code without significant modifications.
- **Support for Out-of-Core Computing:** Dask can handle datasets that are larger than available memory by intelligently loading and unloading data from disk as needed.
- **Scalability:** Dask can scale from a laptop to a cluster of machines, making it suitable for a wide range of computing environments.

Dask is commonly used in data science and machine learning workflows where large datasets need to be processed, analyzed, or transformed efficiently. It provides a user-friendly interface for parallel and distributed computing in Python, making it easier for data scientists and researchers to scale their computations.

II. Installation and Setup

A. Prerequisites

To perform this project, we developed three virtual machines in order to extensively examine parallel data processing. In the first virtual machine we acted as the master while the second and the third ones acted as a workers. For the operating system, we chose Ubuntu 16.04.3 LTS and configured 4 cores, 8 gigabytes of RAM, and 30 gigabytes of hard disk space for each virtual machine. We then configured the installation environment, including Python and the Ray and Dask working frameworks.

B. Ubuntu

First we need to upgrade the linux environment. So we execute the following throught the terminal :

```
sudo apt update
sudo apt upgrade
```

C. Python

In order to setup Python enter the following to a terminal :

```
sudo apt install python3
python3 --version
```

The final line is to make sure that the python environment is set-up correctly. Now since pip will be used in order to install the frameworks, the following step is needed as well :

```
sudo apt install python3-pip
```

Also, since Dask uses 'numpy', installing it is mandatory and can be gone with the following line :

```
python3 -m pip install -U numpy
```

And 'numpy' uses 'sciPy' thus :

```
python3 -m pip install -U sciPy
```

To aid with some machine learning applications we use 'scikit-learn' and 'lightgbm'

```
python3 -m pip install -U scikit-learn
python3 -m pip install -U lightgbm
```

Finally, in order to distribute workloads the package 'python-distributed' is needed and can be installed by :

```
sudo apt install python-distributed
```

D. Ray

After making sure that Python has been installed correctly, the following can be entered to the terminal in order to install the Ray framework.

```
python3 -m pip install -U ray[default]
```

E. Dask

Finally, the Dask framework is installed.

```
python3 -m pip install -U dask[complete]
```

F. Networking

In order for the distribution of tasks to take place, there needs to be a connection between the machines. In real world scenarios in data centers, the machines are typically connected by Ethernet, through network switches and hubs that segment and provide hierarchy and organisation to the whole center. In this case, since the machines are virtual, they are placed in the same NAT network and given public ipv6 addresses so they can communicate with each other and still retain the ability to be connected to the internet. Since the data utilized and produced by this project are not high risk or sensitive, there is no need for extra security measures, but ideally the worker machines should be blocked from accessing or be accessed from public addresses so a potential attacker has fewer access points.

III. Subjects of Analysis

A. Classification [9]

Classification stands as a cornerstone in supervised machine learning, where the primary objective is to train a model using labeled data. This trained model then goes on to predict the class labels of previously unseen instances. The essence lies in deciphering a mapping from input features to predetermined output classes, empowering the model to provide accurate predictions for new data points. In the realm of large-scale data environments, the strategic distribution of classification tasks among multiple machines emerges as a pivotal strategy for achieving scalability and efficiency. In essence, the distributed workload approach is instrumental in handling the computational demands of expansive datasets that may overwhelm the capabilities of a single machine. By harnessing the collective power of multiple machines, we can significantly accelerate the training process, parallelizing computations and reducing the overall time required. This distributed paradigm not only addresses the challenges posed by big data but also optimizes resource utilization, ensuring that each machine contributes meaningfully to the classification process.

B. Grid Search [10]

Grid search is a hyperparameter tuning technique widely used in machine learning to systematically explore a predefined set of hyperparameter combinations for a given model. It involves training and evaluating the model with each combination, helping identify the optimal set of hyperparameters that yield the best performance on a validation set.While grid search is indeed a potent tool for model optimization, its practical application in the context of big data introduces additional complexities, necessitating thoughtful considerations. The strategic distribution of workloads emerges as a pivotal factor in enhancing its efficiency within the landscape of large-scale datasets.This involves not only dealing with the sheer scale of the datasets but also capitalizing on parallelization to streamline the hyperparameter optimization journey. Deliberate attention to nuances such as data distribution, optimal resource utilization, and streamlined communication becomes indispensable when seamlessly integrating distributed grid search into the fabric of big data environments.

C. Clustering [11]

Clustering, a fundamental unsupervised machine learning technique, plays a pivotal role in data analysis by uncovering inherent structures within datasets. The process involves grouping similar data points into clusters, providing insights into patterns and relationships without relying on predefined labels. Common clustering algorithms, including K-means, hierarchical clustering, and DBSCAN, offer versatile solutions tailored to diverse datasets.In the realm of big data, where datasets are vast and dimensionalities are high, challenges arise in the scalability and computational efficiency of clustering algorithms. Pairing clustering algorithms with distributed computing frameworks empowers data scientists and analysts to efficiently analyze vast datasets, uncover meaningful patterns, and gain valuable insights. This combination aligns with the demands of modern data analytics, where the volume and complexity of data necessitate scalable and parallelized solutions for effective clustering.

D. Principal Component Analysis [12]

Principal Component Analysis (PCA) is a dimensionality reduction technique commonly used in machine learning and data analysis to transform high-dimensional datasets into a lower-dimensional space while retaining as much variance as possible. The primary goal of PCA is to identify the principal components, which are linear combinations of the original features, that capture the most significant variability in the data. In the context of distributed computing, performing PCA on large-scale datasets poses challenges related to computational efficiency and scalability. Distributed frameworks such as Ray and Dask provide solutions to address these challenges and enable the application of PCA on big data.

IV. Data generation and Database loading

We decided to create our own data through a Python script, with the aim of achieving better control and more immediate availability. This data, although generated through random number selection, has been parameterized to reflect realistic datasets. This approach allows the data to be used as naturally generated data sets, thus providing a realistic representation of the environments where the project will be implemented. This approach to data generation seeks to enhance the usability and accuracy of the results, while enhancing the flexibility and control of the experiment.

A. Dataset for Classification and Grid Search

We generated a data file that contained synthetic classification data and saved it in the LIBSVM format. The python function that generates it accepts as input 3 variables, which are the number of samples (number of data points) , the number of features (number of features per data point) and the number of classes (number of classes or labels). Since we could be dealing with data-sets that are larger than the usable RAM of the system, the data can be generated in smaller segments (chunks) which will later be concatenated in a singular file. A small file containing the metadata of the data file is also generated to provide an overview of the file. The following settings were used to produce our desired datasets based on the size we wanted to achieve :

	1GB	2GB
Samples	5 M	10 M
Features	10	10

B. Dataset for Clustering and PCA

We used a function to generate synthetic data resembling a Swiss roll and saved it in a file in the LIBSVM format. The parameters of the function are the number of samples (number of data points) and the noise (the standard deviation of the Gaussian noise added to the data). We solve the problem of generating larger than RAM data by dividing the generation into chunks that fit into memory and the concatenate them in a single file. A small file containing the metadata of the data file is also generated to provide an overview of the file.The following settings were used to produce our desired datasets based on the size we wanted to achieve :

	1GB	2GB
Samples	15 M	25 M
Noise	0.2	0.2
K-Means	16	clusters

V. Setting up the clusters

A. Ray

1) Master: After having successfully set up the Ray framework, we can proceed to the terminal of the virtual machine we have assigned as the master, and enter the following:

```
ray start --head
```

This will start the ray head node and provide us with the url of the ray dashboard.

2) Workers: After having successfully set up the Ray framework, and the scheduler is already running we can proceed to the terminal of the virtual machines we have assigned as the workers, and enter the following:

```
ray start --address <IP>:6379
```

This will start a Ray worker process that will connect to the IP provided (in our case the IP that was given to the Master virtual machine).

B. Dask

1) Master: After having successfully set up the Dask framework, we can proceed to the terminal of the virtual machine we have assigned as the master, and enter the following:

```
dask scheduler
```

This will start the scheduler and assign an IP address to it. In addition to the IP address, two ports will also be assigned to it, which by default are 8786 for the scheduler and 8787 for the dashboard.

2) Workers: After having successfully set up the Dask framework, and the scheduler is already running we can proceed to the terminal of the virtual machines we have assigned as the workers, and enter the following:

```
dask worker tcp://<IP>:8786
```

This way, we create a worker that will automatically connect to the scheduler, given that the IP address and port we provided are correct. We can have greater control while creating a worker, and specify how much memory it has access to and how many threads it is allowed to use, with the following parameters :

```
--nthreads 1 --memory-limit 1GB
```

VI. Code for measuring performance

Since Ray and Dask are scaling frameworks, they do not affect the majority of the python code we want to execute. Apart from the different methods to connect to the clusters, there are some notable differences in some small parts of the code.

A. Notable code differences

1) Data handling and loading:

- *Ray :* Data is loaded from .libsvm file into a numpy array.
- *Dask :* Data is loaded from .libsvm file into a Dask dataframe which is then converted into a Dask array.

2) Classification:

- *Ray :* Our data is loaded into a numpy array, then a configuration space for hyperparameter tuning is setup and finally the tuning is done using Ray Tune.
- *Dask :* Our data is loaded into a dask dataframe and then converted to a dask array and finally Dask-ML's train_test_split is used to perform the train-test split step to help evaluate each classifier .

3) Grid Search:

- *Ray :* We use ray.tune.grid_search to perform grid search.
- *Dask :* In order to perform grid search we use dask_ml.model_selection.GridSearchCV using cross-validation.

4) Clustering:

- *Ray :* Ray does not provide any specific APIs for clustering so we use the tools provided by sklearn.cluster
- *Dask :* Dask has dask_ml.cluster that provides us with unsupervides clustering algorithms, such as KMeans.

5) PCA:

- *Ray :* We use PCA from sklearn.decomposition to perform principal component analysis.
- *Dask :* The Dask framework provides us with dask_ml.decomposition.PCA to perform principal component analysis.

B. Connecting to the clusters

1) Ray: In the beginning of the file that contains the python code we want to execute, we need to add the following :

```
ray.init(address="<IP>:6379")
```

Now, the process that is responsible for running the python script will connect to the Ray cluster to distribute the tasks. Between the brackets is the IP address of the machine where the master node is running. At the end of the file we add :

```
ray.shutdown()
```

To close the connection to the head node.

2) Dask: In the beginning of the file that contains the python code we want to execute, we need to add the following :

```
client = Client("tcp://<IP>:8786")
```

Now, the process that is responsible for running the python script will connect to the Dask cluster to distribute the tasks. Between the brackets is the IP address of the machine where the scheduler is running. At the end of the file we add :

```
client.close()
```

To close the connection to the scheduler, so it can be used for another process.

VII. Results

Initially, our project aimed to assess the performance of large datasets. However, constraints imposed by hardware limitations and unforeseen bugs hindered our ability to test datasets of 3, 7, and 10 gigabytes, as originally intended. Despite our best efforts, we restricted our evaluations to smaller datasets of 1GB and 2GB. Consequently, our study focused on comparing the efficiency and accuracy of the Ray and Dask frameworks in handling these reduced dataset sizes.

A. Classification

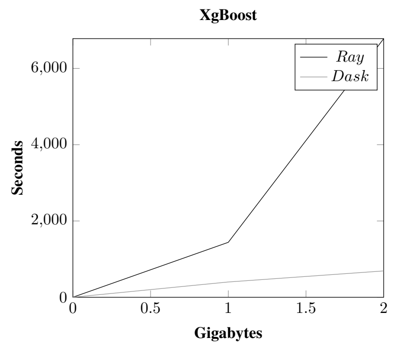

Fig. 1. XgBoost - Time to complete computations.

Time		
	1GB	2GB
Ray	1470	6780
Dask	400	990
Accuracy		
	1GB	2GB
Ray	0.70	0.87
Dask	0.70	0.69

TABLE I
XgBoost Time and Accuracy.

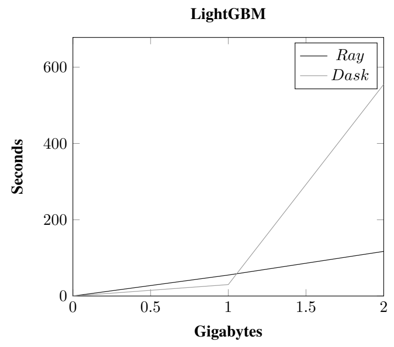

Fig. 2. LightGBM - Time to complete computations.

Time		
	1GB	2GB
Ray	55	117
Dask	30	555
Accuracy		
	1GB	2GB
Ray	0.85	0.79
Dask	0.33	0.40

TABLE II
LightGBM Time and Accuracy.

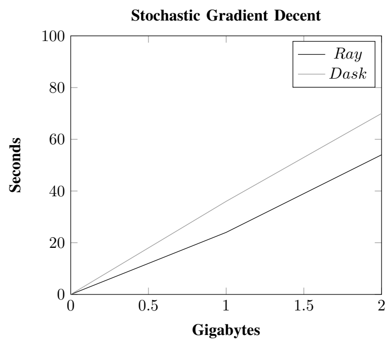

Fig. 3. SGD - Time to complete computations.

Time		
	1GB	2GB
Ray	24	54
Dask	36	70
Accuracy		
	1GB	2GB
Ray	0.60	0.55
Dask	0.60	0.55

TABLE III
SGD TIME AND ACCURACY.

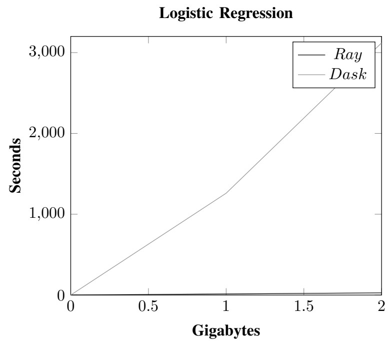

Fig. 4. LR - Time to complete computations.

Time		
GB/Sec	1GB	2GB
Ray	14	27
Dask	1260	3120
Accuracy		
GB/Ac	1GB	2GB
Ray	0.60	0.55
Dask	0.58	0.55

TABLE IV
LR TIME AND ACCURACY.

B. Clustering

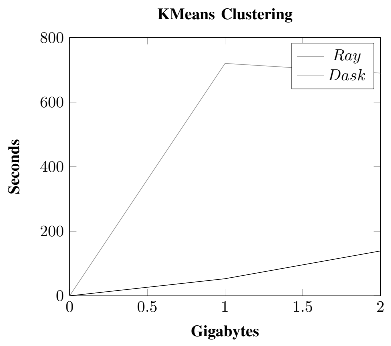

Fig. 5. Clustering - Time to complete computations.

Time		
GB/Sec	1GB	2GB
Ray	53	139
Dask	720	690
Davis-Bouldin Score		
GB/Sc	1GB	2GB
Ray	1.05	1.01
Dask	0.77	0.75

TABLE V
CLUSTERING TIME AND DB SCORE.

VIII. OBSERVATIONS

A. Classification

1) XgBoost: Concerning XGBoost, Ray emerges as the more accurate framework, albeit with a slower runtime. Notably, as the dataset size grows, the computational time required by Ray increases significantly more than that of Dask. XgBoost also seems to require the most ammount of time out of any other computation we performed.

2) LightGBM: LightGBM demonstrates a moderate performance in terms of running speed. Intriguingly, while Ray maintains a higher level of accuracy, Dask exhibits a notable slowdown, particularly with larger dataset sizes.

3) SGD: In SGD, we observe comparable performance and scaling with data size across both frameworks. Notably, Ray exhibits a slight speed advantage, while both frameworks maintain an equivalent level of accuracy.

4) LR: In the case of Logistic Regression (LR), despite similar accuracy levels, the efficiency of Dask is notably worse.

B. Clustering

In Clustering, Ray exhibits superior performance with smaller datasets, whereas Dask's effectiveness appears to improve as datasets expand. Our evaluation, employing the Davies-Bouldin score, consistently reveals that Dask maintains lower scores, indicating superior performance.

IX. Conclusion

Ray and Dask are versatile frameworks, boasting ease of application and scalability. While each excels in different tasks, our findings underscore substantial variations in efficiency even when performing similar operations. The choice between Ray and Dask hinges on the specific use case. Comprehensive research, conducted on more robust hardware and well-established datasets, would provide a deeper understanding.

Dask demonstrates straightforwardness in comparison to Ray, proving simpler to set up and operate. In summary, given the considerable variability in results, users and developers are advised to assess their intended operations and conduct thorough testing with both frameworks to determine the optimal choice. Notably, due to Dask's optimization for handling larger-than-memory datasets, it is expected to outperform in scenarios involving such datasets.

References

[1] "Home — okeanos-knossos IAAS ." https://okeanos-knossos.grnet.gr/home

[2] "What is a virtual machine (VM)?." https://www.redhat.com/en/topics/virtualization/what-is-a-virtual-machine

[3] "Welcome to Python.org." https://www.python.org

[4] "Our Success Stories — Python.org. " https://www.python.org/success-stories

[5] "Productionizing and scaling Python ML workloads simply — Ray." https://www.ray.io

[6] "Ray Documentation." https://docs.ray.io/en/latest/index.html

[7] "Dask — Scale the Python tools you love." https://www.dask.org

[8] "Dask Documentation." https://docs.dask.org/en/stable

[9] "What is Data Classification and Why is it Important?" https://www.techtarget.com/searchdatamanagement/definition/data-classification

[10] "Grid Search for model tuning." https://towardsdatascience.com/grid-search-for-model-tuning-3319b259367e

[11] "Clustering in Machine Learning." https://www.javatpoint.com/clustering-in-machine-learning

[12] "Principal Component Analysis (PCA) Explained." https://builtin.com/data-science/step-step-explanation-principal-component-analysis