Scalable Cloud-Based Distributed Computing for Efficient Big Data Analytics: A Dask Integration Approach

ENGR-E516 Engineering Cloud Computing

Anirudh Penmatcha ✉ Dilip Nikhil Francies ✉ Subhadra Mishra ✉

1. Introduction

The exponential growth of data in recent years has transformed the landscape of data science, presenting both unprecedented opportunities and significant challenges. With approximately 2.5 quintillion bytes of data generated daily, the demand for scalable tools and frameworks to analyze and derive insights from this vast volume of information has never been more pressing. In the single year of 2017, the Weather Channel received an astounding 18 million forecast requests per minute, while an overwhelming 103 million spam emails inundated inboxes every minute. Such statistics underscore the magnitude of data generation and the urgent need for efficient data processing methodologies. This has led to the development of a plethora of frameworks for distributed data analysis, including batch [4], streaming [7], and graph [12] processing systems.

Traditionally, data scientists have leveraged the Python Open Data Science Stack, relying on tools like Pandas, SciPy, NumPy, and scikit-learn for data analysis and building predictive models. While effective for smaller datasets that can fit into memory, these tools often falter when confronted with the monumental scale of contemporary data because these were not designed to scale beyond a single machine. The complexity intensifies as many tools, like PySpark, pose steep learning curves. Analysts often resort to rewriting their computations using more scalable tools, sometimes in different languages, leading to frustration and slowed inference. Additionally, PySpark users may need to migrate their codebase to Scala or Java for optimal Spark utilization [6], further complicating matters. Furthermore, the lag in feature updates for PySpark compared to Java and Scala adds to the challenge. Considering these hurdles, what options do Python-native developers have for running analytics on big data leveraging the distributed computing power of the cloud?

In this project, we propose to build a platform designed to be dynamically scalable based on user demand. At its core, it is the integration of Dask[11], a parallel execution framework, with JupyterHub, containerized and deployed on a cloud instance. We intend to benchmark the performance of Dask as a distributed computing framework on our cluster by conducting computationally intensive hyperparameter tuning of tree-based XGBoost algorithm on big data. Through systematic variations in input format, chunk size, task schedulers, worker nodes, clusters, and threading configurations, we seek to quantify the performance and compare it to baseline values obtained from running the program on the instance without distributing the workload. Our evaluation benchmarking serves two purposes: 1) to compare the performance of running computationally intensive ML algorithms with and without parallelizing the workload with Dask on cloud. 2) To understand in depth the many components of distributed computing that impact its performance.

2. Related Work

2.1. Dask

Dask is a powerful open-source Python library designed for parallel computing tasks [11]. It seamlessly scales Python code from multi-core local machines to expansive distributed clusters in the cloud. The library offers low-level APIs, empowering programmers to execute custom algorithms in parallel efficiently. Dask employs a sophisticated work-stealing scheduler, meticulously optimized to execute task graphs with utmost efficiency. Interestingly, even a completely random scheduler remains surprisingly competitive with Dask's built-in scheduler [1]. However, the primary bottleneck of Dask lies in its runtime overhead. Notably, Dask demonstrates superior performance compared to Spark with up to two workers. Conversely, Spark exhibits improved performance with an increase in the number of workers [1], particularly noticeable with four and thirty-two workers. Dask's effective data partitioning strategy becomes particularly evident with fewer workers, enabling optimal resource utilization and minimizing execution times.

2.2. Sanzu

Sanzu serves as an indispensable tool for assessing systems handling data processing and analytics tasks [13]. Its benchmarking approach encompasses both micro and macro benchmarks. The micro-benchmark isolates and evaluates basic operations, including tasks related to reading and writing data, data manipulation, statistical analysis, machine learning, and time series analysis. Conversely, the macro benchmark assesses analytics applications by simulating real-world scenarios, focusing particularly on sports and smart grid analytics. This evaluation encompasses five prominent data science frameworks and systems: R, Anaconda Python, Dask, PostgreSQL (MADlib), and PySpark.

2.3. Spark

Apache Spark is an open-source general-purpose cluster computing framework developed by the AMP lab at the University of California, Berkeley[10]. The core of Spark is written in Scala, runs on the Java virtual machine (JVM), and offers a functional programming API to Scala. This reference study [14] undertook a rigorous benchmarking analysis comparing Dask and Apache Spark, alongside the single-node SciKit Learn framework. Conducted on the EMNIST dataset, consisting of subsets ranging from 50k to 250k samples, assessments were performed across diverse operating systems and levels of parallelization. Notably, findings revealed a slight edge in favor, Dask showcased prowess in data frame manipulation, Apache Spark exhibited superior end-to-end processing performance, particularly evident on larger datasets, with F1 scores comparable to those achieved by Dask. Additionally Dugre, Hayot-Sasson, and Glatard [5] compared Apache Spark and Dask across three neuroimaging applications, finding no significant performance differences. They noted that differences in engine overheads did not affect performance due to compensating lower transfer times when data transfers utilized bandwidth fully. This suggests a need for future research on strategies to reduce the impact of data transfers on application performance.

2.4. XGBoost

XGBoost is a scalable tree boosting system widely utilized in machine learning, featuring a sparsity-aware algorithm and weighted quantile sketch for improved performance on sparse data. Leveraging insights on cache access patterns, data compression, and sharding, XGBoost achieves scalability beyond billions of examples with minimal resource consumption [3]. Junda Chen et al. [2] demonstrated that the optimal cluster size for the cost-effectiveness of distributed XGBoost training is primarily determined by the sample size, with datasets containing more entries benefiting from increased parallelism. Conversely, increasing dimensionality did not yield significant gains from parallel processing. The block size emerged as a critical factor affecting both matrix creation performance and training efficiency. Furthermore, Zoya Masih [9] demonstrates Dask-ML's linear scalability in execution time and active data storage with increasing LAZ file (popular vector formats for storing LiDAR data as point clouds) sizes. Consistent CPU utilization at around 25 percent indicates efficient resource usage, while worker bandwidth displays satisfactory communication. Memory-mapped access and deferred loading effectively address challenges in LAZ file processing, resulting in moderate memory scaling and efficient resource utilization.

3. Proposed Task

In the initial phase of this study, we aim to install the Dask gateway on Kubernetes and configure JupyterHub to launch private Dask clusters that run Jupyter Notebook. We prefer to deploy the Kubernetes service on AWS, Jetstream2, or Saturn Cloud, preferably on a medium-scale 2Xlarge instance with 8 cores and 64 GB RAM.

In the next phase of our study, we will develop a Dask program specifically tailored for executing hyperparameter tuning via grid search on an XGBoost algorithm. This program will be applied to a dataset [8] comprising 1.4 GB of data, with over 1.5 million records with 119 features. Given the inherently large hyperparameter spaces and computationally intensive configurations characteristic of modern ML models, this task presents an opportunity to investigate the performance evaluation of distributed computing capabilities offered by Dask. Our primary focus will involve a comprehensive examination of compute bandwidth, memory bandwidth, network bandwidth, disk bandwidth, scheduler overhead, and serialization costs of Dask distributed computation.

4. Contribution

Each member of the team has been assigned specific tasks tailored to their strengths and experiences, ensuring a comprehensive approach to our experiment. Dilip is responsible for orchestrating and executing the XGBoost computation with Dask on the cluster. His role extends to observing various benchmarks, analyzing the results, and critically evaluating the performance metrics. To establish a baseline for our experiment, he will primarily consider execution time, the number of tasks per core, total and compressed bytes transferred, and cluster memory usage of the hyperparameter tuning code executed on the instance without any distributed computing attached. Using this baseline, Dilip will compare the performance of Dask distribution with different input data sizes and chunks, numbers of CPU workers, different input formats, etc., and intends to incorporate additional metrics for a more comprehensive analysis as the research progresses. Anirudh will focus on installing the Dask gateway on Kubernetes and configuring JupyterHub which can be containerized and deployed at scale on the cloud. Meanwhile, Subhadra will work on the dataset, ensuring its readiness for experimentation and running the training on the instance without the distributed functionality to help us establish baseline performance metrics essential for comparative analysis and benchmarking. Subhadra will also delve into distributed data formats such as HDFS, Parquet, Dask DataFrames, and Dask Arrays, exploring varying partition sizes to address scalability issues related to input data handling. All three authors have actively contributed to the research process beyond their assigned tasks. They have collectively participated in extensive literature reviews, research paper analysis, and result generation.

5. Timeline

- Week 1: Conduct a comprehensive literature review and draft the project proposal report.
- Week 2: Establish the Dask cluster on a cloud platform and prepare the data model.
- Week 3: Develop Python code to parallelize XGBoost tuning with Dask and conduct initial training on a small subset of data and generate mid term report.
- Week 4-5: Execute the program on the Dask cluster in the cloud and analyze performance metrics.
- Week 6: Review results, delve into metric details, and draw conclusive insights.
- Week 7-8: Compile results, draft discussion segments, conclude findings and complete the references section for the research paper.

References

[1] Baglioni, Michele, Fabrizio Montecchiani, Mario Rosati et al. 2023. Large-scale computing frameworks: Experiments and guidelines. En *CEUR WORKSHOP PROCEEDINGS*, vol. 3606, CEUR-WS

[2] Chen, Junda, Aditya Kumar Akash & Yukiko Suzuki. 2021. Explore optimal degree of parallelism for distributed xgboost training. Relatório técnico.

[3] Chen, Tianqi & Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. En *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 785–794

[4] Dean, Jeffrey & Sanjay Ghemawat. 2008. Mapreduce: simplified data processing on large clusters. *Communications of the ACM* 51(1). 107–113

[5] Dugré, Mathieu, Valérie Hayot-Sasson & Tristan Glatard. 2019. A performance comparison of dask and apache spark for data-intensive neuroimaging pipelines. En *2019 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)*, 40–49. IEEE

[6] Dünner, Celestine, Thomas Parnell, Kubilay Atasu, Manolis Sifalakis & Haralampos Pozidis. 2017. Understanding and optimizing the performance of distributed machine learning applications on apache spark. En *2017 IEEE international conference on big data (big data)*, 331–338. IEEE

[7] Gonzalez, Joseph E., Reynold S. Xin, Ankur Dave, Daniel Crankshaw, Michael J. Franklin & Ion Stoica. 2014. Graphx: graph processing in a distributed dataflow framework. En *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation* OSDI'14, 599–613. USA: USENIX Association

[8] Kaggle. 2021. Tabular playground series - sep 2021.

[9] Masih, Zoya. 2023. Analyzing io performance when using dask-ml

[10] Meng, Xiangrui, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen et al. 2016. Mllib: Machine learning in apache spark. *Journal of Machine Learning Research* 17(34). 1–7

[11] Rocklin, Matthew et al. 2015. Dask: Parallel computation with blocked algorithms and task scheduling. En *SciPy*, 126–132

[12] Vatter, Jana, Ruben Mayer & Hans-Arno Jacobsen. 2023. The evolution of distributed systems for graph neural networks and their origin in graph processing and deep learning: A survey. *ACM Computing Surveys* 56(1). 1–37

[13] Watson, Alex, Deepigha Shree Vittal Babu & Suprio Ray. 2017. Sanzu: A data science benchmark. En *2017 IEEE International Conference on Big Data (Big Data)*, 263–272. IEEE

[14] Zevnik, Filip, Din Music, Carolina Fortuna & Gregor Cerar. ???? Scikit learn vs dask vs apache spark benchmarking on the eminst dataset