Exploring Data at Scale with Arkouda

A Practical Introduction to Scalable Data Science

Ben McDonald
HPC Advanced Dev
Hewlett Packard Enterprise
Houston, Texas, USA
ben.mcdonald@hpe.com

ABSTRACT

Data scientists can be thought of as modern-day explorers, venturing into the vast unknown of information. However, this exciting journey is not without its hurdles. One of the biggest challenges they face is the sheer immensity of data they encounter. Modern datasets cannot fit in laptop memory, containing terabytes or even petabytes of information. Working with such massive data requires specialized tools and techniques to extract meaningful insights. As data sets are growing ever larger, data science demands interactivity, where scientists can learn while working with the data. At the same time, data science demands scalability, where scientists are able to work with data sets in their entirety. Data scientists have naturally been drawn to Python as it provides interactivity through its read, evaluate, print loop and performance through its utilization of libraries written in other languages, like C and Fortran. These libraries typically are not designed for HPC and run into problems when attempting to scale. The gap that Arkouda fills in the data science landscape is a library that is both interactive, providing a familiar Python API, and scalable, leveraging a scalable Chapel server in the backend. Arkouda is a framework for scalable Python packages for interactive data science and has applications ranging from oceanography to net flow analysis.

KEYWORDS

Data science, big data, Python, distributed workflows, parallelization, Chapel language, performance

1 Introduction

Python's interactive nature and its ability to leverage pre-compiled, performant libraries like NumPy have long made it the go-to language for data science. However, as datasets balloon beyond individual machines, traditional Python struggles to keep pace. To address scalability challenges, Python-based solutions like Dask [1] have emerged, as well as systems that combine a Python frontend with different backend technologies. Arkouda [2] falls into the latter category. This NumPy-like package empowers data scientists to conduct exploratory data analysis at supercomputing scale. Arkouda users can tackle terabyte-sized datasets distributed across thousands of nodes.

While initially focused on providing a scalable subset of NumPy's functionality for data science, Arkouda's framework itself has recently undergone a significant overhaul. This framework enables rapid development of high-performance computing code for diverse applications, extending its reach beyond data science.

2 Scalability and Performance

Figure 1 presents the performance profile of Arkouda's argsort operation. Argsort computes the indices required to sort an array, enabling efficient retrieval of sorted elements. These benchmarks were conducted on a Cray-EX supercomputer utilizing a SlingShot interconnect. Notably, without architecture-specific optimizations, Arkouda's argsort demonstrated near-linear scalability, processing 256 TB of data across a cluster of 8,192 nodes in approximately 31 seconds.

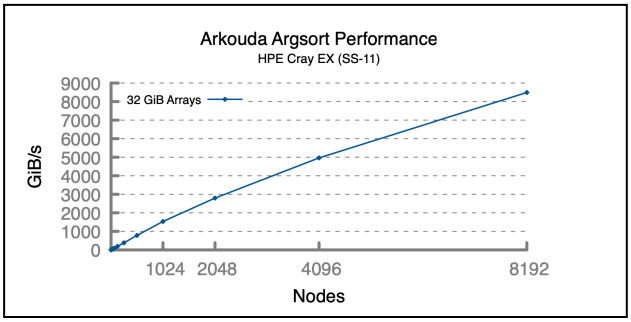

Figure 1: Arkouda Argsort Performance

3 The Arkouda Architecture

Arkouda is a Python package designed for large-scale, distributed exploratory data analysis. Built on a NumPy-like API, it offers a familiar interface for data scientists, minimizing workflow disruptions. The package comprises a Python frontend and a Chapel backend. The frontend manages metadata and interacts with the backend's distributed object store, enabling data to reside on diverse computational resources, including supercomputers and cloud environments. By decoupling where the data is stored in memory from client memory, Arkouda facilitates seamless

analysis of massive datasets. The package's versatility has been demonstrated in various domains, ranging from network flow analysis to brain imaging analysis.

Figure 1 demonstrates the strong scalability and performance of Arkouda's implemented and optimized operations. While Arkouda's capabilities currently encompass a specific set of functionalities, its flexible framework is designed for easy integration with Chapel libraries. Recent advancements have significantly streamlined the development process, reducing the need for extensive boilerplate code and in-depth knowledge of Arkouda's internals to introduce new features.

Figure 2 compares the developer experience before and after the framework changes. Previously, supporting covariance in Arkouda necessitated a cumbersome 97 lines of primarily boilerplate code. In contrast, the new framework requires only 7 intuitive lines to achieve the same functionality, significantly reducing the cognitive load on developers and eliminating the need for deep Arkouda expertise in order to make meaningful contributions to the project.

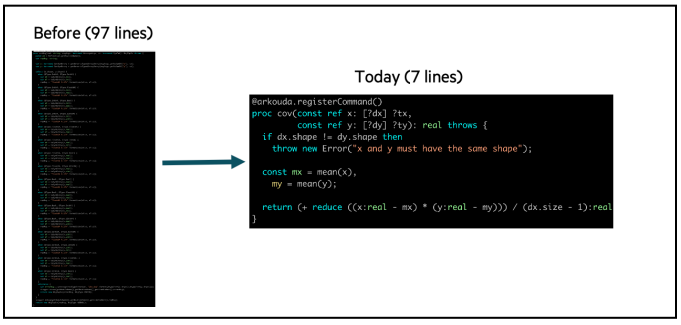

Figure 2. Improved Arkouda developer experience

4 Related Technologies

There are two primary approaches to creating scalable Python packages for HPC environments.

The first approach involves scaling traditional Python code to operate on HPC machines. This method offers several advantages, including seamless integration with existing Python packages like PyTorch [3]. However, it often falls short in terms of performance compared to native HPC code.

The second approach addresses performance limitations by starting with high-performance code and providing a Python interface. While delivering superior performance for specific workloads, interoperability with existing Python libraries can be challenging. Apache Spark [4] exemplifies this approach, using a Scala backend and exposing a Python API.

A key distinction between Apache Spark and Arkouda lies in their evaluation models. Spark's lazy evaluation is well-suited for batch jobs with predefined workflows, enabling the creation of optimized computation graphs. Conversely, Arkouda's eager evaluation excels in dynamic scenarios where the workflow is unknown in advance, allowing for on-the-fly computations and iterative data exploration.

By combining the strengths of both systems, users can leverage Arkouda for initial data exploration and workflow development before transitioning to Spark for optimized batch processing, maximizing the benefits of each platform.

5 Arkouda Usage Demo

5.1 Demo Goals

In this tutorial, attendees will work hands-on with a real-world data set, performing data science through a Jupyter notebook in a GitHub Codespace with Arkouda and Jupyter provided at the click of a button. Through this tutorial, attendees will understand how to launch and connect to an Arkouda server, get an overview of the Arkouda API, understand the file formats supported by Arkouda, and get a picture of how Arkouda can be used for real data science workflows.

5.2 Demo Outline

To enhance accessibility and engagement, the Arkouda demo will be conducted within GitHub Codespaces, allowing attendees to follow along without requiring prior Arkouda installation.

The demonstration will commence by importing the NYC Taxicab [5] dataset into the Arkouda environment. Subsequent steps involve a preliminary data exploration to familiarize attendees with the dataset's characteristics. To enrich the dataset further, integration with Pandas will be showcased, demonstrating Arkouda's interoperability. The demo will culminate in an exploration of Arkouda's advanced features, including GroupBy, before inviting participants to conduct their own analyses within the provided Codespace environment.

REFERENCES

[1] Marta Moreno, Ricardo Vilaca and Pedro G. Ferreira. 2022. Scalable transcriptomics analysis with Dask: applications in data science and machine learning. In BMC Bioinformatics. https://doi.org/10.1186/s12859-022-05065-3.

[2] Ben McDonald. 2023. Removing Temporary Arrays in Arkouda. in Chapel Implementors and Users Workshop (CHIUW'23). 2 pages. https://chapel-lang.org/CHIUW/2023/McDonald.pdf.

[3] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In arxiv. NeurIPS, 12 pages. https://doi.org/10.48550/arXiv.1912.01703.

[4] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker and Ion Stoica. 2016. Apache Spark: A Unified Engine for Big Data Processing. In Communications of the ACM. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/2934664.

[5] NYC Taxi and Limousine Commission (TLC) [Dataset]. (2024, August 6). TLC Trip Record Data. Retrieved from https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page