Arkouda: NumPy-like arrays at massive scale backed by Chapel

Michael Merrill*, William Reus†, and Timothy Neumann‡
U.S. Department of Defense Washington DC, USA
Email: *mhmerrill@mac.com, †reus@post.harvard.edu, ‡timothyneumann1@gmail.com

***Abstract*—Exploratory data analysis (EDA) is a prerequisite for all data science, as illustrated by the ubiquity of Jupyter notebooks, the preferred interface for EDA among data scientists. In order to expand the scale of EDA, we have designed and built a software package for using NumPy-like arrays and Pandas-like data frames at massive scale. We have also integrated this software package into our data science workflow. This software package, called *Arkouda*, is primarily implemented in the Chapel programming language. Arkouda is currently in process to be open-sourced so our work can be shared with a broader community. In the proposed talk we will cover the why, how, and what of Arkouda.**

I. Summary

Exploratory data analysis (EDA) [1] [2] is a prerequisite for all data science, as illustrated by the ubiquity of Jupyter notebooks [3] [4], the preferred interface for EDA among data scientists. The operations involved in exploring and transforming the data are often at least as computationally intensive as downstream applications (e.g., machine learning algorithms), and as datasets grow, so does the need for HPC-enabled EDA. However, the inherently interactive and open-ended nature of EDA does not mesh well with current HPC usage models. Meanwhile, several existing projects from outside the traditional HPC space attempt to combine interactivity and distributed computation using programming paradigms and tools from cloud computing (e.g., Spark [5] and Dask [6]), but none of these projects have come close to meeting our needs for high-performance EDA.

To fill this gap, we have developed a software package, called *Arkouda*, which allows a user to interactively issue massively parallel computations on distributed data using functions and syntax that mimic NumPy [7] and Pandas [8], the underlying computational libraries used in the vast majority of Python data science work-flows.

Data scientists in our organization are using Arkouda on a daily basis to perform interactive, exploratory analyses of terabytes of network meta-data which none of their existing tools could handle. In these applications, users of Arkouda have tended to iterate rapidly between multi-node execution with Arkouda and single-node analysis in Python, relying on Arkouda to filter a large dataset down to a smaller collection suitable for analysis in Python, and then feeding the results back into Arkouda computations on the full dataset. This paradigm has already proved very fruitful for EDA. Our goal is to enable users to progress seamlessly from EDA to specialized algorithms by making Arkouda an integration point for HPC implementations of expensive kernels like FFTs, sparse linear algebra, and graph traversal. With Arkouda serving the role of a shell, a data scientist could explore, prepare, and call optimized HPC libraries on massive datasets, all within the same interactive session.

Because our goal is to make HPC programming and usage more approachable [9], we are seeking to open-source Arkouda. Permission has been granted to open-source the package and legal counsel is determining which open-source license to leverage.

The computational heart of Arkouda is a Chapel [10] [11] interpreter that accepts a predefined set of commands from a client (currently implemented in Python) and uses Chapel's built-in machinery for multi-locale and multi-threaded execution to evaluate computations at scale [12] [13]. EDA operations in Arkouda currently scale to hundreds of HPC nodes comprising tens of thousands of cores and hundreds of terabytes of memory.

The Arkouda package represents a significantly sized Chapel program which exercises many of the language's features. The server code is currently divided into 21 modules consisting of over 8500 lines of code. The client consists of one Python3 module with over 1000 lines of code. Total package code size is relatively small in relation to the amount of functionality it represents. By contrast, an MPI [14] [15] implementation in a less Pythonic language like C, C++, or Fortran would have required about an order of magnitude more code, in order to implement functionality like multi-resolution parallel constructs, distributions, parallel iterators, memory managed objects, and other higher level language features which Chapel enjoys.

Another distinctive feature of Chapel that we rely on in developing Arkouda is portability: our typical workflow includes prototyping on a laptop, multi-locale testing and scaling on a 32-node infiniband cluster, and further scaling and deployment on a supercomputer. The same code compiles and runs in all three of these cases with no modifications and no platform-specific code.

Maintaining interactivity (i.e., an unbroken human thought loop) during analysis is vital to the users of Arkouda. The primary performance metric for Arkouda is, therefore, the execution time of server-side operations. Most of these operations are tuned to run at interactive speed from a Jupyter notebook cell, for example sub-second addition of tera-scale arrays on hundreds of nodes.

This talk will expand upon the above material and cover topics like:

- Why is HPC-enabled EDA necessary?
- How can software enable low-latency hypothesis testing?
- Why did we choose Chapel?
- How did we implement the software and what challenges did we face?
- How does the code perform and scale?
- Why is an integration point (shell) for HPC libraries important?
- What is the future vision for Arkouda?

ACKNOWLEDGMENT

We want to thank Brad Chamberlain and members of the Chapel development team for their enthusiastic support, discussions, and feedback. We would also like to thank our interns for their willingness to incorporate Arkouda into daily usage, and for providing feedback. There are many others we would like to show gratitude to who have provided support and feedback during the development.

REFERENCES

[1] J. W. Tukey, *Exploratory Data Analysis*. Reading, MA: Addison-Wesley, 1977.
[2] D. Donoho, "50 Years of Data Science," *Journal of Computational and Graphical Statistics*, vol. 26, no. 4, pp. 745–766, 2017.
[3] https://jupyter.org.
[4] J. Somers, "The Scientific Paper is Obsolete," *The Atlantic Daily Newsletter*, April 2018, https://www.theatlantic.com/science/archive/2018/04/the-scientific-paper-is-obsolete/556676/.
[5] https://spark.apache.org.
[6] https://docs.dask.org/en/latest/.
[7] https://numpy.org.
[8] https://pandas.pydata.org.
[9] J. Dursi, "HPC is dying, and MPI is killing it," April 2015, https://www.dursi.ca/post/hpc-is-dying-and-mpi-is-killing-it.html.
[10] B. L. Chamberlain, "Chapel," in *Programming Models for Parallel Computing*, P. Balaji, Ed. MIT Press, November 2015, ch. 6, pp. 129–159.
[11] https://chapel-lang.org.
[12] P. Husbands and C. Isbell, "The Parallel Problems Server: A Client-Server Model for Interactive Large Scale Scientific Computation," *Proceedings of VECPAR'98*, June 1998.
[13] P. Husbands, C. L. Isbell, and A. Edelman, "Interactive Supercomputing with MITMatlab," August 2001.
[14] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "A high-performance, portable implementation of the MPI message passing interface standard," *Parallel computing*, vol. 22, no. 6, pp. 789–828, 1996.
[15] https://www.mpi-forum.org.