AXS: Making end-user petascale analyses possible, scalable, and usable

Petar Zečević,[1] Colin T. Slater,[2] Mario Jurić[2] and Sven Lončarić[1]
[1]*University of Zagreb, Croatia;* petar.zecevic@fer.hr , sven.loncaric@fer.hr

[2]*University of Washington, USA;* ctslater@uw.edu , mjuric@astro.washington.edu

Abstract. We introduce AXS (Astronomy eXtensions for Spark), a scalable open-source astronomical data analysis framework built on Apache Spark, a state-of-the-art industry-standard engine for big data processing. In the age when the most challenging questions of the day demand repeated, complex processing of large information-rich tabular datasets, scalable and stable tools that are easy to use by domain practitioners are crucial. Building on capabilities present in Spark, AXS enables querying and analyzing almost arbitrarily large astronomical catalogs using familiar Python/AstroPy concepts, DataFrame APIs, and SQL statements. AXS supports complex analysis workflows with astronomy-specific operations such as spatial selection or on-line cross-matching. Special attention has been given to usability, from conda packaging to enabling ready-to-use cloud deployments. AXS is regularly used within the University of Washington's DIRAC Institute, enabling the analysis of ZTF (Zwicky Transient Facility) and other datasets. As an example, AXS is able to cross-match Gaia DR2 (1.8 billion rows) and SDSS (800 million rows) in 20 seconds, with the data of interest (photometry) being passed to Python routines for further processing. Here, we will present current AXS capabilities, give an overview of future plans, and discuss some implications to analysis of LSST and similarly sized datasets. The long-term goal of AXS is to enable petascale catalog and stream analyses by individual researchers and groups.

1. Introduction

Modern astronomical surveys produce ever larger amounts of data. For example, Large Synoptic Survey Telescope (LSST), which is to start in 2022, will perform approximately 1000 observations of about 20 billion objects (LSST Science Collaboration 2009). This will result in approximately 50 PB of (raw) data over the 10 years of its operations.

Large survey analysis typically involves generating subsets of data by querying the larger catalog (usually with SQL), followed by downloading them (as FITS files, or similar). This is further followed by writing custom (usually Python) scripts to do the bulk of the analysis. While certainly doable, this *subset-download-analyze* workflow can be cumbersome, especially if executed repeatedly. Beyond working on a single catalog, we often want to positionally cross-match objects from two (or more) survey catalogs. This is often solved by pre-computing cross-match tables between catalogs, but as the number of catalogs grows this becomes inefficient ($O(N^2)$, where N is the number of catalogs). Finally, most upcoming servers today will be multi-epoch. There's

therefore a need to enable efficient analyses of *time series*, multiple observations of a single object.

2. Apache Spark as the Workflow Engine for Astronomical Datasets

AXS' goal is to provide a simple, user-friendly, scalable and efficient tool for cross-matching and analyzing data from large astronomical surveys. AXS extends Apache Spark (Zaharia et al. 2016), a general-purpose framework for big data processing. Spark's analytical processing functions are implemented through the Resilient Distributed Dataset (RDD) abstraction. Spark automatically distributes operations on RDDs and executes them in parallel with the minimum involvement of the user.

Spark's rich API can be accessed using SQL, Scala, Java, Python and R interfaces and it offers a breadth of functions: processing of structured and unstructured data, in a streaming or batch fashion, with graph and machine learning algorithms available. Spark can also recover from failures of individual nodes and is efficient and scalable. It enjoys wide and active user base, both in industry and academic world.

These are the main reasons why we chose Spark to be the base for building AXS upon.

3. Minimal Astronomy-specific Extensions for Spark

Spark already implements a significant fraction of functionality needed to support astronomical data analysis and explorations. We've found that with only two simple extensions – a data partitioning scheme built on existing bucketing support and (completely generic) improvements to the underlying Spark sort-merge join algorithm implementations – it is possible to deliver very performant cross-matching and querying functionality. The purposefully minimal nature of our changes may be more maintainable in the long-run relative to approaches which use astronomy-specific partitioning (e.g., HEALPix-based approaches such as Juric (2011), or the recent effort in Brahem et al. (2018)).

3.1. Data partitioning

Data partitioning scheme employed by AXS is based on the *Zones algorithm* (Gray et al. 2007), a well-known algorithm in the Astronomy community, with adaptations needed to support Spark's distributed architecture. In the AXS data partitioning scheme, sky is partitioned into horizontal bands of fixed height called *zones*. Zone height is one arc-minute by default, which makes 10800 zones. An object's zone is determined based on its *Dec* coordinate using this simple formula (Z is the zone height): $zone = \lfloor (Dec + 90)/Z \rfloor$

These are physically stored by Spark into *buckets*, implemented as Parquet files on a (potentially distributed) filesystem, based on the calculated *zone*: $bucket = zone\%N$ (where N is the number of buckets). The default number of buckets is 500, so there are about 21 zones per bucket, by default.

3.2. Distributed cross-matching

To efficiently cross-match tables bucketed by zones, as previously described, AXS partly relies on Spark's sort-merge join and partly on the contributed epsilon-join (Silva

et al. 2010) optimization. To understand this optimization, consider the following query:

```
select * from gaia, sdss where gaia.zone = sdss.zone AND
  gaia.ra BETWEEN (sdss.ra + DELTA, sdss.ra - DELTA) AND
  distance(gaia.ra, gaia.dec, sdss.ra, sdss.dec) < DELTA;
```

Here, two tables are joined based on the `zone` column and the distance between two rows is calculated using the provided `distance` function. Without the range condition on the secondary columns (`ra` columns), all the pairs of objects from the two tables having the same zone would need to be considered, which can involve a huge number of comparisons. However, Spark is not able to optimize this query and take advantage of the range condition on its own. Epsilon-join optimization uses a moving window which slides over the rows in the right table as the left row changes and hence effectively restricts the number of row pairs considered. This is possible because AXS data is sorted inside buckets by `zone` and `ra` columns. The `distance` function is thus evaluated only for the rows in the moving window, only one pass through the data is needed, and the join process uses the minimal amount of memory.

4. Cross-match performance tests

To benchmark AXS' cross-matching performance we used the Gaia DR2 catalog with 1.7 billion objects and the ZTF catalog with 2.9 billion objects. The cross-matching function calculated the distance between two objects using their RA and Dec coordinates, but an arbitrarily complex function could also be used (e.g., one taking the error-ellipses into account).

The cross-match operation for the benchmarked case results in 270 million rows. The tests included only counting of the resulting rows and no further processing on them was performed. The tests were performed on a single large machine with 512 GB memory, 48 CPUs and fast hard disks. Each Spark executor was given 12 GB of memory and a single CPU core.

We looked at how the cross-matching time depends on the number of Spark executors used. The results are shown in Figure 1. The two different lines in the graph show two sets of tests: with the cold and warm filesystem ("buffer") cache. Each data point on the graph is an average of three tests. The results with the cold buffer cache show the "worst-case" performance of the system which includes reading data from disk. The results with the warm cache more closely represent the time needed by the CPU-limited aspects of the cross-matching operation (we note that the dataset is larger than the memory available, so the data was never fully in the cache).

The lowest times we obtained, visible in the Figure, show that AXS can cross-match these two tables (Gaia DR2 and ZTF) in 25 seconds, with the data cached; and 205 seconds, when the data is not cached. Adding more than 24 executors doesn't offer any substantial performance improvements.

5. Summary

In this paper we describe and show initial performance benchmarks for Apache eXtensions for Spark (AXS), a system for performant querying, analyzing, and cross-matching data from astronomical catalogs, built on top of Apache Spark. AXS was

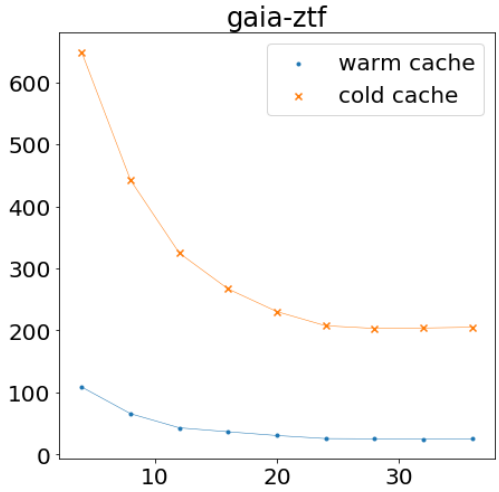

Figure 1. Duration in seconds of cross-matching Gaia DR2 and ZTF catalogs depending on the numbers of Spark executors used. The two curves correspond to tests with data obtained starting with a cold filesystem cache, and tests with some data already residing in the cache. Each data point is an average of three tests.

created with the goal of making the power of an industry-standard tool such as Apache Spark available to non-specialist astronomers for analysis of large-scale datasets. We next plan to further optimize the AXS Python API and overall performance, as well as to make AXS available both individually and as a part of a cloud-based service.

Acknowledgments. PZ, CTS, and MJ acknowledge support from the University of Washington College of Arts and Sciences, Department of Astronomy, and the DIRAC Institute. University of Washington's DIRAC Institute is supported through generous gifts from the Charles and Lisa Simonyi Fund for Arts and Sciences, and the Washington Research Foundation. MJ acknowledges further support from the Washington Research Foundation Data Science Term Chair fund, and the UW Provost's Initiative in Data-Intensive Discovery.

References

LSST Science Collaboration 2009, Lsst science book, version 2.0. arXiv:0912.0201

Brahem, M., Zeitouni, K., & Yeh, L. 2018, IEEE Transactions on Big Data, 1

Gray, J., A. Nieto-Santisteban, M., & Szalay, A. 2007

Juric, M. 2011, in American Astronomical Society Meeting Abstracts #217, vol. 43 of Bulletin of the American Astronomical Society, 433.19

Silva, Y. N., Aref, W. G., & Ali, M. H. 2010, in 2010 IEEE 26th International Conference on Data Engineering (ICDE 2010), 892

Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M. J., Ghodsi, A., Gonzalez, J., Shenker, S., & Stoica, I. 2016, Commun. ACM, 59, 56. URL http://doi.acm.org/10.1145/2934664