A Different Approach to Data Mechanics

Frederick Joossens
CS 591: Data Mechanics
BOSTON UNIVERSITY

Introduction

In this project we present a different set of tools to manage and process datasets exposed through data portals. Concretely, we present an automated and seamless network request caching tool, as well as an automated converter from various data portals (i.e. CKAN and Socrata) to Spark/HDFS/Parquet. The goal is to introduce contemporary tools and workflows to move from proof-of-concept processing to viable real-world deployments, as well as scale up data processing to support vastly larger datasets.

Tools

Prequest

Prequest is a lightweight wrapper around the requests library that facilitates data backups to a configurable data source.

- Seamless drop-in replacement for, with passthrough for other methods
- Use AWS Lambda to handle caching and retrieval of cached files
- Automatically cache each request from whitelisted domains to an S3 bucket

Sparkportal

Sparkportal is a multi-threaded tool that can automatically mirror CKAN or Socrata data platforms and convert the datasets to Parquet files.

- Download entire portal, or particular datasets
- Convert to Parquet on OS' filesystem, or HDFS for added performance
- Column-based indexing offers significant performance improvements for sparse data (memory trade-off)
- Automatically infer column headers and data types

While Spark supports unstructured data, Spark SQL makes it trivial to perform complicated data transformations. In a future version, the dataset and column names will be added to Python templates to provide code hinting for developers.

Performance

Spark is particularly optimized for large datasets, but introduces a ~4 second delay to start a new session. For real-time analysis, especially when connected to the web, sessions should intelligently be started in the background before the user actually runs the computation.

In a comparison between PySpark/Parquet and Scikit/MongoDB, we ran k-means on the 1.1GB 311 dataset of Boston, with 20 clusters, 300 iterations and 10 initialization states. No HDFS was used. On PySpark this took 28.5 seconds, while Scikit took 219 seconds. This is a factor of 7x difference. We anticipate this difference increasing with larger datasets, more iterations, and using HDFS.

As an added bonus, the resulting Parquet file is a factor 10 smaller than the JSON equivalent, and much faster to create as well. Mirroring the entire Boston Data Portal and converting it to Parquet only takes ~200 seconds (1.87GB).

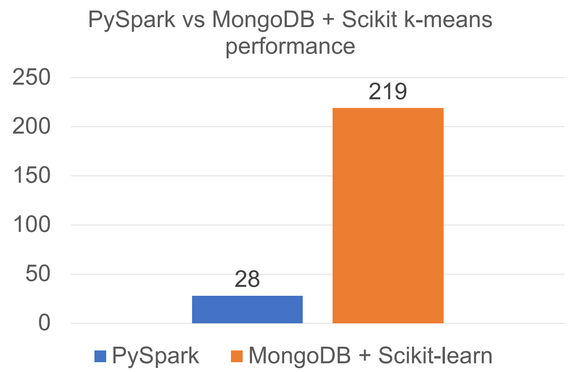

Figure 1 Time in seconds of k-means on 1.1GB 311 dataset with 300 iterations, 20 points and 10 initialization steps. Lower is better.

Future Work

While the current tools are certainly useful, more work is required to make (Py)Spark a viable solution for end-to-end data analysis. The SCC has a fully-featured Spark environment, but with the requirement to connect via the BU network or VPN, and no straightforward way to expose this service to the Internet, a custom Spark environment needs to be set up. A future "hybrid" model makes sense, where Spark writes its results to an external MongoDB instance, that is exposed through a RESTful API.

At first glance, Spark supports all common workflows taught in Data Mechanics, but a more thorough investigation is needed to ensure there are no gaps. This is certainly true for a linear programming solver. Z3 is a mature solution, while Spark's implementations should be classified as "gradware". More work is also needed on Sparkportal to make it more user-friendly: templates for code hinting with column names should be automatically generated, and more control should be offered for common scenarios, such as updating existing datasets, and adding resilience against corrupted datasets (invalid JSON, malformed headers, …)