Alec Bulkin, Alex DiCarlo, Payton McAlice

DS 4300

Prof. Rachlin

Final Project Report (Group 6)

Mongo-Stored WRDS Equity Database

Professor John Rachlin once said, “My trick was to always start with a quote. Maybe we should require all submissions include a notable quote on the topic of distributed databases. Who's with me!?” As discussed in class, relational databases- although widely popular- come with a myriad of drawbacks. Particularly, the process by which data are stored is slow and rigid. This means that it is difficult to quickly execute large queries in something like an SQL database, and inserting new properties is somewhat complicated. While this is not an issue on a smaller scale, large databases will inevitably experience performance issues, and this is especially true for relationally-stored data.

For our final project, we wanted to look closely at how to develop a quick way to access an extremely large dataset. Specifically, we wanted to work with equity data. This is because equity data are retrieved in extremely large batches for most practical uses, which means reading access to these data needs to be as efficient as possible. Those working with the data are often looking to access a variety of different variables to see how they change between different companies and over different periods of time. Thus, we planned to design a database and corresponding application for the storage and retrieval of such information.

The data we settled on working with were provided by Wharton Research Database Services (WRDS), which hosts a multitude of different business-centric datasets. This database is free-to-access for all Northeastern University students via the database collection provided by the school library. We chose to focus on the Compustat Capital IQ dataset, which is a highly comprehensive equity dataset with information spanning tens of thousands of companies over several decades, even well into the twentieth century. Although the entire dataset is much

larger, we chose to focus on 18 (of 74) variables and limited the timeframe to between January 2010 and April 2021. This was queried into a CSV file of over 70 million rows, and sized out around 10 GB.

For storing our dataset, we had considered both Redis and Mongo as viable options. Ultimately, we landed on Mongo because we found it to be more cohesive and allowed for both an easier upload process and a more intuitive formatting structure when preprocessing the data. Using Mongo allowed us to speed up both upload and retrieval time, and formatting the retrieved data to be more user-presentable proved to be rather straightforward. For the purposes of the project, our user-interface and connection were established using Python and the PyMongo library.

To upload data efficiently, we opted to do so in batches. There was a lot of discussion and brainstorming around how to make the data easy to upload and quick to retrieve in queries. Initially, we had agreed to define different key-store values using the CUSIP values for each company, with values then sorted into either a “static” dictionary (for data that remained constant over time) and a “timeseries” dictionary (for things like price and ADR data that change day-to-day). There were also plans to index the timeseries data for faster retrieval, but it was concluded that this would slow down upload speeds and was not worth the marginally improved retrieval time. Ultimately, the idea of a dictionary for static values was scrapped in favor of having all of those variables listed alongside the CUSIP (rather than being put in another dictionary under that CUSIP). Time-sensitive data were still kept in a separate dictionary and keyed by a datetime object. With all data structured to the aforementioned format, documents were uploaded in batches of 100.

For information retrieval, we focused on designing a method that would dynamically allow for the retrieval of static and/or time series data in the same query. Users can provide a list of assets of interest, a list of fields of interest, and optionally a time period of interest, and will be returned a readable dataframe with all the desired information. The user first specifies the

database storing all of the information, and then can submit these queries. A method (set_save) was created to save the query return, which is particularly useful from a user-end because it ensures that queries do not need to be rerun to reassess the same data.

We tested upload and retrieval speeds on two different computers. The first computer had a 2.60 GHz Intel Core i7-9750H processor and 16 GB of RAM. In past homework assignments (namely the SQL and Redis assignments), this computer had relatively low upload and retrieval speeds. On this computer, data are uploaded at a rate of 7.63 inserts per second, finishing at a runtime of 72 minutes. The second computer used had an M1 Mac chip with 16 GB of RAM. This computer, unsurprisingly, was much faster with both uploads and retrievals. This computer had an upload speed of 19.22 inserts per second and a runtime of 28 minutes. Read performance was equally as impressive; getting all timeseries data for 721 unique assets took just 35 seconds, while finding all data for a single unique asset took only 5 seconds. For a dataset that started as a 10 GB CSV file, this query is extremely efficient and provides fast returns for the user.

In conclusion, we wanted to focus on pushing the limits of Mongo in a practical manner. We took an extraordinarily large dataset of equity prices between 2010 and 2021 (with between 10,000 and 20,000 unique assets for every day) and wanted to see how efficient we could make user-friendly uploads and retrievals. The user-friendliness was a focus and motivation for things like the formatting of the data and the inclusion of easy re-accessibility of query results. The results seen are extremely promising, and the project as-in leaves room for expansion by means of more data or possibly a web-based user interface. This project demonstrates a practical purpose for those who look to work with large batches of equity data that they need to be able to quickly query and access.

In terms of contributions, Payton was responsible for the rough, preliminary design of data storage and uploading process. This design was then edited and perfected by Alec. Alex handled a lot of the retrieval process and design. It is worth saying that everyone had

involvement in all of these steps, but different people were directly coding at different points. Payton outlined the structure of the final paper, and Alec and Alex edited that work. The presentation was split evenly between everyone.