Big Data Lake Solution for Warehousing Stock Data and Tweet Data

Paul Adams, paula@smu.edu, Rikel Djoko, rdjoko@smu.edu, and Stuart Miller, stuart@smu.edu

***Abstract*—Purpose - This research paper aims to discover an optimal solution for parallel management and processing of financial markets data into warehousing and analysis engines used for buy-sell decision-making. Methods analyzed herein are components of the Hadoop ecosystem. Included is an in-depth analysis of parallel processing - using the Elastic MapReduce package on Amazon Web Services - and data warehousing with Apache Hive. Apache NiFi is used to direct workflow automation for data migration into the warehouse. Finally, a complete assessment of the combinations of levels of the various Hadoop ecosystem applications used is provided in the context of statistical inference.**

Design, Methodology, and Approach - One pertinent, underlying hypothesis within this study is to prove that there are differences in processing speeds between S3 and HDFS using normalized and optimized schema designs across multiple MapReduce configurations. This analysis is performed using 100 samples of the same volume of data in a repeated measures analysis using a Hotelling-T statistic. The two highest-performing configurations of S3 and HDFS are then assessed. (We need to get this information and report it) for the next section.

Findings - EXAMPLE: Applying S3 with an optimized schema using 10 reduces, map memory allocation of 2,048 mb and a reduce memory allocation of 4,096 mb is optimal for a more expensive S3 approach. Using HDFS from local storage with a configuration of 20 reduces, map memory allocation of 8,096 megabytes and reduce memory allocation of 10,020 megabytes is ideal for batch-level migrations and querying for large-volume processing. Across n repeated measures, using a one-tailed alpha, the resulting p-value is significant at 'Pr ¿ —t—' ¡ x.xxxx (confidence interval (x1, x2)), indicating local storage from HDFS outperforms S3 when using the selected configurations. However, local data storage capacity does not scale well for HDFS compared to cloud-based S3.

I. Introduction

DATA in the twenty-first century is expanding in volumes at exponential rates; every additional source of data that can act as a medium for data communication can obtain useful information, which, with modern technology, can be structured and stored, accessible to any who have the skills and need to make use of it. This information is increasingly profitable. However, with the increasing ability to capture and store data from many disparate sources, the need to store larger volumes of the data is likewise an increasing issue when it comes time to access and apply use, as many of the data gathered exist across very dynamic, diverse, and large partitions. As such, the scalability of storing and accessing big data must increase with it, relevant to the structures and locations of these data repositories. Developing a database in an ecosystem - Hadoop - that supports tools of two major concepts for achieving scalability within big data - data-parallelism and task-parallelism - our team has built a data warehousing solution to structure and store the data based on both optimized and normalized schema designs, drafted from entity-relationship models designed with the intention of enabling rapid storing and accessing through the Hadoop MapReduce process. Through a combination of these systems and data storage within Amazon Simple Storage Service (S3) and Apache Hadoop's native Hadoop Distributed File System (HDFS), we analyze performance between two approaches toward data- and task-parallelism using data gathered from the stock market.

A. Apache Hadoop Ecosystem

Motivated through the opportunity within big data and parallel computing, which enables massive amounts of data to be rapidly accessed for complex analysis and distributed across a scalable, cost-effective distribution of servers, we aligned our project with a modern database application, Hadoop, which provides a data lake "ecosystem" supporting both task- and data-parallelism across the many software applications within the Apache suite in addition to software that can be managed and accessed within a network of servers, called a cluster.

B. Hadoop MapReduce

The cluster of server nodes enables users to read data from the same source, simultaneously, as the data at that source is partitioned and processed across multiple servers – a master and at least one slave – through leveraging a processing framework called MapReduce to assign nodes for "mapping" and "reducing" by applying various configurations, such as related to memory allocation to and volumes of mappers and reducers. As data is mapped, it is reprocessed into a derivative data set, split into tuples that are then processed across the cluster of server nodes, in parallel, and reassembled in the reduce process from which it is delivered to the end-user, whether it be Enterprise Resource Planning (ERP) or a personal user running a SQL "SELECT" statement.

C. Application Integration and Data Ingestion

The parallelizability of Hadoop is central to this study as the primary objective is to rapidly store and access financial market data for buy-sell decision-making. The scope of applied analysis in this study is focused on the ability to scale and process a combination of quantitative stock market data and qualitative Twitter data related to the quantitative data.

Quantitative data was gathered from a markets data vendor through an Application Programming Interface (API) on 15-minute intervals using R programming language. Qualitative

data was gathered and processed through a Twitter API using Python programming language. The data, structured and stored into a Hive data warehouse is created and managed using Hive Query Language (HQL) stored both locally and within an Amazon S3 storage bucket. R will be used via Open Database Connectivity (ODBC) to access and build proof-of-concept models using the data. Once developed, Python will be deployed within the data lake to process and derive predictive data through machine learning decisions.

D. Big Data System Implementation

In order to provide manageable storage repositories that scale well to size and data diversity, we have implemented our solution using S3 and HDFS. S3 and HDFS are designed for large sets of data. Therefore, integrity of data and ingestion systems are able to be well maintained. This is essential in an environment that may need to support many simultaneous users, each with different data needs. The HDFS implementation in this project is housed across three - one master and two slave - Amazon EC2 servers whereas S3 is a standalone repository within Amazon's cloud suite.

Additionally, Apache Hive is used as a data warehouse because it is operated with HQL, which is easily communicable for SQL users. Furthermore, Hive allows for storage of massive databases tables and is well-suited for big data application integration, including the Apache software suite, of which Hadoop is a product. The data warehouse graphical user interface is provided by Cloudera Hue.

E. Data Warehouse Schema and Selection

Two schemas were designed for the warehouse in a star configuration. This first schema was designed in a full normalized fashion, which is known as a snowflake schema. The second schema was based on the snowflake schema, but denormalized to limit the number of tables, limits number of joins required in queries. Query performance will be measured on these schemas to determine which design will be better for this use case.

F. Performance Metrics and Evaluation

Performance is based on speed taken to process our 3 gigabytes of data - once processed into their respective storage systems - into the Hive data warehouse, which includes table creation and loading. We will use a repeated measures analysis and a one-tailed hypothesis test to determine optimal performance among S3 and HDFS groups, which is then used to compare and assess benchmarks between the combinations of MapReduce settings and database schemas among the best-performing S3 and HDFS configuration.

II. Data

This study investigates data warehouse models for housing stock price data and semi-structured alternative data. Both daily and intraday stock price data were collected for use in this analysis. Twitter was chosen as the primary source of alternative data because of ease-of-access to the Twitter API and the large volume of available data. The main features of the collected data are summarized in Table I.

TABLE I
Data Features

Source	Features
Stock Daily	Prices: High, Low, Open, Close
Stock Intraday	Prices: High, Low, Open, Close
	High Bollinger bands
	Mid Bollinger bands
	Low Bollinger bands
	Nominal moving average
	Historical moving average
	Signal moving average
	Exponential moving average
	Stochastic 5-day indicator (Slow K)
	Stochastic 3-day indicator (Slow D)
Tweets	Text, URLs, Hashtags, Mentions, Users

A. Stock Data

The stock price data was collected through an API provided by Alpha Vantage. This API provided access intraday stock prices, intraday price features, and daily prices. Intraday stock data contained 35 features sampled at 15 minute intervals. The intraday features were from the following categories Bollinger bands, stochastic oscillators, moving averages, and exponential moving averages.

Approximately 1 million rows of data were collected, which occurred from Oct. 04, 2019 to Oct. 24, 2019. During the collection process, the data were recorded in files organized by day and category and stored in an S3 bucket. At the end of the collection process, the files for each category were combined and pushed to the HDFS datalake.

B. Twitter Data

Messages on Twitter (called Tweets) are mainly comprised of tweet ID, timestamp, author (screen name), and text. Tweets can also contain many other features such as URLs, hashtags, emojis, and mentions. For this analysis, in addition to the main features, URLs, hashtags, and mentions were also collected in the data warehouse. A mention is a reference to another Twitter user's screen name in the text of a tweet. A hashtag is some collection of characters starting with '#' without white space. Generally, the character portion of a hashtag will be a word or set of words, but this is not necessary.

Approximately 300,000 tweets were collected from over 100 twitter users. The data from Twitter is returned in JavaScript Object Notation (JSON). The features of interest were extracted from the JSON files and reformatted in tab separated files (TSV). After the tweets were extracted from Twitter, mentions of company names or stock symbols were extracted from the tweet text by matching sections of tweet text to company names and company stock symbols. Once the data was collected an processed, it was pushed to the HDFS datalake.

III. Data Warehouse Development

Data warehouses are often conceptually designed with a *star* schema [1]. In the star design, there is a central table (called

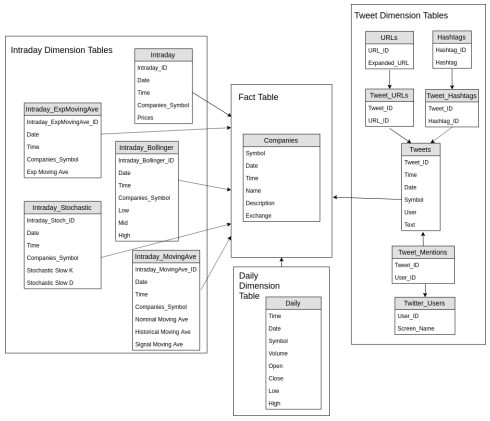

Fig. 1. Conceptual Diagram of the Data Warehouse Snowflake Schema

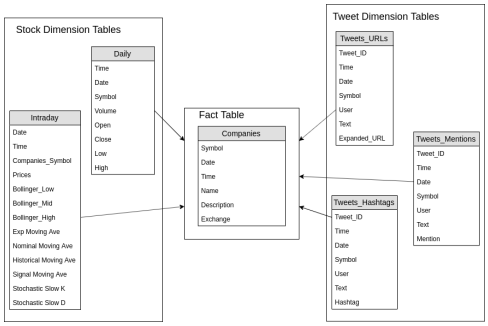

Fig. 2. Conceptual Diagram of the Data Warehouse Star Schema

the fact table), which contains the unifying features of the dataset and keys to other tables [1]. The tables surrounding the fact table (called dimension tables) contain information related to a category related to the unifying feature of the dataset [1]. In the dataset for this analysis, the unifying features are timestamp and company stock symbol, thus the fact table contains these features and several other descriptive features related to the companies. The natural dimensions of the fact table are intraday stock data, daily stock data, and tweet data. In some cases, dimensions of the facts exhibit an inherent hierarchical structure. When the dimensions of a star schema are broken out into the tables that make up the hierarchical structure of the dimensions, the schema is in its *snowflake* form [1]. In this study, two schemas were designed: one in snowflake form and the other in a more optimal form for query speed.

A. Snowflake Schema

As noted previously, the fact table of this star design contained the company information and timestamps. The primary key of the fact table is a composite key that includes stock symbol and timestamp. There are three natural dimensions of fact table: daily stock data, intraday stock data, and tweet data.

Both daily and intraday stock data are naturally keyed by the combination of timestamp and stock symbol, thus naturally keyed to the fact table. The stock data was spilt into two dimensions: daily data and intraday data. The daily data comes in a form suitable for the snowflake design. However, the intraday data was normalized into five tables for the snowflake design: prices, Bollinger bands, moving averages, exponential moving averages, and stochastic indicators. The integration of the fact table and the normalized stock table is shown conceptually in Fig. 1 (left side of the schema diagram).

Unlike the stock data, the tweet data did not naturally join to the fact table of the schema. Like the stock data, the tweet data came with a timestamp, but stock symbol is not nominal feature of tweets. The stock symbol feature was generated by extracting matching strings during the data collection process; therefore, stock symbols are not guaranteed to be non-null in the tweet dimension. Additionally, the combination of timestamp and stock symbol is not guaranteed to be a primary key into the tweet table for tweets with non-null stock symbol fields. However, Twitter assigns a tweet ID for each tweet, which is guaranteed to be a primary key. Thus, the same features can still be used to join the tweets dimension to the fact table, but cannot serve as a primary key. As noted previously, three secondary features of tweets are used in this study. Each of these features, mentions, hashtags, and URLs, represents a hierarchical member of the tweet dimension. Tables were created for each of these features to follow the snowflake design. These additional tables required join tables because there is a many-to-many cardinality between the tweets dimension and each of the three secondary members. The result of the tweet dimension normalization is shown conceptually in Fig. 1 (right side of schema diagram).

B. Denormalized Star Schema

While the snowflake schema is suitable for general use cases. The number of tables should be limited to decrease query time to support fast data transfer to a machine learning system. The number of intraday tables and tweet tables were reduced to support fast query times. All intraday tables were combined into one table. Each secondary member of the twitter dimension hierarchical structure was subsumed into a copy of the main tweet table, reducing the number of tables in the tweet dimension from seven to three. The result of this denomalization is shown conceptually in Fig 2.

IV. Technologies

Rikel

V. Results

Maybe a table comparing read times for the different options

- S3
- HDFS
- HDFS mapreduce settings?

VI. Analysis

Analysis of the results

VII. Conclusion

EXAMPLE CONCLUSION: The scalability of large sets of data is of ever-increasing importance in the data community, which itself is ever-increasing with the advancement of modern technology, which is enabling both an increase in both data volume as well as data diversity being captured and stored. Consequently, parallelism is of utmost importance in making use of this data. Herein analyzed are methods that are proven within the scope of big data. As modern technology continues to advance, these parallel principals will serve as the root from which methodologies will be developed. As with the businesses and industries the data captured will serve, there will remain a trade-off that must be assessed for each business model. This study has provided two primary storage methods - S3 and HDFS - in addition to different data warehousing schema design - normalized versus optimized, star versus snowflake - and data distribution - herein, MapReduce - techniques. Through analysis using repeated measures, S3 outperforms HDFS when X, Y, Z and HDFS outperforms S3 when Z, Y, X. Furthermore, MapReduce configurations are constant across both storage methods - HDFS and S3 - with an optimized star schema performing faster, but a normalized snowflake schema performing more reliably within a conventional three-tiered system architecture.

References

[1] W. H. Inmon, "Unstructured Data and the Data Warehouse," in *Building the Data Warehouse*, 4th ed. Hoboken: Wiley, 2005, ch. x, sec. x. Accessed on Nov. 6, 2019 [Online]. Available: https://learning.oreilly.com/library/view/building-the-data/9780764599446

[2] I. Moalla, A. Nabli, L. Bouzguendam and M. Hammami, "Data warehouse design approaches from social media: review and comparison," Social Network Analysis and Mining., Vol. 7, no. 1, pp. 1-14, Jan. 2017. Accessed on: Nov. 6, 2019 [Online]. Available doi: 10.1007/s13278-017-0423-8