A Datalake Solution for Time Series Data

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

***Abstract*—In this paper, we present a solution for warehousing structured and semi-structured financial markets time series data. Traditional systems, such as SQL databases, are well suited for storing highly structured data, but were not designed to store semi-structured forms of data. Additionally, traditional systems, which operate with ACID transactions, can be slow over large volumes of data. "Big Data" systems were developed to store this type of data, addressing this gap in data storage system technology. We developed a system to store structured and semi-structured time series data based on the Apache Hadoop ecosystem. Based on the results of testing read query performance, schema denormalization and increasing the Hadoop cluster size will reduce query times. A distributed data warehouse framework like Hadoop could be used in cases where data scaling is expected, requiring the computing infrastructure to scale with the data.**

I. Introduction

The ability to capture and generate large volumes of data is an issue many enterprises face. Traditionally, data generation systems were slow and the data was easily structured, which lent to the use of traditional systems. However, "Big Data" is often semi-structured or unstructured and can be generated at high rates, such as server logs or real-time sensor data [5]. "Big Data" technologies were developed to address the gap between the current type of data that are generated and collected and capacity of traditional storage tools [1]. These "Big Data" technologies are able to collect data in raw form and provide access to the data, often using languages similar to Structured Query Language (SQL).

An example of "Big Data" is financial market data and related "alternative" data that may be useful for prediction. For alternative data to be useful to a prediction system, it must be stored in a system that can provide low query time. Additionally, it must be able to work with data that does not follow strict structuring, such as text-based messages or images. In financial market modeling, alternative data is any type of data outside the financial domain [18]. An application of financial market prediction using alternative data is utilizing Twitter data to make predictions on the stock market [19]. A data storage system would need to store large volumes of time series data and provide rapid query access to those volumes to support a financial market prediction engine. In this paper, we will present a "Big Data" system that can be used to store and access large volumes of time series data. Stock data, intraday and daily, will be the primary time series used in the analysis. Data from Twitter will serve as the primary source of alternative data. For this analysis, we collected approximately four-hundred thousand tweets from over one-hundred Twitter accounts and approximately 49 million stock data points.

The Hadoop ecosystem is at the core of this system. Hadoop is an open source framework for distributed computing that is sponsored by the Apache Software Foundation[1] [13]. Since Hadoop is a distributed computing framework, the server cluster used in the implementation can be horizontally scaled to accommodate the need for more computing power as the data storage grows. The Hadoop ecosystem consists of modules for file storage, data warehousing, data flow, data analysis, and machine learning. The primary components of Hadoop used in this system are Hadoop Distributed File System (HDFS) and Hive.

HDFS is a file system distributed over multiple servers (a server cluster) [13]. The processing framework that enables this is called MapReduce, which assigns nodes for "mapping" and "reducing" processes by applying various configurations, such as related to memory allocation and numbers of mappers and reducers [11]. As data is mapped, it is reprocessed into a derivative data set, split into tuples that are then processed across the cluster, in parallel, and reassembled in the reduce process [11]. Hive is a distributed data warehouse that manages data stored in HDFS [13]. Hive provides an SQL-like query language to access and retrieve the data stored in HDFS.

We implemented the solution with Amazon Web Services (AWS), which provides a set of cloud services for data storage and warehousing. Elastic MapReduce (EMR) is a preconfigured service for Hadoop provided by AWS. EMR runs on a cluster of Elastic Compute 2 (EC2) servers, which can be horizontally scaled as needed. We used EMR for the physical implementation of the Hadoop ecosystem in this project.

We designed two schemas for the data warehouse in a star configuration. The first schema was designed in a fully normalized fashion, which is known as a snowflake schema. Our second schema was based on the snowflake schema, but denormalized to limit the number of tables, which limits the number of joins required in queries. Query performance will be measured on these schemas to determine how much improvement in query time is provided by schema denormalziation.

We will test the performance of this system implementation with the collected stock and Twitter data. Performance of the data warehouse is based on time taken to process the data queried in the Hive warehouse. The analysis will examine the improvement in performance from the schema denormalization and increasing the EMR cluster size. We will use a two-way analysis of variance (ANOVA) to compare the differences in combinations of schema and EMR cluster size.

[1] https://hadoop.apache.org/

II. Data

We collected stock data and Twitter data for this study. Both daily and intraday stock price data were collected for use in this analysis. Twitter[2] was chosen as the primary source of alternative data because of ease-of-access to the Twitter API[3] and the large volume of available data. Categories of data are summarized in Table I and a full list of features is given in Appendix A.

A. Stock Data

The stock price data was collected through an API provided by Alpha Vantage[4]. The R[5] programming language was used to collect data from the Alpha Vantage API, using the R package alphavantager[6] [14]. This API provided access to daily prices, intraday stock prices, and intraday price features from the following associated categories: Bollinger bands, stochastic oscillators, moving averages, and exponential moving averages.

The stock data was collected from Oct. 04, 2019 to Oct. 24, 2019 on all 2520 symbols traded on the NYSE and all 3213 symbols traded on the NASDAQ. A single value was stored for each feature of the daily prices. The intraday values were sampled at 15-minute intervals from market open to market close. This resulted in a total of 49,093,917 stock data points.

B. Twitter Data

Messages on Twitter (called tweets) are mainly comprised of tweet ID, timestamp, user (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. An example of a tweet containing one hashtag (`#FF`) and one mention (`@everett`) is given in Listing 1 (the value of '`full_text`').

We collected 423,802 tweets from 108 Twitter users, using the Python[7] module Twython[8]. The data from Twitter is returned in JavaScript Object Notation (JSON); an example is shown in Listing 1. The features of interest were extracted from the JSON files and reformatted in tab separated files (TSV) using Python and the built-in JSON and CSV modules. 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. All other featured were extracted from the JSON, by key.

[2]https://www.twitter.com/
[3]https://developer.twitter.com/en/docs
[4]https://www.alphavantage.co/
[5]Version 3.6.1
[6]https://cran.r-project.org/web/packages/alphavantager/index.html
[7]Version 3.6.8
[8]https://pypi.org/project/twython/3.6.0/

TABLE I
Data Categories

Source	Features
Stock Daily	Symbol
	Time Stamp
	Prices
Stock Intraday	Symbol
	Time Stamp
	Prices
	Bollinger Bands
	Moving Averages
	Stochastic Indicators
Twitter	Symbol
	Time Stamp
	User
	Text
	URLs
	Hashtags
	Mentions

Listing 1. Sample Tweet JSON

```
{'created_at': 'Fri Oct 25 07:44:38 +0000
    2019',
 'id': 1187636067505188864,
 'id_str': '1187636067505188864',
 'full_text': '@everett where do I start
    with this company? #FF!!!!',
 'truncated': False,
 'display_text_range': [0, 52],
 'entities': {
   'hashtags': [
            {'text': 'FF',
             'indices': [45, 48]}],
   'symbols': [],
   'user_mentions': [
          {'screen_name': 'even_everett',
           'name': 'Even Everett',
           'id': 21841004,
           'id_str': '21841004',
           'indices': [0, 10]}],
   'urls': [
     "indices": [32, 52],
     "url": "http://t.co/TZR",
     "display_url": "you.com/watch?v=
         oHg ",
     "expanded_url": "http://www.you.com/
         watch?v=TZR"]}
 'user': {'id': 14216123,
   'id_str': '14216123',
   'name': 'Jim Jim',
   'screen_name': 'jimjim',
   ...}
}
```

As shown in Listing 1, the timestamp, text, and tweet ID can be accessed at the top level of the JSON object with `created_at`, `full_text`, and `id_str`, respectively. The

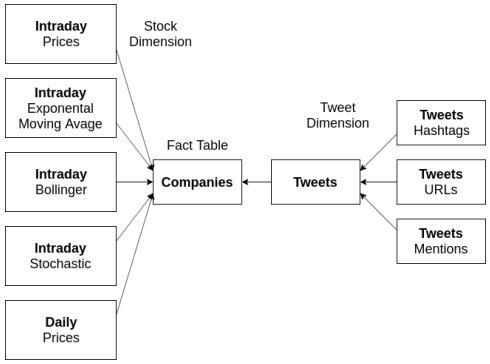

Fig. 1. Conceptual Diagram of the Data Warehouse Snowflake Schema

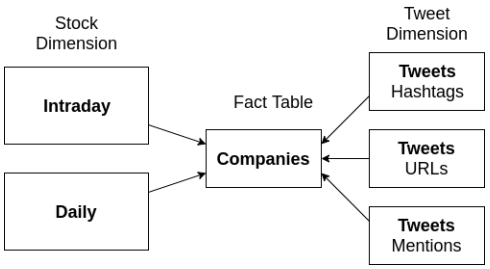

Fig. 2. Conceptual Diagram of the Data Warehouse Denormalized Star Schema

user information can be accessed from the key `user`, which accesses another set of objects. The values for `screen_name` and `id_str` were extracted for the user. The hashtags, mentions, and URLs are located under the key `entities` as `hashtags`, `user_mentions`, and `urls`, respectively. Each of these keys under `entities`, return an array of objects, containing the features of interest. The value for `text` under `hashtags` was the only value collected for the hashtags (`#FF` in the example). The values for `screen_name` and `id_str` under `user_mentions` were collected for mentions (`even_everett` and `21841004`, respectively, in the example). Only the value of `expanded_url` was collected for `urls` (`http://www.you.com/watch?v=TZR` in the example).

III. Data Warehouse Development

Data warehouses are often designed with a *star* schema [2]. In a star schema design, there is a central table (called the fact table), which contains the unifying features of the dataset and keys to other tables [2]. The tables surrounding the fact table (called dimension tables) contain information related to a category of the facts [4]. In the dataset for this analysis, the unifying features are timestamp and company stock symbol. The combination of timestamp and symbol is a natural key into the stock data and the twitter data. These features were included on the fact table as the primary key. The company name and exchange market associated with the symbols were included on the fact table for convenience. The natural dimensions of the fact table are intraday stock data, daily stock data, and tweet data.

In some cases, dimensions can be normalized into multiple tables. When the dimensions of a star schema are are normalized, the schema is in its *snowflake* form [3]. In this study, two schemas were designed: one in snowflake form and the other in a denormalized form. We will consider a star schema in snowflake form when it is normalized into third normal form (3NF), meaning the domains of the attributes are atomic and there are no functional dependencies on a portion of the primary key [17].

A. Snowflake Schema

As noted previously, the fact table of this star design contained the company symbols and timestamps as a composite primary key. The three dimensions of the fact table, daily stock data, intraday stock data, and tweet data, share the primary key of the fact table.

The stock data comes in a form suitable for the snowflake design; it is already in 3NF. However, the intraday data was split into five tables for a more general use schema design: prices, Bollinger bands, moving averages, and stochastic indicators. The daily stock table and the intraday stock tables are shown joining to the fact table in Fig. 1 (left side of the schema diagram).

Like the stock data, the tweet data came with a timestamp, but stock symbol is not a 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. Thus, the same features can still be used to join the tweet dimension to the fact table, but cannot serve as a primary key. However, Twitter assigns a tweet ID for each tweet, which is guaranteed to be a primary key. The tweet ID was used as the primary key of the tweet table.

As noted previously, three secondary features of tweets are used in this study. Each of these features, mentions, hashtags, and URLs, is a normalizable feature of the tweet dimension. To achieve a 3NF design, the unique attributes of the tweets were collected in a central table and tables were created for mentions, hashtags, and URLs. Since there is a many-to-many cardinality between the central tweet table and each of the three secondary members, join tables were created to connect the secondary tables to the central tweet table. 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 read 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 normalized tweet dimension was subsumed into a copy of the main tweet table, reducing the number of tables in the tweet dimension from seven to three. The schema resulting from this denormalization process is shown conceptually in Fig. 2.

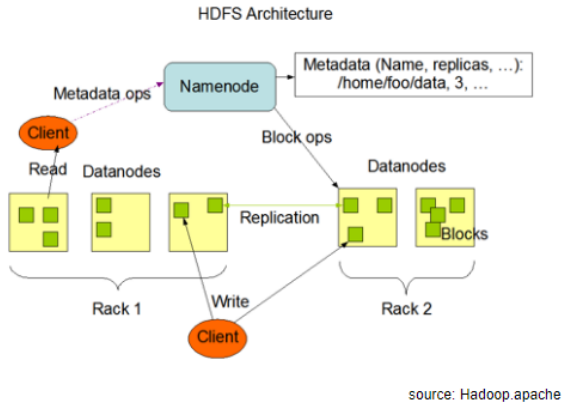

Fig. 3. HDFS Architecture [7]

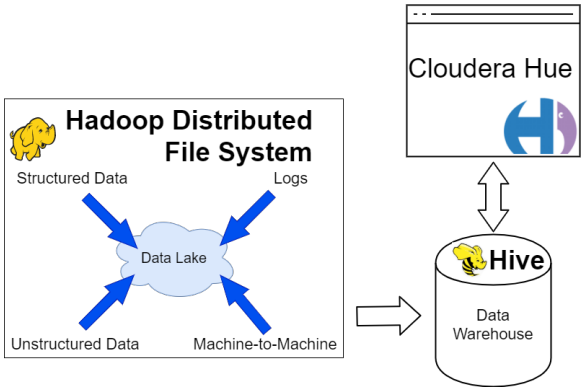

Fig. 4. Overview of Data Warehouse System

IV. Implementation

Multiple vendors provide platforms and infrastructure for data warehousing technologies, this project was implemented using Amazon Web Services (AWS). AWS is a set of cloud computing services provided by Amazon.com[9] that are accessible over the internet. AWS provides multiple services for many different applications. Elastic MapReduce (EMR) was used for the implementation of Hadoop in this project.

A. AWS Elastic MapReduce

AWS EMR is a pre-configured compute cluster for Big Data. The cluster can be provisioned and terminated on demand as needed. It comes with a configurable set of the Hadoop ecosystem elements pre-installed and ready to use. The EMR cluster used in this study was provisioned with three m5.xlarge[10] elastic compute instances using version 5.27.0 of the EMR software[11].

B. Apache Hadoop

Hadoop is an open source software framework for distributed storage and distributed processing of very large datasets. The core of Apache Hadoop consists of a storage part - Hadoop Distributed File System (HDFS) - and a processing part - Hadoop MapReduce.

1) Hadoop Distributed File System (HDFS) and MapReduce: Unlike traditional file systems, the Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware. HDFS architecture consists of a master node, called a "NameNode" and at least one slave node, called "DataNodes." [13]. The NameNode is the controller which manages storing data and providing metadata of the data to the DataNodes. The metadata includes a Namespace lookup table used to locate each file from the Datanodes, which assist in node cluster-computing.

HDFS works by storing duplicate copies of data blocks across multiple servers. Those blocks are then split into chunks, which are then replicated across the cluster, partitioned by server rack. Data jobs take advantage of this feature by breaking down into tasks, where each task pairs with blocks of data and are then processed across the cluster. If one job fails, it can be restarted by the other running tasks until all tasks have been completed and data compiled for the end-user [6].

In reference to Fig. 3, the client - in our case, Hive, but this can be a different application supported by the Hadoop ecosystem - requests a read operation by passing blocks of HDFS files from Datanodes in server Rack 1 with their associated metadata to the Namenode, which then delegates block processing operations to the Datanodes in serverRack 2. Tasks are assigned from the Namenode and partitioned across the racks of servers. Although all data is in both racks - because write operations replicate data across the server racks, as indicated in Fig. 3 - each server rack has unique tasks to operate. As mentioned, in the event a task fails in a node, another node will resume the task since the data is duplicated [6].

2) Hadoop MapReduce: Hadoop MapReduce is a programming model for large-scale data processing. With MapReduce, instead of using one server instance with multiple processors, multiple servers with multiple processors are used for computation. This is refered to as a distributed computing system.

MapReduce uses the "divide and conquer technique" where the input is divided into a set of small tasks. Each task is identified by a key-value pair [9]. The key serves as a task ID and the value as the task output. Each task is then processed and executed by the mapper and the outputs are processed and merged by the reducer [10]. MapReduce couples with HDFS to enable horizontal scaling in a cluster-computing environment.

C. AWS Simple Storage Service

As the name implies, the S3 is a storage system. It is used to store and retrieve any amount of data any time, from anywhere on the web. S3 is reliable, fast, and inexpensive. We used this to share data and load directly into HDFS.

[9]https://aws.amazon.com/

[10]https://aws.amazon.com/ec2/instance-types/m5/

[11]emr-5.27.0 contains Amazon Hadoop 2.8.5, Hive 2.3.5, and Hue 4.4.0. See https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-release-5x.html

TABLE II
READ QUERY RESULTS

Schema	Cluster Nodes	Query Time (seconds)	
		Mean	Standard Deviation
Snowflake	3	121.05	4.12
	5	94.49	4.01
Denormalized Star	3	103.77	2.99
	5	79.68	5.86

D. Apache Hive

Apache Hive is a data warehouse application built on top of Hadoop. We used Hive to structure, organize, model, and query our data using the Hive Query Language, HQL. We structured files from within the Hadoop Distributed File System (HDFS) and loaded them into tables within Hive. Hive operates by using the HDFS storage to generate tables. However, when Hive queries are executed, the data is retrieved from HDFS via the compiler, which uses an execution engine and metastore to return results [12].

E. Cloudera Hue

Cloudera Hue[12] is an open source web application that served as user interface for Hadoop components. Hue provides access to Hadoop from within a browser, allowing users to interact with the Hadoop ecosystem applications. Hue is an alternative to accessing the Hadoop ecosystem applications from the command line interface. Hue was used to interact with the EMR cluster and run Hive scripts.

F. System Design

Once fully configured, the system we developed included three- or five-node EMR clusters. Data collected by members of our team were stored in Amazon S3. The data was loaded into HDFS from S3 prior to creating structured tables in Hive. Local hosts would connect to the Namenode master server by SSH to allow interfacing with Hue. Hive tasks would be executed on the EMR cluster from the Hue interfaces to create and load the data warehouse (as shown in Fig. 4).

V. STUDY DESIGN

In this study, we will investigate the impact of schema normalization and EMR cluster size on the performance of a read query. Both schemas presented in Section III were tested on EMR clusters with 3 nodes and 5 nodes. This is a two-way ANOVA where ERM cluster size and schema are the explanatory factor variables and the response variable is query execution time [15].

During performance testing, we gathered all fields from all tables, joining all tables in the schema together. Joining all tables enabled capturing the entirety of the database, under each schema, to produce the same results. These queries were limited to 75,000 returned records to enable gathering more runtime samples. This number was decided to be reasonable based on the difference in performance time. As noted in the analysis section below, the performance time between queries was significant at 75,000 records; limited a higher volume of returned records would have increased the significance. Therefore, 75,000 was determined reasonable for analysis.

[12]https://gethue.com/

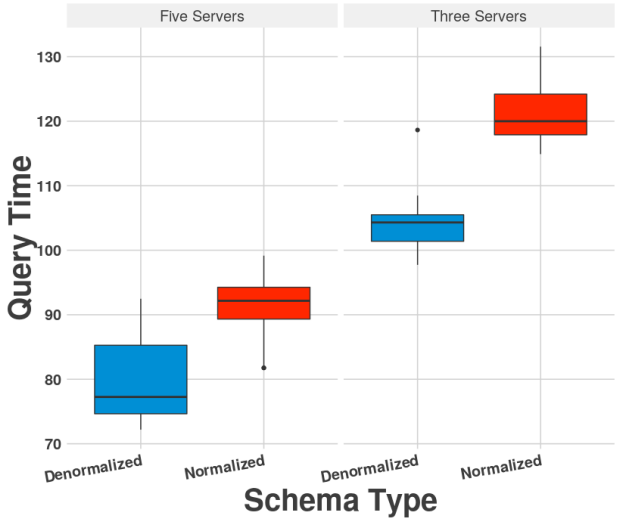

Fig. 5. Average of Query Time by Schema and Block Size with Standard Deviation Bars

VI. RESULTS

As discussed in Section V, a two-way ANOVA test was performed. We identified that the interaction between schema design and server count provide significant results, indicating that there is a difference in the proportions of performance speed between the two schemas when considering count of servers in a cluster. The dependence of performance on combinations of schema and cluster size is apparent in Fig. 5, which shows boxplots of the query times of each combination of schema and cluster size. The red boxes represent the query times for the snowflake schema and the blue boxes represent the query times for the denormalized schema. The boxplot is faceted by cluster size with the three-node results on the right and the five-node results on the left. The query time of the denormalized schema is lower than the normalized schema for both sizes of clusters.

We find that, while holding cluster size constant, schema design is significant on performance across both server sizes (p-value < 0.0001, F $= 999.187$). Denormalizing the snowflake schema provided a 14.27% decrease in mean query time on the three-node cluster and a 15.67% decrease mean in query time on the five-node cluster. Furthermore, while holding schema design constant, performance speed difference was also significant (p-value < 0.0001, F $= 3399.954$) between the two cluster sizes with the five-node cluster outperforming the three-node cluster. A summary of the results is shown in Table II.

VII. ANALYSIS

In analyzing the exploratory graphical analysis of the clusters, the interaction between schema and cluster size is such

that as cluster size increases, the normalized schema approaches a similar level of performance. However, testing indicates there is still a statistically significant difference between performance at the five-server cluster level. Further testing could identify that, while holding database size constant, at some server level, the normalized schema may outperform the de-normalized schema. However, this is based on linear approximation that may become non-linear with more samples and clusters. Regardless, under both cluster sizes, the de-normalized schema outperformed the normalized schema by a level of statistical significance.

VIII. Conclusions

With the increase in data collection activities, data storage systems will need to provide results of read queries on large datasets. This analysis shows that query times of distributed data warehouses can be reduced by denormalizing the schema or scaling up the number of servers in the physical implementation. While schema denormalization does provide improvements for read queries, migration to a new schema may not be possible or may introduce other undesirable effects because the data definition and manipulation logic will be affected. Since the logical design of the warehouse is abstracted from the physical implementation, increasing cluster size would be a better solution. If read performance improvement on a distributed data warehousing system is needed, we recommend increasing the cluster size of the physical system.

Appendix A
Data Features

This section contains a full listing of all the features collected for this study. The features are listed by general category as shown in Table I.

A. Daily Stock Features

The daily stock features are described below and are organized in the following categories: Key Values and Prices. The value for time was imputed to a standard value for the daily values.

1) Key Values:

- Symbol
- Time
- Date

2) Prices:

- Open
- Close
- High
- Low

B. Intraday Stock Features

The intraday stock features are described below and are organized in the following categories: Key Values, Prices, Bollenger Bands, Moving Averages, and Stochastic Indicators.

1) Key Values:

- Symbol
- Time
- Date

2) Prices:

- Open
- Close
- High
- Low

3) Bollenger Bands:

- Open Lower Band
- Open Middle Band
- Open Upper Band
- Close Lower Band
- Close Middle Band
- Close Upper Band
- High Lower Band
- High Middle Band
- High Upper Band
- Low Lower Band
- Low Middle Band
- Low Upper Band

4) Moving Averages:

- Open Convergence Divergence
- Open Convergence Divergence Signal
- Open Convergence Divergence Historical
- Open Exponential
- Close Convergence Divergence
- Close Convergence Divergence Signal
- Close Convergence Divergence Historical
- Close Exponential
- High Convergence Divergence
- High Convergence Divergence Signal
- High Convergence Divergence Historical
- High Exponential
- Low Convergence Divergence
- Low Convergence Divergence Signal
- Low Convergence Divergence Historical
- Low Exponential

5) Stochastic Indicators:

- 5-day indicator
- 3-day indicator

C. Twitter Features

The twitter features are described below and are organized in the following categories: Unique and Common. Unique features are specific to a particular tweet and common features are shared across many tweets.

1) Unique:

- Tweet ID
- Timestamp
- User (Author)
- Text

2) Common:

- Hashtags
- URLs
- User Mentions

REFERENCES

[1] R. Kune, P. Konugurthi, A. Agarwal, R. Chillarige, R. Buyya, "The Anatomy of Big Data Computing," Software: Practice and Experience, Vol. 46 no. 1, pp.79-105, Jan. 2016.

[2] W. H. Inmon, "Unstructured Data and the Data Warehouse," in *Building the Data Warehouse*, 4th ed. Hoboken: Wiley, 2005, ch. 11. Accessed on Nov. 6, 2019 [Online]. Available: https://learning.oreilly.com/library/view/building-the-data/9780764599446

[3] 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

[4] A. Gorelik, "Historical Perspectives," in *The Enterprise Big Data Lake*, 1st ed. Sebastopol, CA: Wiley, 2019, ch. 2, pp. 25-47.

[5] "Extract, Transform, and Load Big Data with Apache Hadoop," Intel, USA, 2013. Available: https://software.intel.com/sites/default/files/article/402274/etl-big-data-with-hadoop.pdf

[6] S. Alapati, Expert Hadoop Administration, 1st ed. Boston, MA: Addison-Wesley, 2017.

[7] D. Borthakur, HDFS Architecture Guide, The Apache Software Foundation, August 22 2019. Accessed on: Nov. 8, 2019. [Online] Available: https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

[8] D. Sitaram, G. Manjunath *Moving To The Cloud*. Boston: Syngress, 2012, pp. 205253.

[9] S. Zhao, R. Li, W. Tian, W. Xiao, X. Dong, D. Liao, S. U. Khan, L. Keqin, Divide-and-conquer approach for solving singular value decomposition based on MapReduce, Abbrev. Title of Journal, vol. 28, no. 2, pp. 331350, Nov. 2016. Accessed on: Nov. 10, 2019 [Online]. Available: doi:10.1002/cpe.3436.

[10] D. Miner, A. Shook, MapReduce Design Patterns, 1st ed. Sebastopol, CA: O'Reilly Media, 2012, pp. 56.

[11] D. Miner, A. Shook, Hadoop MapReduce v2 Cookbook, 2nd ed. Sebastopol, CA: O'Reilly Media, 2015, pp. 1.

[12] H. Bansal, S. Chauhan, S. Mehrota, Apache Hive Cookbook, 1st ed. Birmingham, UK: Packt Publishing, Limited, 2016.

[13] T. White, Hadoop: The Definitive Guide, 1st ed. Sebastopol, CA: O'Reilly Media, 2009, pp. 44.

[14] R Core Team, R: A language and environment for Statistical Computing, Vienna, Austria: 2018. Available: https://www.R-project.org/

[15] M. Crawley, "Analysis of Variance," in *The R Book*, 1st ed. Chichester, West Sussex, United Kingdom: Wiley, 2013, ch. 11, pp. 470-478.

[16] K. Tannir, "Enhancing Map and Reduce Tasks," in *Optimizing Hadoop for MapReduce*, 1st ed. Birmingham, England: Packt Publishing Ltd, 2014, ch. 5. Accessed on: Nov. 20, 2019 [Online].

[17] A. Silberschatz, H. Korth, and S. Sudarshan, "Relational Database Design," in *Database Design Concepts*, 6th ed. NY: McGraw-Hill, 2011, ch. 8, sec. 8.3, pp. 329 - 338.

[18] C. Xiao and W. Chen, Trading the Twitter Sentiment with Reinforcement Learning, arXiv:1801.02243v1 [cs.AI], Jan. 2018. Accessed on: Dec. 7, 2019 [Online]. Available: https://arxiv.org/pdf/1801.02243.pdf.

[19] J. Bollen and H. Mao. "Twitter mood as a stock market predictor," Computer, Vol. 44, no. 10, pp. 9194, 2011. Accessed on: Dec. 7, 2019 [Online]. Available: doi: 10.1109/MC.2011.323