Top-k Equities Pricing Search in the large historical data set of NYSE

Ganesan Senthilvel
Roll No: CS15Z003
Research Scholar, Dept. of Computer Science, IIT Madras
+91-98408-97282
ganesan.senthilvel@gmail.com

ABSTRACT
In this paper, we describe the need to look into large data for analyzing the recorded financial activities in the stock exchange. The study of historic data is very crucial. As the size of the data huge we need better methods to retrieve data efficiently. Big Data Hadoop framework is leveraged to resolve this problem statement.

Categories and Subject Descriptors
D.3.3 [**Top-k Equities Pricing**]: Stock Exchange Equity Constructs and Features – *stock trade component, Historical Pricing, NYSE, Top-k search algorithm, shareholder, liquidity stock, dividend, gain/loss.* NYSE equity product page is available at: https://www.nyse.com/products/equities

Keywords
Stock Market, New York Stock Exchange, Equities, Top-k pricing, Historical data set.

1. INTRODUCTION
Stock exchange is a stock market where brokers and traders can buy and sell stocks. Stock exchange often functions as continuous auction markets. There are ups and downs in the stock market. There are possibility for an investor to experience losses due to many factors. We can reduce the loss by studying historic data prior to investing. It is not enough to just look at a stock's volatility from day by day.

2. PROBLEM STATEMENT
Our problem statement pertains to the area of Top-k pricing search from the equity's historical stock exchange data streams. In our case, we considered the popular New York Stock Exchange (NYSE).

On analyzing the stock exchange site, Top-k pricing search on equity market is not around. Though, other external sites tried to produce Top-k in terms of activity not by pricing. The Wall Street Journal http://www.wsj.com/mdc/public/page/2_3021-activnyse-actives.html. In this scenario, this project is beneficial to retrieve Top-k pricing result set based on user driven date.

3. DATA SET
3.1 Source
Our project is targeting the dataset from world's leading stock exchange – NYSE (New York Stock Exchange) Data is downloadable from the web site eoddata.com based on the subscription level.

3.2 Data Layout
Daily data feed is shipped in the various data formats; our target is CSV file. CSV (for historical stock data) file layout is as follows:

Table 1. Sample NYSE Equity

Ticker	Date	Open	High	Low	Close	Volume
IBM	2015 0728	159.91	160.19	158.50	160.05	272100

3.3 Equity
In general, the definition of equity can be represented with the accounting equation:

Equity = Assets - Liabilities

An equity investment generally refers to the buying and holding of shares of stock on a stock market by individuals and firms in anticipation of income from dividends and capital gains, as the value of the stock rises. Typically equity holders receive voting rights for the board of directors' selection.

3.4 References and Citations
Daily basis data is extracted from NYSE download site at http://eoddata.com/download.aspx .

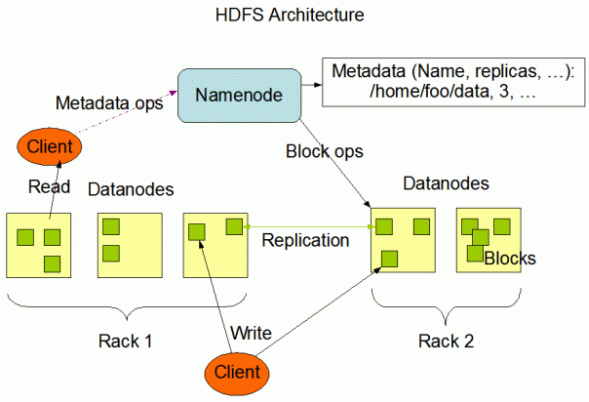
4. HADOOP FRAMEWORK
Hadoop is a framework written in Java for running applications on large clusters of commodity hardware and incorporates features similar to those of the Google File System (GFS) and of the Map Reduce computing paradigm.

Hadoop's HDFS is a highly fault-tolerant distributed file system and, like Hadoop in general, designed to be deployed on low-cost hardware. It provides high throughput access to application data and is suitable for applications that have large data sets. HDFS architecture is represented in the below diagram.

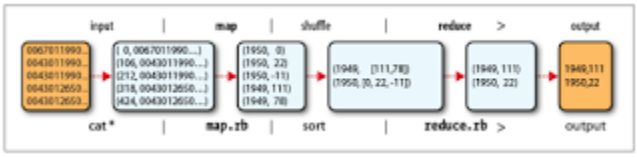
Computers are always driven by 2 key factors. They are (1) storage (2) process. In Hadoop eco system, HDFS is storage and Map Reduce is process for the highly distributed system.

NameNode - 1 per cluster. Meta data's centralized server to provide a global picture of the file system's state. NameNode stores its file system metadata on local file system disks in a few different files, but the two most important of which are fsimage and edits.

DataNode - Many per cluster. Stores block data (contents of files). Data nodes regularly report their status to the NameNode in a heartbeat mode; default 3 mins. It sends Block Report (list of all usable blocks of DataNode disks) to NameNode; default 60 mins.



On submission of the job by the User, Hadoop initiates the Job Tracker process at Master Node. Internally, the execution undergoes 3 major tasks/steps (1) Map (2) Shuffle & Sort (3) Reduce as depicted in the sample.



5. APPROACH

Top-k query processing plays an important role in data retrieval to give an answer to a user quickly. To our best knowledge, the traditional top-k query processing works with a local database. As the size of a database is lager, the database is stored in a distributed network, and it requires the parallel processing. We address a parallel top-k query processing using Google's Map Reduce programming model.

Our experiment with the synthetic datasets, is derived from the authentic historic data from NYSE. So, the financial instrument-equity and its related data are meaningful.

Key advantage on leveraging Map Reduce framework is not only parallelizable processing but also handling large data sets, cluster of processing nodes and scaling out model for elastic computing.

5.1 Strategy

The strategy is to go through the list once, and as you go, keep a list of the top k elements that you found so far. To do this efficiently, you have to always know the smallest element in this top-k, so you can possibly replace it with one that is larger. The heap structure makes it easy to maintain this list without wasting any effort. It is like a lazy family member who always does the absolute minimum amount of work. It only does enough of the sort to find the smallest element, and that is why it is fast.

5.2 Parallel Algorithm

5.2.1 Top-k Query Processing

A top-k query finds the top-k answers with the highest grades on the given query. In the recent years, many assorted algorithms have been proposed to support these top k queries. In this project, we evaluate the existing algorithms for top-k queries using disk based hive and in memory Impala search methodologies.

5.2.2 Role of MapReduce Algorithm

Our approach adopts Google's MapReduce as the parallel programming framework for the top-k query processing. MapReduce gives a programmer simple interfaces for parallel programming, that is, the programmer does not need to consider the parallel issues. Top-k query processing algorithm aims to minimize the number of probing predicates. The main contribution of this work is to design a parallel top-k query processing model using MapReduce. In this process, we evaluate the algorithms through experiments and propose a new approach to improve its performance. The signatures of map and reduce are as follows:

map (k1, v1) → list (k2, v2)

reduce (k2, list (v2)) → list (v3)

5.2.3 Parallel Top-k Processing

In this section, the idea is to parallelize top-k query processing using MapReduce.

We propose an approximation algorithm which estimates the number of top objects to be drawn in each node. The idea is the same with the above ideal case; we want to find only necessary top objects. In order to do that, we can estimate the number of top objects from the result set. We compress a large database to a small database using wavelet and conduct data parallel query processing with the small database. From the result, we identify the ratio of top-k objects in each node and conduct the query processing with the large database and the ratio. That is, we retrieve the different number of necessary top objects in each node according to the ratio.

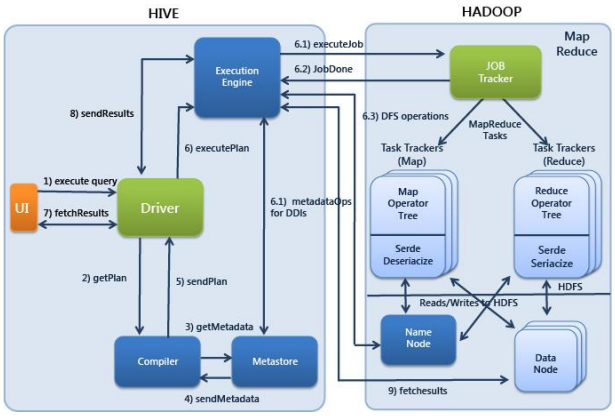
6. Execution

6.1 Cluster Setup

The experiments run on a 4-node cluster with 4 cores, 4GB of RAM running Ubuntu 14.04 operating systems. All algorithms were implemented using Javac Compiler of version 1.7.x. In terms of Hadoop eco system, Cloudera Distributed version 5.2 has been setup. The relevant version of Hive-0.13 and Impala-2.0 are leveraged in this effort. We measure the performance in term of execution time as well as speedup and scale up between disk-based Hive and in memory based Impala.

6.2 Disk based Hive

Apache Hive is a data warehouse infrastructure built on top of Hadoop for providing data summarization, query, and analysis. HiveQL, which is an SQL-like language provided by Hive, provides a mechanism to project structure onto the data and query the data. Also this language allows traditional map/reduce programmers to plug in their custom mappers and reducers when it is inconvenient or inefficient to express this logic in HiveQL. Its execution is depicted as below:



As shown, the data you are storing in Hive databases is in fact a part of Hadoop data directory only. HDFS is a super set where all data including Hive databases is stored.

6.3 Memory based Impala

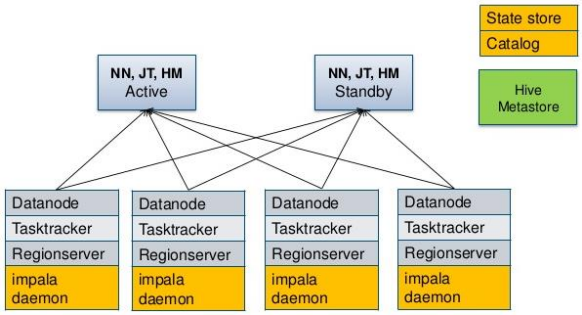
6.3.1 Motivation

Impala doesn't even use Hadoop at all. It simply has daemons running on all your nodes which cache some of the data that is in HDFS, so that these daemons can return data quickly without having to go through a whole Map/Reduce job.

The reason for this is that there is a certain overhead involved in running a Map/Reduce job, so by short-circuiting Map/Reduce altogether you can get some pretty big gain in runtime

6.3.2 Design

Impala is designed to scale with 3 key daemons. (1) Impala – collocate the data nodes (2) State Store – confirm the healthy node to accept the new work (3) Catalog – broadcast meta data changes to all Impala daemons.



6.4 How Impala is faster

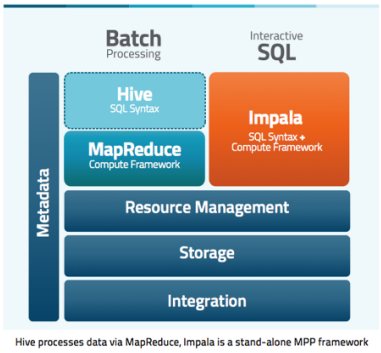
To avoid latency, Impala circumvents MapReduce to directly access the data through a specialized distributed query engine that is very similar to those found in commercial parallel RDBMSs.

In the experiments, it is clear that Impala has outperformed Hive *by 6x to 10x* times.

Table 2. Impala performance over Hive

#	Query Category	X times faster
1	Interactive	6x
2	Reports	8x
3	Deep Analytics	10x

The result is order-of-magnitude faster performance than Hive by the architecture itself.



Hive processes data via MapReduce, Impala is a stand-alone MPP framework

That is the reason Impala is heavily used Cloudera, MapR, Amazon Web Services, Oracle, etc.

6.5 User Defined Function - Ranking

User-defined functions (frequently abbreviated as UDFs) let you code your own application logic for processing column values during the Hive/Impala query.

Depending on your use case, you might write all-new functions, reuse Java UDFs that you have already written for Hive, or port Hive Java UDF code to higher-performance native Impala UDFs.

We can code either scalar functions for producing results one row at a time, or more complex aggregate functions for doing analysis across. In this project, ranking algorithms UDF is developed not only weights based on term importance of the data set.

7. Experiments

This section reports the evaluation of the proposed system, top-k equity pricing query processing using MapReduce algorithm. Evaluation process involves two methodologies (1) disk based Hive (2) memory based Impala.

7.1 Performance Evaluation

We test few months of multi-dimensional datasets with varied dataset size in our experiments. Specially, we retrieve several synthetic with varying the number of data records from 10,000 to 200,000 from NYSE historical download site.

In our experiments, we evaluate the effects of the dataset size n, the result number k, and the number of machines s using Hive & Impala. Our use case has the variation of ‘s’ from 2 to 4. ‘n’ is based on the pre-loaded historical date wise records.

7.2 Observation

As the source data file is downloaded from NYSE historical data site, it is loaded into Hadoop storage. Top-k Query is executed based on the input criteria (usually date wise) given by the end user. It is executed in two shells (1) Hive (2) Impala.

On experimenting, the results are observed as below:

Table 3. Hive Execution in sec

Node(s)	100 million	200 million	300 million
1	6.31	11.17	16.30
2	6.10	10.81	15.43
3	5.82	9.87	15.01
4	4.80	9.12	14.69

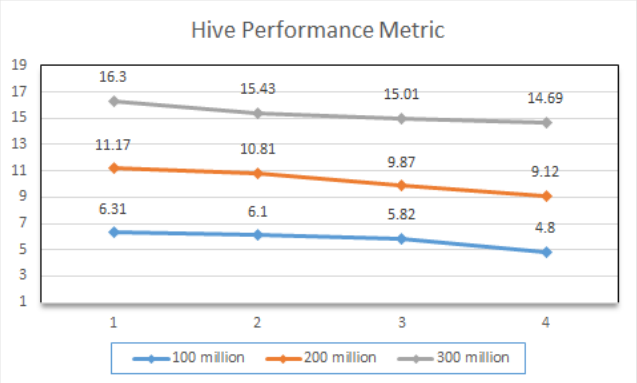
The same data set D is executed for Top-k equity pricing query Q and the results are observed as:

Table 4. Impala Execution in sec

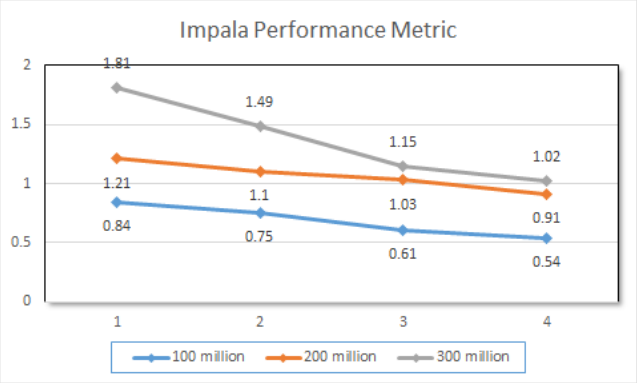
Node(s)	100 million	200 million	300 million
1	0.84	1.21	1.81
2	0.75	1.10	1.49
3	0.61	1.03	1.15
4	0.54	0.91	1.02

Key observation is the parallel execution is linearly scalable against data set volume and number of nodes.

It is completely depicted in the observed result table and graph as attached.



In the performance metric chart, number of Hadoop nodes (1,2,3,4) is recorded in X axis, whereas the execution time is placed in Y axis. In our experiment result, the execution time is represented in seconds.



Based on the above analysis, it is inferred that disk based Hive processing is taking 8 times of memory based Impala.

A linearly scalable application can scale just by adding more machines and/or CPUs, without changing the application code. In our observation, adding nodes is linearly scalable against the performance of the execution process.

8. RELATED WORK

In this project, two areas are addressed (1) Top-k equity pricing search @ business domain (2) disk based vs. in memory based @ technology domain. On analyzing the stock exchange site(s), Top-k pricing search on equity market is not around and so it is kind of new entry. In terms disk vs memory based Hadoop frameworks (hive vs impala), there are few industry white papers available to consume.

9. CONCLUSION & FUTURE WORK

This work studies parallel top-k equity queries over large multidimensional data using disk based and memory based Hadoop framework. By experiment results, this study shows the better effectiveness and scalability of in-memory based Impala than disk based Hive framework.

As the next step, the existing analysis pattern can be extended to the rest of commonly available financial stock exchange data.

10. ACKNOWLEDGMENTS

I would like to thank our professor Dr. Sayan Ranu and TAs for their helpful comments and suggestions. This material is based upon the project work for IIT 2015 odd semester effort.

11. REFERENCES

[1] Hadoop: The Definitive Guide, 3rd Edition by Tom White Publisher: O'Reilly Media, Inc. Release Date: May 2012. http://cdn.oreillystatic.com/oreilly/booksamplers/9781449311520_sampler.pdf

[2] Data-Intensive Text Processing with MapReduce by Jimmy Lin and Chris Dyer. Morgan & Claypool Publishers, 2010. http://lintool.github.io/MapReduceAlgorithms/ed1.html

[3] Paper on MapReduce Online by Tyson Condie, Neil Conway, Peter Alvaro, Joseph M. Hellerstein at UC Berkeley & Khaled Elmeleegy, Russell Sears by Yahoo! Research, http://db.cs.berkeley.edu/papers/nsdi10-hop.pdf

[4] NYSE EOD data downloader site: http://eoddata.com/download.aspx

[5] An Efficient Parallel Top-k Similarity Join for Massive Multidimensional Data by Dehua Chen, Changgan Shen, Jieying Feng and Jiajin Le of Computer Science and Technology Academy, Donghua University, Shanghai,China, http://www.sersc.org/journals/IJDTA/vol8_no3/6.pdf

[6] THUSOO, A., SARMA, J. S., JAIN, N., SHAO, Z., CHAKKA, P., ANTHONY, S., LIU, H., WYCKOFF, P., AND MURTHY, R. Hive — a warehousing solution over a Map-Reduce framework. In VLDB (2009)

[7] Dynamic file-access characteristics of a production parallel scientific workload by David Kotz and Nils Nieuwejaar of The Pennsylvania State University, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.120.4167 &type=pdf

[8] Y. Kim, K.Shim,"Parallel top-k similarity join algorithms using MapReduce", Data Engineering (ICDE), 2012 IEEE 28th International Conference on. IEEE, (2012)