5-2016

# Data Warehousing Modernization: Big Data Technology Implementation

Phani Vivekanand Kandalam
*St. Cloud State University*, kaph1202@stcloudstate.edu

# Data Warehousing Modernization: Big Data Technology Implementation

by

Phani Vivekanand Kandalam

A Starred paper

Submitted to the Graduate faculty of

St. Cloud State University

In Partial Fulfilment of the Requirements

for the Degree of

Master of Science

In Computer Science

May, 2016

Starred paper Committee:

Donald Hamnes, Chairperson

Dr. Ramnath Sarnath

Dr. Dennis Guster

## Abstract

Considering the challenges posed by Big Data, the cost to scale traditional data warehouses is high and the performances would be inadequate to meet the growing needs of the volume, variety and velocity of data. The Hadoop ecosystem answers both of the shortcomings. Hadoop has the ability to store and analyze large data sets in parallel on a distributed environment but cannot replace the existing data warehouses and RDBMS systems due to its own limitations explained in this paper. In this paper, I identify the reasons why many enterprises fail and struggle to adapt to Big Data technologies. A brief outline of two different technologies to handle Big Data will be presented in this paper: Using IBM's Pure Data system for analytics (Netezza) usually used in reporting, and Hadoop with Hive which is used in analytics. Also, this paper covers the Enterprise architecture consisting of Hadoop that successful companies are adapting to analyze, filter, process, and store the data running along a massively parallel processing data warehouse. Despite, having the technology to support and process Big Data, industries are still struggling to meet their goals due to the lack of skilled personnel to study and analyze the data, in short data scientists and data statisticians.

## Acknowledgement

Though the following paper is an individual work, I could have never reached the heights without the support, efforts and guidance of certain people.

First and foremost, I would like to thank my advisor Prof. Donald Hamnes for providing me endless guidance throughout the project tenure.

I would like to thank and express my deepest gratitude to my committee members Prof. Ramnath Sarnath and Prof. Dennis Guster for all their support and encouragement working through this project. I would also like to thank Prof. Jayantha Herath for providing me guidance throughout my tenure at St. Cloud State University.

This project could not have been completed without tremendous support from my manager at my workplace who supported and assisted me at every step. It is with his strong knowledge, motivation and vast experience I have acquired enough notion to explore the overall scope of the project.

Finally, I would like to thank my family and all my friends for the continuous support and encouragement without whom I could not have come this long way.

# Table of Contents

**Table** **Page**

Chapter V

# List of Tables

# List of Figures

**Figure** **Page**

## Definition of Terms

| | | |
|---|---|---|
| ABT | ……. | ABSTRACT SYNTAX TREE |
| AMPP/MPP | ……. | ASYMMETRIC MASSIVELY PARALLEL PROCESSING/ MASSIVELY PARALLEL PROCESSING |
| BI | ……. | BUSINESS INTELLIGENCE |
| CPU | ……. | CENTRAL PROCESSING UNIT |
| DAG | ……. | DIRECTED ACYCLIC GRAPH |
| DML | ……. | DATA MANIPULATION LANGUAGE |
| ETL | ……. | EXTRACT TRANSFORM LOAD |
| FPGA | ……. | FIELD PROGRAMMABLE GATE ARRAYS |
| GRA | ……. | GUARANTEED RESOURCE ALLOCATION |
| GUI | ……. | GRAPHICAL USER INTERFACE |
| HDFS | ……. | HADOOP DISTRIBUTED FILE SYSTEM |
| HiveQL | ……. | HIVE QUERY LANGUAGE |
| JDBC | ……. | JAVA DATABASE CONNECTIVITY |
| JVM | ……. | JAVA VIRTUAL MACHINE |
| LDAP | ……. | LIGHTWEIGHT DIRECTORY ACCESS PROTOCOL |

| | | |
|---|---|---|
| OBIEE | ……. | ORACLE BUSINESS INTELLIGENCE ENTERPRISE EDITION |
| ODBC | ……. | OPEN DATABASE CONNECTIVITY |
| OLAP | ……. | ONLINE ANALYTICAL PROCESSING |
| OLE | ……. | OBJECT LINKING AND EMBEDDING |
| OLTP | ……. | ONLINE TRANSACTION PROCESSING |
| PB | ……. | PETA BYTE |
| PQE | ……. | PRIORITY QUERY EXECUTION |
| QB | ……. | QUERY BLOCK |
| RAID | ……. | REDUNDANT ARRAY OF INDEPENDENT DISKS |
| RAM | ……. | RANDOM ACCESS MEMORY |
| RDBMS | ……. | RELATIONAL DATABASE MANAGEMENT SYSTEM |
| RSG | ……. | RESOURCE SHARING GROUP |
| SPA | ……. | SNIPPET PROCESSING ARRAY |
| SPU | ……. | SNIPPET PROCESSING UNIT |
| SQB | ……. | SHORT QUERY BIAS |
| SQL | ……. | STRUCTURED QUERY LANGUAGE |
| TB | ……. | TERA BYTE |

TCP ……. TRANSMISSION CONTROL PROTOCOL

# Chapter I

## INTRODUCTION

"The world contains an unimaginably vast amount of digital information which is getting ever vaster ever more rapidly" [1]. Currently, vast amount of data is generated in short amount of time leaving middle scaled as well as large scaled industries in havoc. According to [18], by 2020, there will be 5,200 GB of data for every person on Earth. During the next eight years, the amount of digital data produced will exceed 40 zettabytes. Also, until the year 2002 the total amount of data generated is equal to the data generated in a single day today. This is mainly due to the increase in number of sources the data is generated from. Also, the number of sources and devices generating the data are increasing constantly. We are all data generators on this planet, all of our behavior from purchasing a book online to uploading a picture to Facebook produces data. These data contain information which is critical for decision making, market research, future prediction, etc. Such data reflects many aspects of this world in many fields such as: sport, finance and banking, science, marketing, journalism and medicine. For example, the data related to the baseball players are statistically accumulated to measure a player's performance and also to predict his/her future performance. The finance and banking have been mathematized and all transactions are logged. These logs reflect customer's preference, purchasing behaviors, etc. The data recorded by each of the experiments at the Large Hadron Collider (LHC) in Geneva, Switzerland is enough to fill around 100000 DVDs every year [19]. Streams and streams of data are being generated ready to be used, to make more money. Most of the data generated these days is unstructured, which means, the data cannot be stored in structured, predefined tables (Traditional Database Systems) anymore (external applications can be used to store such data but every application has its own limitations). Such data which is

beyond the capabilities of typical database systems to store, process, and analyze it because of its volume, velocity and variety is termed as "BIG DATA" [2].

***Volume:*** Big Data cannot be defined in terms of being more than a certain number of terabytes. As technology advances, the volume of data that can be handled by typical database systems also increases. Hence, Big Data is defined as the volume of data that is beyond the storage capacity of typical database systems [3].

***Velocity:*** Big Data is not just defined by volume but also by the velocity at which the data is being generated. If the rate at which the data is generated exceeds the rate at which the data can be processed by a typical database system, then the data can be defined as Big Data [3].

***Variety:*** Big Data is also about the variety of information in the data being generated. Information includes images, videos, logs, graphs, documents etc. All this is raw data and is unstructured and so cannot be stored and processed by typical databases [3].

## Problem Statement

According to Gartner, as of 2012, 64% of organizations were either planning to or have already invested in Big Data technology [11]. Out of that 64%, 35% still do not know what they are doing or going to do [11]. Only less than 8% were actually able to deploy big data technology [11]. These numbers show that most of the Big Data projects are either failing completely or failing to make any progress.

Reasons for this failure in Big Data projects were analyzed with the help of current and previous research articles from Gartner and Computing Research. The main reasons for the failure are:

I. Asking the wrong questions: Data science is a complex blend of domain knowledge and technical knowledge. Too many organizations hire data scientists that are technically fit but lack the domain (business) knowledge [12].

   Example use case: A manufacturing company with many dealerships ran a sentiment analysis project to learn about its customers. After spending six months and $10 million on Big Data technology, the findings were distributed to all the dealerships. These dealerships already knew this information that the Data scientist dug out, which eventually lead to the project failure [12].

II. Data access capabilities: Being able to access and process the data is critical after analyzing the data. This applies to networking issues as well.

   Example use case: A retailer tried to run a Big Data project in the cloud. The project could not move forward due to network congestion [12].

III. Enterprise strategy: Big Data projects can be highly successful if they are not isolated. Sharp boundaries between various departments within an organization are a great hindrance to Big Data growth [12].

   Example use case: One of the challenges of a leading telecom provider was that, even though analytic capabilities were spread across the enterprise, each department had its own data analysts and technologies. Many of their efforts were duplicated across departments because of their disconnected structure [26].

IV. Speed of data retrieval: Hadoop is a low cost system but at a cost of latency. Hence data retrieval rates are not significant compared to a regular RDBMS system or a Data Warehouse system [13].

Example use case: An online advertising broker found out that they needed a better platform than Hadoop for instant data analytics as Hadoop is a batch processing system optimized to load large data sets but not for fast data analytics [29].

V. Amateur and Limited Big Data resources: Even though a few companies have deployed Big Data systems, it is still in an amateur stage, and a majority of the developers and architects are still working for the creators of the Big Data technologies (Google, Yahoo, Facebook, etc.) [13].

Example use case: A research conducted by PwC and Iron Mountain reveals that 75% of organizations (companies of all sizes) surveyed were lacking the skills and technology in using their data to gain competitive edge over other similar organizations. Moreover, they were not able to hire an expert data analyst [49].

VI. Poor Data Quality: Poor data quality can have a big impact on the analysis based on the project (Sometimes, even exponential impact) [13].

Example use case: A company generated a sales report and made a hiring and business decision based on the report. But since the data was of poor quality, the report was flawed as it was generated on poor data. The company was shocked when their sales took a dive because of that decision [50].

VII. Inefficient Data Warehouse: Data warehouses should be capable of handling large volumes of data as well as process multiple requests concurrently at a very high speed. A fine tune of workload management and concurrency handling yields results.

Example: As an enterprise matures, their data warehouse has to face more complex data requests, larger volumes of data, data source incompatibilities, etc. but efficient operation of data warehouse is still the number one priority and so, data warehouse technology and resources have to mature as well.

Hence, a better way of analyzing, visualizing and manipulating Big Data is very much needed.

## Nature and Significance of the Problem

To analyze a business is to study their data, which is termed as "Business Intelligence" (BI). Such data is used to make strategic decisions that drive the business into the future [4].

Evolution of Business Intelligence (BI):

As shown in Figure 1, in the 1980's, data was used by businesses for reporting purposes, mainly static reporting, which basically explains 'what happened' in the past. However, just figuring out 'what happened' does not add much value to the business. In the 1990's, analysis tools were released which helped to figure out 'why things happened' but they still give information only about the past. In the 2000's, it became more about live monitoring with dashboards or scorecards instead of figuring out what happened. This gives information about 'what is happening right now'. Today, we are in the world of prediction. It is about 'what is going to happen' and making decisions based on those predictions [6].

So, for efficient functioning of a business, getting the data and using the data is the key [5]. Existing enterprise data warehouses and relational databases are efficient ways of processing structured data and can store massive amounts of data. Although, such requirements for structure restricts the kinds of data that can be processed since the relational database model and data warehousing concepts were built based on traditional relational database modeling.

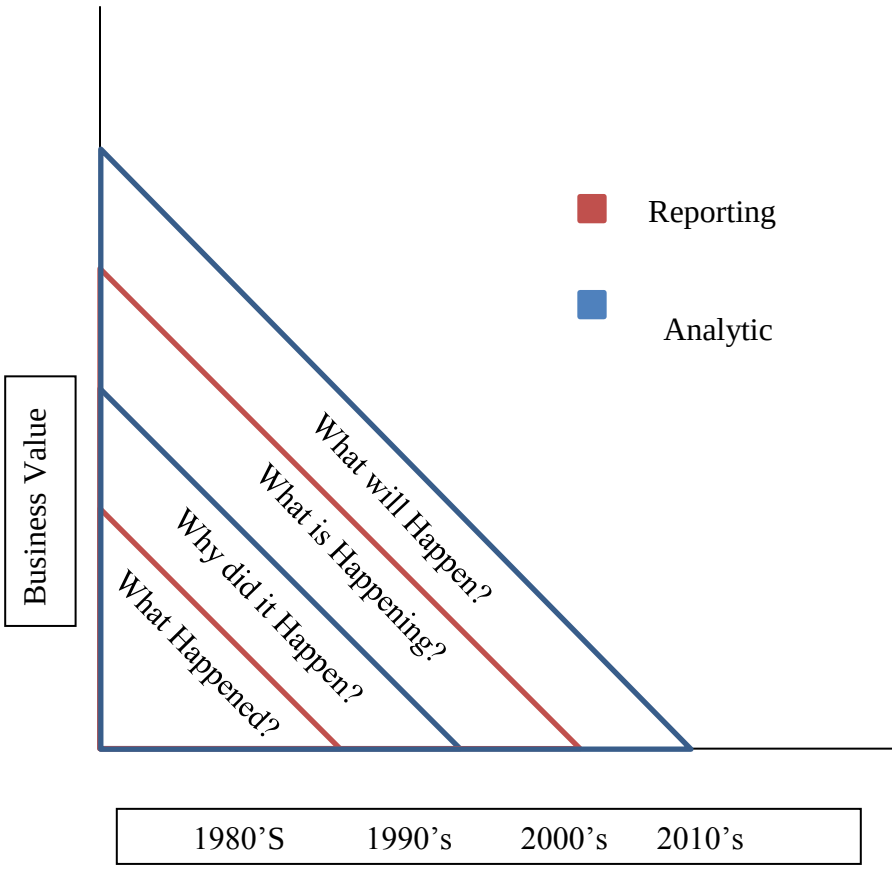


**Figure 1: Evolution of Data Analysis [6]**

With the growth of social media and mobile devices the world has already started producing huge amounts of unstructured data (Big Data). All this unstructured data can play a big part in business intelligence and so this data should be brought to a single platform. Also, there has been an immense increase in terms of data production. However, the rates at which data can be read

from drives have not increased impressively over time and there is a mismatch in terms of data production rate and the data warehousing and retrieval rate [7]. Since traditional database modeling has its own limitations with handling this kind of data, there is a need for new technologies in handling and maintaining this Big Data.

## Objective of the Survey/Study

- Introduce Big Data and Hadoop (Big Data technology)
- Perform a survey to identify the problems with the existing usage of Big Data technologies and analyze how successful Big Data deployments are functioning.
- Analyze how companies like Facebook, etc. are performing data warehousing operations along with Big Data solutions.
- Suggest Big Data solutions based on the analysis performed.

## Study Questions/Hypotheses

1. What is Hadoop and can it solve the Big Data problem?
2. How big is the Big Data problem and is it worth the time and effort of the organizations trying to solve Big Data issues?
3. Can Hadoop run together, in parallel with the existing data warehouse systems?

## Limitations of the Study

Implementing Hadoop, Hive and/or MPP architecture data warehouse like Netezza cannot be an ideal solution to every organization that deals with Big Data. Simulations, analysis, implications and results may vary from one enterprise to another not limited to retail, social media, e-commerce, etc. The only purpose of the analysis carried out between Oracle DB and Netezza

appliance in this paper is to show which one fits better in a data warehouse environment but not to compare the two.

# Chapter II

## BACKGROUND AND LITERATURE REVIEW

Companies thrive to put in every effort and time to achieve the ultimate aim of providing lighting speed data support to the end clients or customers. Growth of data over the years have increased multiple folds and to support such a huge volume there is always a need to have appliances supporting parallel request simultaneously along with a server capable of handling bulk requests streaming at high speeds. Several industries have spent millions of dollars hiring widely technical personnel to analyze the business and have failed. Over the years, companies were to shell out additional expenditure worth millions of dollars to buy more and more servers to support 'Big Data'. Though implementation is considered to be a secondary factor on why the companies are failing at achieving 'Big Data ', the primary is the decision making and lack of suitable/talented personnel. Considered crucial, with no experienced data scientists, data analysts, data statisticians and data architects it is a nightmare for a company to achieve 'Big Data' solutions since it takes a decisive team work of such personnel to understand the in-and-outs of a business from end to end in order to make any crucial decisions towards the welfare of a business to keep it competing in the current market.

## Background Related to the Problem

In earlier days, data was stored in a flat file format with no structure to it. Retrieving the data from the flat file used to be a project by itself. When efficient data retrieval from a database and data integrity looked a distant dream, **Edward Frank Codd** invented the relational model for database management [8]. With this invention, the information retrieval from databases was made efficient and all the database users started to adopt the relational database model [8]. Some

vendors have built tools and applications to support the relationship between data and improve the performance of data retrieval. Entity Relationship model also evolved which is a data model to describe data in an abstract manner.

With the popularity and ease of use, almost all organizations started using relational database management systems and ultimately there was a rapid growth in data. This enormous data growth posted a big challenge to the organizations that wanted to build intelligent systems based on the data. Hence organizations started data warehousing where data was stored and then processed [8]. Data was retrieved overnight from the transaction system and business intelligence reports were built on it. By this process, business intelligence has become a necessity.

Existing Data warehouse Architecture:

A data warehouse is a relational database that is optimized for query and analyzing purposes as opposed to transactional processing. It is usually the data system used by enterprises to store their transactional data but is not restricted to just that.

Data warehouses are designed to help analyze data. An example can be to analyze a company's sales data. A general rule of a data warehouse would be that the data should not be changed once it enters a data warehouse. On OLTP systems, historical data needs to be archived on a regular basis as having a large amount of data can affect performance of the data warehouse. Nevertheless, a data warehouse needs to maintain historical data as businesses perform analytics and generate reports based on the data in a data warehouse. Also, analysts need large amounts of data to discover trends in business accurately. [36]

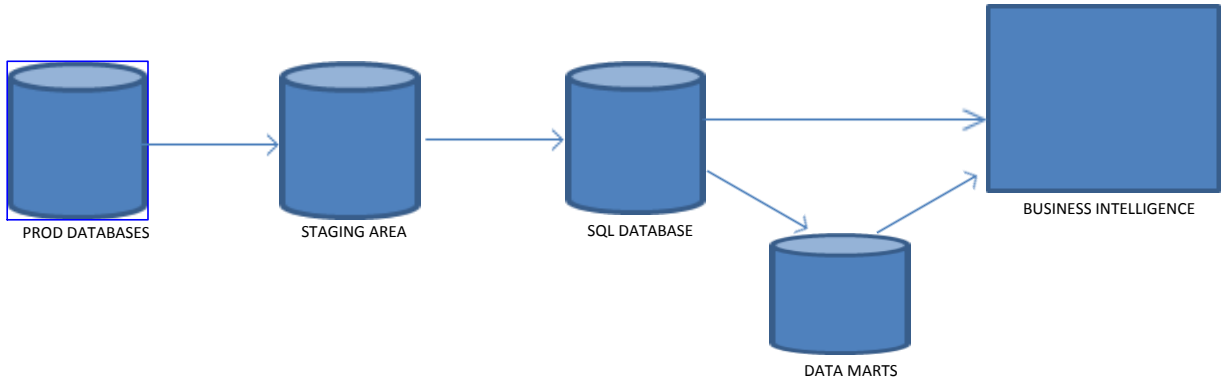


**Figure 2: A typical Data Warehouse Architecture,** adapted from [20]

Data warehouses are typically designed to accommodate huge workloads as most of the queries would be adhoc and the workload can be variable unlike OLTP systems.

Existing enterprise data warehouses and relational databases are very good at processing structured data and can store massive amounts of data. This requirement for structure restricts the kinds of data that can be processed as the relational database model and data warehousing concepts were built based on traditional relational database modeling.

## Literature Review

[14] shows a performance comparison between a four node Hadoop cluster and MySQL DB based on data from campus library circulation log. In the experiment performed, Hive was used as the front end for the Hadoop file system. Results from the queries executed in [14], show an increased performance ranging from 3% for simple queries to 37% or even more in case of queries involving larger data sets. In one example, Hadoop was able to execute a query that caused memory exception in MySQL in 4 minutes 22 seconds. Overall, Hadoop system outperformed traditional RDBMS system in each of the four queries executed. Also, this paper suggests further research on optimizing the processing nodes of distributed file systems.

From [15], it can be understood that any organization from any industry (no matter if it is atmospheric, biological science or manufacturing etc..,) can take benefit from careful analysis of their big data for the problem solving purpose. This paper explains the Big Data scope, samples, advantages and challenges of data. The challenge is not anymore to collect or manage the data but it is how to extract useful information from meaningful analysis of the collected data [15].

[16] talks about the bottlenecks prevalent in the existing enterprise computing with using centralized databases. Cassandra which is an open source distributed database management system was considered in [16]. The hardware included four 250 GB hard disks, nodes ranging from 2 – 16 with each node having two 64 bit 2 Ghz processors and 2 GB memory. Network connecting the nodes was a 1 Gbps Ethernet connection. In case of accessing 5000 records, the least latency of 0.0020 seconds was achieved for 16 nodes. But for 1 million records, the least latency of 0.0008 seconds was achieved for a 16 node system. The results from [16] show that adding additional nodes did not always result in improved latency and this can be because of increase in communication overhead with increase in number of nodes. The same logic applies for data insert rates too. It can also be understood that significant performance gains can be achieved by using a distributed database over a centralized but still, more number of nodes in a distributed system does not always result in an improved performance. Considering network speed between the nodes plays a very important role in performance of a distributed system. Also, a distributed system like Cassandra can be a highly scalable and cost effective solution as it uses multiple commodity disks.

[17] provides survey results about what type of data is currently being captured and analyzed which is also shown in Figure 3. Several forms of unstructured and semi structured data dominates enterprise data. This paper also provides information on how much of the collected

data is actually being analyzed, rate of data growth and also information about some of the obstacles in implementing Big Data solution.

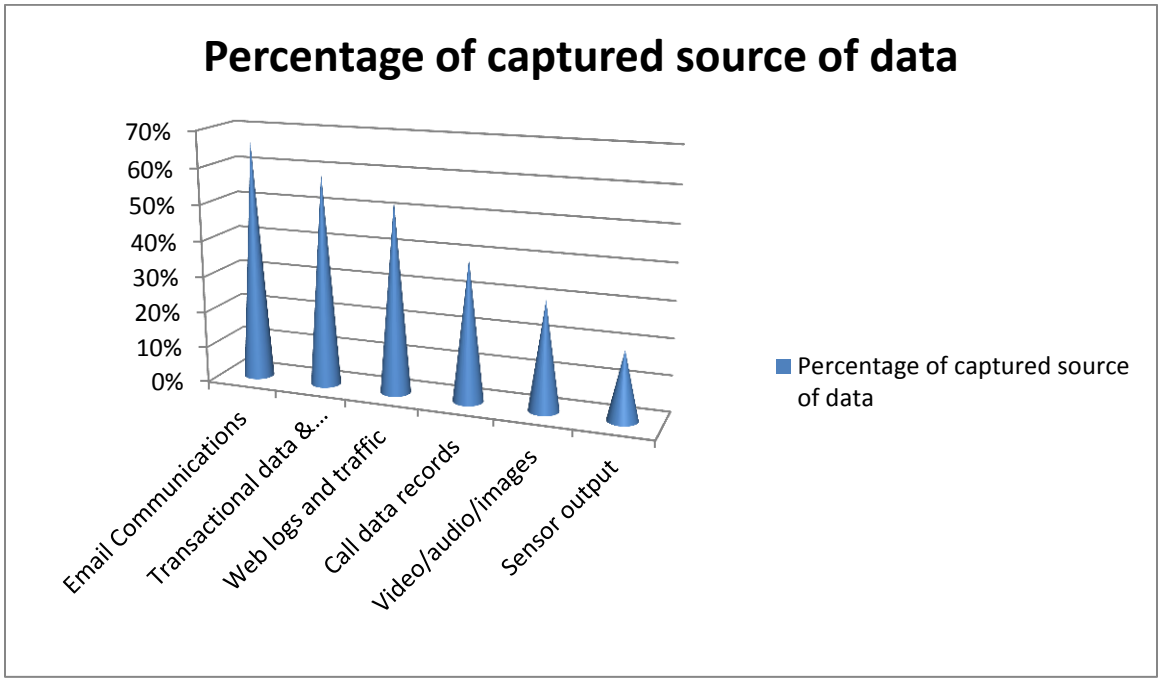


**Figure 3: Primary sources of Data,** adapted from [17]

## Literature Related to the Methodology

To solve this Big Data problem, Google proposed the Map-Reduce algorithm for processing their huge amounts of input data [8]. To explain in general, a 'Map' operation is applied to each logical record in the input and a set of intermediate key/value pairs are computed. The Map/Reduce library functions group together all intermediate values associated with an intermediate key and passes them to the 'Reduce' function. Then, a 'Reduce' operation is applied to all the values that shared the same key. By this way, the intermediate data is computed accordingly. This functionality allowed Google to parallelize large computations easily [8].

These innovations from Google were incorporated into Nutch, [7] an open source web search engine. Later, Hadoop was created by Doug Cutting, the creator of Apache Lucene, the widely used text search library of which Apache Nutch is a part [7].

"**Apache Hadoop is an open source software framework that supports data-intensive distributed applications under a free license**" [7]. It is a Java-based programming framework that supports reliable, scalable, distributed processing of large data sets in a distributed computing environment. This system is designed to run on a large number of machines. It changes the dynamics of how the large scale data is looked at and worked with currently, rather than relying on high-end hardware. The software has the ability to detect and handle failures at the application layer. Described by the judging panel of the MediaGuardian Innovation Awards as a "Swiss army knife of the 21st century", Apache Hadoop received the ***innovator of the year*** award for having the potential to change the face of media innovations [9].

The two major components of the Hadoop framework are the Hadoop Map/Reduce and Hadoop Distributed File System.

Hadoop Map/Reduce: This method is used to split larger data into smaller parts and distribute it to many different commodity servers. Each server has its own share of resources to process the data. Once the data is processed, the data is sent back to the main server collectively. The Hadoop Map/Reduce is effectively used to process large data effectively and efficiently [9].

Hadoop Distributed File System (HDFS): The HDFS is a file system designed to work with Hadoop Map/Reduce. When a file is moved to HDFS, it is automatically split into multiple pieces. These small chunks are again replicated and stored in different servers. This provides both fault tolerance and high availability [10].

Data processing at high speeds will need to run concurrently on a data warehouse server in order to process multiple requests at the same time. Each concurrent query/request running against a data warehouse server requires and acquires available resources. It is also equally important to set a limit to the number of concurrent queries. Restricting concurrent queries can save network congestion, thereby providing efficiency and throughput. [37] illustrate a model which states the query response time for each query submitted to the system. Priorities need to be assigned based on the role of a user/user group in a data warehouse environment. Resources will need to be nurtured round the clock to provide enough efficiency to the higher priority jobs. Dynamic allocation of resources plays a key role in such an environment. [37] also stated that a control loop has been designed which allows to have a control over the number of active concurrent sessions running against a system. [37] also states how the dynamic allocation of system resources, re-assigning the priorities and re-allocating the system resources based on the requirement, importance and impact, can result in a large variance in terms of performance thereby meeting business requirements.

In [38], Benoit proposed a model which illustrates the performance of a data warehouse server based on assigning memory resources across users and user groups. Assigning correct combination of system resources can impact the performance of a server in a great way. Each resource assigned to a user/user group has an impact on the rest of the other resources, which is thereby causing interdependency across each resource assigned on a data warehouse server. Similarly, [39] describes the factors contributing towards the performance of a system as well as the available metrics to measure the performance factor. Several parameters contribute towards a server's performance not limited to tweaking system configurations, Work Load Management policies, resource setting adjustments, and appliance configuration at the server/global level.

# Chapter III

## METHODOLOGY

The objective of this survey is to introduce and analyze the concept and implementation of Big Data and analyze why the organizations are failing to successfully implement Big Data solutions. Analyzing the enterprise architecture of successful implementations and there by providing solutions to overcome problems faced by mid-scale companies is also part of the objective.

Between 1998 - 2003, Google implemented many special-purpose computations that process large amounts of raw data. However, the input data was usually huge and so the computations had to be distributed across thousands of machines to finish in a reasonable amount of time. Even though the computations were straightforward, the issues of parallelizing the computations, distributing the data and handling failures made this process complex [8].

Reacting to this complexity, a new abstraction was developed. This was inspired by the 'Map' and 'Reduce' primitives present in Lisp and many other functional languages [8]. To explain in general, a 'Map' operation is applied to each logical record in the input and a set of intermediate key/value pairs were computed. The Map/Reduce library functions group together all intermediate values associated with an intermediate key and passes them to the 'Reduce' function. Then, a 'Reduce' operation was applied to all the values that shared the same key. By this way, the intermediate data is grouped accordingly. This functionality allowed Google to parallelize large computations easily [8].

Here, I am considering an example of '**Finding the right partner to a person based on his/her interests on a dating website**' to explain the Map/Reduce algorithm which is inspired by the idea of 'Finding mutual friends on Facebook' in an article by **Steve Krenzel** [21].

For this example, consider a dating website www.XdatesY.com. Every user in this website will have a unique list of their interests listed in their account. Also, every user would like to date a person whose interests closely match with the interests of theirs. So, whenever a person searches on the website to find a dating partner, a list of mutual interests should be displayed between the two. The list of mutual interests can be calculated using the map reduce functions once a day and these results can be stored so that whenever a person opens the profile of another person, the mutual interests are displayed.

The interests are stored as Person -> [List of Interests]. Here, we have the following lists:

A -> 1 2 3 4 6 9

B -> 2 4 6 8 3

C -> 1 5 2 9 8

D -> 3 6 5 9 2 4

Each line will be an argument to a mapper function. For every person, the mapper will output a key-value pair. The key will be a person along with another person of opposite gender and near to the current location. The value will be the list of all the interests of that person. (I will not be discussing on how to differentiate the person based on gender and location in this paper. For convenience, I am considering A, C as males and B, D as females and all are at the same location.) The key will be sorted so that the persons are in order, causing all pairs to go to the same reducer function. After all the mappers are done running, we have the following list:

For map (A -> 1 2 3 4 6 9):

(A, B) -> 1 2 3 4 6 9

(A, D) -> 1 2 3 4 6 9

For map (B -> 2 4 6 8 3):

(A, B) -> 2 4 6 8 3

(B, C) -> 2 4 6 8 3

For map (C -> 1 5 2 9 8):

(B, C) -> 1 5 2 9 8

(C, D) -> 1 5 2 9 8

For map (D -> 3 6 5 9 2 4):

(A, D) -> 3 6 5 9 2 4

(C, D) -> 3 6 5 9 2 4

These are grouped by their keys before they are sent to the reducers:

(A, B) -> (1 2 3 4 6 9) (2 4 6 8 3)

(A, D) -> (1 2 3 4 6 9) (3 6 5 9 2 4)

(B, C) -> (2 4 6 8 3) (1 5 2 9 8)

(C, D) -> (1 5 2 9 8) (3 6 5 9 2 4)

After grouping, each key-value pair is passed as an argument to the reducer. The reduce function intersects the lists of values and output the key with the intersection result. (This is nothing but getting the list of mutual interests between the two persons)

The resulting key-value pairs after reduction are:

(A, B) -> (2 3 4 6)

(A, D) -> (2 3 4 6 9)

(B, C) -> (2 8)

(C, D) -> (2 5 9)

When A visits B's profile, (A, B) is looked up and their common interests are listed which are (2 3 4 6). In this way, Map/Reduce algorithm can be used to find the common interests between two persons.

## Architecture of Hadoop

Since Hadoop is based on Google white papers, it subscribes to a Master/Slave Architecture. It stores file system metadata and application data separately. HDFS stores metadata on a dedicated server, called the ***NameNode***. Application data are stored on other servers called ***DataNodes***. These servers are connected and communicate using TCP-based protocols. The file content is replicated on multiple DataNodes for data durability. Figure 4 shows the connections between the various components in the HDFS architecture.

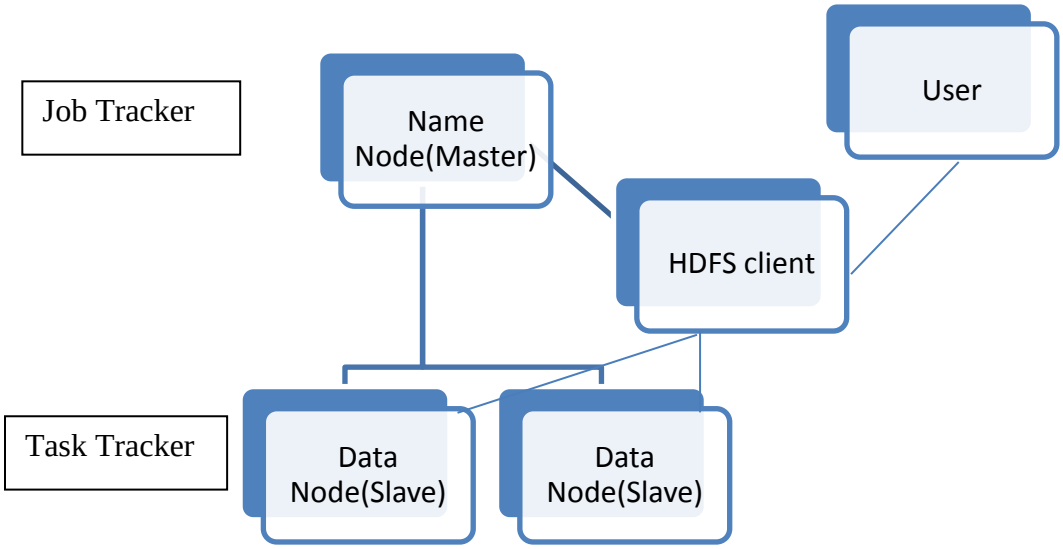


**Figure 4: HDFS Architecture**

NameNode: Files and directories are represented on the NameNodes in the form of inodes. These inodes save permissions, access and modification times, name space and disk spaces. To access the file, the HDFS client sends requests to the NameNode to retrieve the locations of the DataNodes comprising the file. Then the block content is accessed by the client. A block is the amount of data a drive can read or write at a time. The client always tries to read the content from its nearest data node. Users can access the file system only through the HDFS client (exports file system interface) [10].

To write data, the client requests NameNode to select DataNodes to replicate the blocks. This writing of data to these DataNodes is processed in a pipeline fashion. For each cluster, there exists a single NameNode, several DataNodes (in thousands) and many HDFS clients (in tens of thousands). This helps to execute multiple tasks concurrently [10].

The *Image* is the metadata of the file system which is the combination of inodes and the list of blocks that belong to each file. This image stored in the file system of the local host is called

*Checkpoint*. The log that maintains the changes made to the image is called the *Journal*. These can be used to recover a NameNode whenever it experiences a crash or a system restart. For each transaction a client initiates, the changes are recorded in the journal. This journal is updated and synchronized before the change is committed in the HDFS client. Unlike journal, if a new checkpoint is created by the administrator or during a restart the new checkpoint entirely replaces the old checkpoint. The HDFS is generally configured to store directories containing the checkpoint and journal at multiple locations to preserve the information in case of data corruption [22, pp. 200-204].

The NameNode can also play the role of a *CheckpointNode* or a *BackupNode* [10]. A CheckpointNode creates a new checkpoint from the existing checkpoint and the journal. This new checkpoint replaces the old checkpoint and the journal is cleared. This increases the stability and consistency of the system. If the CheckpointNode is not created and the journal continues to grow, then when the NameNode restarts, it takes more time to get back to work because of the large journal file [22, pp. 200-204].

A BackupNode is similar to a CheckpointNode creating checkpoints but also creates an image of the filesystem namespace which is regularly synchronized with the NameNode. If the NameNode fails, this BackupNode becomes the record of the last namespace state. So as soon as the NameNode fails, the BackupNode creates a Checkpoint and saves the namespace of the BackupNode into the local disk. Hence the BackupNode can be viewed as a read only NameNode which doesn't have block locations [22, pp. 200-204].

DataNodes: [10] The DataNode basically maintains the file data which is stored as blocks. HDFS has a default of 128mb block size (can be modified by user). Each block of a file is

replicated at multiple nodes independently which is by default three (can be modified by user). The reason for the HDFS blocks being large compared to the normal disk blocks is to minimize the cost of seeks. Each file is divided and saved into multiple blocks which together form the data of the file. The name spaces of all these blocks and the mapping of file blocks are stored in the NameNode. During start up, the DataNode performs a *Handshake* with the NameNode to verify the 'Namespace ID' and 'Software Version'. If Namespace ID and software version of the DataNode do not match to the NamespaceID and software version in NameNode, the DataNode shuts down. This provides consistency to the data and helps in preventing data loss.

A new DataNode is given permission to join the cluster by receiving the cluster's namespace ID. After a Handshake, the DataNode gets registered to a particular NameNode. *Storage ID* is a unique id of the DataNode which helps in identifying a DataNode even if it is restarted with a different IP address. Each DataNode sends a block report to the NameNode which contains the information of block replicas it contains. A block report contains the *block id*, the *generation stamp* and the length for each block replica the server hosts. After the registration of the DataNode, a first block report is sent. Consecutive block reports are sent with a time interval of one hour to update the NameNode with the location of the block replicas.

A *heartbeat* is sent to the NameNode every three seconds to show the normal operation of the DataNode. A heartbeat typically consists of total storage capacity, storage in use and number of data transfers in progress. The NameNodes replies to the heartbeats by sending instructions to the DataNodes to perform operations. A NameNode can process up to thousands of heartbeats per second depending on the hardware platform on which Hadoop is running. If a NameNode doesn't get heartbeat from a DataNode for ten minutes, then the NameNode considers the DataNode to be dead and schedules creation of new replicas on other DataNodes.

Hadoop Map/Reduce

Hadoop Map/Reduce layer consists of one JobTracker and many TaskTrackers.

JobTracker: [22, pp. 31-34] The JobTracker is the service within Hadoop that performs the Map/Reduce tasks to the nodes in the Hadoop cluster. The client application submits the jobs to be performed to the JobTracker. The JobTracker contacts the NameNode to determine the location of the data. Then the JobTracker locates TaskTracker nodes with available DataNodes near the data and submits the work to be done. The TaskTracker nodes are monitored by the JobTracker by the help of heartbeat signals which are sent by the TaskTracker to the JobTracker. If the TaskTracker fails to send heartbeat signals, then the JobTracker assumes the TaskTracker has failed and schedules the work to a different TaskTracker. Once the job is completed, the JobTracker updates its status and becomes available to the client application.

TaskTracker [22, pp. 31-34]: A TaskTracker is a node in the cluster that accepts jobs from the JobTracker. The TaskTracker spawns a separate JVM process to do the actual work. This is to ensure that any process failure does not take down the TaskTracker. The TaskTracker not only captures the output but also monitors the spawned processes. It notifies the JobTracker about the output when the process is finished. The heartbeat signals sent by the TaskTracker to the JobTracker not only indicates proper functioning of the TastTracker but also informs the JobTracker of the number of available DataNodes for the JobTracker to stay up to date.

Snapshots [10]: Whenever a software upgrade is applied to the HDFS, it is vulnerable to data loss. Hence, a backup of the state of the file system is needed before the upgrade is applied. This is done by the snapshot mechanism. It creates a snapshot of the current file system before an upgrade. Whenever the administrator chooses to snapshot, the NameNode copies the checkpoint

and journal files into a specific user defined memory and writes a new checkpoint with empty journal in place of the old checkpoint and journal. This is done to make sure that the old checkpoint and journal remain unchanged. During handshake, the DataNodes get the instruction from the NameNode to create a snapshot. The DataNodes then copy the storage directories and links the block files to it.

Whenever there is a software upgrade failure, the administrator can select the snapshot and the system gets back to the previous state. After the system rollback is done, the administrator can delete the snapshot and recover that memory.

Reads and Write: Any application accessing the HDFS can create a new file and write data to it. This way, it can add data to HDFS. Once the file is closed, the data in the file can neither be removed nor modified and any remaining block size after the last write of the file becomes unavailable. Whenever a client is writing on a file, the duration for which a client can write to a file is defined by two components: soft limit and hard limit (time limits). Until the soft limit expires, the user is certain to have access to the file. Within the soft limit, if the client fails to send a heartbeat to the NameNode, the hard limit time starts. By default, the hard limit is set to one hour. In the hard limit period, any other client can have access to write on to this file. If another client chooses to write on this file, then the hard limit automatically expires and the other client will have access. If even the hard limit has expired and there is no response from the client, then HDFS will assume that the client has quit and closes the file [22, pp. 62-75]].

Whenever a file needs to be written, NameNode picks a new block and assigns it a unique block ID and selects the list of DataNodes to hold the replicas. Bytes are sent through a pipeline as packets. Packets can be sent into the pipeline before receiving the acknowledgement of the

previous packet. The interaction between the client and the DataNodes include control messages to set up and close the pipeline, data packets and acknowledgement messages [10].

A file system design like this is especially optimized for Batch programming systems like MapReduce which needs a high throughput for reads [10].

Replica Management: Replica placement in HDFS plays a vital role in data reliability and read/write performance.

The rules for HDFS replica placement:

1. A single DataNode should contain more than one replica of any block.

2. No rack contains more than two replicas of the same block, provided there are sufficient racks on the cluster [22, pp. 32-50].

A NameNode gets to know the information about the number of replicas when it gets the block report from the DataNodes. Whenever, the number of replicas is high, the NameNode deletes a replica from the DataNode that has the least available space. If the number of replicas is low, the replica is put in the replication priority queue. The priority of the replica in the queue is based on the number of replicas already present. If there is only one replica present, then the priority is high. Now this process becomes similar to a new block replication. The NameNode makes sure that no two replicas of a block are placed on the same rack [10].

Hadoop is particularly useful in environments where massive server farms are used to collect data from a variety of sources. Hadoop processes parallel queries as big, background batch jobs on the same server farm. This saves the user from acquiring additional hardware for a traditional

database system to process the data. Hadoop also reduces the time and effort required to load data into another system. It can be processed directly within Hadoop. [23]

But the map reduce programming model is very low level and hence, requires custom programs to be built on top of it. These custom programs were hard to maintain and reuse and so, custom programs were needed on top of Hadoop that are standard, reusable and easy to maintain which formed the Hadoop Eco System.

Hadoop Eco System

The Hadoop ecosystem includes other tools built by companies like Yahoo, Facebook etc.., to address their particular needs. Figure 5 shows some of the tools at different layers of a Hadoop Ecosystem. The purpose of each layer is self-explanatory.

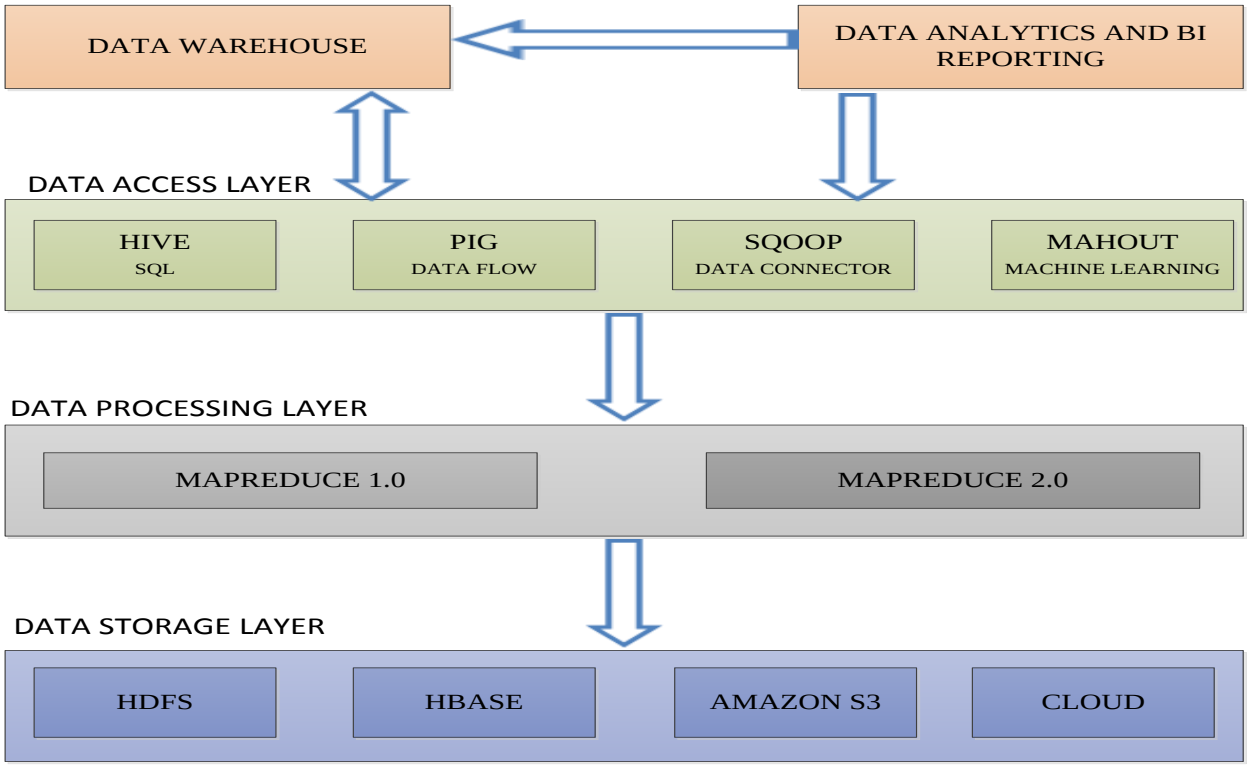


**Figure 5: Hadoop Eco system,** adapted from [30]

The concepts of MapReduce and HDFS have already been discussed in this paper.

Some components from Data Access Layer:

Pig: Pig is a platform for constructing data flows for ETL processing and analysis of large datasets. *Pig Latin* is the programming language for Pig. It provides common data manipulation operations, such as filtering, grouping and joining. Pig Latin is designed to fill the gap between the declarative style of SQL and the procedural style of MapReduce. It also has operators similar to SQL such as *FILTER* and *JOIN* that are translated into a series of map and reduce functions [23]. From [24], it can be understood that Pig itself generates the Hadoop MapReduce operations to perform the data flows. This simplifies the process by allowing users to inspect the stored data without attaching any complexities of MapReduce framework.

Hive: [25] Hive is data warehouse software that facilitates ad hoc querying, analyzing of large datasets and structuring of data. It is a SQL oriented system that supports the parts of SQL specific to querying data. Hive jobs are optimized for scalability but not latency. HiveQL is the dialect for Hive's SQL. HDFS data is stored in the form of tables. The queries are translated into MapReduce jobs to utilize the Hadoop system's scalability.

Some components from Data Storage layer:

HBase:[31] HBase is a database management system that is column oriented but does not support any query language like SQL. Most of the applications are written in Java similar to Map/reduce. It runs on top of HDFS. HBase system consists of tables with rows and columns similar to a RDBMS system. Each column is an attribute of an object. Many attributes are grouped together known as column families. All the elements of a column family are stored together which is different compared to a row based DBMS where all the columns of a row are

stored together. HBase is built on similar concepts of HDFS with a NameNode or MasterNode which manages the cluster and DataNode stores portions of data.

Amazon S3: [32] Amazon S3 (Simple Storage Service) is a storage designed for the internet. It can be used to store or retrieve any amount of data from around the web anytime. In Amazon S3, objects are organized in the form of buckets and each bucket is identified by a unique key given to the user. This service is mainly used to host websites. Amazon S3's design is aimed to provide scalability, high availability and low latency at commodity costs.

The main focus in this paper is on Hive providing details about Hive's data model, architecture and Hive Query Language.

## Hive

Hive is considered to be a core component of the Hadoop ecosystem. Hive is placed above the Hadoop to provide a summary of data. Hive turns the HiveQL queries into MapReduce jobs. Hive provides a platform to develop SQL scripts for MapReduce operations. Hive brings the traditional data warehousing tools and techniques such as SQL, metadata, partitioning etc.., to the Hadoop eco system [33].

Hive was first introduced by Facebook, and then was moved into apache foundation to be developed as an open source. Hive is not a design for OLTP, rather a more sophisticated design for OLAP. Hive is not a relational database, rather it actually stores schema in a database. Hive is a language for real time queries. It provides SQL type language called HiveQL or HQL [33].

Hive Architecture

Figure 6 shows the major components of Hive and their interaction with HDFS.

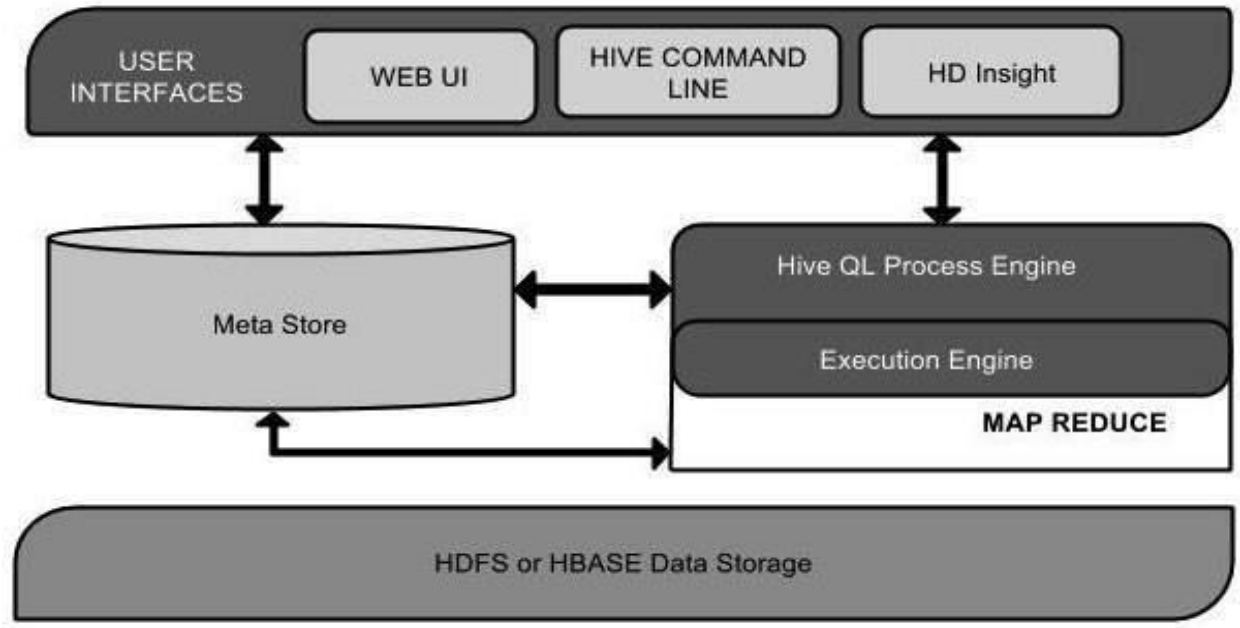


**Figure 6: Architecture of Hive, adapted from [33, 34]**

User Interface: [34] Hive provides user interfaces in the form of Web User Interface and Command Line Interface and also Application Programming Interfaces in the form of JDBC and ODBC.

MetaStore: [34] The MetaStore is like a system catalog for Hive which stores information about tables, partitions, schemas, columns and their types, table locations, etc. This information is stored in an RDBMS system as it needs to be sent to a compiler promptly and can be viewed or modified using the thrift [40] interface. Without MetaStore it is not possible to impose a structure on Hadoop files and so, it is very critical to the Hive system. Hence the information is backed up periodically. Also, it is ensured that the MetaStore server scales with the number of queries submitted by the users. This is done by ensuring that no calls are made from the reducers or mappers of the job. Any Metadata needed by them is passed through xml plan files that are generated by the compiler.

Query Compiler: [34] The Hive query compiler uses the metadata stored in the MetaStore to execute query plans to identify the estimated cost of a query as well as generate an execution plan. Hive compiler processes Hive SQL through the following steps:

- **Parse -** Abstract Syntax Tree (ABT) for the query is generated.
- **Type checking and Semantic Analysis -** The compiler fetches all the information needed from the MetaStore to build a logical plan to convert it into an operator DAG. The AST is converted into a Query Block (QB) where all nested queries are converted into parent child relationships.
- **Optimization** – The optimization logic consists of a chain of transformations. This can be customized or new optimization logic can be added, by adding it to the chain of transformations. The five interfaces involved here are Node, GraphWalker, Dispatcher, Rule and Processor. Nodes in the operator DAG are implemented by the Node interface. Every general transformation involves walking the DAG and for every Node, check if the Rule is satisfied and then invoke the corresponding Processor for the Rule. The mapping from Rule to Processor is maintained in the Dispatcher.

Under the optimization stage the following transformations are done in Hive:

- **Column trimming:** Only columns that are needed in the query processing are projected.
- **Predicate pushdown:** Predicates are pushed down so that rows are filtered early in the processing.
- **Partition trimming:** The files of partitions that do not satisfy the predicate are trimmed.
- **Map-side joins:** If the tables are very small in a join, they are replicated in all the mappers and joined with other tables.

- **Join reordering:** Join operation does not exceed the memory limit on the reducer side as larger tables are not saved in the memory of the reducer.

Execution Engine: [33] Tasks are executed in the execution engine in the order of their dependencies i.e. a task is executed only if all of its prerequisites have been executed. Any map/reduce task initially serializes its part of the plan to a plan.xml file which is added to the job cache of the task. Each of the ExecMapper and ExecReducer deserializes the plan.xml file and the relevant part of the operator DAG is executed. These results are temporarily stored in a location. For DML's, the final result is moved to the target location after the end of the entire query. For other queries, the data is sent as is from the temporary location.

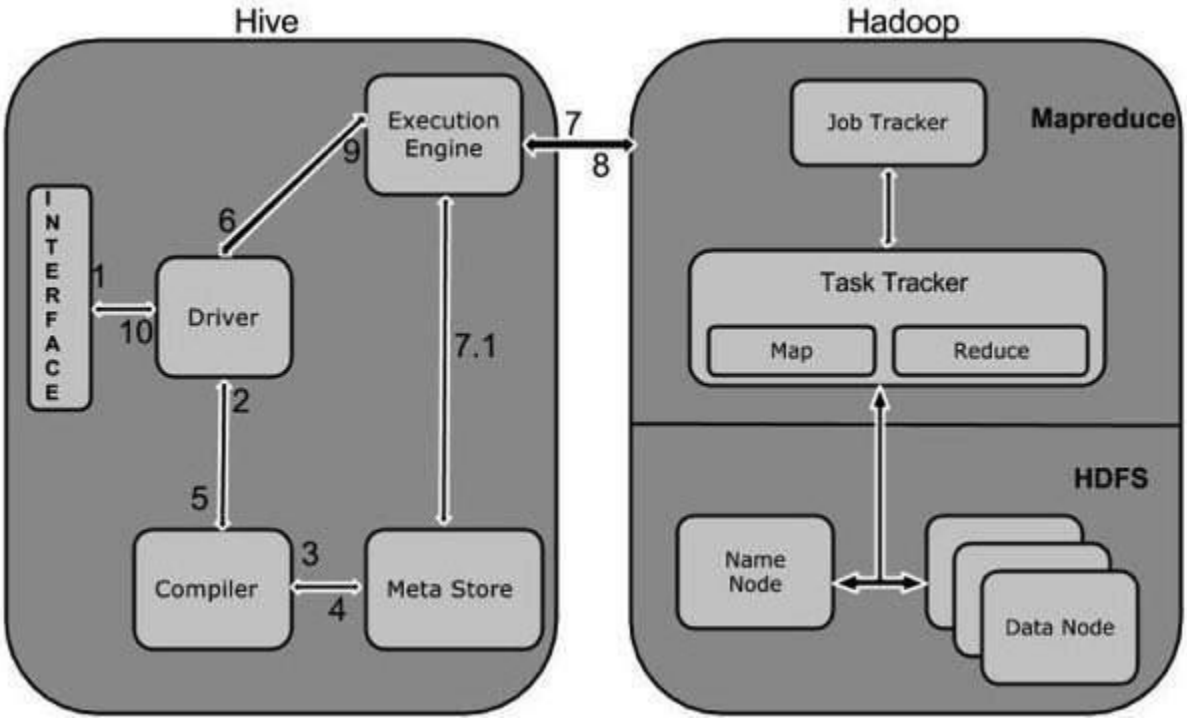


**Figure 7: Data Flow in a Hive – Hadoop environment,** adapted from [33]

Figure 7 provides the data flow in a Hadoop with Hive environment in the form of incrementing numbers starting from 1. Hive GUI interfaces or Hive command line sends a query to the driver

(ODBC, JDBC, etc.) asking it to send instructions to execute the query. Then the driver ensures the request is sent to the compiler. Compiler, using the query syntax, compiles the query to obtain the query plan as well as the cost estimates. Then it forwards metadata request to the MetaStore with the respective instructions pertained to the query attributes. Meta Store looks up and returns the Metadata information back to the compiler. Compiler checks and forwards the plan back to the driver which is eventually forwarded to the execution engine. The engine passes the information to the Hadoop machine where the query executes MapReduce job. The execution engine receives the resultant data from the data nodes and sends them to the driver which is eventually forwarded to the Hive interfaces [33].

Hive Data Model [33]

The data model of Hive is similar to any traditional database system data model. Hive structures the data in the form of tables, columns, rows, tuples and data partitions. Apart from supporting the common data types such as integers, floats, doubles and strings, Hive comes with a complex list of data types which are explained in the next section. Although Hive provides an additional feature by combining multi set of data types to make it look more complex, this is hardly used and implemented. Hive allows a user to design new data types and custom functions. The following sections will drive through the details and comparisons of data types on Hive as well as on other database management systems. Hive stores the data into the table without having to transform it.

Hive Data Types [33]

Data types in Hive are divided into primitive data types and complex/type constructors. Primitive data types include TINYINT, SMALLINT, INT, BIGINT, FLOAT, BOOLEAN, DOUBLE,

STRING, BINARY, TIMESTAMP. Complex types include ARRAYS, LIST, STRUCT, and MAPS.

Complex types syntax:

- ARRAY < primitive-type >
- MAP < primitive-type, data-type >
- STRUCT < col-name : data-type, ... >
- Associative arrays – map<key-type, value-type>
- Lists – list<element-type>
- Structs – struct<file-name: field-type,... >

Associative arrays map strings to structs that in turn contain fields. For example, consider two integer fields named p1 and p2. "*list<map<string, struct<p1:int, p2:int>"* [33] represents the list of associative arrays. Data in associative arrays and lists can be accessed using '[]' operator. Query expressions within the constructive type structs must access the data in it using a '.' operator. All the above described primitive and constructive data types can be used to create a database table.

```
CREATE TABLE EXAMPLE1
(
x string,
y float,
z list<map<string,
struct<a1:int,
a2:int>>
);
```

In the above example adapted from [33], EXAMPLE1.z[0] gives the first element of the list and EXAMPLE1.z[0]['key'] gives the struct associated with 'key' in that associative array. The a2

field of this struct can be accessed by EXAMPLE1.z[0]['key'].a2. Hive contains default serializers and deserializers which help to serialize and deserialize a table. However, before making any changes to these settings, one must ensure the correct source of data, to lessen the impact of failures since the data could come from certain programs or even legacy scripts written in C, Java, etc. which are coded to extract data from other sources. Hive incorporates data into the table without having to transform the data by providing a jar that implements the SerDe java interface to Hive. In basic terms, any arbitrary complex types can be plugged into Hive by providing a jar file that contains the implementation methods for the SerDe and ObjectInspector interfaces. The following syntax adapted from [33] adds a jar containing the SerDe and ObjectInspector interfaces and then proceeds to create the table with the custom serde.

add jar /jars/mysample.jar;

```
CREATE TABLE EXAMPLE2
ROW FORMAT SERDE 'com.mysample.MySerDe';
```

Hive Query Language [33]

Query set of a Hive query language (HiveQL) is similar to the ones on a traditional SQL. Any user with minimal SQL notion will be able to run queries on a Hive command line interface. Query set in a HiveQL includes select clauses, where (restrict/filter) clause, group bys, aggregations, unions, join operations which includes inner, outer, left outer and right outer, as well as several other complex primitives. HiveQL has a different set of commands unlike traditional SQL to describe tables, and other database objects. Query Explain plans are readily available for each query that is run against the Hadoop although the interpretation is different from a traditional SQL plan file.

Query language in Hive has the following drawbacks:

- Joins need to be specified only using ANSI join syntax.
- Join operations in HiveQL supports only equality predicates.
- Insert operation overwrites existing data.

Data loaded periodically into a data warehouse server resides in each of the data partitions. Newly loaded data resides in an empty partition space unless guided to move across a different partition space. Since Hive does not support INSERT INTO semantics, this can be a huge drawback when loading the data. With frequent data loads the size and the number of partitions increase rapidly. To resolve this constraint, old data must be overwritten, deleted or updated. Since Hive does not support these features, read and write operations can be performed easily without implementing any complex locking protocols that come with performing INSERT INTO, UPDATE and DELETE operations.

As mentioned earlier, only equality predicates are supported in a join predicate and the joins have to be specified using the ANSI join. Table 1 compares the Join queries between Hive and traditional SQL adapted from [33]:

**Table 1: HiveQL syntax vs Traditional SQL syntax,** adapted from [33]

| **HiveQL Syntax:** | **Traditional SQL:** |
|---|---|
| `SELECT EXAMPLE1.a1 as COL1,`<br>`  EXAMPLE2.b1 as COL2`<br>`FROM EXAMPLE1 JOIN EXAMPLE2`<br>`ON (EXAMPLE1.a2 =`<br>`EXAMPLE2.b2);` | `SELECT EXAMPLE1.a1 as COL1,`<br>`  EXAMPLE2.b1 as COL2`<br>`FROM EXAMPLE1, EXAMPLE2`<br>`WHERE EXAMPLE1.a2 =`<br>`EXAMPLE2.b2;` |

One of the biggest drawbacks in Hive is the 'Insert' statement functionality. Instead of inserting new records into an existing table, Hive overwrites the entire table when doing insert operations. The following statement adapted from [33] is the syntax for performing Insert operation in Hive.

```
INSERT OVERWRITE TABLE EXAMPLE1
SELECT * FROM EXAMPLE2;
```

Data Storage: HDFS holds the data pertaining to a table. A table may be partitioned or non-partitioned. As per Hive, tables are logical data units which map the metadata information to the HDFS directories. For example, a sample table *employees* gets mapped to *<warehouse_root_directory>/employees* in HDFS. The *hive.metastore.warehouse.dir* configuration parameter defines the *warehouse_root_directory* in the 'hive-site.xml'. A table created on a HDFS file system can come in two types, partitioned and non-partitioned. A partitioned table can be created using the syntax shown in the below example adapted from [33]. The table partitions are stored in HDFS under the directory path /user/hive/warehouse/{table_name}. For every unique row (or value/tuple) there is a partition which exists on the HDFS file system.

```
CREATE TABLE employees
(
     C1 int,
     C2 string
)
PARTITIONED BY (date string, hour int);
```

For the above example, a partition exists for every unique value of 'date' and 'hour' fields given by the user. Partitioning columns are not part of the table data but the partition column values are encoded in the directory path of the partition. Partitioned data is often used to distribute the data in a horizontal manner. Organizing the data in this fashion has great performance benefits.

Partitioning tables alter how a Hive compiler can structure the data storage. One of the critical drawbacks is maintaining a large number of partitions. This causes an overhead to NameNode since it must keep all the metadata information.

To add a new partition to the existing table, Either INSERT or ALTER statement can be used as given in the example below adapted from [33]. The only difference is the INSERT operation adds a new partition along with the data from the existing table 't', whereas the ALTER statement creates an empty partition. Both the operations create new directories in the HDFS directory - /user/hive/warehouse/employees/date=<date string1>/hour=<hour1> and /user/hive/warehouse/employees/date=<date string2>/hour=<hour2>.

Insert Statement:

```
INSERT OVERWRITE TABLE
employees PARTITION(date='<date string1>', hour=<hour1>)
SELECT * FROM t;
```

Alter Statement:

```
ALTER TABLE employees
ADD PARTITION (date='<date string2>', hour=<hour2>);
```

Partitioning offers great benefits when they are less in number. Partitioning segregates Hive tables into multiple partitions aka multiple directories/files. Partitioning yields significant results only when there are limited (and less) numbers of partitions of comparatively equal sized. Though partitioning method has great benefits there is still a critical drawback in it. For example, say, a table is partitioned based on the country. Larger countries contribute to large data sets whereas smaller countries contribute to smaller data sets. This makes the partitioning concept unequal.

To overcome this problem, Hive has come up with a new concept of segregating and managing data in a more beneficial way – bucketing. Bucketing concept is based on hashing function on a bucketed column divided by the total number of buckets. Hash function is decided by the type of the chosen bucketing column. CLUSTERED BY clause is used to define the bucketing column in a table. Records are distributed based on the value of the bucketing column and are distributed almost equally across all the buckets. Bucketing provides efficient sampling of data. Bucketing should be implemented when you have a large data set. Map-side joins are faster on bucketed tables than non-bucketed tables. Map-side joins maps the matching rows on the left and right side of the tables and throws them in their corresponding bucket. When defining a bucket, a user needs to specify the number of buckets along with the column on which the data needs to be segregated in a bucket.

File Formats: The type of file format defined in a Hadoop environment specifies how the records are stored in a file. There are currently two types of file formats – TextInputFormat and SequenceFileInputFormat. Text files are stored in the TextInputFormat and the binary files are stored in the SequenceFileInputFormat. Though these are the two defined standard formats, Hive does not restrict the users from creating custom formats to store the records. The format needs to be defined along with the table definition.

Going back to the data warehousing process where most of the data entering would be structured, this type of data can also be called Big Data because of the huge volume of data and the velocity at which it arrives. In this kind of Big Data environment that includes various front end and back end appliances which support the overall functionality of the environment, there is always a need and requirement to install high end performance servers. Since the data generated is in huge volumes, the appliance will need to handle multiple requests at any given time. There

is a need to implement a data warehouse server, an appliance, capable of handling concurrent multiple requests. The following section explains the architecture, functionality and workload management techniques in Netezza aka PureData System for Analytics™ which is an example for a data warehouse appliance in my paper.

## Netezza

Netezza was first introduced in the year 1999 by Jit Saxena and Foster Hinshaw. Soon, International Business Machines (IBM) Corporation acquired Netezza in the year 2010 for $1.7 Billion. Netezza supports Hadoop, Java, C++, Python and MapReduce programming models which is favorable for any industry trying to adapt Big Data by setting up a Hadoop server along with a data warehouse server like Netezza in parallel. Netezza requires a very minimal to absolutely no tuning requirement. Maintenance level is very low when compared against the current traditional database servers. [28]

TwinFin Architecture of Netezza

Netezza's TwinFin model integrates server, database, storage and advanced analytics into a single system. It is designed to run complex analytics faster on large volumes of data [42]. Netezza adapted to the TwinFin model in the year 2009 [44]. Figure 8 shows the Netezza's TwinFin architecture.

The key building blocks of the Netezza appliance are:

Netezza Host: The Netezza Host is a High Availability Linux Server optimized for high performance. It acts as an interface between the appliance and the external applications. . *NzLoad* is a feature in Netezza to load data from external sources into the appliance. Also, external tools like IBM's DataStage or other ETL tools can be used to load or retrieve data [27].

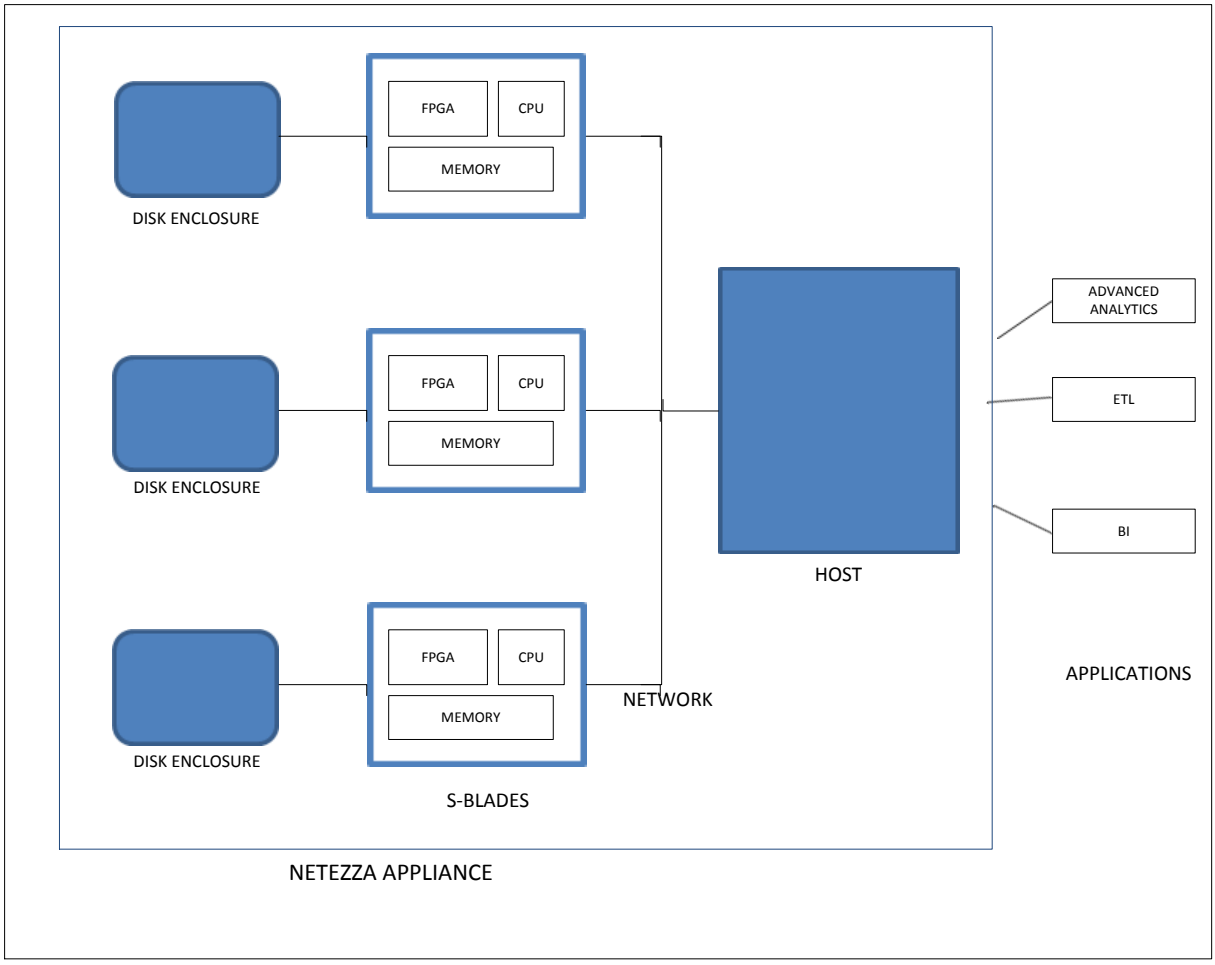


**Figure 8: Netezza TwinFin Architecture, adapted from [27]**

It has two Red Hat Linux operating systems where one is primary and the other secondary which acts as a backup. The secondary OS will be activated when the primary OS has either failed or is corrupted. The Netezza Host compiles SQL queries into executable code blocks called Snippets which is the language Netezza understands. For each query that is submitted, Netezza initially creates a plan to optimize the query before running it which betters the query and engine performance. The Snippets and the query plan are distributed to the multiple parallel processing units by the Host for execution by multiple disks in parallel. The Host also returns the final results back to the requesting application [28].

Snippet Blades (S-Blades): [27] S-Blades or Snippet Processing Units (SPUs) are the MPP nodes that make up the core of the Netezza appliance. Each S-Blade is an individual server which consists of high processing multi core CPU, multi engine FPGA and a multiple Gigabyte RAM. The CPUs are designed to run complex algorithms on large data sets. All operations that can be performed in parallel including query processing, parsing, filtering etc. are performed by the S-Blades. This reduces the amount of data moved within the system and increases the overall performance. Other operations such as sorts, joins, aggregations etc. can be done either by the S-Blades or by the Host depending on the query execution plan.

Disk Enclosures: Any data loaded into the Netezza appliance is stored in the high performance disks in the Disk Enclosures. These disks are RAID protected and are connected to the S-Blades through the Network Fabric. Disks and S-Blades work together in parallel to produce maximum throughput and performance [27]. Data in a table is split among multiple disks based on a distribution key for that table. Distribution key is a combination of one or more columns of the table based on which the table data is distributed. Data is replicated and maintained on different disks to ensure data availability on failure of one of the disks [28].

Network Fabric: [27] All the Netezza components are connected through a high speed network fabric. Netezza's customized IP protocol utilizes the total bandwidth of the network avoiding congestion. This allows large data transfers at high speeds between multiple nodes and components.

Features of Netezza

The following are the important features of Netezza compared to traditional data warehouse systems.

Performance: [42] The Asymmetric Massively Parallel Processing architecture of the Netezza appliance with a combination of S-blades, high performance disks, high availability server host provides a definite performance advantage over traditional data warehouse systems. It provides exceptionally fast query performance on highly complex workload environments which can involve petabytes of data. The TwinFin model of Netezza brings the complex computation (analytics) to the data which is similar to a Hadoop environment.

Speed: Netezza's massively parallel processing capabilities ensure rapid response to applications that send complex queries to the Netezza appliance. Some of the most commonly accessed data is compressed and stored in cache. This ensures fast retrieval of data instead of accessing disks. Field Programmable Gate Arrays (FPGA) in the S-Blades decompresses this data and filter out 95-98% of the table data keeping only data required to answer the query. FPGA contains embedded engines that perform operations like transformations, filtering, etc. on the data. These are custom configured for every snippet based on the parameters that are provided in the query plan by the Host during query execution [27]. Netezza's newest model integrates Directed Data Processing algorithms (explained in the following section) which increased the throughput to 20 times over previous Netezza models [28].

Simplicity: [42] The TwinFin model of Netezza is always on a ready-to-go state with no indexing or tuning required. Since it is an appliance, all the integration of hardware, software and storage comes right out of the box. It integrates with almost all ETL, BI and analytics applications through Netezza's supporting interfaces: ODBC, JDBC and OLE DB. Redundancy of data is another feature in Netezza which does not impact system performance upon disk failures.

Value: [42] Netezza is a low cost commodity appliance. The installation and implementation costs are also lower as it comes as a *plug and play* device. It also requires minimal administration to maintain a Netezza system. Also, loading large amounts of data into the Netezza appliance does not require any additional tasks to optimize performance.

User Privileges [43]

The user privileges provide the level of privileges a user is assigned to in the Netezza system. The two types of privileges in Netezza are administrative level privileges and object level privileges.

Administrative level privileges include creation of database objects and executing global objects. Object level privileges are restricted to a particular object like a database, a table, etc. A database user inherits the access level privileges from a database group they are assigned to. User authentication is either 'local' or LDAP authentication.

Query Optimizer and Planner [43]

The Netezza Query Optimizer makes sure that smaller tables are before larger tables in any query involving join statements. The Netezza Optimizer depends on a number of statistics about the objects in the query being executed like the number of total rows in the tables involved, the number of columns in the query, etc. This helps the Query Optimizer to determine the best execution plan.

Whenever a query is submitted by an external application to the Netezza appliance, the query is compiled and an execution plan is created by the Netezza Host. The query execution plan has the information relating to the order of operations to be performed, distribution of data over the SPUs either through redistribution (HASH – distribution key specified or RANDOM – no

distribution key specified) or broadcasting. Consider an example of counting distinct users of a retail transactions table (RETAIL_TRANSACTIONS) which is distributed randomly.

```
SELECT COUNT (DISTINCT USER_ID) FROM RETAIL_TRANSACTION;
```

To perform the above query, Netezza has to break up the job as it cannot perform count distinct user_id on each SPU and then add up the numbers at the end. There can be same users on multiple SPUs and counting distinct users on each SPU and adding them at the end can include duplicate users. Therefore, to follow the above procedure, the table has to be redistributed on USER_ID column to avoid duplicate users. If the above table is already distributed on USER_ID, the cost of redistribution could be saved and improve the overall performance. The Netezza system optimizes a query such that larger tables are not redistributed to save the cost of redistribution. This can make a huge impact on the performance of the query.

Workload Management of Netezza

During query execution, the Netezza Host allocates system resources based on the priority given to the resource groups. When the resource groups are assigned equal resources, priority is assigned to the user group or individual user accessing the resources. In an enterprise environment, there will be large complex queries running around the clock pertaining to business needs. These queries utilize most of the system resources affecting the performance of smaller adhoc queries. Hence, Netezza implements certain workload management techniques for managing resource allocations:

Guaranteed Resource Allocation and Resource Groups: [43] A Resource Group in a Netezza system is a group with resource settings that calculates the net system resources (total resources minus resources allocated for special, high priority jobs) and allocates them to the

query plans associated to that group. Each Netezza system has a minimum of one Resource Group. Each user is assigned to only one Resource Group. Whenever a user submits a query, the query planner assigns that particular Resource Group associated with the user to the query.

Each Resource Group has a resource minimum and resource maximum which define the minimum and maximum amount of system resources that can be granted to a Resource Group. It is the responsibility of the Guaranteed Resource Allocation to make sure that each Resource Group gets its resource minimum. GRA is enabled in a Netezza system by default but is not effective until the Resource Groups are defined with their respective resource minimum and resource maximum. Each Resource Group has a priority for queries defined in such a way that short queries have higher priority than longer queries. The GRA scheduler takes care of the scheduling by making sure that high priority queries are run before lower priority queries. This helps in optimizing the performance of the Netezza server.

Priority Query Execution: [43] Even though the GRA scheduler makes sure that resources are allocated correctly to a resource group, there can be situations where, multiple users belonging to the same resource group might be running multiple jobs. There can be both high priority jobs and low priority jobs that are run by the users with the same resource allocation. Netezza implements a feature called Priority Query Execution (PQE) which makes sure that high priority jobs are allocated more resources compared to low priority jobs. Once GRA allocates resources based on Resource Groups, it reallocates the system resources for jobs run by users within a Resource Group. Netezza has six different levels of PQE of which four are available for at user level and two are designed specifically for the Netezza system's administrative purposes.

Short Query Bias: [43] Within each Resource Group, Netezza system provides the functionality of reserving memory resources for short queries. In this way, short queries utilize the additional resources allocated to them when the standard scheduler is busy. Also, short queries do not have to wait for the longer queries to finish which are in queue. A query is defined as a short query if the cost (number of seconds to run the query) estimates of the query is less than the specified threshold (default to two seconds).

Directed Data Processing: [45] Netezza distributes workload automatically to the processing nodes that contains the relevant data. This enables greater processing efficiency. For example consider an item details table in a retail company ITEM_DETAILS distributed on the column ITEM_KEY and notice how the queries are executed simultaneously:

**Query_1:**
```
SELECT ITEM_NAME, ITEM_DESC, INVENTORY_CNT, PRICE FROM
ITEM_DETAILS WHERE ITEM_KEY = 100;
```

**Query_2:**
```
SELECT ITEM_NAME, ITEM_DESC, INVENTORY_CNT, PRICE FROM
ITEM_DETAILS WHERE ITEM_KEY IN (101, 102);
```

The above queries would be serialized as the system assumes it could do only one disk operation at a time. Therefore, Query_1 is executed first. Query_2 is executed only after the completion of Query_1.

But, since the table is distributed on ITEM_KEY, records for ITEM_KEY = 100 is stored on a different data slice, records for ITEM_KEY = 101 are stored on a different data slice and records for ITEM_KEY = 103 are stored on a different data slice for the table ITEM_DETAILS. Hence, while executing the above queries, the system actually goes after records that are on different data slices.

With Directed Data Processing, Netezza system realizes this and the relevant snippets for ITEM_KEY = 100, ITEM_KEY = 101 and ITEM_KEY = 102 are sent in parallel to the S-Blades which are related to the data slices having the item details. So, the above two queries are executed in parallel by the Netezza system. The Direct Data Processing feature is particularly affective while running short queries that executes against small chunks of data and also, the number of queries that run in parallel is also increased without impacting the performance.

# Chapter IV

## DATA PRESENTATION AND ANALYSIS

In this chapter, a brief introduction is illustrated based on how the implementation of Hadoop and Hive changed the overall functionality and performance at Facebook. The following sections describe how the decisions undertaken by Facebook have yielded positive results and the configurations and settings they have developed to adapt to Big Data. Apart from Hadoop and Hive, there is always a need to maintain a parallel processing data warehouse server instead of a RDBMS server at the back end capable of handling hundreds of requests at the same time. The following section also provides a brief report illustrated at the end based on the performance of a BI query run against an Oracle database and a Netezza system.

## Hadoop and Hive At Facebook

Before implementing Hadoop at Facebook, RDBMS systems and traditional data warehouses were used to load, process and store the data. This infrastructure was very inadequate and some daily processing jobs were taking more than a day to process. The situation was getting worse every day as the data coming in was increasing at a rapid pace and data processing was getting slower. Then Facebook implemented Hadoop and the same jobs that were taking more than a day to complete were completed in a few hours. But then programming with Hadoop was not easy especially since Map/Reduce was still fairly new. It is not as expressible as SQL and also Map/Reduce is low level programming. This inspired the developers at Facebook to build Hive [33].

Hive and Hadoop are extensively used for huge data processing purposes at Facebook. As of 2010, Facebook has a 2 PB Hadoop cluster. They add about 5TB of compressed data every day

with a compression ratio of 1:7. This becomes 15 TB after replication in Hadoop. On any given day, more than 7500 jobs run on the Hadoop cluster and more than 75TB of compressed data is processed. The continuous growth in the Facebook network means that the data they process every day keeps increasing constantly [35].

Adhoc queries place more than a fifty percent load on the Hadoop cluster, and the rest of the capacity is for reporting dashboards. This kind of workload management by Hive helps in adhoc analysis. However, resource sharing between adhoc queries and scheduled BI and other reporting queries post significant challenges because of the unpredictability of the adhoc queries. Also some of the adhoc queries might not be optimized and can consume valuable Hadoop cluster resources. Therefore separate clusters are maintained for adhoc queries and scheduled reporting queries as resource scheduling is weak in Hadoop [33].

A wide variety of Hive jobs are run ranging from simple summarization jobs to machine learning algorithms. Overall, the system is able to provide data processing services at a fraction of the cost of a traditional RDBMS [33].

As Hive brings the traditional data warehousing tools which are familiar, it improved the developer/analyst productivity tremendously and created new use cases and usage patterns to Hadoop. The same task that would take hours or days to program can be expressed in a few minutes in Hive as most of it would be programming in SQL. Therefore, more users are using Hive for adhoc analysis at Facebook [34].

Here is one highly simplified application called Status-Meme at Facebook [34]:

When users update their status, the updates are logged into a flat file on NFS directory. `/logs/status_updates`

This data is loaded into the Hive table `status_updates (userid int, status string, ds string)` using the load statement :

```
LOAD DATA LOCAL INPATH '/logs/status_updates' INTO TABLE status_updates PARTITION (ds='<status update date>')
```

The Status_Updates table contains the user information who updates the status, the actual status updated and the date the status was updated. User profile information like gender, school attending is present in the table `profiles (userid int, school string, gender int)`

The frequency of daily updates based on the gender and the school are computed:

The multiple-insert statement in Figure 9 generates the daily counts of status updates by school. HiveQL allows the query results to be entered into a specific partition of the output table by specifying the partition (date of status update in this example).

```
FROM (SELECT t1.status, t2.school
FROM status_updates t1 JOIN profiles t2 ON (t1.userid = t2.userid and t1.ds='<date of status update>')
) query1
INSERT OVERWRITE TABLE school_summary PARTITION(ds='<date of status update>')
SELECT query1.school, COUNT(1) GROUP BY query1.school
```

**Figure 9: Daily Count of Status Updates by School,** adapted from [34]

In this example, to display the ten most popular status updates by users who attend a particular school, the map reduce constructs in Hive are used (displayed in Figure 10). A custom Python mapper script (meme-extractor.py) is executed to parse the result of the join between status updates and profiles tables. This script uses natural language processing techniques to extract the

status strings. For example, consider top10.py which is a simple custom python script to perform aggregation and get the top 10 memes per school.

These can be reused and run as scheduled queries to get the top 10 popular status updates every day from a school. Hive makes the life of a developer/analyst better as it can be programmed quickly as well as effectively without much effort.

```
REDUCE query2.school, query2.meme, query2.cnt
USING 'top10.py' AS (school,meme,cnt)
FROM (SELECT query1.school, query1.meme, COUNT(1) AS cnt
FROM (MAP t2.school, t1.status
USING 'meme-extractor.py' AS (school,meme)
FROM status_updates t1 JOIN profiles t2 ON (t1.userid =
t2.userid)
) query1
GROUP BY query1.school, query1.meme
DISTRIBUTE BY school, meme
SORT BY school, meme, cnt desc
) query2;
```

**Figure 10: Query to get the top 10 popular status updates,** adapted from [34]

Figure 11 provides the order in which the operations are performed for the above Status- Meme example:

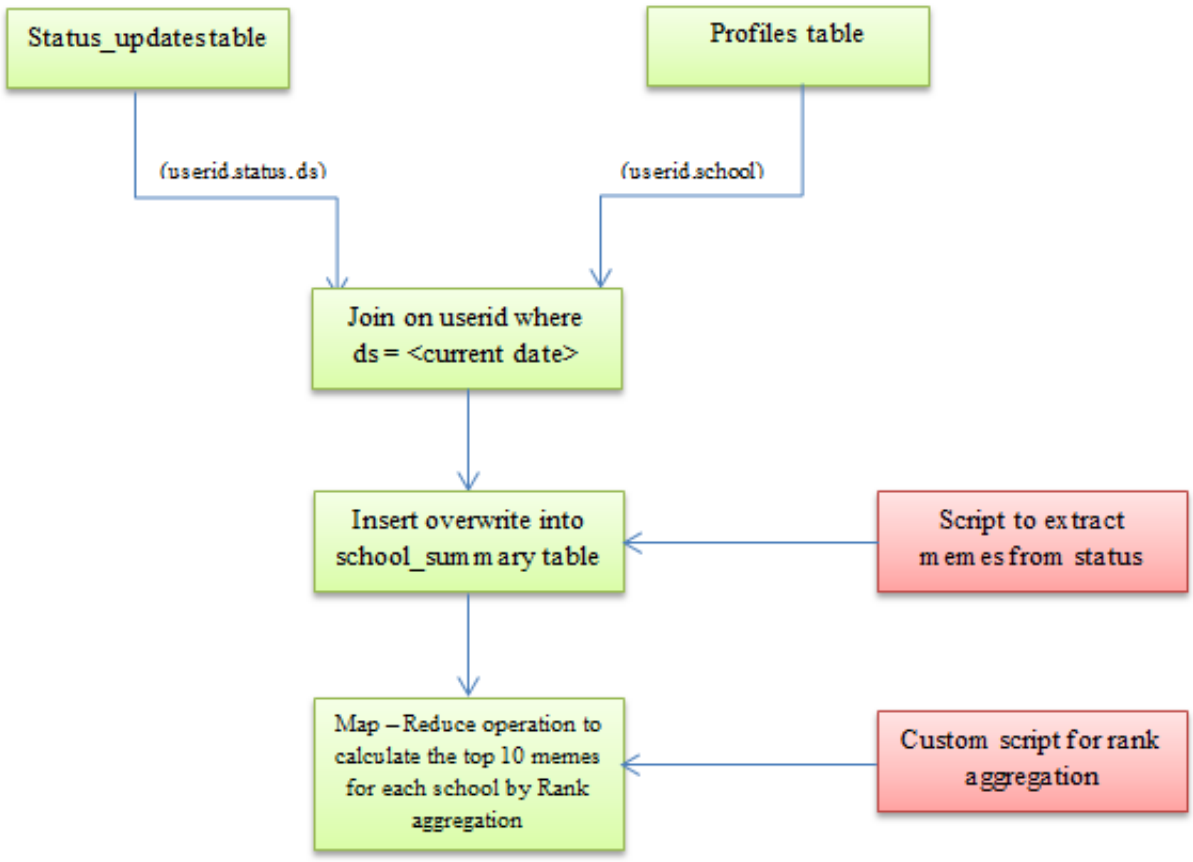


**Figure 11: Calculating top 10 status memes per school**

## Netezza Over RDBMS System

For their parallel processing capabilities, Massively Parallel Processing (MPP) databases like Netezza are desired more favorably and can be utilized to execute complex queries more efficiently. Hence, they are suggested as an accepted choice for typical data warehouse implementations. This allows companies to scale up effortlessly with MPP databases like Netezza. Also, with Netezza, it is possible to have the ability to build and deploy refined analytics models on most recent data that emulate real-world complexities more easily and effectively with better performance. Companies can continuously experiment, expand and tune

analytics to determine trends and to find ways to lower business risk, reduce cost, increase revenues, and make fact based decisions, all with better performance.

To provide a simple example that Netezza has the potential to provide better performance over a traditional RDBMS system like Oracle, I considered a query that generates a BI report to identify the top seven items sold for a given day. This BI query performs a join on four different tables which contain typical inventory and sales information. This report is generated on a daily basis and is used to analyze daily sales information. To perform this experiment, I used the following applications/tools:

- Oracle database version 11.2.0.3 as the traditional RDBMS
- IBM Netezza appliance N1001-005, software version 7.0.4 as the MPP database machine (Table 2 provides the hardware details)
- SQL Developer 4.0.3.16 to connect to the Oracle database
- WinSQL Lite 7.5 to connect to Netezza
- IBM DataStage 11.3.0.1 as the ETL tool to load the data from a file into the tables of both of the data systems defined.

Due to the sensitivity and propriety nature of the data owned by the company, data in the tables was not provided in this paper. Some of the information on the figures below will be masked and/or not provided as deemed by the company with prior approval from the director of Data Integration team. In the experiment, I will record and analyze the execution time of this BI query. This experiment is performed only to support the idea of having a MPP machine like Netezza instead of a RDBMS system like Oracle DB in the process of Data Warehousing involving Big Data in an enterprise. More information comparing architectures of Oracle and Netezza is provided in [41].

**Table 2: Netezza N1001-005 Hardware configuration**

| **Netezza N1001-005 components** | **Capacity/Quantity** |
|---|---|
| Number of Racks | 1 |
| Active S-blades/SPUs | 7 |
| CPU cores | 56 (2 * 4 core Intel CPUs per blade) |
| FPGA cores | 56 |
| Total disk space | 64 TB |
| Number of disks | 48 |
| Operating System | Red Hat Enterprise Linux 5.9 |

Table 3 provides the number of records for each table involved in this experiment:

**Table 3: Tables with record counts**

| TABLE | RECORD COUNT |
|---|---|
| TABLE_1 | 7597433 |
| TABLE_2 | 0 |
| TABLE_3 | 5876654 |
| TABLE_4 | 1079 |

Figure 12 shows the time taken to load TABLE_1 in Oracle 11g through DataStage. This was 13:12 minutes.

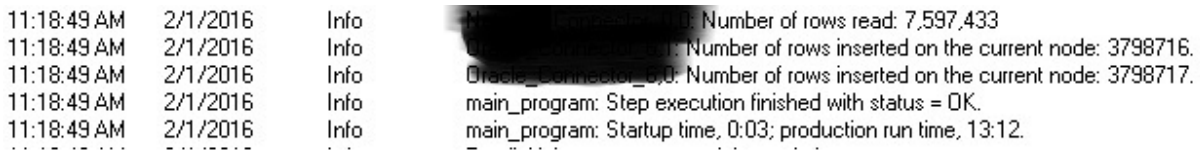


**Figure 12: Load of TABLE_1 successful on Oracle db**

Figure 13 gives the time it took to load TABLE_1 in Netezza through DataStage which was 7:50 minutes.


**Figure 13: Load of TABLE_1 successful on Netezza**

We can observe a difference of 5 minutes to load 7.6 million records. Such difference can be very critical in a real data warehousing environment where records on the order of billions need to be processed every few hours.

After running the BI query:

Figure 14 shows the query run time on Oracle database which is close to 53 seconds.


**Figure 14: Execution time of the BI query on Oracle db**

Figure 15 gives the query run time on a Netezza machine which took around 6 seconds.


**Figure 15: Execution time of the BI query on Netezza**

Here we can see a difference of more than 45 seconds for a very simple query that retrieves only 7 records and with an increase in data, this time difference is going to increase exponentially. This difference is vital as many of these queries are executed on a daily basis.

This experiment has been performed multiple times and the results can be seen in Table 4.

**Table 4: BI query run time on both machines**

| BI Query run time on Oracle | BI Query run time on Netezza | Difference |
|---|---|---|
| 52.46 seconds | 5.83 seconds | 46.63 seconds |
| 53.04 seconds | 5.87 seconds | 47.17 seconds |
| 52.96 seconds | 5.88 seconds | 47.08 seconds |
| 52.37 seconds | 5.87 seconds | 46.50 seconds |
| 52.68 seconds | 5.79 seconds | 46.89 seconds |

Analytics that once seemed impossible or impractical to run are now possible with Netezza. The capability to analyze data, foresee outcomes and find ways to improve business is driving companies to fully exploit advanced analytics. Making sense of enormous volumes of data and turning it into meaningful results can be overwhelming or even technically impractical in companies with traditional databases.

# Chapter V

## IMPLICATIONS AND CONCLUSION

After looking at the above statistics, I think that Hadoop and/or a Massively Parallel Processing system should have a definite presence in the data warehouse architecture of any enterprise.

Considering the existing data warehouse architecture to be similar to Figure 2, there are four ways in which Hadoop can be added to that architecture based on functionality, usability and the nature of data and BI. This addition of Hadoop to a data warehouse can make it cheaper, more scalable and very efficient.

### Hadoop for Cold Data Storage

Current data warehouses typically hold data as old as ten years which might no longer be used frequently. The older the data becomes, the lesser it is used. Therefore, data can be classified as cold, warm or hot where hot data is the data that is used every day, and cold data is the data that is occasionally or never used. Having this cold data in the database decreases the speed of data retrieval as, size of a table can have huge impact on the performance. In this case, Hadoop can be used to store that cold data as this form of storage is cheaper and is also accessible. The data warehouse in this scenario is technically the combination of Hadoop and a MPP appliance/database. Cold data from the MPP appliance can be pushed into the Hadoop system periodically [20].

Figure 16 represents this architecture.

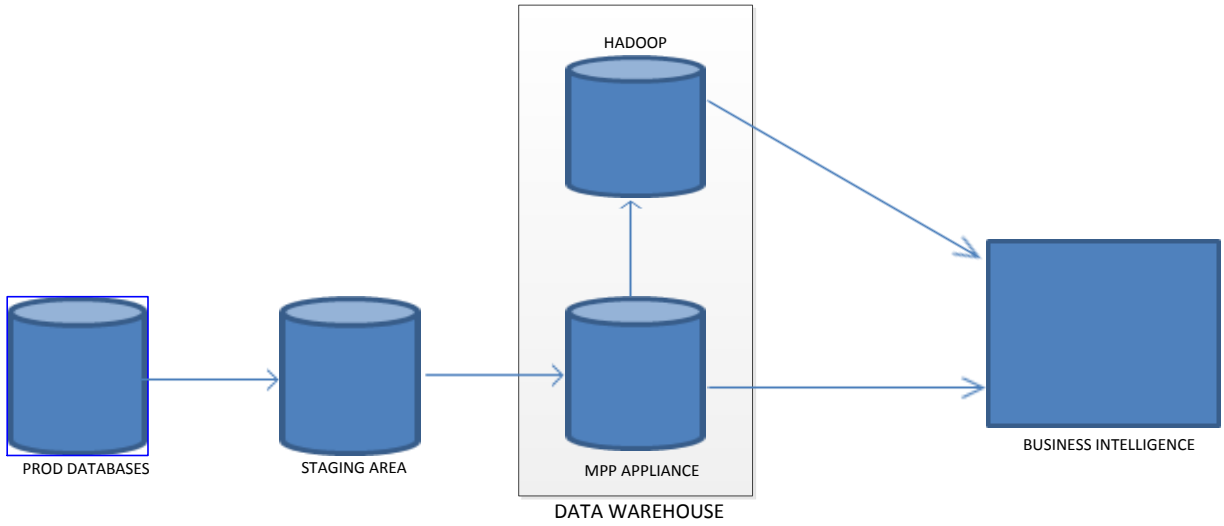


**Figure 16: Architecture using Hadoop as a cold storage, adapted from [20]**

Real time Example: Hortonworks along with Composite software (which is now part of Cisco) developed storage architecture for a global investment bank to archive cold (old) data that is still accessible in a cost effective manner. The bank was looking to reduce risk in the credit business by identifying risk trends in the past five years while also making profitable credit decisions. They could not meet the expense to maintain the entire past five years of data in the data warehouse as that can impact the performance of the data warehouse. Likewise, the old method of archiving old data on tapes makes it harder to perform business analytics on that data. To address this issue, Cisco developed a platform that can perform data virtualization on both the data warehouse (that consists of market data which is hot data) and Hortonworks data platform which is Hadoop (that consists of the cold data). This platform seamlessly integrated data between the two data stores and made it accessible to business intelligence tools. This architecture allowed the bank to perform predictive analysis based on old and current data in a fast and cost effective manner [46].

Hadoop as an extra data warehouse

In the first approach, Hadoop consists of only those tables that are in the MPP appliance. Tables that are not in the MPP appliance will not be in the Hadoop system. In this approach, both data stores contain different sets of tables and different data. Data is copied directly from the production database into the Hadoop system. Even business intelligence reports extract data directly from the Hadoop system itself. In this approach also, data warehouse is a combination of both Hadoop and MPP appliance [20].

Figure 17 represents this kind of architecture.

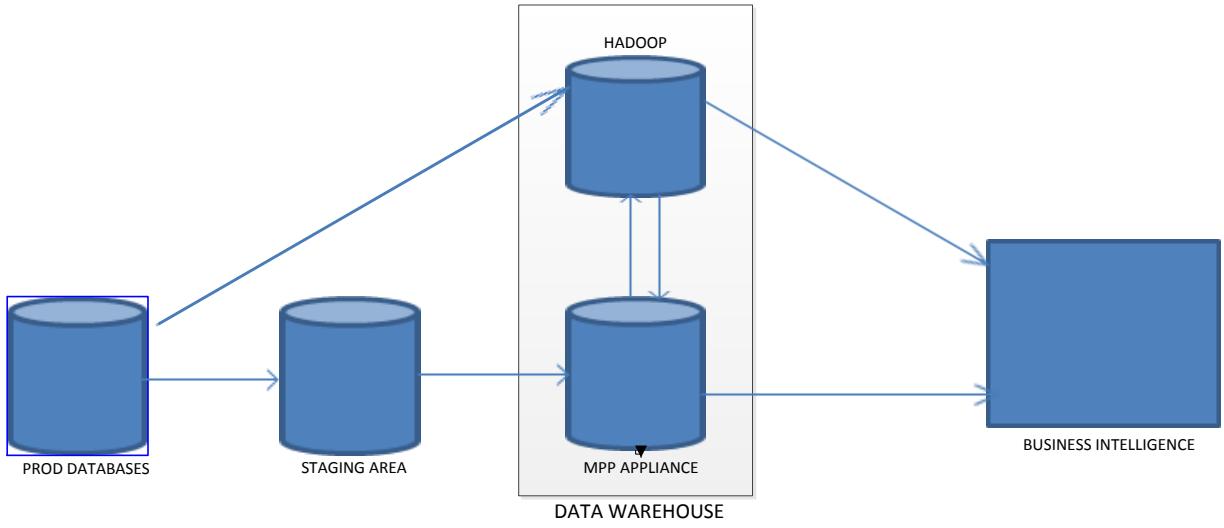


**Figure 17: Architecture using Hadoop as an extra Data Warehouse, adapted from [20]**

This architecture can be implemented where a huge amount of new data is generated and MPP databases in the staging area cannot handle such data.

Real time Example: The example of how Facebook is using Hadoop to collect and process every day data has already been discussed in this paper. Hadoop along with Hive made the

development of programs easy and efficient. Any processed and refined data that is needed for analytics is moved to the database for faster data retrieval.

Hadoop for ETL processing

Having huge amounts of data in a data warehouse can slow down the performance. In those situations, data can be aggregated and then loaded into the warehouse. Hadoop will be the data store which maintains all the detailed data and Map/Reduce operations are performed on it to aggregate data and load to the data warehouse. The data is loaded directly into Hadoop, and it does all the processing. This architecture is particularly useful in case of loading multi-structured data. A schema on read is performed to develop the schema while reading and loading the data. Map/Reduce operations are used to generate the schema that is used while loading the data into the MPP appliance [20].

Figure 18 represents this architecture.

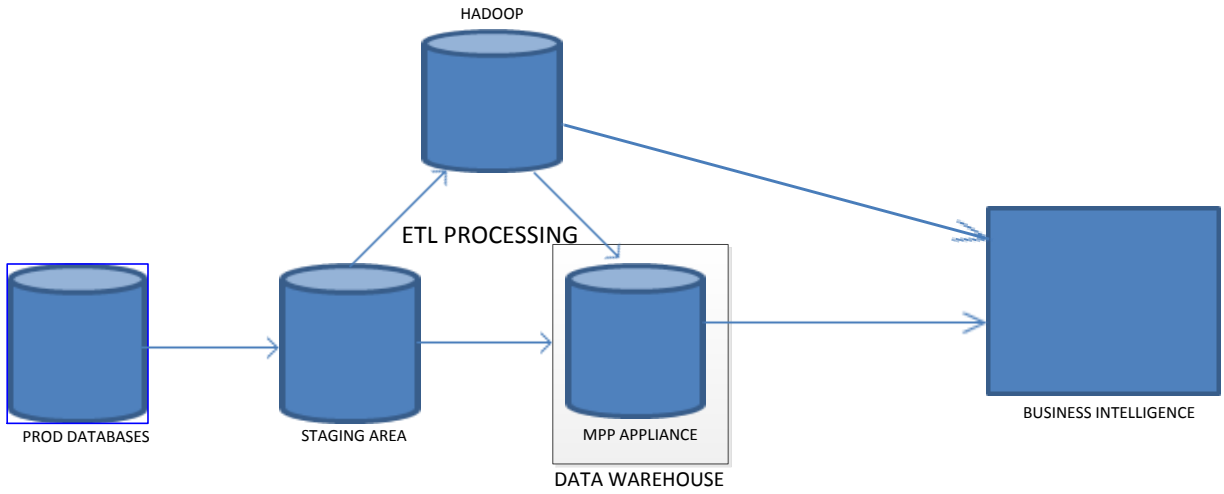


**Figure 18: Architecture where Hadoop being used for ETL processing, adapted from [20]**

Real time Example: Intel has mentioned various advantages of using Hadoop for ETL processing. Hadoop is a 'no schema on write' which means, the data schema does not need to be predefined before loading and so, any type of data (unstructured or structured) data can be loaded into Hadoop. Once the data is loaded into Hadoop, regular ETL tasks like normalizing, aggregations, cleansing, etc. can be performed utilizing the Map Reduce scalability. This process helps in performing ETL tasks rapidly and also avoids any transformation bottlenecks. Analytics can be performed directly on the Hadoop data as well without having the necessity to move the data to the data warehouse. Using Hadoop for ETL processing reduces the overall operational cost of data warehousing [47].

Hadoop as a staging area

Figure 19 shows architecture with Hadoop as the staging area in a Data Warehouse environment. New data is loaded into Hadoop (staging area). Then it is transformed and loaded into the MPP appliance of the data warehouse. One of the main capabilities of Hadoop is being able to load a huge amount of data rapidly and hence this would suit perfectly. Data is then extracted by a batch oriented approach. Data can also be transformed if needed using the map/reduce operations. Hive can completely replace the data warehouse in this scenario and the processed data can be loaded back to Hadoop on which the BI operations can be performed [20].

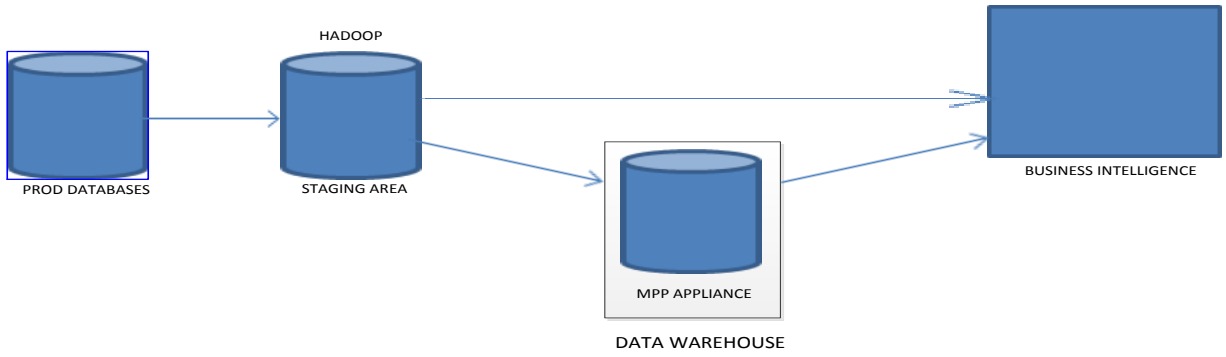


**Figure 19: Architecture using Hadoop as a staging area, adapted from [20]**

Real time Example: The CEO of a supply chain company of many manufacturers wanted to estimate the impact of recent floods on quarterly earnings in a short span of time. The analytics team needed access to various structured and unstructured data sources rapidly. They accessed not only their only supply chain data but also data from top 25 suppliers. They also had to augment this data with weather forecast and road conditions. But all this data being accessed is never needed in their data warehouse as a situation like this does not occur frequently. Hence they used Hadoop which is a staging area to host this data and performed analytics on that data. Once the analysis was completed, the analytics team archived information that was useful in the future. Similarly, Hadoop as a staging area can load huge amounts of unstructured data rapidly and later this data can be processed and moved to data warehouse [48].

In each of the above four approaches, the MPP appliance/database in the data warehouse can be a massively parallel processing appliance like IBM's Netezza or Oracle's Exadata, etc. For example, if the data that is being loaded is huge but mainly structured data, the staging area or the entire data warehouse can be replaced with these systems as they can process billions or even trillions of records with ease.

To conclude this section, the study questions imposed earlier in this paper will be answered.

1. ***How big is the Big Data problem and is it worth the time and effort of the organizations trying to solve Big Data issues?***

   Data analytics has become hugely popular and collecting each and every particular in a business and analyzing it appropriately has become a major challenge in the data warehousing world. And with this data being unstructured or semi-structured, loading, managing, extracting and storing this data became a huge concern. But successful extraction of this data and careful analyzation has helped in improving businesses in many ways and this has been proven by a few enterprises. So, the time and effort put forth by organizations in trying to solve the Big Data issues is definitely worth it.

2. ***What is Hadoop and can it solve the Big Data problem?***

   The earlier sections of this paper gave a clear insight about Hadoop, it's origin, the technology behind it and it's features. With all these, Hadoop can definitely take care of the problems faced by data warehouses with Big Data provided it is implemented in a correct manner. In many cases, Hadoop can be more efficient if implemented with one or more technologies in the Hadoop eco system as Hadoop can only perform map/reduce operations and these have to be coded at a low level. Also, developing programs in a technology like Hive can make the programmer's life easy as it's HiveQL is very similar to SQL which most of the programmers working on databases or data warehousing will be familiar with.

3. ***Can Hadoop run together, in parallel with the existing data warehouse systems?***

Hadoop can run in parallel with an existing data warehouse, with a MPP appliance, a SQL database or one or more of it's eco system technologies using one of the above provided data warehouse architecture approaches. By adding Hadoop to any data warehouse system, it definitely is going to provide more efficiency at a lesser cost. Apart from that Hadoop can also be used to store massive amounts of data and BI reports can run directly on the Hadoop cluster. Also, Hadoop can be used to archive data which helps reduce costs in many ways and makes data retrieval easy.

## Conclusion

By the addition of Hadoop or Hadoop along with its eco system of either Hive or HBase etc., data warehousing has been proven to be more efficient, cost effective and scalable. If most or all of the data that is coming into a data warehouse is structured or transactional data but in huge volume, enterprises can turn to MPP appliances like IBM Netezza, Oracle Exadata, etc. Also, four different approaches were provided in which Hadoop can be used in optimizing the whole data warehousing process. Furthermore, these approaches have been proven successful by many large enterprises like Facebook, EBay, Yahoo, Amazon, etc.

For the most part companies are reluctant when it comes to implementing BIG DATA as its too early for most organizations and there is a fear of project failure coupled with the cost of implementing BIG DATA. On the other hand companies have already invested vast amounts of resources and money on Business Intelligence to drive meaningful insights into their existing data for decision making. Traditional Data warehouse and Business Intelligence have been proven to be a success for most organizations over BIG DATA which is fairly in its early stages.

## References


1. Cukier, K. (2010, February 25). Data, data everywhere. Retrieved Spring, 2015, from http://www.economist.com/node/15557443
2. Dave, P. (2013, October). 2013 - Journey to SQL Authority with Pinal Dave. Retrieved from http://blog.sqlauthority.com/2013/10/30/big-data-learning-basics-of-big-data-in-21-days-bookmark/
3. Soubra, D. (2012, July 5). The 3Vs that define Big Data. Retrieved from http://www.datasciencecentral.com/forum/topics/the-3vs-that-define-big-data
4. Inmon, B. (2014, April 19). Evolution of Business Intelligence. Retrieved from http://www.forestrimtech.com/evolution-of-business-intelligence
5. Patil, S. (2014, May 5). Evolution and transformation of Business Intelligence (BI). Retrieved from http://scn.sap.com/community/business-intelligence/blog/2014/05/05/evolution-tranformation-of-business-intelligence
6. Wayne E. (2014, Sept 8). The New Analytical Ecosystem: Bridging the Worlds of BI and Big Data. Retrieved Spring 2015 from: http://tdwichapters.org/Blogs/Minneapolis/2014/08/The-New-Analytical-Ecosystem.aspx
7. Turner, J. (2011, January 12). Hadoop: What it is, how it works, and what it can do. Retrieved from https://www.oreilly.com/ideas/what-is-hadoop
8. Dean, J., & Ghemawat, S. (2008, January). MapReduce: Simplified data processing on large clusters. *Communications of the ACM - 50th Anniversary Issue: 1958 - 2008, 51(1),* 107-113.

9. Rao, R. (2013, January). When to Use Hadoop Instead of a Relational Database Management System (RDBMS). Retrieved March, 2015, from http://quaero.csgi.com/blog/464-when_to_use_hadoop_instead_of_a_relational_database_management_system_rdbms

10. Shvachko, K., Kuang, H., Radia, S., & Chansler, R. (2010). The Hadoop Distributed File System. *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*. Retrieved from http://dl.acm.org/citation.cfm?id=1914427

11. Gartner. (2013, September 23). *Gartner Survey Reveals That 64 Percent of Organizations Have Invested or Plan to Invest in Big Data in 2013* [Press release]. Retrieved February, 2015, from http://www.gartner.com/newsroom/id/2593815

12. Sicular, S. (2014, July 29). Big Botched Data [Web log post]. Retrieved February, 2015, from http://blogs.gartner.com/svetlana-sicular/big-botched-data/

13. Talend. Four Key Pillars To A Big Data Management Solution [White paper]. Retrieved February, 2015, from http://tdwi.org/whitepapers/2013/05/four-key-pillars-to-a-big-data-management-solution/asset.aspx?tc=assetpg

14. Elliot, T. (2014, April 7). No, Hadoop Isn't Going To Replace Your Data Warehouse [Web log post]. Retrieved from http://timoelliott.com/blog/2014/04/no-hadoop-isnt-going-to-replace-your-data-warehouse.html

15. Sagiroglu, S., & Sinanc, D. (2013, May). Big data: A review. *Collaboration Technologies and Systems (CTS), 2013 International Conference on*. Retrieved from http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6567202

16. Guster, D., O'Brien, A. Q., & Lebentritt, L. CAN A DECENTRALIZED STRUCTURED STORAGE SYSTEM SUCH AS CASSANDRA PROVIDE AN

EFFECTIVE MEANS OF SPEEDING UP WEB ACCESS TIMES. Retrieved from http://micsymposium.org/mics_2013_Proceedings/submissions/mics20130_submission_2.pdf

17. *Big Data: What's holding you back?* [Pdf]. (2013, April). Talend.
18. Mearian, L. (2012, December 11). By 2020, there will be 5,200 GB of data for every person on Earth. Retrieved from http://www.computerworld.com/article/2493701/data-center/by-2020--there-will-be-5-200-gb-of-data-for-every-person-on-earth.html
19. Computing. (n.d.). Retrieved from http://home.cern/about/computing
20. Van der Lans, R. F. (2013, September). *Data Warehouse Optimization: Embedding Hadoop in Data Warehouse Environments* [PDF]. Talend. Retrieved from http://info.talend.com/rs/talend/images/WP_EN_BD_DataWarehouse_Optimization.pdf
21. Krenzel, S. (n.d.). MapReduce: Finding Friends. Retrieved from http://stevekrenzel.com/articles/finding-friends
22. White, T.(May 2012), *Hadoop: The Definitive Guide*, O'Reilly Media. Retrieved Spring, 2015, from: http://download.bigbata.com/ebook/oreilly/books/Hadoop.The.Definitive.Guide.3rd.Edition.May.2012.pdf
23. Hadoop Ecosystem | Think Big Analytics. (n.d.). Retrieved from https://thinkbiganalytics.com/leading_big_data_technologies/hadoop/
24. IBM (n.d.). What is Pig? Retrieved September, 2015, from https://www01.ibm.com/software/data/infosphere/hadoop/pig/

25. IBM (n.d.). Apache Hive. Retrieved September, 2015, from https://cwiki.apache.org/confluence/display/Hive/Home;jsessionid=13A6E9DCA7964EA6334A30761F6CB32C

26. *Helping a leading telecom provider turn big data into actionable insights*[Pdf]. (2015, March 23). Deloitte. Retrieved from http://www2.deloitte.com/content/dam/Deloitte/us/Documents/deloitte-analytics/us-da-telecom-case-study-final-03232015.pdf

27. Francisco, P. (2016, February). *IBM PureData System for Analytics Architecture* [Pdf]. IBM Retreived from http://www.redbooks.ibm.com/redpapers/pdfs/redp4725.pdf

28. Muddu. S. (2015). Puredata Systems for Analytics: Concurrency and Workload Management. *Culminating Projects in Information Assurance.* Retrieved from http://repository.stcloudstate.edu/msia_etds/3

29. Burns, E. (2013, December). Hadoop still too slow for real-time analysis applications? Retrieved April 17, 2016, from http://searchbusinessanalytics.techtarget.com/feature/Hadoop-still-too-slow-for-real-time-analysis-applications

30. Patil, M. R. (2013, August 23). Hadoop Ecosystem. Retrieved from http://www.manojrpatil.com/2013/08/hadoop-ecosystem.html

31. IBM (n.d.). What is HBase? Retrieved from: https://www-01.ibm.com/software/data/infosphere/hadoop/hbase/

32. Amazon Simple Storage Service: Developer Guide. (n.d.). Retrieved Spring, 2016, from http://docs.aws.amazon.com/AmazonS3/latest/dev/Introduction.html

33. Thusoo A., Sarma S. J., Jain N., Shao Z., Chakka P., Zhang N., Antony S., Liu H., & Murthy R. (2009, June 10). Hive – A Petabyte Scale Data Warehouse Using Hadoop *Yahoo Hadoop Summit*. Retrieved from http://infolab.stanford.edu/~ragho/hive-icde2010.pdf

34. Facebook Data Infrastructure Team. (2009, Aug 28). Hive - A Warehousing Solution Over a Map-Reduce Framework. *Proceedings of the VLDB Endowment 2(2)*. 1626-1629. Retrieved from http://dl.acm.org/citation.cfm?id=1687609&dl=ACM&coll=DL&CFID=750092345&CFTOKEN=82647425

35. Sarma, J. S. (2008, June 5). Hadoop. Retrieved from https://www.facebook.com/notes/facebook-engineering/hadoop/16121578919/

36. Database Data Warehousing Guide. (n.d.). Retrieved from https://docs.oracle.com/database/121/DWHSG/concept.htm#DWHSG001

37. Schroeder B., Harchol-Balter M., Iyengar A., Nahum E., & Wierman A., (2006). How to Determine a Good Multi-Programming Level for External Scheduling. *Data Engineering, ICDE '06. Proceedings of the 22nd International Conference on*. Retrieved from http://www.cs.toronto.edu/~bianca/papers/icde_camera.pdf

38. Benoit D. G. (2013, June 17). *Automated Diagnosis and Control of DBMS Resources* [PDF]. ResearchGate. Retrieved from https://www.researchgate.net/publication/2465040_Automated_Diagnosis_and_Control_of_DBMS_Resources

39. Weikum G., Hasse C., M¨onkeberg A., & Zabback P. (1994, July). The COMFORT automatic tuning project. *Information Systems*. 19(5). 381-432. Retrieved from http://dl.acm.org/citation.cfm?id=189748

40. Apache Thrift. (n.d.). Retrieved March 6, 2016, from http://thrift.apache.org/

41. Vykhodtsev, A. (2014, November 19). 41. Things to know when switching from Oracle Database to IBM Pure Data System for Analytics [Web log post]. Retrieved from http://expertintegratedsystemsblog.com/2014/11/things-to-know-when-switching-from-oracle-database-to-ibm-puredata-system-for-analytics-part-1/

42. *Netezza TwinFin™: High-Performance Business Intelligence and Advanced Analytics for the Enterprise* [Pdf]. (n.d.). IBM. Retrieved from http://www.netezza-twinfin.com/pdfs/ibm-netezza-twinfin-data-sheet.pdf

43. *IBM Netezza System Administrator's Guide* [PDF]. (2015, March). IBM. Retreived from https://download4.boulder.ibm.com/sdfdl/v2/sar/CM/IM/063mk/0/Xa.2/Xb.jusyLTSp44S02VJbZVlYWHGMx7bsEBnGM3gL5hH9h9cCmrAoIlpqP0NDRUo/Xc.CM/IM/063mk/0/nz-winclient-v7.2.1.2.zip/Xd./Xf.Lpr./Xg.8558945/Xi.habanero/XY.habanero/XZ.t_NjGWFPNyORVZDdBUxBuyxy0jw/nz-winclient-v7.2.1.2.zip

44. Dignan, L. (2010, August 27). Netezza's TwinFin fuels profit surge | ZDNet. Retrieved from http://www.zdnet.com/article/netezzas-twinfin-fuels-profit-surge/

45. Ronthal, A. (2013, February). What's New in the N2001 (Striper) Architecture? Retrieved from https://www-

304.ibm.com/connections/forums/html/topic?id=e5106045-3be9-44cc-916f-378af57f8359

46. Sensmeier, L. (2013, August 29). Modernizing Data Archiving with Hadoop - Hortonworks. Retrieved from http://hortonworks.com/blog/modernizing-data-archiving-virtualization-for-big-data-analytics/

47. *Extract, Transform, and Load Big Data with Apache Hadoop* [Pdf]. (n.d.). Intel. Retrieved from https://software.intel.com/sites/default/files/article/402274/etl-big-data-with-hadoop.pdf

48. Schmarzo, B. (2012, May 06). Understanding the Role of Hadoop In Your BI Environment | InFocus. Retrieved from https://infocus.emc.com/william_schmarzo/understanding-the-role-of-hadoop-in-your-bi-environment/

49. White, S. K. (2015, November 10). Study reveals that most companies are failing at big data. Retrieved April 17, 2016, from http://www.cio.com/article/3003538/big-data/study-reveals-that-most-companies-are-failing-at-big-data.html

50. Hogan, C. (2014, December 2). Facing The Consequences Of Poor Data Quality. Retrieved April 19, 2016, from http://www.insightsquared.com/2014/12/facing-the-consequences-of-poor-data-quality/