3-2019

Performance Evaluation of Structured and Unstructured Data in PIG/HADOOP and MONGO-DB Environments

Sri Rama Chandra Charan Tej Dasari
scdasari@stcloudstate.edu

Sri Ram
dasarisriramchandra@gmail.com

Performance Evaluation of Structured and Unstructured Data in Pig/Hadoop and Mongo-Db Environments

by

Sri Rama Chandra Charan Tej Dasari

A Starred Paper

Submitted to the Gradaute Faculty of

St. Cloud State University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in Information Assurance

March, 2019

Starred Paper Committee:
Dennis Guster, Chairperson
Susantha Herath
Balasubramanian Kasi

Abstract

The exponential development of data initially exhibited difficulties for prominent organizations, for example, Google, Yahoo, Amazon, Microsoft, Facebook, Twitter and so forth. The size of the information that needs to be handled by cloud applications is developing significantly quicker than storage capacity. This development requires new systems for managing and breaking down data. The term “Big Data” is used to address large volumes of unstructured (or semi-structured) and structured data that gets created from different applications, messages, weblogs, and online networking.

Big Data is data whose size, variety and uncertainty require new supplementary models, procedures, algorithms, and research to manage and extract value and concealed learning from it. To process more information efficiently and skillfully, for analysis parallelism is utilized. To deal with the unstructured and semi-structured information NoSQL database has been presented. Hadoop better serves the Big Data analysis requirements. It is intended to scale up starting from a single server to a large cluster of machines, which has a high level of adaptation to internal failure.

Many business and research institutes such as Facebook, Yahoo, Google, and so on had an expanding need to import, store, and analyze dynamic semi-structured data and its metadata. Also, significant development of semi-structured data inside expansive web-based organizations has prompted the formation of NoSQL data collections for flexible sorting and MapReduce for adaptable parallel analysis. They assessed, used and altered Hadoop, the most popular open source execution of MapReduce, for tending to the necessities of various valid analytics problems. These institutes are also utilizing MongoDB, and a report situated NoSQL store. In any case, there is a limited comprehension of the execution trade-offs of using these two innovations. This paper assesses the execution, versatility, and adaptation to an internal failure of utilizing MongoDB and Hadoop, towards the objective of recognizing the correct programming condition for logical data analytics and research. Lately, an expanding number of organizations have developed diverse, distinctive kinds of non-relational databases (such as MongoDB, Cassandra, Hypertable, HBase/ Hadoop, CouchDB and so on), generally referred to as NoSQL databases. The enormous amount of information generated requires an effective system to analyze the data in various scenarios, under various breaking points. In this paper, the objective is to find the break-even point of both Hadoop/Pig and MongoDB and develop a robust environment for data analytics.

Keywords: MapReduce, Fault Tolerance, MongoDB, Pig, NoSQL Database.

Table of Contents

Chapter Page

Chapter Page

List of Tables

Table Page

List of Figures

Chapter I: Introduction

Introduction

This Chapter is an introduction to Pig and MongoDB which explains the nature and significance of the problem statement, which helps in understanding the experiments, comparing the performance of Pig with MongoDB.

Databases are an accumulation of information. In spite of this fact, when utilizing the term database, it refers to the entire database framework and the term refers not to the gathering of information alone. The framework refers to handling data, its transfer, transformation or other aspects of the database and is called the Database Management System (DBMS). The next step is to define how different frameworks write the data into their database. Early models and usage depended on the utilization of connected records to make relations amongst data and to identify patterns. For example, in a typical Relational Database Management System (RDBMS), the data from different tables are inter-related through the primary-key and foreign-key connections. These models were not standardized as they required broad preparing with a concrete end goal to make effective utilization of their architecture. Databases were created keeping in mind the end goal to fulfill this need of storing and accurately analyzing information. Since the inception of the conventional databases in the 1960s, diverse kinds have been developed, each utilizing its own and specific methods of deriving information and distinctive innovations for taking care of data transfer and transformation. Developers started with navigational databases which depended on connected records, proceeded onward to relational databases with joins, and, a short time, later developed systems without joins in the late 2000s, such as NoSQL (MongoDB, Cassandra,

Hypertable, HBase/ Hadoop, CouchDB and so on). NoSQL was developed and has turned into an excellent platform for storing and managing unstructured data.

The data stored in a typical RDBMS can be created, altered, updated and extracted using SQL queries. DBMS stores backup files on the Hard Disk. If DBMS fails, one can retrieve data from backup. Figure 1 shows this relationship.

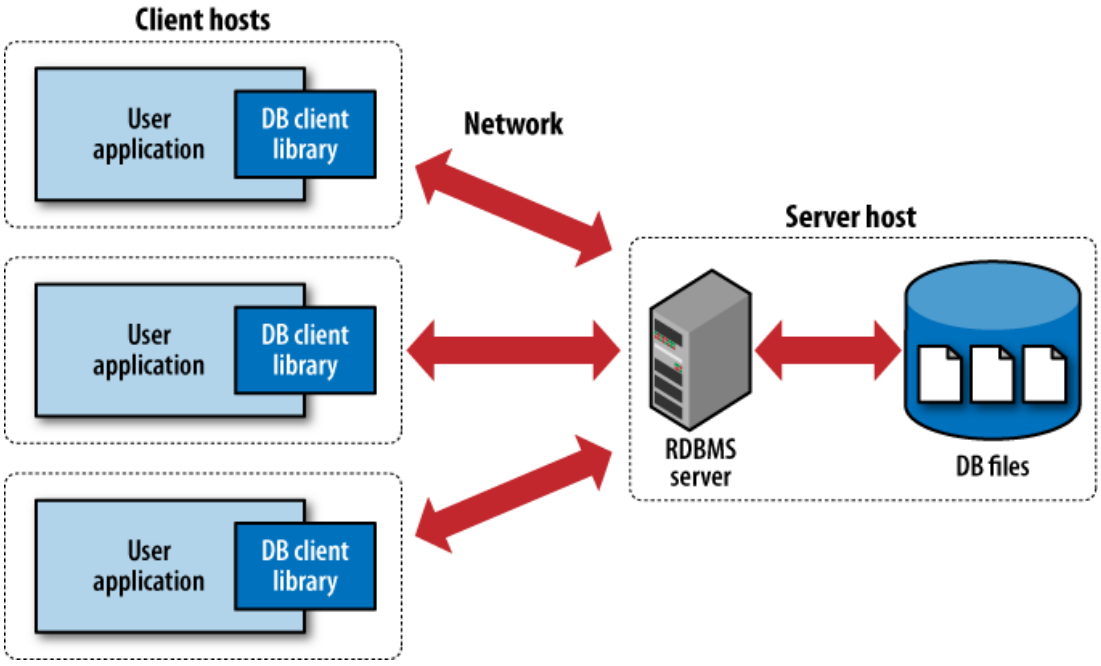


Figure 1. Conventional RDBMS architecture.

Later, a need for storing, retrieving and processing of large data lead to the development of Big-Data Technologies. Thus, began the quest to design and adopt Distributed File Systems for storing petabytes of data, while minimizing data-latency. Thus, Google released its Google designed distributed storage system called Big-Table to store, retrieve and process peta bytes of structured data, at a faster rate.

Introduction to Hadoop

Hadoop was an open-source project from the beginning; made by Doug Cutting of Yahoo (also known for his work on Apache Lucene, a common search and sort platform). Hadoop initially originated from an undertaking called Nutch, an open-source web crawler made in 2002. Throughout the following couple of years, Nutch overtook and developed superior improvised versions of various Google Projects. In 2003, when Google released their Distributed File System (GFS) to store, retrieve and process peta bytes of structured data, Nutch released their own, which was called NDFS (Ghemawat, Gobioff, & Leung, 2003). In 2004, Google presented the idea of MapReduce, with Nutch declaring the release of their MapReduce engineering soon after in 2005. It was not until 2007 that Hadoop was formally released. Utilizing concepts extended from Nutch, Hadoop turned into a platform for parallel handling of huge amounts of data scaling over clusters of production servers. Hadoop is designed to address data analytics of large datasets, and it is not an alternative for relational database frameworks.

History of Hadoop

In the 1990s, Google needed to gather more information and to get the correct layout; it has taken 13 years to accomplish this. In 2003, they presented GFS (Google File System) which is a distributed file system to store extensive information designed to interact with applications using distributed data (Ghemawat et al., 2003). In 2004, they introduced MapReduce which performs simplified data processing on large clusters through functional programming. They published a “white paper” which had a depiction of GFS and MapReduce. Yahoo took the white paper which was written by Google and began implementing and published HDFS (Hadoop Distributed File System) and MapReduce. These are the two main segments of Hadoop. Doug

Cutting then presented Hadoop in 2005. Meanwhile, Google presented Bigtable enabling its application a dynamic control over the format and structure of data, thus providing data to its projects (Chang et al., 2006). Yahoo, which is known as the next best web browser company after Google, presented their HDFS in the year 2006-2007 and their MapReduce in 2007-2008. In 2008, Yahoo introduced a distributed Data Serving System called as PNUTS. "PNUTS provides data storage organized as hashed or ordered tables, low latency for large numbers of concurrent requests including updates and queries, and novel per-record consistency guarantees" (Cooper et al., 2008, p. ii).

These cloud services providing data using distributed systems with low latency, are not comparable with each other due to the difference in the size, speed of data requests made by various applications. Yahoo published a paper "*Benchmarking Cloud Serving Systems with YCSB*" in 2010 Cooper, Silberstein, Tam, Ramakrishnan, and Sears (2010) creating a benchmark between multiple Cloud Serving systems such as Cassandra, Yahoo's PNUTS, HBase and a sharded system of MySql.

Components of Hadoop

Hadoop is an open source system provided by Apache programming establishment for storing and preparing enormous data sets with the cluster of commodity machines which is finished by these segments.

HDFS (Hadoop Distributed File System)

HDFS is an exceptionally designed File System for storing large information collections with clusters of commodity equipment with gushing access design which supports "Write Once Read Many Times." The block size by default is 64MB or 128MB. Shvachko, Kuang, Radia, and

Chansler (2010) expressed the way HDFS is economical and scalable as, “In a large cluster, thousands of servers both host directly attached storage and execute user application tasks. By distributing storage and computation across many servers, the resource can grow with demand while remaining economical at every size” (p. 1).

Pig

Apache Pig is a data flow handling (scripting) language. It consists of a high-level language called Pig Latin for articulating complex data-analytics programs and is used as a platform for analyzing massive datasets. The principal feature of Pig programs is that their architecture is compliant to significant parallelization, empowering them to deal with massive datasets and a straightforward language structure. Its ability of parallelization of jobs gives a reflection that makes Hadoop jobs faster and less demanding to compose than usual Java MapReduce employments.

Brief History of Pig

The pig was initially created by Yahoo in 2006, for analysts to have an impromptu method for making and executing MapReduce jobs on extensive data sets. It was built to diminish the overhead time through its multi-query approach. Pig is likewise made for experts (primarily programmers) from a non-Java background, to make their activity less demanding.

Introduction to MongoDB

Initially, MongoDB was created by the organization 10gen in 2007 as a cloud-based application motor, which was planned to run grouped programming and services. They built two primary segments, Babble (the application motor) and MongoDB (the database). The project did not take off, driving 10gen to scrap the application. Later, they released MongoDB as an open-

source project. In the wake of turning into open-source programming, MongoDB thrived, attracting support from a development group with different upgrades created to enhance and incorporate in the new release. While MongoDB consists of a Big Data design, it is vital that it has to be universally useful, intended to supplant or upgrade existing RDBMS frameworks, giving it a sound assortment of utilization cases.

Overview of MongDB database. MongoDB (the term derived from the word humongous) is an open source, and report arranged NoSQL database that has of late achieved some recognition in the data science and analytics community (Chodorow & Dirolf, 2010). Then, it is a standout amongst the most well-known NoSQL databases, because it favors master-slave replication. The responsibility of a master is to perform peruses and composes through the slave limits to duplicate the information obtained from the Master, to carry out the real task, and reinforce the jobs on the data. The slaves do not partake in compose assignments yet may choose a substitute master if there should be an occurrence of the current master failure. MongoDB utilizes parallel configuration of JSON-like archives underneath and has an advantage of building a robust framework, not at all like the standard relational databases. In case of the arrangement in MongoDB, it can return specific fields, and query sets the range to seek by fields, run sub-queries, customize the articulation view, and so on and may incorporate the client characterized complex JavaScript capacities. As implied as of now, MongoDB hones adaptable construction and the report structure in a gathering, called Collection. Thus, collections may help in change of primary fields of different records in accumulation that can store different kinds of data.

There are sufficient tools available in MongoDB, required for interacting with many programming languages, which are utilized to create a customized framework that uses MongoDB as their backend technology. There is an excellent requirement of using MongoDB as a sophisticated in-memory database; in such cases, the application dataset is dependably small. However, it is simple for support and can make a database designer's work easier; this can be an advantage for sophisticated applications that require massive database administration capacities. A portion of the prominent clients of MongoDB is MetLife, Craigslist, Forbes, The New York Times, Sourceforge, eBay, and so on. For instance, The New York Times has its frame building application that permits photograph storage. MongoDB database is used to design these applications. Then, Sourceforge has shown more interest in MongoDB and used it to store back-end pages.

History of MongoDB. MongoDB was released in 2009 and is composed using C++, and it is one the most famous NoSQL database framework. MongoDB stores information in JSON-like reports that can shift in structure. Related data can be stored together for quick query access through the MongoDB query language. MongoDB utilizes dynamic patterns, which makes records without first characterizing the structure, for example, the properties or the information writes. It is conceivable to change the structure of documents by just including new qualities or erasing existing fields. This model speaks to serial connections, to store clusters, and other more perplexing structures effortlessly. Archives in a record are not required to have the same arrangement of fields. MongoDB is outlined with high accessibility and adaptability and incorporates replication and auto-sharding.

For instance, consider a **General Store Management System** which deals with the business activities in a grocery store. These include keeping up the records of stock points of interest, keeping track of the deals that increased sales for a specific month/year and so forth. Therefore, clients need less time for computation and the business action can be finished within less time through a standard framework. Thus, the time saved by quickly maintaining records is used to focus more on implementing better business decisions. The database stores the information, which reduces paperwork, and the client can invest additional energy in examining the store.

MongoDB design. The replication of collections provides superior replication including automated failure handling, while sharded groups make it conceivable to separate large data sets over multiple machines, directly connected to the Client machines.

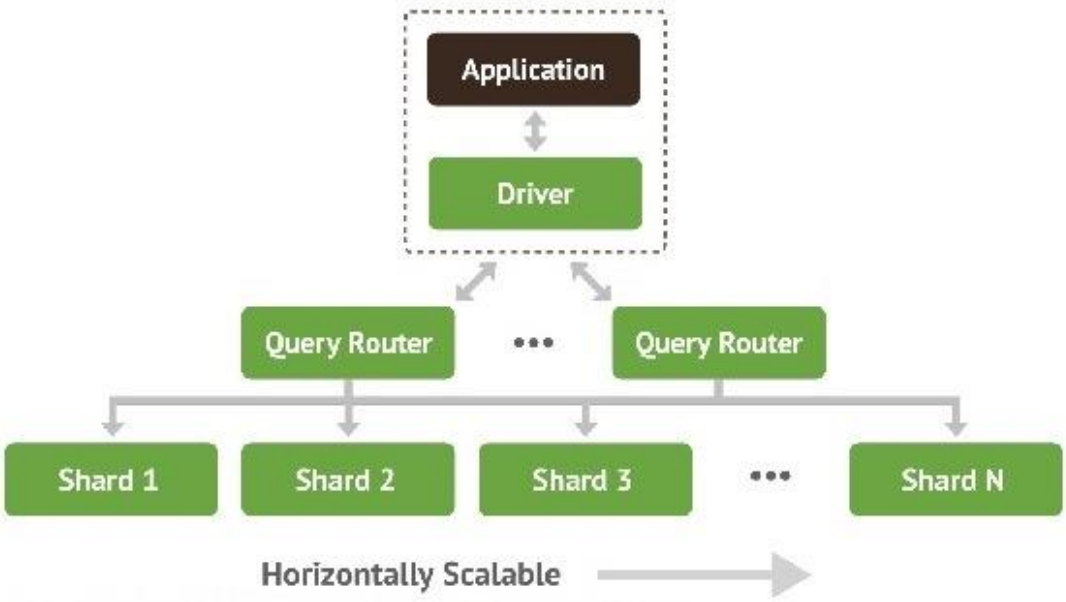


Figure 2. MongoDB architecture.

MongoDB clients consolidate the replicated collections and sharded groups to give elevated amounts of the additional data sets, which are directly accessible for applications. MongoDB supports sharding through the configuration of sharded clusters.

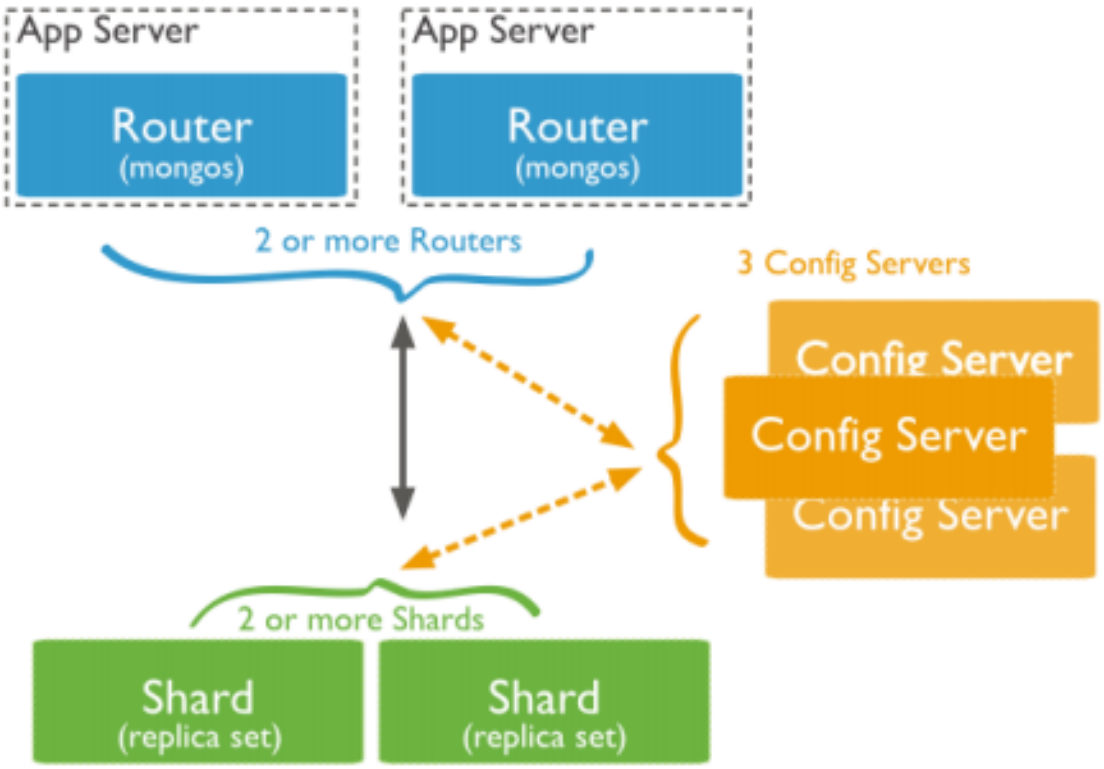


Figure 3. Sharding in MongoDB.

Following is a explanation of the essential components of MongoDB:

- **Shards**. Utilized to store information. They give high accessibility and data consistency. Internally, every shard is a different replicated collection.
- **Config Servers**. Config servers store the collection's metadata, which contains a mapping of the collection's data index to the shards. The query router utilizes this metadata to target activities to specific shards. Inherently, sharded collections have precisely three config servers.

- **Query Routers**. Query routers are essential in MongoDB; they interface with client applications and direct activities to the appropriate shard. The query routers processes and focuses on the events to shards and later returns results to Client Machine. A shared group can contain more than one query router to split the client machine's request load. A Client machine sends a request to one of the query routers. Usually, a sharded collection has numerous query routers, and the Mongos assigns the requests.

Some features of MongoDB.

- **Adaptability:** MongoDB stores information in reports organized by utilizing JSON. It uses less archive blueprint and maps to local programming language write.
- **Rich query language:** It gives the components required by RDBMS. Dynamic queries, sorting, backup files, frequent updates, simple collection, up sort (refresh if the record exists and embed on the off chance that it does not) are a few RDBMS highlights. Adaptability and versatility are the additional features.
- **Auto-sharding:** Auto-sharding enables to scale the cluster linearly by including more machines. It is conceivable to expand the effectiveness which is critical on the web when the load can increase abruptly and bring down the site.
- **Ease of usage:** Feature of being easy to utilize, keep up and arrange.
- **High performance:** It gives the best data determination and, decreases I/O movement on the database framework by supporting implanted archives, and the use of fast and robust queries.

- **High accessibility:** MongoDB uses a replica set. The replica set is a cluster of servers that keep up the same dataset. It gives automatic failover, excess and expanded data accessibility.
- **Support for multiple storage engines:** It supports different storage engines, for example, Wired Tiger stockpiling motor, MMAPv1 stockpiling motor. It likewise supports a pluggable capacity motor API that enables an outsider to create a capacity motor for MongoDB.

Table 1

Terms and Concepts of Different Databases

Relational Database	MongoDB Database
Fixed Schema	Schema Less
Table	Collection
Rows, Columns	Documents
Joins	Embedded Documents
Vertical Scaling	Vertical Scaling, Horizontal Scaling

Table 2

Basic Queries Used in Two Different Databases

Query	Relational Database	MongoDB Database
Create Command	CREAT TABLE table_name (column_name1 datatype, Column_name2 datatype)	Noneed for defining schema
Insert Command	INSERT INTO table_name (column_name 1, column_name 2) values (value 1,value 2)	Dlb.collection_name. Insert ({name1: alu 1, Name2: value 2})
Delete Command	DELETE FROM table_name WHERE (condition)	db.collection_name.remove ({condition})
Import Command	BULK INSERT Table_name FROM File_name WITH { FIELDTERMINATOR = ; ; ROWTERMINATOR = +/N+ } GO	Mongoimport --db Database_name -- collection colletion_name --type csv –file “file_name”
Select Command	SELECT column_name FROM table_name	db.collection_name ({}, {condition})

Problem Definition

This section gives a short definition of Hadoop and MongoDB. After the definition, now the execution of both the frameworks in term of usage in the data science market is assessed. The MongoDB database comprises a group of databases in which every database contains various collections. Since MongoDB works with dynamic diagrams, each collection may hold a distinct variety of data. Each query gets sent as a record. These records are created in form the of JSON format: a list of key-value pairs. The value can be off for the most part three writes: a set of unprocessed values, a non-homogeneous tuple or a list of key-value pairs. To query these key-

value pairs, the user can convert the collections joined as a list of key-value sets. It is additionally possible to query related fields. The queries are likewise JSON-organized; subsequently, a random query can consume significantly more memory than a similar query for the other databases. If the implicit queries are excessively constrained, which is possible, making it impossible to send JavaScript logic to the server for more complicated queries.

MongoDB provides principally two kinds of replication: master-slave and replica sets. In the master-slave replication, the master has control of the complete access to the data, and it composes each change made to its slaves. The slaves can be conceivable to analyze the information. A replica set works same as master-slave replications. However, it is possible to choose another master if the first master is down. Another critical element that is supported by MongoDB is programmed sharding. Utilizing this feature data can be allocated and distributed to various nodes. The master needs to confirm a sharding key for every collection which characterizes how to store the documents. In such a domain, the Client interacts with a Secondary master node called mongo process which investigates and side-tracks the query to the proper node or nodes. For preventing the loss of data, each logical node contains physical servers which replicate the data present in the node. By utilizing this framework, it is equally conceivable to use MapReduce having a decent execution.

Numerous organizations are utilizing incorporated systems instead of using them independently to consolidate the strengths of every one of these frameworks (Apache Pig and MongoDB). Both Apache Pig and MongoDB can store in distributed data frameworks and can perform data analytics. The information investigation process includes a few phases, for example, importing semi-organized information, storing in conveyed document frameworks, at

that point execute the required tasks utilizing MapReduce. The vast majority of the above activities have a scope to be implemented on both the structures Apache Pig and MongoDB. To decide the structure required for every one of these stages, an examination of the execution amongst MongoDB and Pig is fundamental.

Nature and Significance of Problem

This study includes investigation of the performance of Pig/Hadoop with MongoDB to decide the engineering and functionalities of these two structures in a coordinated framework. Subsequently, this examination requires performing data transformations on specific large chunks of data. After that, the study illustrates the execution of both the frameworks for different functionalities, for example, bringing in the required information, investigating the information and storing the results of the analysis. A set of detailed experiments are performed to quantify the performance of two different frameworks under the influence of various parameters.

Chapter II: Background and Review of Literature

Introduction

The literature review covers all the research papers, books, online articles and online recordings related to Pig/Hadoop, MongoDB systems, and other Big Data Analytics tools. Additionally, review and experimental data from past research papers and articles which should empower the comprehension and improvement of an Integrated Big This report also uses Data Analytics tools such as Hadoop and MongoDB.

Background

As of late, in the wake of understanding the significance of NoSQL databases, critical work is being done. Arora and Aggarwal (2013) proposed an algorithm to transform SQL databases (MySQL) to NoSQL databases (MongoDB). This work can get extended to other NoSQL databases in the future.

The proposed algorithm gets implemented in NetBeans Java IDE. Rao and Govardhan (2013) proposed the algorithm to enhance the execution of online aggregation which is, “Sharded Parallel MapReduce in MongoDB for Online Aggregation.” It was estimated to produce results in less time when contrasted with the customary MapReduce framework. A technique was proposed to incorporate two kinds of databases, to be specific MySQL and MongoDB, by including a middleware between the application layer and the database layer. The middleware consists of metadata which contains various kinds of bundles. Three mainstream NoSQL databases were considered, to be specific Cassandra, MongoDB, and CouchDB. Also, the paper infers that every arrangement was produced for various applications and had their upsides and downsides.

Mapanga and Kadebu (2013) made proposals for addressing the security issues of NoSQL databases: influencing utilization of outsider to open source devices for review and logging, worked in verification, input approval and to get control. Obligations for segregation and encryption of data are the goals of Firesmith's security prerequisites.

Hadoop is a structure which is utilized for the capacity and handling of vast volumes of information in distributed record framework or condition, across a group of nodes through programming models as a solution for the conventional RDBMS frameworks.

The development of this Integrated Big Data Analytic instrument requires the best possible comprehension of the functionalities of every one of the components in the Hadoop and MongoDB systems. This paper clarifies these essential elements and the features of these systems, in the accompanying pages.

Literature Review

The Shvachko et al. (2010) HDFS (Hadoop Distributed File System) is the File System created in light of the guideline of a distributed file system with parallel processing. HDFS is profoundly a fault tolerant and uses minimal effort equipment hardware, as large memory is not required.

Data Localization is one of the features of Hadoop, enabling it to handle data at their res of pective DataNodes. Thus, Hadoop system controls the data part instead of the coding part. The Hadoop framework contains the accompanying segments which perform the Data Localization: Client, Name Node, and Data Nodes. Every one of these segments has various functions in managing the framework.

- **Name Node:** The name node works as a master. It can keep up the namespace of the entire data in the Hadoop framework. The name node contains a document framework tree and the metadata for every one of the records and registries put away in the tree. The name node consistently stores the above data as two locations: fs-image and the edit-log. The Namenode also stores the metadata containing the physical location of the actual data in the DataNodes.
- **Client Node:** The Client Node is the mechanism through which the user connects with a Hadoop Cluster (Name node and Data Node). It resembles the interface between the client and the name node. At whatever point, a client needs to compose information into a Hadoop Cluster, and the correspondence must be set up by the customer. The Client sends a request to the name node, asking to name a gathering of information nodes to which the client composes the information in a pipeline technique.
- **Data Node:** The Data Nodes are an integral component of the whole HDFS. They are the nodes on which extensive records are separated into blocks and distributed as suggested by the name node.

The components of a conventional Hadoop Cluster connect in an organized manner, and they lose coordination amongst them if any disconnection in the cluster occurs. Mainly, if any data node loses connection with its Name node, at that point the whole information is inaccessible until the connection between the name-node and data-node is re-established back. Organizations like Facebook and YouTube, which rely on storing and maintaining the Big Data of their clients, cannot afford to cause any inconvenience to their users in accessing their data.

The Hadoop Cluster guarantees data integrity and adaptation to non-critical failure by providing replication. Each document imported is separated into blocks and after that replicated into various Data-nodes in the Hadoop Cluster.

MapReduce

Dean and Ghemawat (2008) explained about MapReduce concept. MapReduce is a programming model for handling the information stored in the distributed data framework. MapReduce programs support diverse programming languages, for example, Java, Python Ruby, C++. The MapReduce, as the name proposes, is the mix of two functions: Mapping and Reducing.

The Mapping procedure includes the isolation of the key-value information from the large and voluminous details. At this stage, the data is transformed and segregated based on certain conditions which ease the aggregations performed in the reducing stage. Hence, only the analyzed data elements are separated, and they receive a key-value combination to simplify the process of identifying each record. The key enables in identifying subgroup, and the value is the data to be analyzed. Together, they are the required key-value pair.

Subsequently, the key-value pair generated as the output in the Mapper Stage works as the input in the Reduce stage. The Reduction procedure, as the name proposes, works to draw out a pattern or trend of the data being analyzed through aggregations, while the basic aggregation operations being, the calculation of max values, the summation, average, count, etcetera.

YARN (Yet Another Resource Negotiator) is the redesigned version of the MapReduce, correcting the issue of bottlenecking when more than one task gets executed on the MapReduce.

In YARN, the Application-Master and Resource-Manager share the responsibilities of a job tracker between theManagerm. The basic idea of YARN is that the Application-Master consults with the Resource-Manager and allocates new memory resources, along with the processors required for performing that specific task (Vavilapalli & Murthy, 2013).

MongoDB

MongoDB is a flexible, versatile and capable document-oriented NoSQL database. MongoDB has numerous unique features which influence it to stand out among the other NoSQL databases, which include but not limited to auxiliary lists, Ad hoc queries, indexing, replication, load balancing, capped collections and sharding.

- **Indexing:** Indexing is one of the various components controlled by MongoDB, to query and extract data at a quicker rate. Queries which do not use the element of order perform a collection-scan (this term has its origin in connection with the RDBMS. It means searching the whole collection to get hold of the required record, and consequently, indexing decreases the preparing time of the Data Base).
- **Shards:** A shard stores a small portion of the entire data stored in a sharded cluster. Many such shards together contain the entire data stored by the Sharded cluster.
- **Sharding:** Sharding is a way of dividing a large amount of information into small subsets of data to, improve the preparation of the data distributed across many machines. This group of nodes, which contain subsets of the whole MongoDB collections, is called Shards. Shards are similar to the distributed filesystem in Hadoop. One of the essential objectives of Sharding is preparing to visualize these groups as a single machine to the client while saving time in gathering every one of

these bits of information from different shards to generate the required report. Hence, the MongoDB Sharding architecture includes a unique machine called MongoS (Mongo Server) whose essential function is to retrieve the information without noticeable time delay by directing the expected information to the client machine from the various Shards. To empower this component, the Mongos maintains a table containing the lists of every datum set or piece of information coordinated to that Shard.

After Yahoo published its paper on establishing benchmark for creating a comparibility between the various Distributed Cloud services, a new study by , provided an insight into the influence of technical choices over the elasticity of cloud databases in the paper by Dory, Mejas, Roy, and Tran (2011). Since, then the focus has shifted on developing models which can store much larger data with ease and provide data on request with less latency and fault tolerance. Meanwhile, Fadika and Govindaraju (2010) published a MapReduce model flexible to process large and small data-sets, i.e, for both on-disk and in-memory applications called LEMO-MR. Later, FCM algorithms are applied on large data sets for clustering and performing efficient analytics (Havens, Bezdek, Leckie, Hall, & Palaniswami, 2012). The Materials Genome project insisted on developing robust open-source computing platforms for identify all possible properties of in-organic materials and has also influenced the data analytics field of computer science (Jain et al., 2013).While they were surveys conducted aiming at reviewing the process of generation, acquisition, storage and analysis of data (Chen, Mao, & Liu, 2014), simultaneous efforts were made to optimize the Hive queries performing transaction on large datasets through Indexing and Join ordering algorithms (Jain & Kakhani, 2015).

Meanwhile, facebook created its NoSql distributed database, Cassandra in 2009, which is designed to meet the data requirements of Facebook's through using low-cost hardware infrastucture and providing high end read and write efficiency (Lakshman & Malik, 2009). Soon after the inception of NoSQL databases, studies were performed to evaluate their impact on traditional RDBMS by comparing the cloud scalability of NoSQL database and its performance (Pokorny, 2011). Later in 2014, an architecture was developed to integrate the NoSql and Sql platforms (in this case MongoDB and MySQL), through a virtual layer built on top of the NoSQL System (Lawrence, 2014). Further studies were made to evaluate the performance of Hadoop integrated with NoSQL databases such as MongoDB and Cassandra (Seema & Ayush, 2014).

The Big Data Distributed Models and NoSQL Database models, both were believed to threaten the existence of RDBMS. But, later studies revealed that Sql stays alive with SQL NoSQL models have different benchmark standards thus, both have their own standards in developing data models (Floratou, Teletia, Dewitt, Patel, & Zhang, 2012).

"Therefore, research is needed to delineate the advantages of distributed databases and this study addresses this with data obtained from a basic configuration of a Cassandra database. Data collected from the Cassandra test bed revealed that in general, a distributed database using additional nodes could reduce latency. However, diminishing returns were observed as additional nodes were added into the experiment" (Guster, O'Brien, & Lebentritt, 2013, p. i). Guster et al. has already researched the inactivity of distributed document frameworks on a NoSQL Database framework called Cassandra, and how the execution changes as the quantity of slave nodes increments. Presently, this paper draws motivation from the Guster et al. (2013) work, and

reports on a comparative execution examination conducted between the distributed frameworks of Pig/Hadoop and MongoDB.

Chapter III: Methodology

Introduction

Traditional Relational Database Management Systems (RDBMS) are designed around relations and tables to arrange and structure data in a combination of rows and columns. Current conventional database frameworks such as RDBMS is likely to remain as such for a long time to come. For some organizations, RDBMS arrangements are sufficient in maintaining, managing and analyzing their data; however, they are not feasible for every use case. These frameworks regularly keep running into bottlenecks with versatile data sets and data replication when dealing with a large amount of data/data sets.

Study of Hadoop

Hadoop, as already mentioned, is a framework which supports a distributed environment. The essential parts of Hadoop are the Hadoop Distributed File System (HDFS) and MapReduce programs, programmed in Java. Supplementary tools are a collection of other Apache items, including:

1. Hive (for querying data)
2. Pig (for analyzing large data sets)
3. HBase (column-oriented database)
4. Oozie (for planning Hadoop jobs)
5. Sqoop (for interfacing with different frameworks, for example, BI, investigation, or RBDMS) and
6. Flume (for conglomerating and pre-processing information).

Like MongoDB, Hadoop's HBase database achieves flat file adaptability through database sharding. Hadoop is intended to keep running on multiple nodes, with the capacity to

process information in various configurations, including collecting data from numerous sources. Transfer of information gathered is taken care of by the HDFS, with a discretionary information structure actualized with HBase, which distributes information into segments (unlike the two-dimensional designation of an RDBMS into columns and rows). Data would then be able to be sorted (through utilization of programming like Solr), queried with Hive, or have different analytics or cluster jobs keep running on it with decisions accessible from the Hadoop environment.

Study of MongoDB

MongoDB is an exciting technology, as it is a NoSQL database adopted by a large number of organizations, although it does not have much endorsement. A noteworthy objection about MongoDB is its adaptation to non-critical failure issues, which can cause resources to be scattered. However numerous occurrences of these issues can be discovered on the internet. Additional complaints against MongoDB are composing bolt imperatives, information collection issues, poor reconciliation with RBDMS, and that's only the tip of the iceberg. MongoDB likewise can import information in CSV or JSON formats only, which may require additional data transformation.

Hadoop's critical issue used to be the Namenode, which is a single point of failure of the HDFS clusters; if the Namenode fails, then the framework no longer works. Although this issue was addressed with the arrival of HDFS High Availability (HA), which gives the capacity to arrange two excess Name nodes, so the framework will failover to the Secondary NameNode should an issue emerge. Additional concerns with Hadoop are the measure of the time it takes to finish data handling job and its inefficiency with regulating resources.

As the products Hadoop and MongoDB develop with newer versions, a large number of these issues tend to develop later on, with the arrival of new updates or new programming brought through their innovation communities. RBDMS arrangements are additionally progressing, and so are other NoSQL platforms. As versions of these tools' functionalities keep improving, it is exciting to see how all these distinctive platforms advance and change by addressing the issues of the developing innovation and user requests.

Hive. Hive is a data repository framework based over Hadoop. Hive gives instruments to empower simple information outline, specially appointed querying and investigation of comprehensive datasets stored in Hadoop records. It provides a device to put a structure on this data, and it likewise provides a primary query language called Hive-QL, similar to SQL, empowering clients comfortable with SQL to search the data through these queries.

HCatalog. It is a storage administration layer for Hadoop that empowers clients with various data handling instruments. HCatalog's table abstraction presents clients with a relational perspective of information in the Hadoop distributed file system (HDFS) and guarantees that clients do not need to worry about where or in what kind of format their information got stored.

MapReduce. Hadoop MapReduce is a programming model and programming structure for composing programs that quickly processes large measures of data in parallel on the cluster of PC nodes. MapReduce utilizes the HDFS to get to record sections and to lessen inappropriate outcomes.

HBase. HBase is a distributed, column-based database. HBase utilizes HDFS to take leverage of its fault tolerance. It maps HDFS information into a database-like structure and gives Java API access to this DB. It reinforces cluster style computations utilizing MapReduce and

instant queries (random scan of that database). HBase is as a part of Hadoop Development plugins used for random real-time reading/writ irequiredng. Its objective is the facilitating of large tables running over clusters of custom nodes.

Hadoop distributed file system. Hadoop Distributed File System (HDFS) is the essential storage system utilized by Hadoop applications. HDFS is, as its name suggests, a distributed file system that gives high throughput access to application information making numerous replicas of data blocks and dispersing them on register nodes all through a group to empower dependable and quick algorithms.

Core. The Hadoop core comprises an arrangement of segments and interfaces which gives access to the distributed file system and general I/O (Serialization, Java RPC, Persistent information structures). The core segments additionally give “Rack Awareness,” an enhancement which considers the topographic grouping of servers, limiting data movement between servers from different geographic clusters.

High-Level Architecture of Hadoop

The architecture of Hadoop is a MapReduce system that works on HDFS or HBase. Fundamentally, Hadoop breaks down a task into a few similar but smaller tasks that can get executed adjacent to the data (on the Data Node). In this way, each job gets executed parallelly: the Map stage. Later on, all the different outcomes boil down to one outcome: the Reduce stage. In Hadoop, the Job Tracker (Java process) is in charge of observing the activity, dealing with the MapReduce stage, and dealing with the repeats if there should be an occurrence of errors. The Job Tracker (Java process) is running on the various Data Nodes. Each Job Tracker executes the activity on the data nodes through Task Trackers. The center of the Hadoop Cluster Architecture

is beneath HDFS (Hadoop Distributed File System): HDFS is the essential record storage, fit for capturing a substantial number of large documents.

MapReduce

MapReduce is the programming model by which information is dissected utilizing the processing resources inside the cluster. Every node in the Hadoop cluster is either a master or a slave. The slave nodes are always both a Data Node and a Task Tracker.

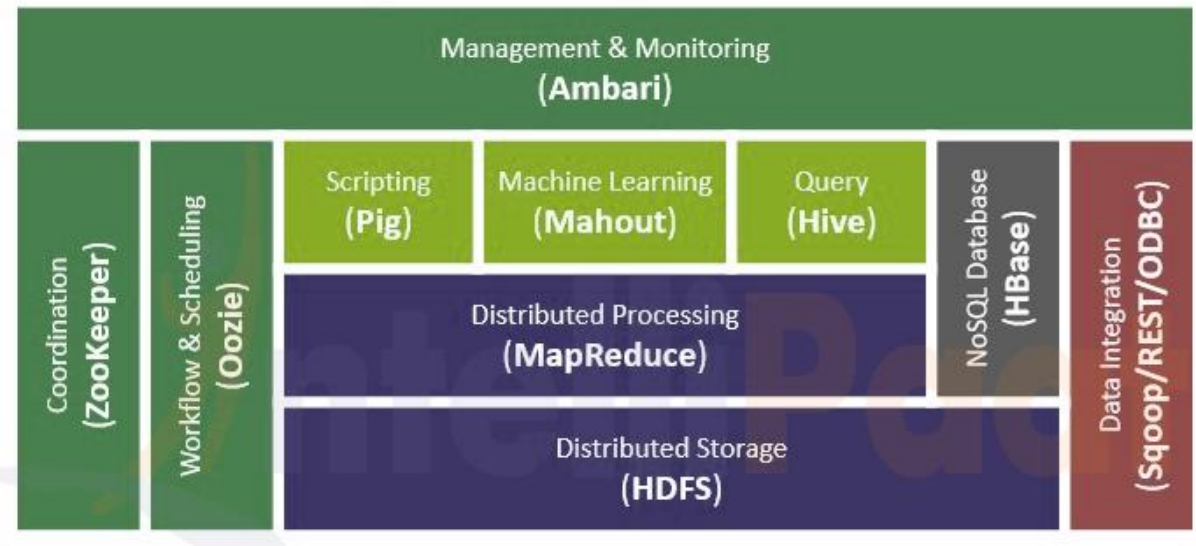


Figure 4. High-level architecture of Hadoop ecosystem.

Figure 4 shows the high-level architecture of Pig and Hive and that these two components interact with one another using MapReduce for getting access to various data.

- Name Node: Stores and manages metadata of the data stored in DataNodes and its access control. There is precisely one Name Node in each cluster.
- Secondary Name Node: Transfers checkpoints at configured time intervals from the name Node for adaptation to internal failure. There is precisely one Secondary Name Node in each cluster.

- Job Tracker: Hands out a task to the slave nodes. There is just one Job Tracker in each cluster.
- Data Node: It is used to store data in the form of blocks, and the meta-data gets stored in the Name Node. Every data node deals with its own privately connected storage and stores a replica of a few or all blocks in the record framework. There is at least two Data Node in each cluster.
- Task Tracker: It is one of the services in the Hadoop framework; it performs tasks assigned by the Job Tracker, at the DataNodes which are in the same cluster. There is at least one Task Tracker in each node.

Hadoop Distributed File System (HDFS)

HDFS group has two kinds of nodes working in a master-slave design: a Name node (the master) and various DataNodes (slaves). The name node deals with the file system namespace. It keeps up the file system tree and the metadata for every one of the records and registries in the tree. The name node likewise knows the data nodes on which every one of the blocks for a given file got recorded. Data nodes are the essential components of the Hadoop Distributed file system. They store and recover parts when they are requested to (by the client or the name node), and they send feedback to the name node intermittently with whereabouts of data that they are capturing. The Name Node implements replication factor of data blocks which is the number of replicas of each block across the cluster. In a run of the mill HDFS, block-size is 64MB, and replication factor is 3 (second duplicate on the adjacent rack and third on the remote rack).

Figure 5 depicts a Hadoop Distributed File System HDFS. Hadoop MapReduce applications utilize capacity in a way that is unique concerning broadly useful processing. To

examine an HDFS record, for analysis the client applications use a standard Java document input stream, as though the document was in the local file system. In the background, however, this stream is controlled to recover information from HDFS. To begin with the process of analysis, the NameNode is approached to ask for authorization. Internally, the Name Node interprets the HDFS filename into a rundown of the HDFS block IDs containing that document and a summary of Data Nodes that store each block and restore the summaries to the Client. Next, the client opens a connection with the “nearest” Data Node (because of Hadoop rack-awareness, yet ideally a similar node) and requests a particular block ID. The requested block returns through the same connection, and the data gets conveyed to the application.

In the process of writing data into HDFS, customer applications see the HDFS record as a standard YARN stream. Inside, be that as it may, streamed data first gets divided into HDFS-sized blocks (64MB) and after that little packet (64kB) by the client thread. Every packet gets enqueued into a FIFO that can hold up to 5MB of information accordingly decoupling the application thread from Storage system inactivity amid ordinary activity. An instant thread is in charge of dequeuing packets from the FIFO, coordinating with the Name Node to allocate HDFS Block IDs and targets, and transmitting bits to the Data Nodes (either neighborhood or remote) for storage. A third thread oversees affirmations from the Data Nodes that information has been resolved to plate.

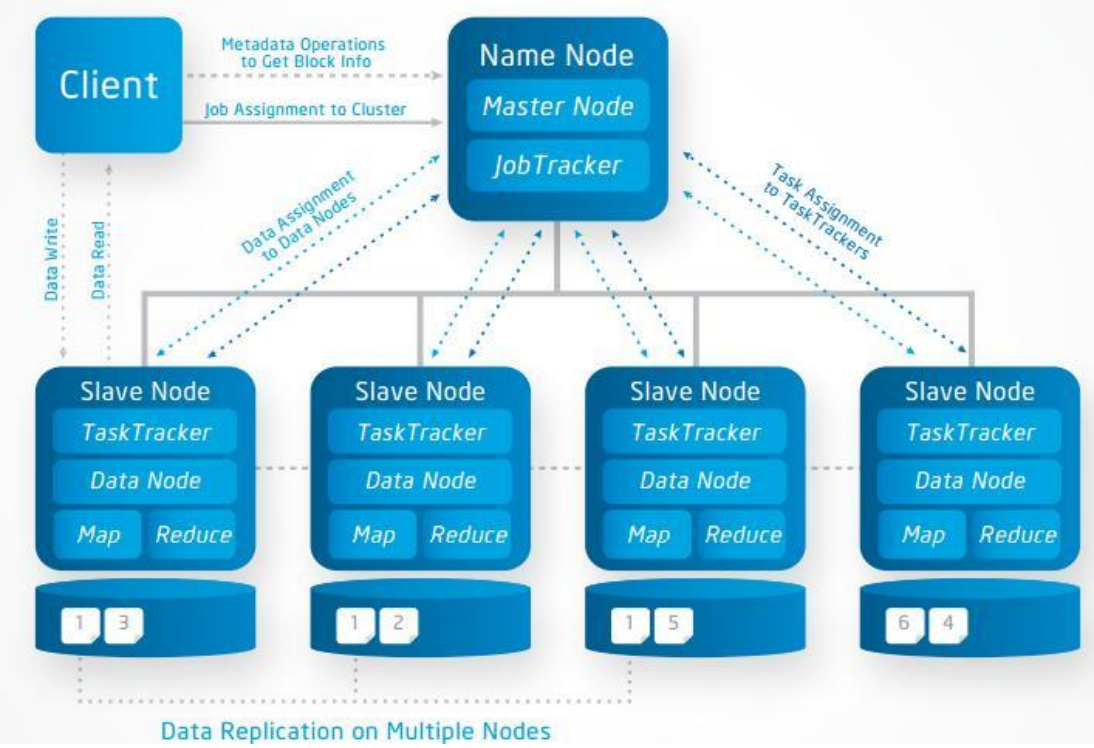


Figure 5. Hadoop distributed file system cluster architecture.

The Hadoop Distributed File System (HDFS) is an approach to store and investigate substantial static data files over different machines instead of a single machine holding the whole circle limit of the collected records. HDFS utilizes information replication and dispersion of the information and is made to be fault tolerant. A record is stacked into HDFS and is duplicated, and divided into units called blocks, which are commonly 64 MB of information and handled and put away with a bunch of hubs or machines called Data Nodes. HDFS utilizes the Master and Slave design where the Master (Name node) is in charge of the administration of metadata and execution of jobs to the Data Node.

Hadoop daemons. As indicated by the Apache Hadoop, A simple Hadoop cluster incorporates a single master and different slave nodes. The master node includes a Namenode, Job Tracker, Task Tracker, and Data Node.

Hadoop comprises five daemons. They are separate from the master node and slave nodes. Master daemons include three Hadoop daemons, for example, the Namenode, Secondary-name node and a Job Tracker. The remaining daemons are the two slave daemons, the Data Nodes, and the Task Tracker. A daemon is a foundation procedure. The master daemons can converse with each other, and all slave daemons coordinate amongst each other. On the off chance that a Namenode is a Master node, its associated slave node is a Data Node. Job Trackers converse with Task Trackers. If the Name node is a Job Tracker, its associated slave node is a Task Tracker as shown in Figure 6 below**.**

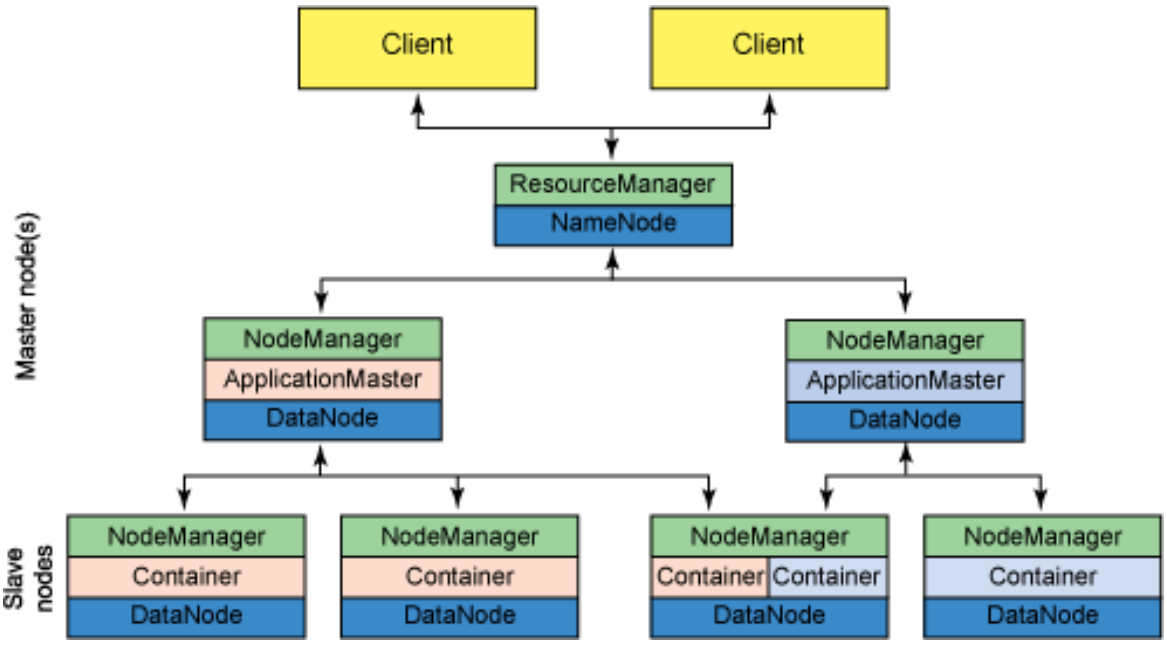


Figure 6. HDFS daemons and Hadoop core components.

- Namenode: The Namenode is used to hold the Metadata (information about the location, size of files/blocks) for HDFS. The Metadata gets stored on RAM or Hard-Disk. There is only one Namenode in a cluster. Failure of the Namenode causes complete failure of the Hadoop cluster.

- Secondary Namenode: It acts as a backup for the metadata stored in Namenode. It holds the fs-image and edit-log file information. When the Namenode fails, the Secondary-Namenode shares the latest fs-image and edit-log stored by it so that the data stored in the data nodes are not orphaned.
- Data Node: While the Namenode stores Metadata, the actual data gets stored on Data Nodes. The number of Data Nodes required depends on the data size. Users can additionally add them if required. The Data Node communicates to the Namenode on a frequent basis (every 3 seconds). However, this frequency gets modified by altering the settings.
- Job Tracker: The Namenode and Data Nodes store details and actual data on HDFS. It is also essential to process this data as per the client's requirements. The developer writes a MapReduce code to process the data, and the MapReduce engine sends the code across Data Nodes, creating jobs. The Job tracker monitors and manages these jobs continuously.
- Task Tracker: Task trackers perform the jobs given by Job trackers. Each Data Node has at least one task tracker. Task trackers communicate with Job trackers to send statuses of the jobs.

MapReduce Architecture and Implementation

MapReduce is a data grooming or parallel programming model developed by Google. In this model, a user determines the algorithm by two functions, Map and Reduce. In the mapping stage, the mapper takes the information and stores every record to the mapper. In the reducer stage, the reducer receives each one of the records from the mapper and arrives at final output. In

basic terms, the mapper is intended to channel and change the results to something that the reducer can aggregate over. The original MapReduce library consequently parallelizes the algorithm and handles complicated problems like data transmission, load balancing, and adaptation to internal failure. As enormous information get spread across numerous machines, there is a need to parallelize events such as transferring the data and giving booking, adjusting to non-critical failure. The first MapReduce execution by Google, and also its open source partner, Hadoop, attempted parallel processing in expanding clusters of machines. MapReduce has picked up a remarkable prominence as it effortlessly accomplishes adaptation to non-critical failure. It naturally handles the social affair of results over the various nodes and returns a single outcome or set. MapReduce demonstrates an advantage in the simple scaling of information handling over various processing nodes.

MapReduce is a system for handling extensive data sets simultaneously over a cluster of machines. Data Analysis utilizes a two-stage Map-Reduce process. The Resource Manager supplies MapReduce Analytics capacities, and the Hadoop system gives the scheduling, distribution, and parallelization services.

- Fault tolerance: MapReduce is intended to be fault tolerant because failures are an inevitable event in a large cluster which distributes data across various machines. The Hadoop Framework achieves fault tolerance through Data Replication. The NameNode splits a file into smaller chunks of data called Blocks. Each of these blocks gets stored in a different DataNode. Now if one of these data node fails, the block in that node is lost, and thus the file cannot be completely retrieved. At this

juncture, Hadoop Fault tolerance comes into play. The Data Replication feature enables Hadoop to store replicas of a block in a different Data Node.

- DataNode failure: The NameNode receives the heartbeat from each DataNode every three seconds or at specific intervals of time, which gets configured by the user. This heartbeat indicates the physical availability of each DataNode, and when the NameNode stops receiving the heartbeat for 10 minutes or a specific duration as configured by the client, the NameNode assumes the DataNode failure. The current task and any tasks unfinished by this DataNode are re-allotted to another DataNode and executed from the earliest starting point. Finished tasks do not need to be re-executed because their results get stored in the Hadoop Distributed File System.
- Master failure: As the master is a single machine, the likelihood of master failure is high. MapReduce re-executes the whole job if the master comes up short. There are as of now three prevalent executions of the MapReduce programming model specifically Google MapReduce, Apache Hadoop, Stanford Phoenix.

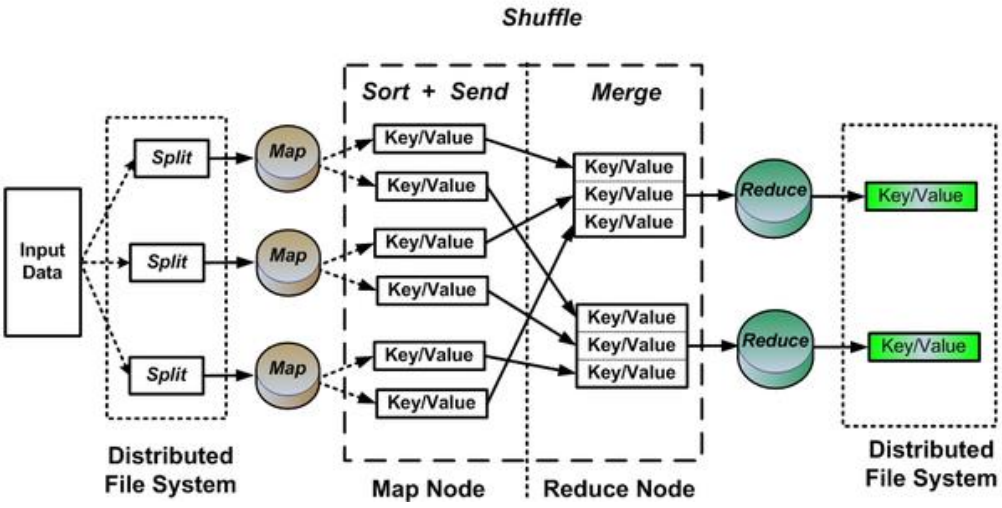


Figure 7. The architecture of MapReduce.

In the MapReduce programming model, MapReduce job comprises Map tasks and Reduce tasks. When a Job calls the Mappers: MapReduce first divides the data into N blocks with size ranging from 16MB to 64MB. Then it initiates numerous daemons on a group of various machines. One of the daemons is the Job-Tracker on the NameNode program; the others are Task-Trackers on the DataNodes, which can execute their work assigned by the Client machine. Master can provide an appropriate Map task or a Reduce task to a DataNode without having to move the Data because of the Data Localization feature of Hadoop Framework. Whenever a slave gets a Map task, it parses the Data Block and yield the key/value sets, at that point and passes the pair to a client characterized for Map work. The Map function keeps the transitory key/value matches in memory. The sets occasionally are composed of a closed set and divided into P pieces. From that point forward, the next machine advises the master of the area of these sets. If a DataNode is given a Reduce assignment and informed about the location of these data sets, the Reducer reads the whole buffer by utilizing remote technique calls. From that point onward, it sorts the brief information in light of the key. At that point, the reducer manages the more significant part of the records. For each key and agree(NOUN)on the set of qualities, the reducer passes key/value sets to a client characterized for Reduce work. This yield is the last yield of this segment. After the more significant part of the mapper and reducers has finished their work, the master restores the outcome to clients' project. The yield is put away in singular documents.

Pig Architecture and Components

In the Map-Reduce framework, jobs (programs) must be converted into the continuous format of Map and Reduce stages. Furthermore, as this is not user-friendly for someone like data

analysts who are not familiar with the programming language. So, to cover up this difference, an abstraction known as Pig was created over Hadoop. Pig is a high-level programming language used for analyzing large data sets. The Yahoo! Team developed the pig.

Pig's main agenda was to allow people to focus more on managing and analyzing massive data sets, and along with that to spend little time coding Map-Reduce programs. Similar to a pig, which eats anything, the Pig programming language is designed to work with any data. That's why the name, Pig.

Pig consists of two components: **Pig Latin** which is a scripting language, and **the runtime environment,** for running Pig Latin programs.

A Pig Latin program consists of a sequence of transformations which are used as input data to give specific output. These transformations are known as a data flow which is used to convert into an executable format, by Pig's execution environment. As a result of these transformations, Pig creates sequences of MapReduce tasks automatically without the interference of a programmer. So, in a way, Pig allows the programmer to focus on data rather than the nature of execution. PigLatin is a relatively simple language which uses common keywords from data processing, e.g., Join, Group, and Filter.

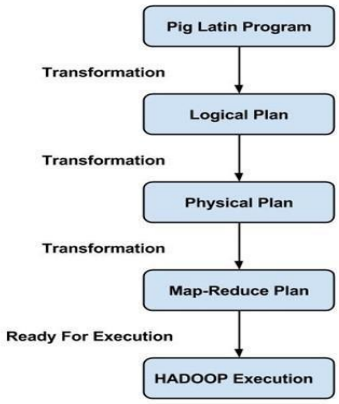


Figure 8. Pig components and execution mode.

Pig has two distinct modes of execution:

1. **Local mode:** Here Pig executes in a single JVM where it makes use of the local file system. The Local mode is suited explicitly for analyzing small data sets by Pig.
2. **MapReduce mode:** In this mode, queries written in Pig Latin are translated into MapReduce jobs and get executed on a Hadoop cluster (cluster may be pseudo or fully distributed). MapReduce mode with the fully distributed cluster is useful for running Pig on large datasets.

For processing of massive datasets in substantial parallelization, Apache Pig is used as a platform. The structure of Pig gets described in two layers: an infrastructure layer with an inbuilt compiler that can produce MapReduce programs and a language layer with a text-processing language called “Pig Latin.”

Pig makes data analysis and programming more manageable and understandable for beginners in the Hadoop framework by utilizing Pig Latin which can be known as a parallel data

flow language. The pig setup also has provisions for further optimizations and user-defined properties.

Pig Latin queries are merged and compiled into MapReduce tasks. Then, they get executed in distributed Hadoop cluster environments.

Pig Latin looks like SQL [Structured Query Language], but yet it got designed for Hadoop's data processing environment, just like SQL for RDBMS environment. Next, another member of the Hadoop family is Hive, which is a query language (similar to SQL). Before using Hive, the data should get loaded into tables. It works on schema-less or inconsistent nature and can be operated on the available data as soon as it gets loaded into HDFS environment. Pig is similar to scripting languages like Perl and Python in certain aspects as it is flexible in syntax and dynamic. So, Pig Latin is efficient as a native parallel processing language for distributed systems such as Hadoop.

MongoDB Architecture

MongoDB does not organize data in tables with columns and rows. Instead, data gets stored in "documents," every one of which is an affiliated cluster of scalar values, lists, or nested associative arrays. MongoDB records are typically serialized as JavaScript Object Notation (JSON) objects and are in reality stores utilizing a binary encoding of JSON called BSON. To scale its execution on a cluster of machines, MongoDB utilizes a procedure called Sharding, which is the process of splitting data uniformly over a group of machines to parallelize data.

The parallelization gets executed by classifying the MongoDB server into two groups; a group of front-end routing servers (MongoS) that redirect activities to another group of back-end data servers (MongoD). MongoDB queries inspect one record at any given moment, which

implies that queries over different documents must be executed on the clients or utilized in different MongoDB's integrated MapReduce (MR). Even though MongoDB's MR can get executed in parallel at every shard, there are two noteworthy downsides. One is the language for MR code is JavaScript, and which is sluggish and has insufficient analytics libraries, and the second one is the SpiderMonkey JavaScript usage utilized by MongoDB is not thread safe, so multiple MapReduce programs cannot get executed simultaneously.

HDFS vs. MongoDB Design

While HDFS is ideal for sequential analysis from substantially large chunks of data, MongoDB is enhanced for random and parallel processing, i.e., through queries to the data. The outcomes of this paper also demonstrate that MongoDB displays inefficiency for parallel storage of data because of the global write lock. Through data replication, both Hadoop and MongoDB offer data reliability. With MongoDB, the client can pick the number of replicas stages completed a writing data before the activity gets completed; this provides scalability of data, but on the other hand, this could lead to data loss if the client connection gets misconfigured. HDFS replicates the data present in the data-nodes by the replication factor which is either pre-defined by default in the HDFS or re-configured by the user.

Chapter IV: Experiment Setup

Installing and Configuring Apache Hadoop

The Hadoop Installation process involves the following steps:

Checking for any updates

user@ubuntu:~$ sudo apt-get update

Installing JDK 6

user@ubuntu:~$ sudo apt-get install sun-java6-jdk

Creating a dedicated Hadoop User account to run Hadoop

user@ubuntu:~$ sudoaddgrouphadoop_group

user@ubuntu:~$ sudoadduser --ingroup hadoop_group hduser1

Adding hduser1 to the sudo group

user@ubuntu:~$ sudoadduser hduser1 sudo

Now, configuring SSH

user@ubuntu:~$ su – hduser1

hduser1@ubuntu:~$ ssh-keygen -t rsa -P ““

Providing access to the new machine using the key created

hduser1@ubuntu:~$ cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys

Finally set up SSH connection

hduser@ubuntu:~$ ssh localhost

Installing Hadoop and its related extensions

Switching to hduser1

hduser@ubuntu:~$ su - hduser1

Now, download and extract Hadoop latest version Hadoop 3.0.0-alpha1

Setup environment variable for Hadoop

export HADOOP_HOME=/usr/local/Hadoop

Adding Hadoop bin/ directory to PATH

export PATH= $PATH:$HADOOP_HOME/bin

Change the file: conf/hadoop-env.sh

#export JAVA_HOME=/usr/lib/j2sdk1.5-sun

export JAVA_HOME=/usr/lib/jvm/java-6-openjdk-amd64 (for 64 bit)

export JAVA_HOME=/usr/lib/jvm/java-6-openjdk-amd32 (for 32 bit)

Setting up required ownerships and permission

hduser@ubuntu:~$ sudomkdir -p /app/Hadoop/tmp

hduser@ubuntu:~$ sudochownhduser:hadoop /app/Hadoop/tmp

hduser@ubuntu:~$ sudochmod 750 /app/Hadoop/tmp

Paste the following configuration in conf/core-site.xml

<property>

<name>hadoop.tmp.dir</name>

<value>/app/Hadoop/tmp</value>

<description>The location for other temporary directories.</description>

</property>

Eg: fileName- default(hadoop_uri)

<property>

<name>fs.default.name</name>

<value>hdfs://localhost:54310</value>

<description>

The name of file system is default. To determine the File System, authority, and scheme a URI is required. The config property got defined by Uri's scheme and (fs.SCHEME.impl). Then, identify the "FileSystem Implementation Class."

Using, the uri's authority the host, port, etc. for a file system is known.

</description>

</property>

Paste the following configuration in conf/mapred-site.xml

<property>

<name>mapred.job.tracker</name>

<value>localhost:54311</value>

<description>The port number that the MapReduce job tracker is hosted

at. For local machines, then jobs are run in-process as a single map and reduce task.

</description>

</property>

Pasting the following code in file conf/hdfs-site.xml

<property>

<name>dfs.replication</name>

<value>1</value>

<description>Default block duplication.

The exact number of duplication can be known when the file gets generated.

The default gets used if duplication is missing during the creation time.

</description>

</property>

Format the HDFS Filesystem via name node

hduser@ubuntu:~$ /usr/local/Hadoop/bin/Hadoop name node –format

Installing and Configuring Pig

Downloading the latest version of Pig from http://hadoop.apache.org/releases.html

cd Downloads/

Unzip the tar file.

$ tar -xvf pig-0.11.1.tar.gz

Create a directory

$ sudomkdir /usr/lib/pig

move pig-0.11.1 to pig

$ mv pig-0.11.1 /usr/lib/pig/

Set the pig_HOME path in the bashrc file

To open bashrc file use this command

$ gedit ~/.bashrc

In bashrc file append the below two statements

export pig_HOME=/usr/lib/pig/pig-0.11.1

export PATH=$PATH:$pig_HOME/bin

Restart the computer or use [. .bashrc]

Now let’s test the installation

On the command prompt type

$ pig -h

It shows the help related to Pig, and its various commands.

Starting pig in local mode

$ pig -x local grunt>

Starting pig in MapReduce mode

$ pig -x MapReduce

4.3 Installation and configuring MongoDB:

Importing the Public Key

sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 7F0CEB10

Creating a List File

echo “deb http://repo.mongodb.org/apt/ubuntu “$(lsb_release -sc)”/mongodb-org/3.0 multiverse” | sudo tee /etc/apt/sources.list.d/mongodb-org-3.0.list

Installing and Configuring MongoDB

sudo apt-get install -y mongodb-org

Verifying if the MongoDB is up and running service MongoDB status.

Now, executing the various phases of Data Analytics such as Storing, processing and retrieving with the unstructured data on both PIig/Hadoop and MongoDB. After careful comparison of the processing time taken by both Pig/Hadoop and MongoDB, then, the functionalities that are to be performed by the frameworks are determined.

Chapter V: Evaluation

In this paper, my data analysis gets performed over unstructured data. Hence, the HTTP web-log data belonging to a client-server environment is extracted from open source GitHub. This dataset contains rows with eight columns, each column delimited by commas. The eight columns and the kind of data contained are as follows:

1. **Time:** The timestamp when the log got recorded.
2. **Remote_ip:** The source IP address.
3. **Remote user:** The name of the user.
4. **Request:** The request made by the user or the client machine.
5. **Response:** The response given from the server to request made by the client.
6. **Bytes:** How many bytes of data got sent in response.
7. **Referrer:** Header information.
8. **Agent:** The machine and the application that made the request.

The entire data sums up to 80 GB, and only subsets of this data get utilized for the analysis. The analysis process involves two basic operations, a find with a filter (or called Select in SQL) and an aggregation (Sum). By using the filter operation, the records are segregated based on successful and unsuccessful responses made by the server. By using the sum function, a summation of the bytes sent for all records that have a successful response gets executed.

These operations help in understanding a security breach, to find a pattern in the unsuccessful attempts. A further analysis gets performed by filtering which IP-address are causing more unsuccessful attempts and if they represent a Brute-Force attack or a DOS attack. Thus, the operations performed here are especially useful, when analyzing a large amount of log data, and thus they help streamline the focus on the potential cause for the security breach.

Any data analytics problem, in general, would involve searching for a subset of the data, based on a particular condition and an aggregation over the obtained data. Hence, the most basic operations which are as simple as find() and sum() got chosen.

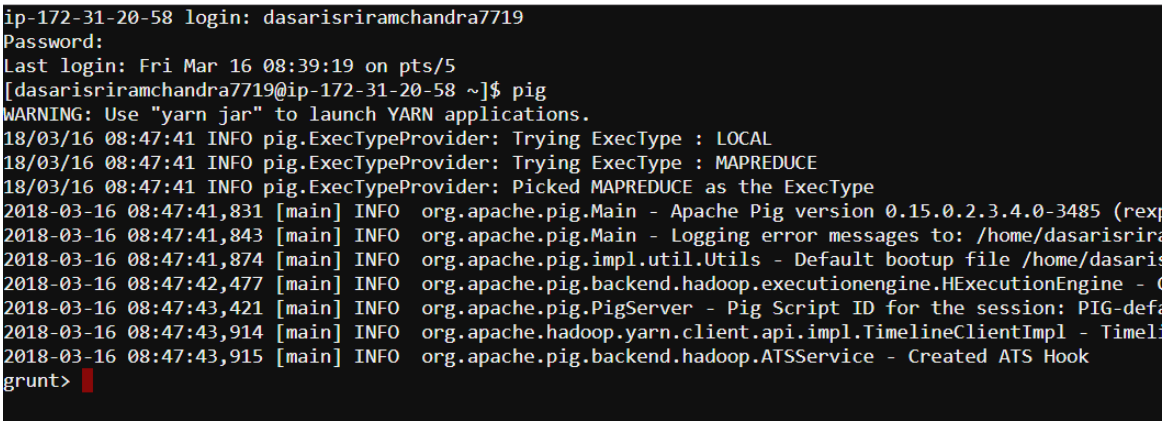
Table 3

HDFS and MongoDB

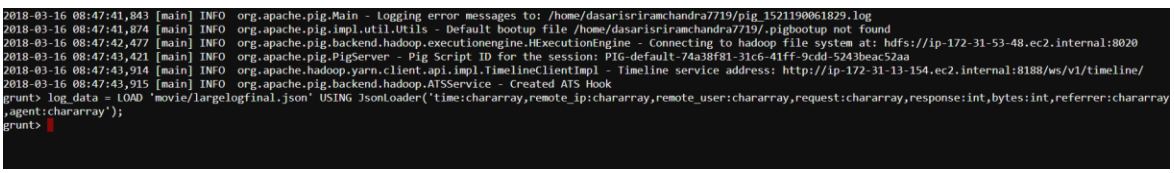
Characteristic	HDFS	MONGODB
Storage	Distributed file system	Distributed schema-less Database, in-memory at each node
Reading	Sequential and block access	Random and sequentiaal access
Writing	Cached locally then sent to Data Node	Journaled writed to index and data blocks
Reliability	Replication	Replication

Pig Experimental Results

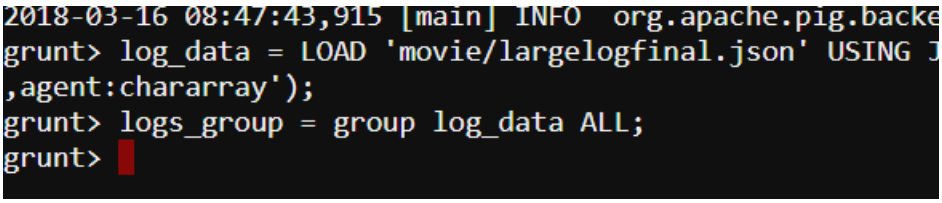
Login into the Pig environment.

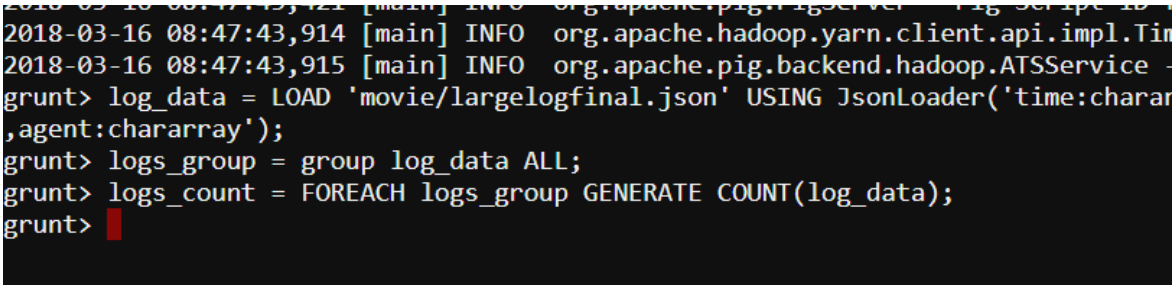


Loading Data.

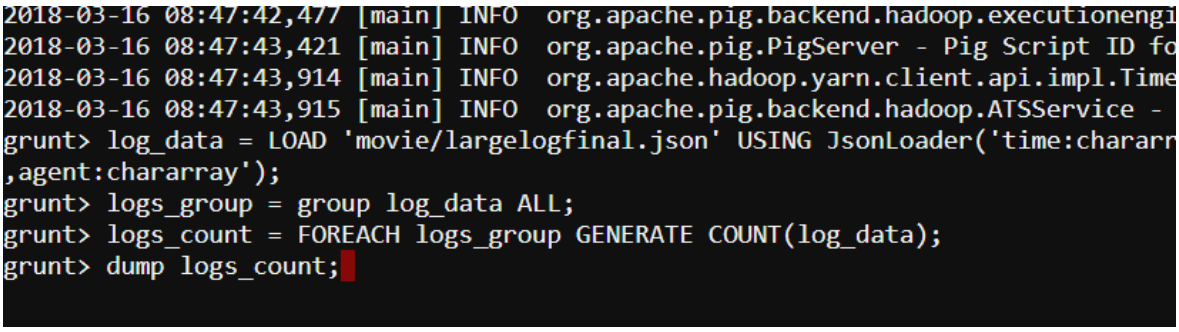


Grouping Data.

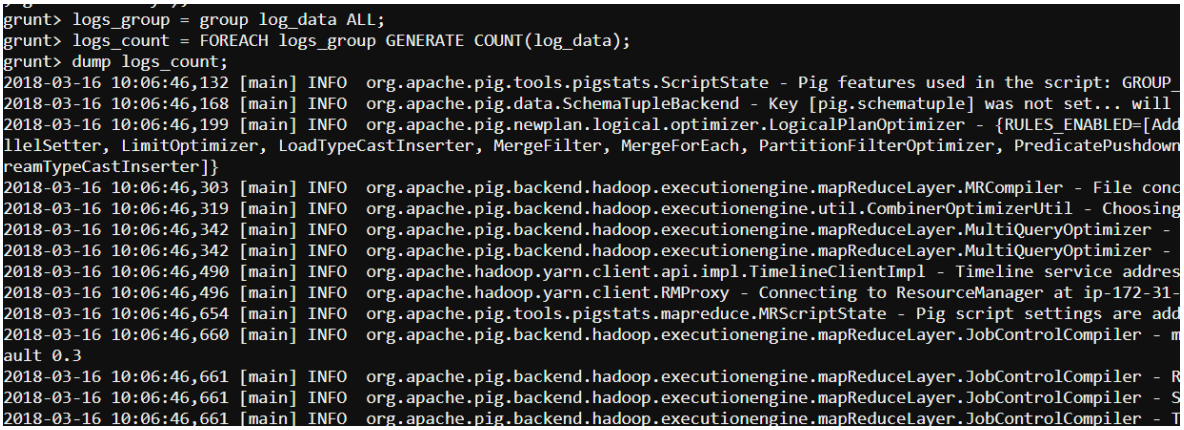




Performing count on the Grouped Data.



Displaying the count on the Group data.



The count job ran successfully.

```
r
2018-03-16 10:07:23,531 [main] INFO  org.apache.hadoop.yarn.client.api.impl.TimelineClientImpl - Timeline service addr
2018-03-16 10:07:23,532 [main] INFO  org.apache.hadoop.yarn.client.RMProxy - Connecting to ResourceManager at ip-172-3
2018-03-16 10:07:23,538 [main] INFO  org.apache.hadoop.mapred.ClientServiceDelegate - Application state is completed.
r
2018-03-16 10:07:23,696 [main] INFO  org.apache.hadoop.yarn.client.api.impl.TimelineClientImpl - Timeline service addr
2018-03-16 10:07:23,697 [main] INFO  org.apache.hadoop.yarn.client.RMProxy - Connecting to ResourceManager at ip-172-3
2018-03-16 10:07:23,701 [main] INFO  org.apache.hadoop.mapred.ClientServiceDelegate - Application state is completed.
r
2018-03-16 10:07:23,759 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher -
2018-03-16 10:07:23,762 [main] INFO  org.apache.pig.tools.pigstats.mapreduce.SimplePigStats - Script Statistics:

HadoopVersion   PigVersion      UserId  StartedAt       FinishedAt      Features
2.7.1.2.3.4.0-3485      0.15.0.2.3.4.0-3485     dasarisriramchandra7719 2018-03-16 10:06:46     2018-03-16 10:07:23

Success!

Job Stats (time in seconds):
JobId   Maps    Reduces MaxMapTime      MinMapTime      AvgMapTime      MedianMapTime   MaxReduceTime   MinReduceTime
```

The Job Statistics of the above count job.

```
Success!

Job Stats (time in seconds):
JobId   Maps    Reduces MaxMapTime      MinMapTime      AvgMapTime      MedianMapTime   MaxReduceTime   MinReduceTime   AvgRedu
job_1517296050843_7523  5       1       22      6       14      10      13      13      13      13      log_data,logs_count,log
c2.internal:8020/tmp/temp-320246660/tmp1929978005,

Input(s):
Successfully read 2573100 records (609988343 bytes) from: "hdfs://ip-172-31-53-48.ec2.internal:8020/user/dasarisriramchandra771

Output(s):
Successfully stored 1 records (9 bytes) in: "hdfs://ip-172-31-53-48.ec2.internal:8020/tmp/temp-320246660/tmp1929978005"

Counters:
Total records written : 1
Total bytes written : 9
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0
```

Performing a Sum of the bytes received.

```
grunt> bytessum = FOREACH logs_group GENERATE SUM(log_data.bytes);
grunt> dump bytessum;
2018-03-16 10:17:26,111 [main] INFO  org.apache.pig.tools.pigstats.ScriptState - Pig features used
2018-03-16 10:17:26,137 [main] INFO  org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple]
2018-03-16 10:17:26,137 [main] INFO  org.apache.pig.newplan.logical.optimizer.LogicalPlanOptimizer
llelSetter, LimitOptimizer, LoadTypeCastInserter, MergeFilter, MergeForEach, PartitionFilterOptimiz
reamTypeCastInserter]}
2018-03-16 10:17:26,141 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.M
2018-03-16 10:17:26,142 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.CombinerOpt
2018-03-16 10:17:26,144 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.M
2018-03-16 10:17:26,144 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.M
2018-03-16 10:17:26,238 [main] INFO  org.apache.hadoop.yarn.client.api.impl.TimelineClientImpl - Ti
2018-03-16 10:17:26,238 [main] INFO  org.apache.hadoop.yarn.client.RMProxy - Connecting to Resource
2018-03-16 10:17:26,241 [main] INFO  org.apache.pig.tools.pigstats.mapreduce.MRScriptState - Pig sc
2018-03-16 10:17:26,241 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.J
ault 0.3
2018-03-16 10:17:26,241 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.J
2018-03-16 10:17:26,241 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.J
```

The Job performing the sum of received bytes ran successfully.

```
2018-03-16 10:17:52,285 [main] INFO  org.apache.hadoop.mapred.ClientServiceDelegate - Application state is completed. FinalApplicationSt
r
2018-03-16 10:17:52,311 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - 100% complete
2018-03-16 10:17:52,312 [main] INFO  org.apache.pig.tools.pigstats.mapreduce.SimplePigStats - Script Statistics:

HadoopVersion   PigVersion      UserId  StartedAt       FinishedAt      Features
2.7.1.2.3.4.0-3485      0.15.0.2.3.4.0-3485     dasarisriramchandra7719 2018-03-16 10:17:26     2018-03-16 10:17:52     GROUP_BY

Success!

Job Stats (time in seconds):
JobId   Maps    Reduces MaxMapTime      MinMapTime      AvgMapTime      MedianMapTime   MaxReduceTime   MinReduceTime   AvgReduceTime
job_1517296050843_7524  5       1       13      6       11      12      6       6       6       6       bytessum,log_data,logs_group
c2.internal:8020/tmp/temp-320246660/tmp888614124,
```

The Job Statistics of the sum job.

```
Success!

Job Stats (time in seconds):
JobId   Maps    Reduces MaxMapTime      MinMapTime      AvgMapTime      MedianMapTime   MaxReduceTime   MinReduceTime   AvgReduceT
job_1517296050843_7524  5       1       13      6       11      12      6       6       6       6       bytessum,log_data,logs_gro
c2.internal:8020/tmp/temp-320246660/tmp888614124,

Input(s):
Successfully read 2573100 records (609988343 bytes) from: "hdfs://ip-172-31-53-48.ec2.internal:8020/user/dasarisriramchandra7719/m

Output(s):
Successfully stored 1 records (13 bytes) in: "hdfs://ip-172-31-53-48.ec2.internal:8020/tmp/temp-320246660/tmp888614124"

Counters:
Total records written : 1
Total bytes written : 13
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0

Job DAG:
job_1517296050843_7524
```

MongoDB Experimental Results

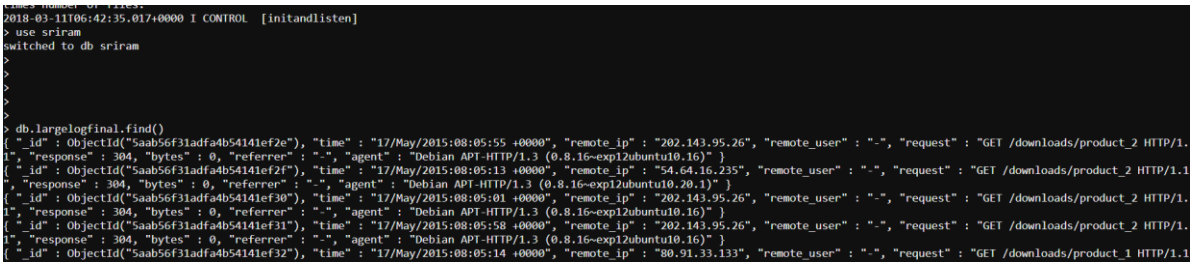
Loading file from the local path to MongoDB.

```
ip-172-31-20-58 login: dasarisriramchandra7719
Password:
Last login: Fri Mar 16 05:08:39 on pts/10
[dasarisriramchandra7719@ip-172-31-20-58 ~]$ ls
cloudxlab_jupyter_notebooks  pig_1521141938966.log  starredpaper
[dasarisriramchandra7719@ip-172-31-20-58 ~]$ ls starredpape
ls: cannot access starredpape: No such file or directory
[dasarisriramchandra7719@ip-172-31-20-58 ~]$ ls starredpaper
cardata.json  largelog.json  movie.json  website.json
[dasarisriramchandra7719@ip-172-31-20-58 ~]$ hadoop fs -ls ./movie/
Found 3 items
-rw-r--r--   3 dasarisriramchandra7719 dasarisriramchandra7719   12189237 2018-03-15 18:32 movie/largelog.json
-rw-r--r--   3 dasarisriramchandra7719 dasarisriramchandra7719  621651189 2018-03-16 04:55 movie/largelogfinal.json
-rw-r--r--   3 dasarisriramchandra7719 dasarisriramchandra7719    1641221 2018-03-15 17:56 movie/movie.json
[dasarisriramchandra7719@ip-172-31-20-58 ~]$ hadoop fs -get ./movie/largelogfinal.json starredpaper/largelogfinal.json
[dasarisriramchandra7719@ip-172-31-20-58 ~]$ cd starredpaper
[dasarisriramchandra7719@ip-172-31-20-58 starredpaper]$ ls
cardata.json  largelogfinal.json  largelog.json  movie.json  website.json
[dasarisriramchandra7719@ip-172-31-20-58 starredpaper]$ mongoimport --db sriram --collection largelogfinal --file largelogfinal.json
2018-03-16T05:32:35.287+0000    connected to: localhost
```

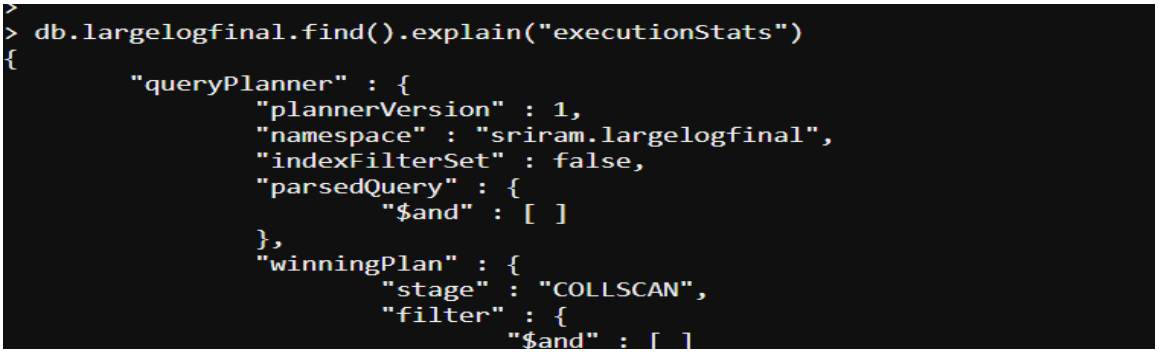
Open MongoDB.

```
ip-172-31-20-58 login: dasarisriramchandra7719
Password:
Last login: Fri Mar 16 05:27:24 on pts/9
[dasarisriramchandra7719@ip-172-31-20-58 ~]$ ls
cloudxlab_jupyter_notebooks  pig_1521141938966.log  starredpaper
[dasarisriramchandra7719@ip-172-31-20-58 ~]$ mongo
MongoDB shell version: 3.2.9
connecting to: test
Server has startup warnings:
2018-03-11T06:42:35.016+0000 I CONTROL  [initandlisten]
2018-03-11T06:42:35.017+0000 I CONTROL  [initandlisten] ** WARNING: /sys/kernel/mm
2018-03-11T06:42:35.017+0000 I CONTROL  [initandlisten] **        We suggest setti
2018-03-11T06:42:35.017+0000 I CONTROL  [initandlisten]
2018-03-11T06:42:35.017+0000 I CONTROL  [initandlisten] ** WARNING: /sys/kernel/mm
2018-03-11T06:42:35.017+0000 I CONTROL  [initandlisten] **        We suggest setti
2018-03-11T06:42:35.017+0000 I CONTROL  [initandlisten]
2018-03-11T06:42:35.017+0000 I CONTROL  [initandlisten] ** WARNING: soft rlimits t
times number of files.
2018-03-11T06:42:35.017+0000 I CONTROL  [initandlisten]
> use sriram
switched to db sriram
```

Simple find function.



Find function with execution stats.



Sum function with Time stats.

```
> db.largelogfinal.explain("executionStats").count()
{
        "queryPlanner" : {
                "plannerVersion" : 1,
                "namespace" : "sriram.largelogfinal",
                "indexFilterSet" : false,
                "winningPlan" : {
                        "stage" : "COUNT"
                },
                "rejectedPlans" : [ ]
        },
```

Task Running in MongoDB

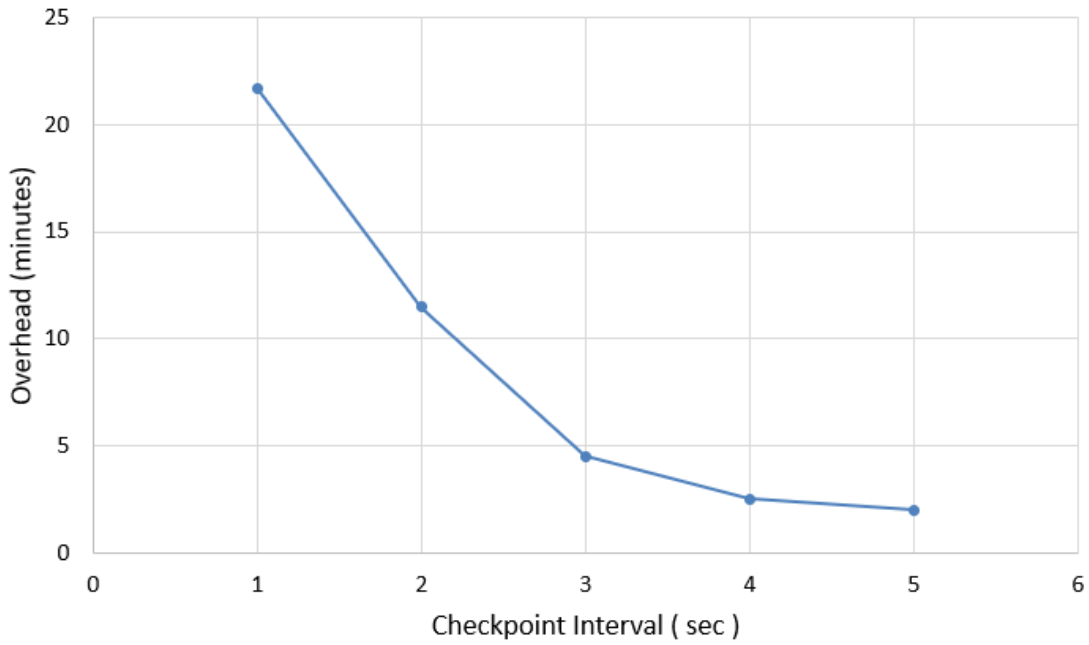


Figure 9. Overhead processing time caused by the frequency of checkpointing.

Evaluation of MongoDB

While performing various experiments throughout this paper, the first experiment's objective is to compare the performance of MongoDB with that of Hadoop in a large setup.

Hence, through this experiment, an effort is made to quantify the capability of MongoDB to handle multiple jobs on distinct worker nodes. To monitor the reliability and fault tolerance capability, MongoDB needs to adopt the method of checkpointing, in which the data nodes or the Mongo-worker nodes regularly report their status to the central node or the MongoDB central server node. Thus, this feature tests the ability of MongoDB in multi-tasking several jobs in parallel. In Figure 10, the graph illustrates the time taken to complete 520 tasks with checkpoints at various regular intervals. The 520 tasks take approximately 8 minutes when all of them run across 130 cores (i.e., four tasks per core). The total time for executing all the 520 tasks without any checkpointing is approximately 8 minutes, so this time is subtracted, and the difference is Overhead processing time. This Overhead processing time is the additional time consumed by the worker nodes to continually report their status to the MongoDB central server or any other additional time taken apart from the actual execution of the Pig script. The rate of the checkpointing is configured to occur at different time intervals in different instances for the same data-set, to observe the impact of the checkpointing on the performance of MongoDB. With the checkpoint interval configured to 5 seconds, the overhead processing time is less than 1 minute. As the checkpoint interval is gradually decreased from 5 seconds to 1 second, the frequency of checkpointing increases which leads to a rise in overhead to approximately 20 minutes. Thus, it is evident that the performance drops as the checkpoint interval gets decreased, because of the rise in the checkpointing connections per second from 130 to 520, and the rate of writing data to 520MB/sec.

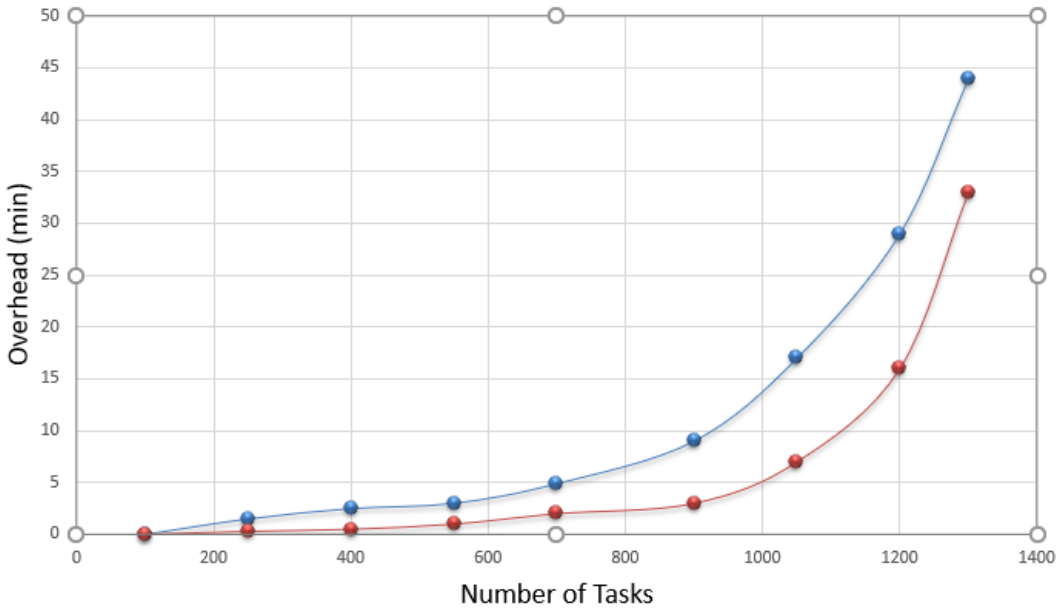


Figure 10. Overhead processing time generated due to a rise in the number of tasks.

In the previous experiment, the checkpointing influenced the performance when the number of nodes and tasks are kept constant. However, Figure 11 illustrates that the checkpoint size with the varying number of tasks per nodes also causes overhead processing time and influences the performance1. As the number of nodes and tasks increases, the overhead also increases differently for different checkpoint sizes. The time taken for each task is 8 minutes, and all the tasks run in parallel. Instead of varying the checkpoint interval, in the current experiment it is fixed at 10 seconds. However, the checkpoints do not influence the overhead processing time significantly, until the number of tasks is increased to 900 tasks approximately. When the checkpoints get configured to a size of 1MB, the performance dropped by four times between 1132 to 1300 tasks. On the contrary, if the checkpoint size gets configured as 64KB, the performance decreased almost three times from 1230 to 1300 tasks.

Both the size of data and the number of connections has an unfavorable effect on the performance, but from the outcomes of the above experiment, it is evident that the number of connections has a far more significant influence on performance when compared to the size of the dataset. Despite the datasets of the 1MB checkpoints being approximately 15 times that of the 64KB checkpoints, at the maximum tasks performed, i.e., 1300, the overhead of 1MB tasks are hardly 25% greater than the 64KB checkpoints. Thus, this data illustrates how multiple threads caused due to the overhead of multiple connections decrease the performance of MongoDB. MongoDB causes this lag in performance because it allocates a new thread for every new connection.

MongoDB vs. Pig/HDFS Performance

In the following experiment, the performance of MongoDB and Hadoop is measured, by comparing the read and write performance. For the analysis to be unbiased a java program and an equivalent python script got written which can read 30 million records and write 15 million records. The objective(aim) of the current experiment is to compare the performance of HDFS and MongoDB based on the reads and writes capabilities.

The setup for the current experiment consists of a MongoDB setup with two sharding servers and two HDFS data nodes. For the reading tests, each node is expected to import 30 million records. The data set consists of 100 columns, and it has the structure similar to Medicare Claims and Enrollment data sent to CMS by any Health Plan, but the data is fictitious and extracted from Github. The data includes member enrollment dates, service dates, diagnosis codes, premium amounts, benefit package codes, Member-Provider-Practice demographic data. Pig reads the current dataset read at a rate of 7.8 million records per minute from the HDFS and

on the other hand, reads the same data at a rate of 1.7 million records per minute from MongoDB. The write tests got performed with only two columns as the benchmark for both Pig/HDFS and MongoDB.

Thus, both Pig/HDFS and MongoDB read records that are 100 times larger than the records they wrote. For these records, the Pig/HDFS composes 13 million records in 13 seconds (60 million records per minute), while MongoDB takes 5 minutes (2.6 million records per minute). Therefore, this experiment, shows a 1:5 reading efficiency ratio between MongoDB and Pig/HDFS while writing huge records and for writing small records, the ratio is an enormous 1:23, which shows that reading performance of MongoDB and Pig/HDFS is closer when compared to the when compared to the writing performance.

MongoDB MapReduce

In spite of the inbuilt MongoDB MapReduce, the MongoDB-Hadoop connector enables the usage of Hadoop's MapReduce with MongoDB storage. MongoDB's MapReduce is scalable with its resources for executing MapReduce programs, i.e., for every MongoDB server, a mapper or reducer task is initiated. Figure 11 compares MongoDB's local MapReduce (MR) with Mongo-Hadoop's MapReduce. For this experiment, a single node MongoDB server and 2-node Hadoop cluster got installed. The results of this experiment show that the Mongo-Hadoop connector takes six times lesser time than that of MongoDB's MapReduce. The graph further demonstrates how the processing time of MongoDB MapReduce steeply rises with the increase in the number of records to be processed when compared to that of Hadoop-MongoDB MapReduce.

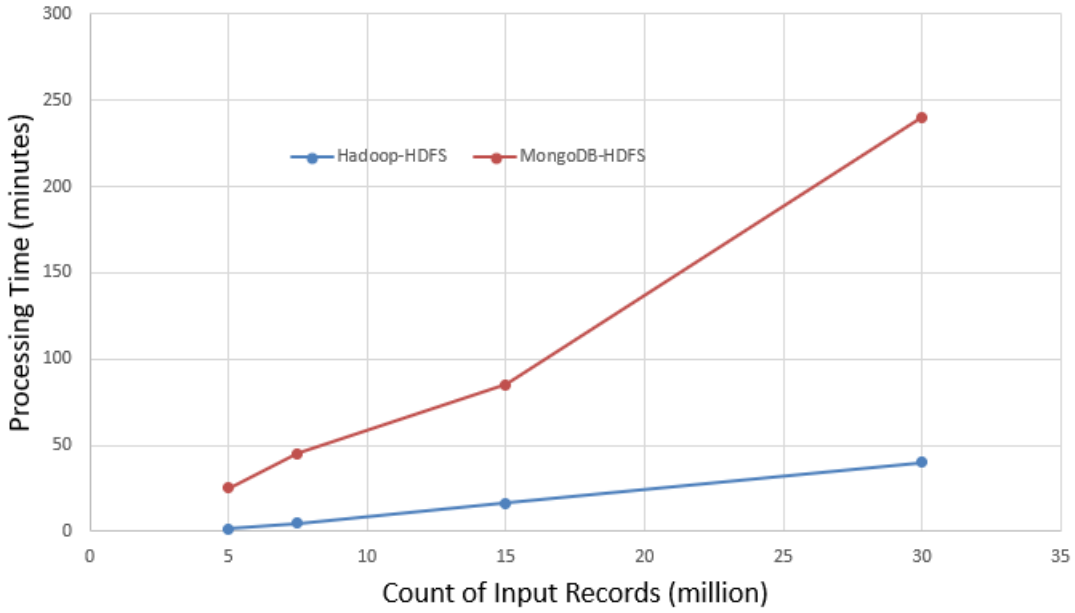


Figure 11. Performance of MongoDB vs. Hadoop based on the number of records.

Evaluating MongoDB Configurations

In both Hadoop and MongoDB, the pre-MapReduce process involves converting the large datasets into smaller subsets called Splits. Every mapper or reducer task deals either with Splits or aggregation of the split. The splitsize determines, the number of mappers that get initiated as each Split gets analyzed as a separate mapper. The smaller the Splitsize, higher is the number of mappers initiated. The number of mappers means more overhead resource management. Hence, optimizing the Splitsize plays a vital role in enhancing the performance of the cluster. The current experiment uses a 3.6GB dataset containing 8.4 million records for understanding the influence of Splitsize on Overhead processing time. If the split size is 8MB by default, then Hadoop initiates over 450 mappers. As mentioned earlier, too many mappers lead to a significant overhead on Resource Manager. Hence, increasing the Splitsize as shown in the

Figure12 below, reduces the number of mappers initiated and releases the stress on Resource Manager; thus, decreasing the processing time. However, the graph also illustrates that beyond 128MB (with 29 mappers initiated) the split-size does not seem to improve the processing time significantly. Therefore, for all the remaining experiments, the 128MB split-size is considered ideal.

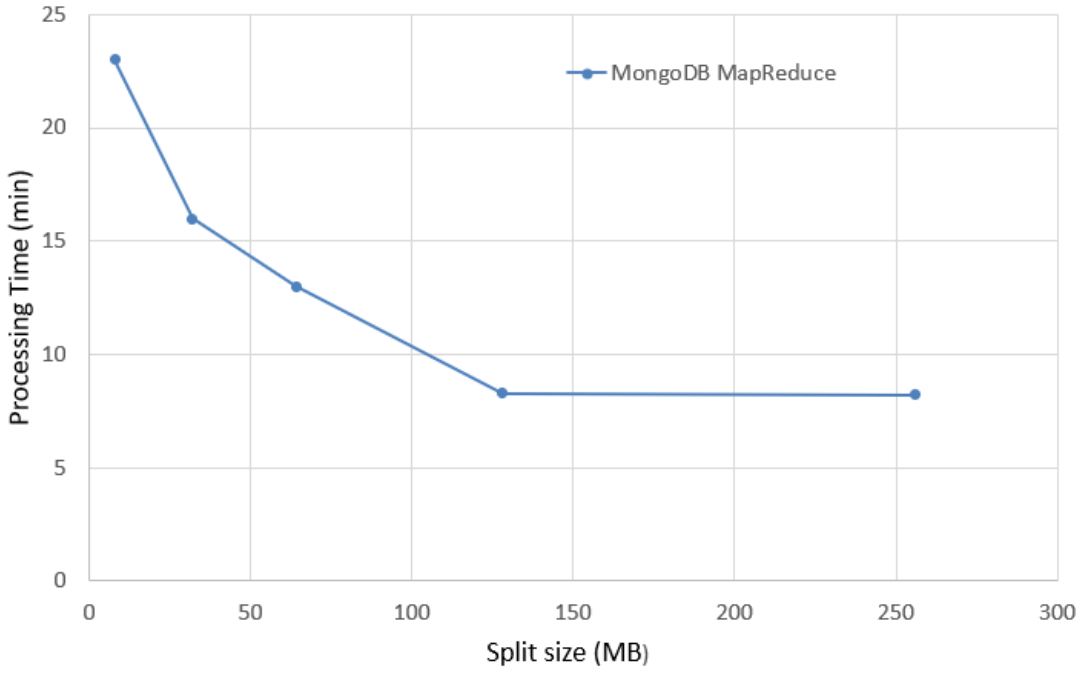


Figure 12. Effect of splitsize on the processing time of MongoDB's MapReduce.

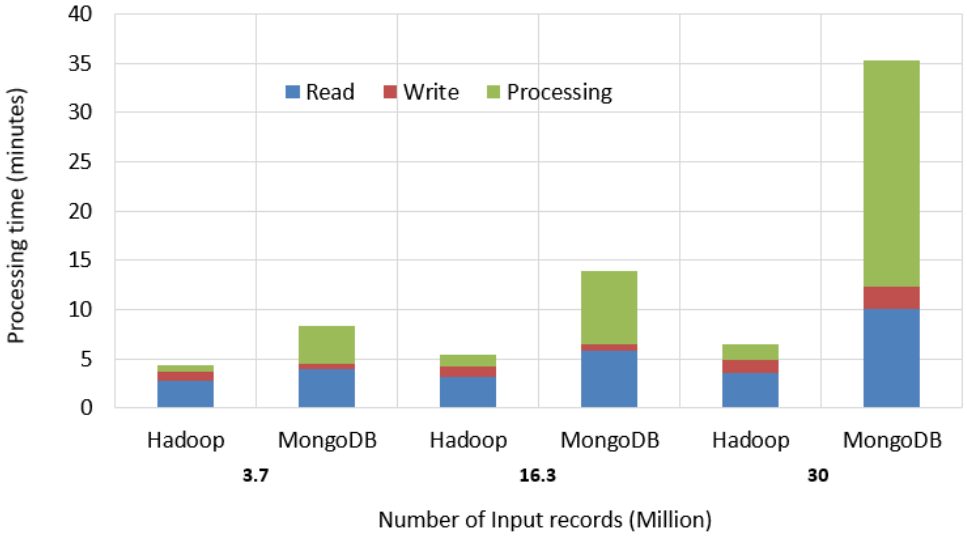


Figure 13. Effect of increasing records on processing time.

Figure 13 demonstrates the next experiment, which is comparing the read, write and processing time of Hadoop-HDFS on a 2-node cluster to MongoDB with two sharding servers setup, when dealing with large data sets. The graph illustrates that the performance of Hadoop is consistently higher than that of MongoDB. The gap between the processing time gradually increases with the size of the dataset. For a dataset of 3.7 Million records, Hadoop's processing time is half of MongoDB processing time, and at 30 Million records, the MongoDB is seven times slower than Hadoop.

Scalability Tests

The next experiment is to understand and compare the effects of horizontal scaling over Pig/Hadoop and MongoDB. As illustrated in Figure 13, increasing the cluster sizes and thereby increasing the number of cores from 8 to 64, decreases the reading time of MongoDB significantly, because the rise in the number cores improves the capability of handling multiple connections. However, there is no significant change in the writing time of both Hadoop and

MongoDB with increased scaling. Also, the previous experiments explain that the write times are influenced by reduce step of the MapReduce program. In these cases, MongoDB can handle multiple connections through a sharding setup. However, in this case, the growth in performance is constrained by the overhead time caused due to data-routing among the shards.

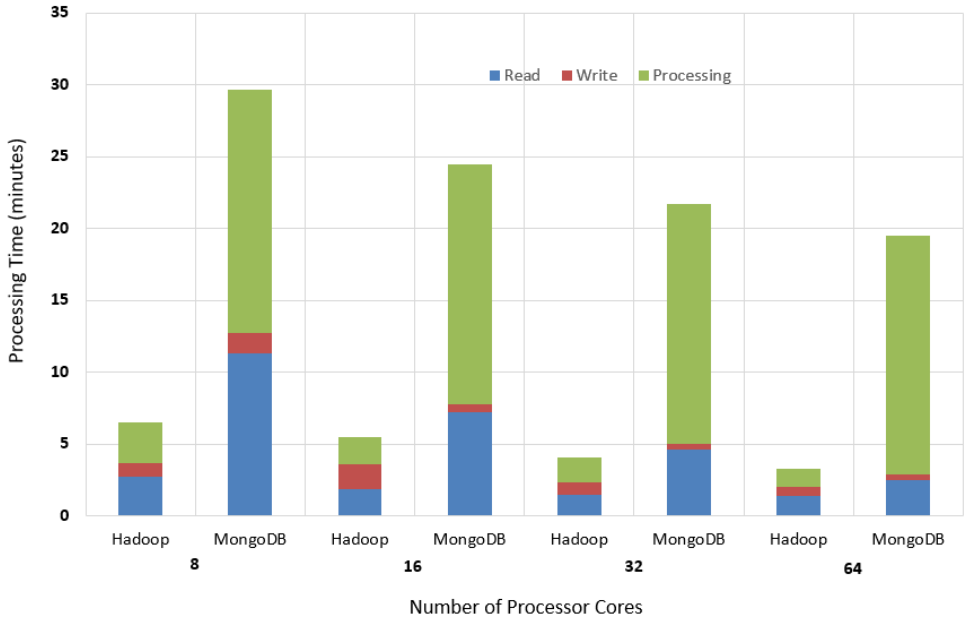


Figure 14. Effect of processor cores on processing time.

The next experiment compares the performance between Pig/Hadoop and MongoDB based on the reads and writes when the number of records is increasing. Figure 15 illustrates that the read and write times of Hadoop with HDFS are more efficient when compared to MongoDB read and write times. In spite of the Sharding servers in a MongoDB setup, the HDFS of Hadoop seems to be more efficient in reading and writing which helps in decreasing the overall run-time and thereby increasing the performance of Hadoop.

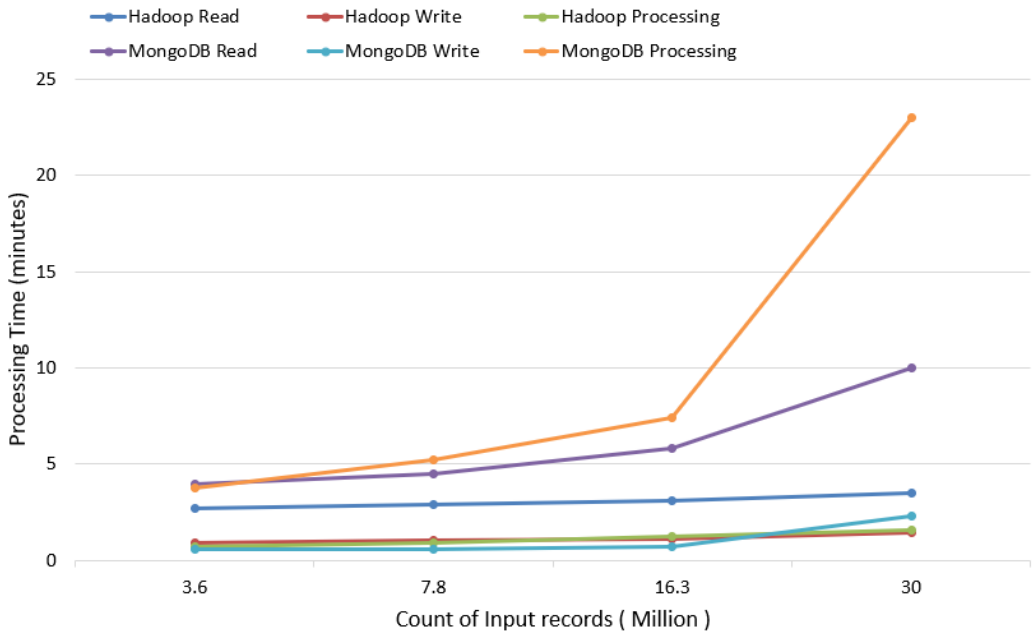


Figure 15. Comparing read, write, and processing individually with an increasing number of records.

The next graph in Figure 16 illustrates the comparison between the effects of scalability on MongoDB with that of Hadoop. As the cluster size gradually increases, thereby increasing the number of cores, the performance of Mapper programs increases because of the increasing mappers per Data Node. However, the reduce times do not change, significantly as the cluster does not require many reducers and therefore, the increase in the size of the cluster does not have an impact on Reduce part of MapReduce. Further, there is not a significant rise in performance beyond 32 nodes as the overhead processing time for managing multiple parallel connections for a relatively smaller data set, compensates any drop in the processing time.

Fault Tolerance in MongoDB

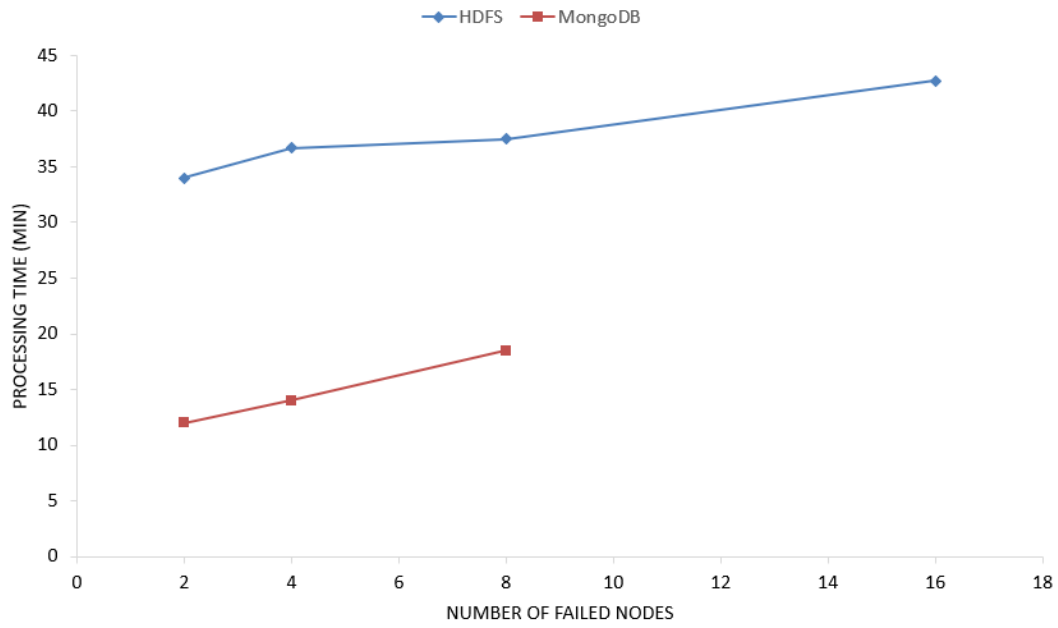


Figure 16. Fault tolerance of HDFS vs. MongoDB.

The next experiment is performed to compare the Fault Tolerance between Pig/Hadoop and MongoDB. For this experiment, a 32 node Hadoop cluster is used to read to write and process a data sample containing 40 million records. For consistency in the comparison, the same dataset is also used to calculate the read, write and processing duration with MongoDB. As the graph indicates, in spite of Hadoop having relatively lower processing durations, the Hadoop cluster encounters too much loss of data when it loses more than eight eight nodes, i.e., 25% of the total nodes installed. This loss of data nodes leads to incomplete MapReduce programs. On the other hand, the MongoDB is also started to lose to data after losing four shards, each present in different nodes.

Chapter VI: Conclusion and Future Works

Conclusion

The checkpoint intervals had a significant influence on the overhead performance duration of MongoDB. As the interval is decreased the number of connections increased and caused a rise in the overhead processing time. the rise in overhead processing duration is caused due to both larger data sets and the rise in a number of connections, tThe experiments illustrated in this paper indicate that rise in the number of connections has a more significant impact on overhead processing time, increasing it by approximately five times which is more than the overhead caused by large data sets.

The performance of the MapReduce program can be improved significantly by increasing the split size, which leads to a rise in number Mappers. However, the rise in the number of mappers also increases the number of connections which drops the performance. Thus, as illustrated in this paper, the split size must be chosen based on the cluster configuration and the number of nodes the cluster contains.

The comparison of performance between Hadoop and MongoDB involves several criteria out of which, only a handful got analyzed in this paper. This paper does not conclude that one of these two tools is the best. However, the findings of this paper can be used to choose a better tool between Pig/Hadoop and MongoDB, also considering the size of the data, the organization or an individual is trying to store and analyze and the kind of infrastructure and architecture that they can adopt.

Future Works

In the future, further studies in this direction can lead to the development of a better tool which might adopt the concepts of data locality from Hadoop and the indexing capability of MongoDB.

References

Arora, A., & Aggarwal, A. (2013). *An algorithm for transformation of data from MySQL to NoSQL (MongoDB)*. Retrieved from http://ijascse.org/volume-2-special-issue-1/An_algorithm.pdf.

Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., …Gruber, R. E. (2006). Bigtable: A distributed storage system for structured data. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation*, Vol. 7, OSDI '06, pp. 5-15, Berkeley, CA, USA, USENIX Association.

Chen, M., Mao, S., & Liu, Y. (2014). *Big data: A survey*. Wuhan 430074 China: Springer, School of Computer Science and Technology, Huazhong University of Science and Technology.

Chodorow, K., & Dirolf, M. (2010). *MongoDB: The definitive guide*. Boston, MA: O'Reilly Media Inc.

Cooper, B. F., Ramakrishnan, R., Srivastava, U., Bohannon, P., . . . Yerneni, R. (2008). Pnuts: Yahoo!'s hosted data serving platform. *Proceedings VLDB Endowment, 1*(2), 1277-1288.

Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., & Sears, R. (2010). Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC '10*, (pp.143-154). New York, NY, USA, ACM.

Dean, J., & Ghemawat, S. J. (2008). MapReduce: Simplified data processing on large clusters. *Communications of the ACM, 51*(1), 107-113.

Dory, T., Mejas, B., Roy, P. V., & Tran, N-L. (2011). Measuring elasticity for cloud databases. In *Proceedings of the Second International Conference on Cloud Computing, GRIDs, and Virtualization*.

Fadika, Z., & Govindaraju, M. (2010). Lemo-MR: Low overhead and elastic MapReduce implementation optimized for memory and CPU-intensive applications. In *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, CLOUDCOM '10 (pp. 1-8). Washington, DC, USA, 2010. IEEE Computer Society.

Floratou, A., Teletia, N., Dewitt, D., Patel, J., & Zhang, D. Z. (2012). Can the elephants handle the NoSql onslaught? In *International Conference on Very Large Data Bases, VLDB, 5*(12), 1712-1723.

Ghemawat, S., Gobioff, H., & Leung, S-T. (2003). The Google file system. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles, SOSP '03* (pp. 29-43). New York, NY, USA, 2003. ACM.

Guster , D., O'Brien, A. Q., & Lebentritt, L. (2013). *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.

Havens, T. C., Bezdek, J. C., Leckie, B., Hall, L. O., & Palaniswami, M. (2012). Fuzzy c-Means algorithms for very large data. *IEEE Trans. on Fuzzy Systems, 20*(6), 1130-1146.

Jain, A., Ong, S. P., Hautier, G., Chen, W., Richards, W. D., Dacek, S., . . . Persson, K. A. (2013). The materials project: A materials genome approach to accelerating materials innovation. *APL Materials, 1*(1), 011002. doi:10.1063/1.4812323.

Jain, B., & Kakhani, M. K. (2015). Query optimization in hive for large datasets. *Advances in Computer Science and Information Technology (ACSIT), 2*(4).

Lakshman, A., & Malik, P. (2009). Cassandra: A decentralized structured storage system. In *Proceedings of the 28th ACM Symposium on Principles of Distributed Computing, PODC '09* (pp. 5-15). New York, NY, USA, ACM.

Lawrence, R. (2014). Integration and virtualization of relational SQL and NoSQL systems including MySQL and MongoDB. *International Conference on Computational Science and Computational Intelligence (CSCI), IEEE*.

Mapanga, I., & Kadebu, P. (2013). Database management systems: A NoSQL analysis. *International Journal of Modern Communication Technologies and Research, 1*(7), 12-18.

Pokorny, J. (2011). NoSQL databases: A step to database scalability in the web environment. In *Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services*, iiWAS '11, (pp. 278-283). New York, NY, USA, ACM.

Rao, B. R., & Govardhan, A. (2013). Sharded parallel mapreduce in Mongodb for online aggregation. *International Journal of Engineering and Innovative Technology, 3*(4), 119-127.

Seema, B., & Ayush, T. (2014). Performance analysis of NoSQL databases having Hadoop Integration. In *Proceedings of International Conference on Emerging Research in Computing, Information, Communication and Applications (ERCICA-14)*

Shvachko, K., Kuang, H., Radia, S., & Chansler, R. (2010). *The Hadoop districuted file system*. Retrieved from https://cs.uwaterloo.ca/~tozsu/courses/CS742/W13/presentations/QimingHsu.pdf.

Vavilapalli, V. K., & Murthy, A. (2013). *Apache Hadoop YARN: Moving beyond MapReduce and batch processing with Apache Hadoop 2*. Boston, MA: Addison-Wesley.

Bibliography

Adomavicius, G., & Zhang, J. (2012). Stability of recommendation algorithms. *ACM Trans. on Information Systems, 30*(4), 1-31.

Bansal, K., & Chawla, P. (2017). A study of Big Data analysis using Apache Pig. *International Journal for IJCTA,* 8665-8672.

Biliris A. (1992). An efficient database storage structure for large dynamic objects. *IEEE Data Engineering Conference*, (pp. 301-308). Phoenix, Arizona.

Cattell, R. (2011). *Scalable SQL and NoSQL data stores*. Retrieved from http://www.cattell.net/datastores/Datastores.pdf.

Condie, T., Conway, N., Alvaro, P., & Hellerstein, J. M. (2010). *Online aggregation and continuous query support in mapreduce*. SIGMOD'10, Indianapolis, Indiana, USA. Copyright 2010 ACM 978-1-4503-0032- 2/10/06. Retrieved from http://www.neilconway.org/docs/sigmod2010_hop_demo.pdf.

Dhawan, S., & Rathee, S. (2013). Big data analytics using Hadoop components like Pig and Hive. *Americal International Journal of Research in Science, Technology, Engineering & Mathematics,* 88-93.

Eldawy, A., & Mokbel, M. F. (2013). A demonstration of spatialhadoop: An efficient mapreduce framework for spatial data. Proceedings of the *VLDB Endowment, 6*(12). VLDB Endowment 21508097/13/10.

Ghemawat, S., Hsieh, W. C., Wallach, D. A. Burrows, M., Tushar, c., Fikes, A., . . . Dean, J. (2006). *Bigtable: a distributed storage system for structured data*. Retrieved from https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf.

Gupta, V., & Lehal, G. S. (2013). A survey of common stemming techniques and existing stemmers for Indian languages. *Journal of Emerging Technologies in Web Intelligence, 5*(2), 157-161.

Kedar Dixit. (2017). *Workshop on big data using hadoop technology*. Jawaharlal Engineering College, Aurangabad, Maharashtra.

Lee, K-H., & Choi, H. (2011). Parallel data processing with MapReduce: A survey. *SIGMOD Record, 40*(4).

Li, M. J., Ng, M. K., Cheung, Y. M., & Huang, J. Z., (2008). Agglomerative fuzzy k-means clustering algorithm with a selection of the number of clusters. *IEEE Trans. on Knowledge and Data Engineering, 20*(11), 1519-1534.

Lu, C. H. Y., & Swanson, D. (2010). Matchmaking: A new mapreduce scheduling. In *10th IEEE International Conference on Computer and Information Technology* (CIT'10), pp. 2736-2743.

Madden, S. (2012). From databases to big data. *IEE Internet Computing*, *16*(3), 4-6.

Mridul, M., Khajuria, A., Dutta, S., & Kumar, N. (2014). *Analysis of Bigdata using Apache Hadoop and MapReduce, 4*(5), 27.

Niknam, T., Taherian Fard, E., Pourjafarian, N., & Rousta, A. (2011). An efficient algorithm based on modified imperialist competitive algorithm and K-means for data clustering. *Engineering Applications of Artificial Intelligence, 24*(2), 306-317.

Olmsted, J. P. (2014). *Scaling at scale: Ideal point estimation with big-data*. Princeton, NJ: Princeton Institute for Computational Science and Engineering.

Oracle. (2011). *Hadoop and NoSQL technologies and the Oracle database*. Retrieved from https://www.oracle.com/technetwork/database/hadoop-nosql-oracle-twp-398488.pdf.

Palanisamy, B., Singh, A., & Liu, L. (2010). *Cost-effective resource provisioning for MapReduce in a cloud*. Stanford, CT: Gartner.

Pansare, N., Borkar, V., Jermaine, C., & Condie, T. (2011). *Online aggregation for large mapreduce jobs*. Seattle, WA: VLDB Endowment.

Platzer, C, Rosenberg, F., & Dustdar. S. (2009). Web service clustering using multidimensional angles as proximity measures. *ACM Trans. on Internet Technology, 9*(3), 11-26.

Rodriguez, A., Chaovalitwongset, W. A., Zhe L., Singhal, H., & Pham, H. (2010). Master defect record retrieval using network-based feature association. *IEEE Trans. on Systems, Man, and Cybernetics, Part C: Applications and Reviews, 40*(3), 319-329.

Russum, P. (2011). *Big data analytics: TDWI best practices report*. India: The Data Warehousing Institute.

Sandeep, R. S., Vinay, C., & Hemant, S. M. (2013). Strength and accuracy analysis of affix removal stemming algorithms. *International Journal of Computer Science and Information Technologies, 4*(2), 265-269.

StratApps. (n.d.). *Big data: An introduction to Pig*. Retrieved from https://stratapps.net/intro-pig.php.

Thilagavathi, G., Srivaishnavi, D., & Aparna, N. (2013). A survey on efficient hierarchical algorithm used in clustering. International Journal of Engineering, 2(9), 2553-2556.

Ward, J. S., & Barker, A. (2012). *Undefined by data: A survey of big data definitions*. Stanford, CT: Gartner.

Yang, Z., Tu, Q., Fan, K. Zhu, L., Chen, R., & Peng, B. (2008). Performance gain with variable chunk size in GFS-like file systems. *Journal of Computational Information Systems, 4*(3), 1077-1084.

Zielinnski, T., Szydlo, T., Szymacha, R., Kosinski, J., & Kosinska, J. (2012). Adaptive soa solution stack. *IEEE Trans. on Services Computing, 5*(2), 149-163.