REPUBLIC OF ALBANIA
UNIVERSITY OF TIRANA
FACULTY OF NATURAL SCIENCES

Master of Science in Informatics

Project

Big Data & Hadoop

Linux applications

Content

Abstract

We live in a digital space on demand, with the dissemination of data by institutions, individuals and machines of a very high degree of intelligence. This data is defined as "Big Data" because of their apparent volume, variety and speed. Most of these data are unstructured, almost unstructured or semi-structured and are heterogeneous in nature. The volume and heterogeneity of data at the speed at which it is generated makes it difficult for the current computing infrastructure to manage Big Data. Traditional data management, storage and analysis systems do not have the tools to analyze this data. Due to its specific Big Data nature, they are stored in distributed file system architectures. Hadoop and HDFS by Apache are widely used for storing and managing big data. Analyzing big data is a challenging task as it involves large distributed file systems which must be fault tolerant, flexible and scalable. Map Reduce is widely used for efficient big data analysis. Traditional DBMS techniques such as JOIN and INDEXING and other techniques are used to classify and collect big data. These techniques are being adopted for use in Map Reduce. In this task we suggest one of the methods to handle this using HDFS. Map Reduce is a technique that uses indexing of files with marking, sorting, rearranging and finally a scaling. Analyzing big data is a challenging task as it involves large distributed file systems which must be fault tolerant, flexible and scalable. Map Reduce is widely used for efficient big data analysis. Traditional DBMS techniques such as joining (JOIN) and indexing (INDEXING) and other techniques are used to classify and collect big data. These techniques are being adopted for use in Map Reduce. In this task we suggest one of our methods to handle this using HDFS. Map Reduce is a technique that uses indexing of files with marking, sorting, rearranging and finally a scaling. Analyzing big data is a challenging task as it involves large distributed file systems which must be fault tolerant, flexible and scalable. Map Reduce is widely used for efficient analysis of big data. Traditional DBMS techniques such as JOIN and INDEXING and other techniques are used to classify and collect big data. These techniques are being adopted for use in Map Reduce. In this task we suggest one of the methods to handle this using HDFS. Map Reduce is a technique that uses indexing of files with marking, sorting, rearranging and finally a scaling. Map Reduce is widely used for efficient analysis of big data. Traditional DBMS techniques such as JOIN and INDEXING and other techniques are used to classify and collect big data. These techniques are being adopted for use in Map Reduce. In this task we suggest one of the methods to handle this using HDFS. Map Reduce is a technique that uses indexing of files with marking, sorting, rearranging and finally a scaling. Map Reduce is widely used for efficient big data analysis. Traditional DBMS techniques such as joining (JOIN) and indexing (INDEXING) and other techniques are used to classify and collect big data. These techniques are being adopted for use in Map Reduce. In this task we suggest one of the methods to handle this using HDFS. Map Reduce is a technique that uses indexing of files with marking, sorting, rearranging and finally a scaling. In this task we suggest one of our methods to handle this using HDFS. Map Reduce is a technique that uses indexing of files with marking, sorting, rearranging and finally a scaling. In this task we suggest one of our methods to handle this using HDFS. Map Reduce is a technique that uses indexing of files with marking, sorting, rearranging and finally a scaling.

1. Introduction

Big Data & Hadoop: Introduction to Big Data and Hadoop. We will learn about Hadoop and its architecture. How to install hadoop and how to manage it with commands. We will understand the difference between different versions of Hadoop and how hadoop uses ETL.

Hive: We will understand what types of problems Hive solves in Big Data. We will use different data models, get acquainted with the different types of files that Hive has and use data management in Hive via of the query.

Pig: We will understand how Pig solves problems in Big Data. mode of operation, differences between Pig and SQL.

Cases of use: Real-time applications to better understand Hadoop and its components, so we will learn how to build a simple Data Pipeline in Hadoop to process big data.

Practical: In each case we will have an illustration in practice.

2. Big Data

Big Data is a field that deals with ways to analyze, systematically extract information, or deal with datasets that are too large or complex to be handled by the traditional data processing application program. Multi-field data (columns) provide greater statistical power, while data with higher complexity (more attributes or columns) can lead to a higher false detection rate. The major challenges of data analysis include data capture, data storage, data analysis, search, sharing, transfer, visualization, query, updating, information privacy, and data source.

Big Data was initially associated with three main concepts: volume, variety and speed. Their analysis presents challenges in sampling, and thus previously only allowed observations and sampling. Therefore, big data often includes sized data that exceeds the capacity of traditional software to be processed within an acceptable time and value.

Why is Big Data Important?

The importance of big data does not revolve around how much data we have, but what we do with it. We can take data from any source and analyze it to find answers that enable: 1) cost reduction, 2) time reduction, 3) new product development and optimized offerings, 4) intelligent decision making . When we combine big data with high-powered analytics, we can accomplish business-related tasks like:

- Determining the root causes of failures, issues and defects in near real time.
- Generating coupons at the point of sale based on the customer's buying habits.
- Complete recalculation of risk portfolios in minutes.
- Detecting fraudulent behavior before it affects our business.

benefit

A major practical application of Big Data Development has been "fighting data poverty". In 2015, Blumenstock and colleagues estimated poverty and wealth predicted by cell phone metadata and in 2016 Jean and colleagues combined satellite imagery and mechanical learning to predict poverty. Using data as digital footprints to study the labor market and digital economy in Latin America, Hilbert and colleagues argue that digital footprint data has several benefits such as:

Thematic coverage
Includes areas that were previously difficult or impossible to measure.

Geographical coverage
International sources provided substantial and comparable data for almost all countries, including many small countries that are not normally included in international inventories;

Level of detail
Provide as detailed data as possible with many interrelated variables, and new aspects, such as network connections;

Time and deadlines:
Graphs can be produced within days of collection.

Challenges

At the same time, working with digital data as a clue instead of traditional survey data does not eliminate the traditional challenges involved when working in the field of international quantitative analysis. Priorities vary slightly, but the underlying discussions remain the same. Among the main challenges it is important to consider:

Representation
While traditional development statistics are primarily concerned with the representation of random survey samples, digital trace data is never a random sample.

Generalizability
Observation data always very well represent this source, they represent only those elements for which they have data, and nothing more. While it is tempting to generalize from the specific observations of a platform to wider environments.

Harmonization
Digitally tracked data still require international harmonization of indicators. Added to this is the challenge of so-called 'data integration', the harmonization of different sources.

Data overload
Analysts and institutions are not used to effectively handle a large number of variables, which is done efficiently and interactively. Practitioners do not yet have a standard workflow that will allow researchers, users, and policymakers to access data efficiently and effectively.

COVID-19

During the COVID-19 pandemic, big data was gathered as a way to minimize the impact of the disease. Important big data applications included minimizing the spread of the virus, identifying the case, and developing medical treatment. Governments used big data to track infected people and minimize the spread.

3. Hadoop

Apache Hadoop is an open source framework software used to develop data processing applications that run in a distributed computing environment. Applications built using HADOOP run on large data sets distributed across several sets of computers. These computers are cheap and widely available. These are mainly useful for achieving greater low cost computing power. Similar to data that resides in a local file of a personal computer system, in Hadoop, the data resides in a distributed file which is referred to as a Hadoop Distributed System. The processing model is based on the concept of 'Data Locality' where the computational logic is sent to the nodes of the groups (servers) that contain data. This computational logic is nothing but a compiled version of a program written in a high-level language such as Java. Such a program processes the data stored in Hadoop HDFS.

3.1 Hadoop Ecosystem

Apache Hadoop consists of two architectural subsystems:

Hadoop MapReduce

MapReduce is a computing model and part of a software framework for writing applications that run in Hadoop. These MapReduce programs are capable of processing large amounts of data in parallel in large groups of computational nodes.

HDFS (Hadoop Distributed File System)

HDFS takes care of the storage part of Hadoop applications. MapReduce applications consume data from HDFS. HDFS creates multiple copies of data blocks and distributes them to compute nodes in a group. This distribution enables reliable and extremely fast calculations. Although Hadoop is best known for MapReduce and its distributed file system - HDFS, the term is also used for a family of related projects that fall under the umbrella of distributed computing and large-scale data processing. Other Hadoop-related projects in Apache include Hive, HBase, Mahout, Sqoop, Flume and ZooKeeper.

3.2 Hadoop Architecture

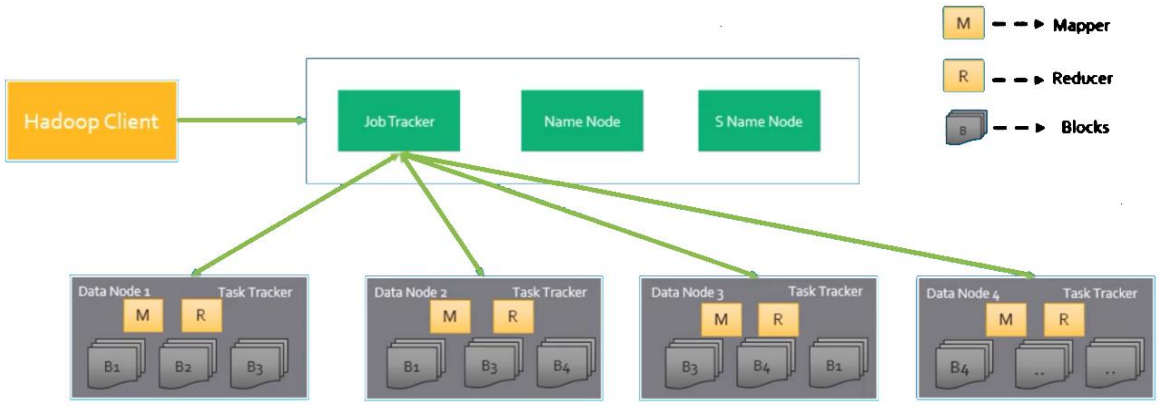
Hadoop has a Master-Slave architecture for data storage and distributed data processing using MapReduce and HDFS methods.

NameNode -represents each file and directory used in the namespace.

DataNode -helps us manage the state of an HDFS node and allows us to interact with blocks.

MasterNode - allows us to perform parallel data processing using Hadoop MapReduce.

SlaveNode- are additional machines in the Hadoop group which allow us to store data to perform complex calculations. Moreover, all of SlaveNode comes with Task Tracker and a DataNode. This allows us to synchronize processes with NameNode and Job Tracker respectively. In Hadoop, the master or slave system can be placed in the cloud or on the premise.



3.3 Hadoop 1.x vs Hadoop 2.x

Hadoop Components V.1.x

Apache Hadoop V.1.x has the following two main components:

1. HDFS (HDFS V1)
2. MapReduce (MR V1)

In Hadoop V.1.x, these two are also known as the two pillars of Hadoop.

Components of Hadoop V.2.x

Apache Hadoop V.2.x has the following three main components:

1. HDFS V.2
2. Fije (MR V2)
3. MapReduce (MR V1)

In Hadoop V.2.x, these two are also known as the three pillars of Hadoop.

Hadoop 1.x and 2.x:

If we look at the components of Hadoop 1.x and 2.x, the Hadoop 2.x Architecture has an additional and new component that is:

YARN (Yet Another Resource Negotiator).

- Hadoop 1.x only supports one namespace for managing the HDFS file system while Hadoop 2.x supports multiple namespaces.
- Hadoop 1.x supports one and only one programming model: MapReduce. Hadoop 2.x supports multiple programming models with the YARN Component such as MapReduce, Interactive, Streaming, Graphic, Storm, etc.
- Hadoop 1.x has many limitations in scalability. Hadoop 2.xe has overcome that limitation with the new architecture.
- Hadoop 1.x HDFS uses a fixed-sized Slots mechanism for storage, while Hadoop 2.x uses a variable-sized Container.
- Hadoop 1.x supports a maximum of 4,000 nodes for info where Hadoop 2.x supports more than 10,000 nodes for info.

3.4 ETL vs ELT

The data explosion has brought about a massive strain on the database architecture. Businesses handle large volumes and different types of data, including sensor, social media, customer behavior, and big data. If the business has a database, the extract, transform and load (ETL) or extract, load, transform (ELT) data integration method will most likely be used. ETL and ELT are two of the most popular methods for collecting data from multiple sources and storing them in a database that can be accessed by all users in an organization. ETL is the traditional method of data storage and analysis, but with advances in technology, ELT has now emerged.

What is the difference between ETL and ELT?

- In ETL, data is extracted from various sources such as ERP and CRM systems, transformed (calculations are applied, raw data is changed to the required format / type, etc.), and then uploaded to the database.
- In ELT, after extraction, the data is first uploaded to the target database and then transformed. Data transformation takes place within the database.
- That said, the difference between these two processes is not limited to the order in which the data is integrated.

To understand their differences, we must also consider:

- Basic storage technologies
- Design approach to data warehouse architecture
- Cases of business use for the database

Hadoop and advanced data integration tools enable ELT

Tools such as Apache Hadoop have renewed the interest of businesses in ELT. Previously, large data sets were split into smaller ones, processed and transformed remotely, and then sent to data warehouses. With Hadoop integration, large data sets that once circulated around the clouds and were now processed can be transformed into the same place, i.e., within Hadoop.

ELT is a good option if we are moving to a data storage facility to support big data initiatives using Hadoop or a NoSQL analytical DBMS.

The ETL process feeds traditional repositories directly, while in ELT, data transformations occur in Hadoop, which then feeds the data repositories. Therefore, poor quality data or data requiring essential integration should not be uploaded to Hadoop, unless we have a team of highly skilled programmers capable of writing custom code for complex data transformations.

3.5 HDFS management from Command Line

Manage HDFS through Shell HDFS Commands

We can use HDFS in different ways:

From the command line using simple file system commands like Linux, as well as through an Internet interface, called WebHDFS. Use the HttpFS port to access HDFS in a firewall through the Hue file browser (and Cloudera Manager and Warehouse, if you are using Cloudera, or Hortonwork Hadoop distributions).

It is important to know that HDFS file systems are just one way for Hadoop to implement a file system. There are several other Java file implementations that work with Hadoop. These include local file systems (file), WebHDFS (WebHDFS), HAR (Hadoop archive files), View, S3 (s3a) and others. For each file system, Hadoop uses a different URI schema for the file system instance in order to connect to it.

We can use two types of HDFS shell commands:

1. The first set of shell commands is very similar to common Linux file system commands like ls, mkdir etc.
2. The second set of HDFS shell commands are specific to HDFS, such as commands that allow us to set the file replication factor.

Some commands:

```
$ hdfs dfs –ls file: ///

Displays a list of files stored on the local Linux file system.
```

```
hdfs dfs [GENERIC_OPTIONS] [COMMAND_OPTIONS]

Using the hdfs dfs program, we can execute the file system commands on the
Hadoop-based file system, which happens to be HDFS.
```

List of HDFS Files and Directories:

As with regular Linux file systems, the ls command is used to sort HDFS files.

```
$ hdfs dfs -usage ls
```

Listing of two files and directories:

If the ls command is used for a single file, it displays the statistics for the file, and if it is a directory, it lists the contents of that directory. We have such examples:

```
$ hdfs dfs –ls /

drwxr-xr-x - hdfs hdfs 0 2013-12-11 09:09 / data

drwxr-xr-x - hdfs supergroup 0 2015-05-04 13:22 / lost + found

drwxrwxrwt - hdfs hdfs 0 2015-05-20 07:49 / tmp

drwxr-xr-x - hdfs supergroup 0 2015-05-07 14:38 / user
```

Change groups:

We can only change one user group with the chgrp command:

```
$ sudo –u hdfs hdfs dfs –chgrp marketing /users/sales/markets.txt
```

Changing HDFS file permissions:

We can use the chmod command to change the permissions of a file or directory.

```
hdfs dfs –chmod [-R] <mode> <file / dir>
```

In this section we are using HDFS commands from the command line to view and manipulate HDFS files and directories. There is an even easier way to access HDFS, and that is through Hue, the web-based interface, which is easy to use and allows us to perform HDFS operations through a GUI. Hue comes with an application that lets you sort and create files and directories, download and upload files from HDFS, and copy / move files.

4. Hive

4.1 Introduction to Hive

Hive is a database infrastructure tool for processing structured data in Hadoop. It is one of the key components of Hadoop for summarizing Big Data and making them easy to analyze.

Hive was originally developed by Facebook, later Apache Software took it and further developed it as an open source called Apache Hive. Hive is used by various companies. For example, Amazon uses it in Amazon Elastic MapReduce.

Hive is not:

1. A relational database
2. A Model for OnLine Transaction Processing (OLTP)
3. A language for real-time queries and line-level updates

4.2 Hive Architecture

Hive Clients:

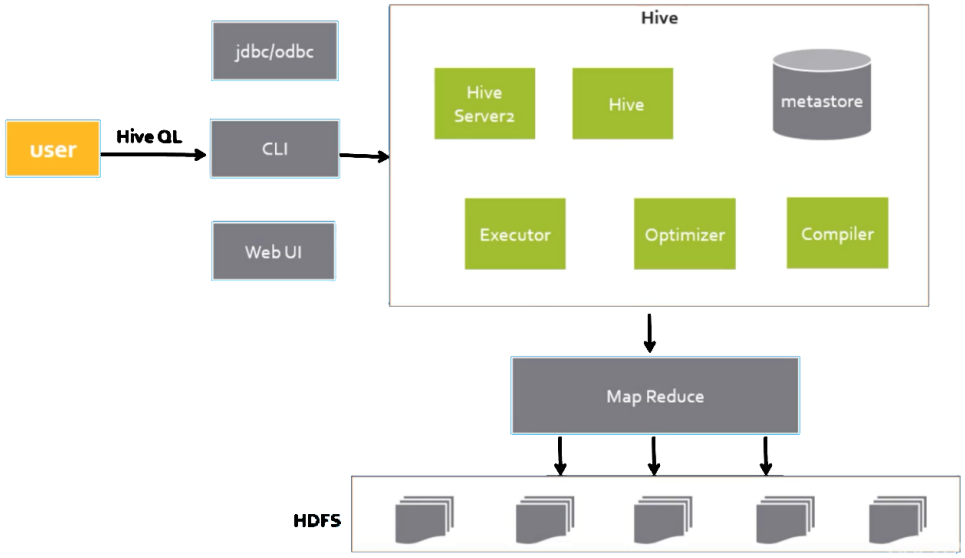
Hive allows writing applications in a variety of languages, including Java, Python, and C ++. It supports different types of clients such as:

- Thrift Server - is a cross-language platform of service providers that serves the application of all those programming languages that support Thrift.
- JDBC Driver - used to establish a connection between Hive and Java applications. The JDBC Director is present in the class org.apache.hadoop.hive.jdbc.HiveDriver.
- ODBC Driver - Allows applications that support the ODBC protocol to connect to Hive.

Hive Services:

- Hive CLI - Hive CLI (Command Line Interface) is a dimension where we can execute queries and Hive commands.
- Web User Interface - Hive Web UI is just an alternative to Hive CLI. It provides an internet-based GUI for executing Hive queries and commands.
- Hive MetaStore - is a central database that stores all the information of the table structure and various partitions in the database. It also includes the column metadata and its type information, the serializer and deserializer used to read and write data, and the corresponding HDFS files where the data is stored.
- Hive Server is referred to as the Apache Thrift Server. It accepts the request from various clients and provides it to the Hive Driver.
- Driver hive - Receive queries or requests from various sources such as web UI, CLI, Thrift and JDBC / ODBC Director. Transfers information to the compiler.

- Hive Compiler - The purpose of the compiler is to analyze information and perform semantic analysis on various blocks and expressions of information. It converts HiveQL statements into MapReduce jobs.
- Hive Execution Engine - The optimizer generates the logical plan in the form of DAG of tasks for reducing mapping and HDFS tasks. Finally, the executable engine executes the input tasks in the order of their dependency.



4.3 Hive data model

Hive data models

Hive data models contain the following elements:

- Database
- TABLES
- Divisions
- Buckets or groups

Table:

Apache Hive tables are the same as the tables present in a relational database. The table in Hive logically consists of the stored data and the associated metadata describe the presentation of the data in the table. We can perform filtering, projection, merging operations on Hive. In Hadoop the data usually resides in HDFS, although it can be in any Hadoop file system, including the local file system or S3, but Hive stores the metadata in a relational database rather than in HDFS.

Hive has two types of tables which are as follows:

1. Managed table
2. External table

Divisions:

Partitioning means dividing a table into granular parts based on the value of a partition column such as 'data'. This makes it faster to make requests for data. So what is the Division function? Partition keys determine how data is stored. Here, each unique value of the partition key defines a table partition. The dividers are named according to dates for convenience.

Bucket:

Bucket gives additional data structure that can be used for requests and information as efficiently as possible. A merge of two tables that are linked in the same column, including the merge column can be implemented as a Mapping Join. Bucketing by the ID used means that we can quickly evaluate a user-based request by executing it on a random record of the total group of users.

4.4 SQL vs HQL

Structured language, popularly known as SQL, is a database language that uses the concept of relational database management to manage data. Data management includes selecting (pulling data from a single or multiple database), adding (adding one or more rows to a table), updating (responsible for changing the value of one or more rows in a table) , delete (responsible for deleting one or more rows in a table) and creating the schema through queries.

HQL, or Hibernate Query Language, extends the concept of object-oriented programming to existing SQL. It is easy to learn and similar in syntax to SQL. It has features like general functions and clustering that we often see in SQL.

Differences between SQL and HQL:

- SQL is based on a relational database model while HQL is a combination of object-oriented programming with relational database concepts.
- SQL manipulates the data stored in tables and modifies its rows and columns. HQL is about its objects and properties.
- SQL is about the relationship that exists between two tables while HQL considers the relationship between two objects.

4.5 Practice in Hive

```
--Hive Practice
create external table employee (
      id string,
      fname string,
      lname string,
      department_id string
 )
 ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' location '/ user /
sample_data / employee /';
-
create external table department (
      id string,
      dept_name string
 )
 ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' location '/ user /
sample_data / department /';

--External Table in Hive
create external table salary (
      salary_id string,
      employee_id string,
      payment double,
      date string
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' location '/ user /
sample_data / salary /';

--Internal Table
create table salary_internal as
select salary_id, employee_id, payment, date from salary;

--File Formats in Hive:
--Text File:
```

```
create table salary_text (
      salary_id string,
      employee_id string,
      payment double,
      date string
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' stored as textfile;
INSERT OVERWRITE TABLE salary_text SELECT * FROM salary;

--RCFile:
create table salary_rcfile (
      salary_id string,
      employee_id string,
      payment double,
      date string
) stored as rcfile;
SET hive.exec.compress.output = true;
SET mapred.output.compression.type = BLOCK;
SET mapred.output.compression.codec = org.apache.hadoop.io.compress.SnappyCodec;
INSERT OVERWRITE TABLE salary_rcfile SELECT * FROM salary;

--Parquet:
create table salary_parquetfile (
      salary_id string,
      employee_id string,
      payment double,
      date string
) stored as parquetfile;
INSERT OVERWRITE TABLE salary_parquetfile SELECT * FROM salary;

--ORC:
create table salary_orc (
      salary_id string,
      employee_id string,
      payment double,
      date string) stored as orc;
```

```
INSERT OVERWRITE TABLE salary_orc SELECT * FROM salary;

--finds the number of employees per department with a salary higher than 1000
in August 2008.
select d.dept_name, count (e.id) as employee_count
from department d
join employee e on d.id = e.department_id
join salary s on s.employee_id = e.id
where s.payment> 1000 and month (s.date) = 8 and year (s.date) = 2008
group by d.dept_name;

- from commandline
hive -e 'select * from hive_demo.salary limit 2'
hive -S -e 'select * from hive_demo.salary_orc limit 5'
```

5. Pig

5.1 Introduction to Pig

Pig Hadoop Manuals - Objectives

When it comes to analyzing large data sets as well as representing them as data streams, we use the Apache Pig. It is nothing but an abstraction on MapReduce.

What is Hadoop Pig?

Hadoop Pig is nothing but an abstraction on MapReduce. When it comes to analyzing large data sets as well as representing them as data streams, we use the Apache Pig. Generally, we use it with Hadoop. Using Pig, we can perform all data manipulation operations in Hadoop.

In addition, Pig offers a high-level language for writing data analysis programs which we call Latin Pig. One of the main advantages of this language is, it offers several operators. Through them, programmers can develop their functions for reading, writing and processing data.

It has key features as follows:

Ease of programming

Basically, when all the complex tasks consisting of multiple interconnected data transformations are clearly encoded as data flow sequences, this makes them easy to write, understand, and maintain.

Optimization opportunities

This allows users to focus on semantics rather than efficiency, to optimize their execution automatically, in which tasks are encoded and those allowed by the system.

Map Reduce

In order to do special-purpose editing, users can create their own functions. Therefore, programmers need to write scripts using the Pig Latin language to analyze data using the Apache Pig. However, all of these scripts have been converted internally to Map Reduce tasks.

5.2 Pig Architecture

Pig Architecture:

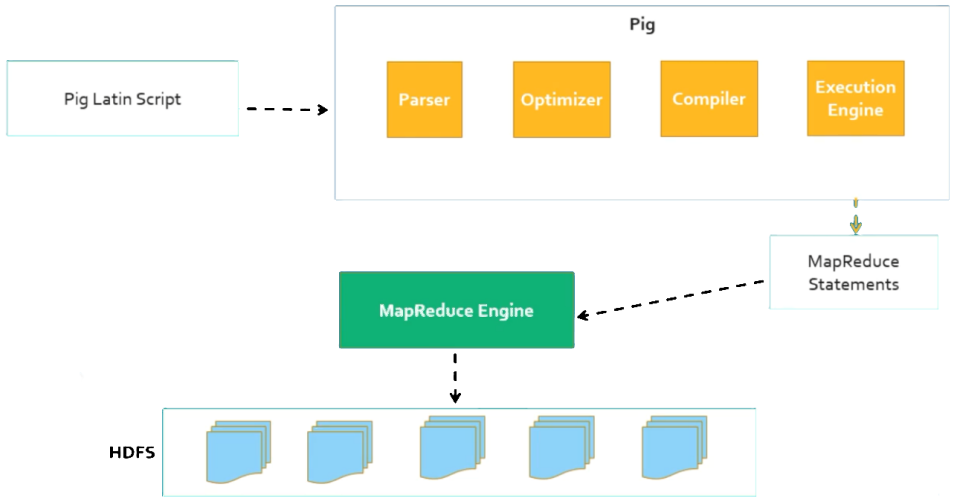
The Pig architecture consists of two components:

1. Pig Latin, which is a language
2. A timely environment for executing PigLatin programs.

A Pig Latin program consists of a series of operations or transformations which are applied to input data to produce results. These operations describe a data stream which translates into an executable view, from the Hadoop Pig executable environment. Below the results of these transformations are a series of MapReduce jobs that a programmer is unaware of. So in a way, Pig in Hadoop allows the programmer to focus on data rather than on the nature of the execution. PigLatin is a relatively rigid language which uses keywords known from data processing eg, Join, Group and Filter.

Execution methods:
Pig in Hadoop has two execution modes:

1. Local mode: In this mode, the Hadoop Pig language runs in a single JVM and uses the local file system. This method is only suitable for analyzing small data sets using Pig in Hadoop
2. Map Reduce Mode: In this way, queries written in Pig Latin are translated into MapReduce jobs and executed in a Hadoop group (the group can be pseudo or fully distributed). MapReduce mode with fully distributed group is useful for executing Pig in large data sets.



5.3 SQL vs Pig

- DBMS systems running SQL are considered to be faster than MapReduce (operated by Pig through the PigLatin platform). However, loading data is more challenging in the case of RDBMS, making it difficult to configure.
- PigLatin offers a number of advantages in terms of executing execution plans, ETL routines, and transmission channel modification.
- SQL is declarative and PigLatin is procedural to a large extent.
- What we mean by this is in SQL, we mainly specify "what" will be accomplished and in Pig, we mention "how" a task should be performed.
- A script written in Pig essentially turns into a MapReduce job before it is executed.
- A Pig scenario is shorter than the corresponding MapReduce job, which significantly reduces development time.

5.4 Practice in Pig

```
--Practical Pig
--load with schema

e = LOAD '/user/sample_data/employee/employee.csv' USING PigStorage (',')
as (eid: chararray, fname: chararray, lname: chararray, department:
chararray);

e_sample = limit e 10;
dump e_sample;
```

```
--load without schema
e1 = LOAD '/user/sample_data/employee/employee.csv' USING PigStorage (',');
e1_sample = limit 10 e1;
dump e1;

- a simple query query1
d = LOAD '/user/sample_data/salary/salary.csv' USING PigStorage (',') as
(salary_id: chararray, employ_id: chararray, payment: double, p_date:
datetime);
d_sample = limit d 10;
dump d_sample;

--a simple query query2
g = LOAD '/user/sample_data/employee/employee.csv' USING PigStorage (',') as
(eid: chararray, fname: chararray, lname: chararray, department: chararray);
f = group g by department;
f_sample = limit f 2;
dump f_sample;


- JOIN in Pig
emp = LOAD '/user/sample_data/employee/employee.csv' USING PigStorage (',')
as (emp_id: chararray, fname: chararray, lname: chararray, dept_id:
chararray);

dep = LOAD '/user/sample_data/department/department.csv' USING PigStorage
(',') as (dept_id: chararray, dept_name: chararray);

dep_emp_join = JOIN emp by (dept_id), dep by (dept_id);
grouped = group dep_emp_join by dept_name;

group_by_count = foreach grouped {
 uniq_emp = DISTINCT dep_emp_join.emp_id;
 generate group, COUNT (uniq_emp) as emp_count;
 };
```

```
dump group_by_count;

--store pig results
emp = LOAD '/user/sample_data/employee/employee.csv' USING PigStorage (',')
as (emp_id: chararray, fname: chararray, lname: chararray, dept_id:
chararray);

dep = LOAD '/user/sample_data/department/department.csv' USING PigStorage
(',') as (dept_id: chararray, dept_name: chararray);

dep_emp_join = JOIN emp by (dept_id), dep by (dept_id);
grouped = group dep_emp_join by dept_name;

group_by_count = foreach grouped {
 uniq_emp = DISTINCT dep_emp_join.emp_id;
 generate group, COUNT (uniq_emp) as emp_count;
 };
store group_by_count into '/ user / sample_data / pig_output';
```

6. Hive & Pig (Case Use)

Task: Design a simple Data Pipeline using the knowledge gained in Hive & Pig

A Data Pipeline is a series of data processing steps. If the data is not currently loaded on the data platform, then they are first retrieved from the Data Pipeline. Then there are a series of steps in which each step gives an exit which is the entrance to the next step. This continues until the Data Pipeline is completed. In some cases, the independent steps can be run in parallel.

The Data Pipeline consists of three main elements: a source, a processing step or steps, and a destination. Data Pipelines enable the flow of data from an application to a database, from a Data Lake to an analytical database, or to a payment processing system, for example.

```
--Hive Script
--create external table (for data parsed by pig)
create external table parsed_logs (
```

```
      logdate    string,
      URIs       string,
      ip         string,
      city       string,
      state      string,
      country    string,
      swid       string
 )
 ROW FORMAT DELIMITED FIELDS TERMINATED BY '\ t' location '/ user /
sample_data / pig_out_parsed_logs';

--Pig scripts
--parsing logs

logs = LOAD '/user/sample_data/click_stream_logs/Omniture.0.tsv' USING
PigStorage ('\ t')
neither
(col_0: chararray, col_1: chararray, col_2: chararray, col_3: chararray,
...
 .. ... ... .. col_176: chararray, col_177: chararray, col_178: chararray);

logs1 = FOREACH logs GENERATE col_1 AS logdate,
col_7 AS ip, col_12 AS url, col_13 AS swid, col_49 AS city, col_50 AS
country, col_52 AS state;

logs2 = FOREACH logs1 GENERATE logdate, url, ip, city, UPPER (state) AS
state, UPPER (country) AS country, swid;

store logs2 into '/ user / sample_data / pig_out_parsed_logs';
--load products
products = LOAD '/user/sample_data/click_stream_logs/products.tsv'
USING PigStorage ('\ t') AS (url: chararray, category: chararray);

store products into '/ user / sample_data / pig_out_products';

--load users
```

```
users = LOAD '/user/sample_data/click_stream_logs/users.tsv'
USING PigStorage ('\ t') AS (SWID: chararray, BIRTH_DT: chararray,
GENDER_CD: chararray);
store users into '/ user / sample_data / pig_out_users';
```

7. Practice – Big Data Hadoop

7.1 Analyzing NYC taxi travel data

Analyzing NYC Taxi Trips Data

Data Source: This is a public data set provided by NYC Taxi. The yellow and green taxi travel records include fields that capture pick-up and drop-off dates / times, pick-up and drop-off locations, travel distances, specified fares, fare types, payment types and number of passengers reported by driver.

Url: https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page

We will analyze the Yellow Taxi data.
Below is an example of travel records:

```
VendorID,                   2,
tpep_pickup_datesime,       2016-01-0100:       00:       00,
tpep_dropoff_datetime,      2016-01-0100:       00:       00,
passenger_count,            2,
trip_distance,              1.10,
pickup_longitude,           -73.990371704101563,
pickup_latitude,            40.734695434570313,
RatecodeID,                 1,
store_and_fwd_flag,         N
dropoff_longitude,          73.981842041015625,
dropoff_latitude,           -40.732406616210937,
payment_type,               2,
```

```
fare_amount,                    7.5,
extra,                          0.5,
mta_tax,                        0.5,
tip_amount,                     0,
tolls_amount,                   0,
improvement_surcharge,          0.3,
total_amount                    8.8,
```

There are more than 100 million + records each month, and, on average, the approximate file size of a single month is 1.6+ GB. We downloaded the January 2016 data.

objective: We will design a Data Wareouse schema of data in Hive. The DWH scheme designed will be based on the Star Schema where we can move or clear data based on business date and time dimensions.
The following scheme is designed in MySQL, and we need to design the same model in Hive.

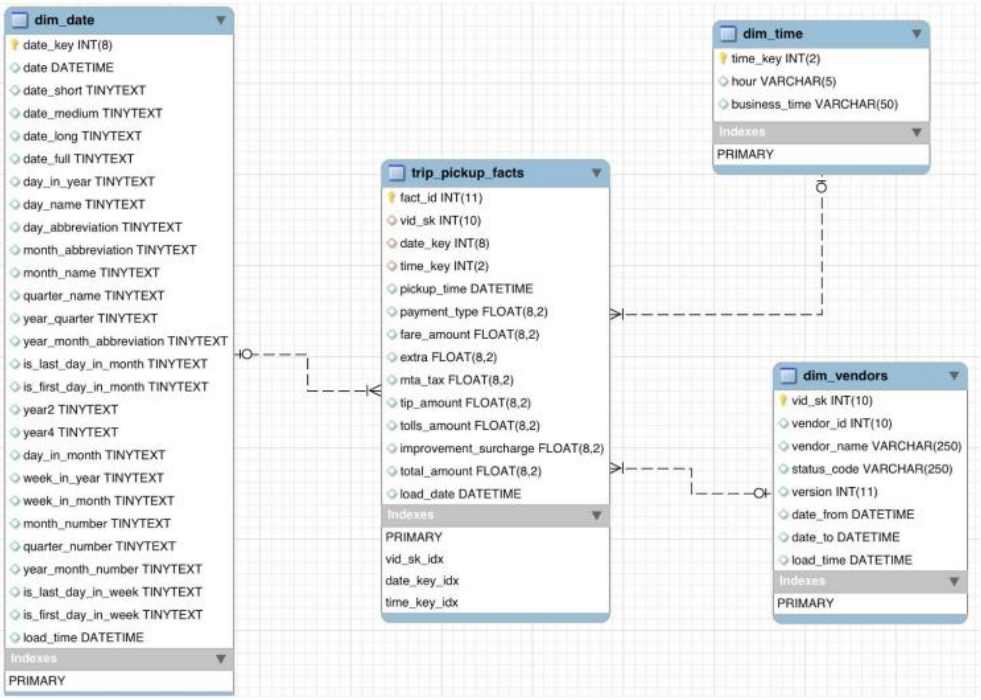


chart trip_pickup_facts: This is a table of transactional facts where each record represents a business event (acquisition event). It has three dimensional tables which are: date dimension, time dimension and vendor dimension. The time dimension keys hold the clock numbers ie. 0,1,2, .., 23. To give the aggregation flexibility at minute level, we have (pickup_time) in the fact chart.

The dimension of time: Below is how the business time is determined. This is a table that is generated only once. Whenever we enter data into the fact table, we will extract the hour numbers from the pick_time of the time as a time_key.

time_key	hour_name	Business time
0	12am	Late Night
1	1am	Late Night
2	2am	Late Night
3	3am	Early Morning
4	4am	Early Morning
5	5am	Early Morning
6	6am	Early Morning
7	7am	AM Peak
8	8am	AM Peak
9	9am	AM Peak
10	10am	Mid Morning
11	11am	Mid Morning
12	12pm	Lunch
13	1pm	Lunch
14	2pm	Mid Afternoon
15	3pm	Mid Afternoon
16	4pm	Mid Afternoon
17	5pm	Evening
18	6pm	Evening
19	7pm	PM Peak
20	8pm	PM Peak
21	9pm	PM Peak
22	10pm	Night
23	11pm	Night

Date dimension: The date dimension corresponds to many derived date attributes (as shown in the diagram above). We have a generated table of a date search (date_lookup) that has 10 years of future search dates (this can be generated according to the business need).
New date attributes will be added during each ETL process (if date attributes do not exist in the dimension table). The ETL process will search the date_lookup table to retrieve all the data_key data, date attributes which will be added to the data_dimension table.

Step 1:
We create a directory in HDFS, hadoop fs -mkdir / user / taxi_trips, and upload the data to this directory.

Step 2: We create an external spreadsheet to read the data.

```
create external table trips_raw (
      VendorID                    int,
      tpep_pickup_datesime        timestamp,
      tpep_dropoff_datetime       timestamp,
      passenger_count             int,
      trip_distance               double,
      pickup_longitude            string,
      pickup_latitude             string,
      RatecodeID                  string,
      store_and_fwd_flag          string,
      dropoff_longitude           string,
      dropoff_latitude            string,
      payment_type                string,
      fare_amount                 double,
      extra                       double,
      mta_tax                     double,
      tip_amount                  double,
      tolls_amount                double,
      improvement_surcharge       double,
      total_amount                double
);
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' location '/ user /
taxi_trips /';
```

Step 3:

We create a database (dwh).

```
create database dwh;
```

Step 4: We create the time dimension table and fill it with values.

```
create table dim_time (
      time_key int,
      hour string,
      business_timing string
);
```

```
        // filling with values…
    insert into table dim_time values
        (0, '12am', 'Late Night'), (1, '1am', 'Late Night'), (2, '2am',
        'Late Night'), (3, '3am', 'Early Morning' ), (4, '4am', 'Early
        Morning'), (5, '5am', 'Early Morning'), (6, '6am', 'Early
        Morning'), (7, '7am', 'AM Peak '), (8,' 8am ',' AM Peak '), (9,'
        9am ',' AM Peak '), (10,' 10am ',' Mid Morning '), (11,' 11am ',
        'Mid Morning'), (12, '12pm', 'Lunch'), (13, '1pm', 'Lunch'),
        (14, '2pm', 'Mid Afternoon'), (15, '3pm', 'Mid Afternoon'), (16,
        '4pm', 'Mid Afternoon'), (17, '5pm', 'Evening'), (18, '6pm',
        'Evening'), (19, '7pm', 'PM Peak'), (20, '8pm', 'PM Peak'), (21,
        '9pm', 'PM Peak '), (22,' 10pm ',' Night '), (23,' 11pm ','
        Night ');
```

Step 5: We create the date dimension.

```
    create external table dim_date (
        date_key                int,
        year_month_number           string,
        quarter_number          string,
        month_number            string,
        year_number                 string,
        year_month_abbreviation     string,
        year_quarter                string,
        quarter_name            string,
        month_name              string,
        day_name                string
    )
    ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' location '/ user /
sample_data / dim_date /';
```

Step 6: We create the trip_pickup_facts fact and set the values in it.

```
    create table trip_pickup_facts (
        vendor_id int,
        date_key int,
```

```
        time_key int,
        pickup_time timestamp,
        payment_type string,
        fare_amount decimal,
        extra decimal,
        mta_tax decimal,
        tip_amount decimal,
        tolls_amount decimal,
        improvement_surcharge decimal,
        total_amount decimal
) stored as orc;
// fill the fact table with data
insert overwrite table trip_pickup_facts
        select
        vendorid,
        from_unixtime (unix_timestamp (tpep_pickup_datetime),
        'yyyyMMdd') as date_key,
        hour (tpep_pickup_datetime) as time_key,
        tpep_pickup_datesime,
        payment_type,
        fare_amount,
        extra,
        mta_tax,
        tip_amount,
        tolls_amount,
        improvement_surcharge,
        total_amount
        from trips_raw;
```

Step 7: Create a test query.

```
select dd.day_name, dt.business_timing, tpf.vendor_id, count
(tpf.vendor_id) as trips_count, sum (tpf.total_amount) as
revenue_earned
from trip_pickup_facts tpf
join dim_date dd on dd.date_key = tpf.date_key
```

```
join dim_time dt on dt.time_key = tpf.time_key
group by dd.day_name, dt.business_timing, tpf.vendor_id;
```

7.2 UDF Design in Hive

Designing Hive UDF

As we know Functions Defined for Hive Users, known as UDF, allow us to create custom functions to process records or sets of recordings. For example, a UDF can perform calculations using an external math library, combine several column values into one, do geospatial calculations, or other types of tests and transformations that are outside the scope of operators and integrated functions. SQL.

In the data for Taxi Trips, we saw that there was information on latitude and longitude for each taxi ride. One of the questions that the business is interested in could be: What if we were to do an analysis of pick-up and drop-off trends in Midtown?

There can be many ways to solve this problem; by reading some Map APIs to generate area names by latitude and longitude or by generating Geohash of given latitude and longitude and then solving problems further.

In this practical exercise we will write a UDF in Hive to generate Geohash, and using that UDF we will solve this problem. First of all, let's understand what Geohash is.

What is Geohash?

A Geohash is a convenient way to express a location, anywhere in the world, using a shorter alphanumeric string, with greater accuracy obtained with longer strings. It returns the given width and length in a string layout.

Example: Geohash (lat, long) Geohash (-73.9832763672,40.7138175964) -> hfugn7

A Geohash can be represented at different levels. On the examples and data we have in this task we will use level 6 Geohash (six character string).

Benefits of Geohash:

- Can be used as a unique Identifier
- Can be used to store point data in the database
- The search point can be faster due to storing it as a string and creating index in it.
- It can be a good use case in Geographic Data Analysis.

Below is the code in Python that generates a Geohash:

```
from math import log10
```

```
import sys
import sys
__base32 = '0123456789bcdefghjkmnpqrstuvwxyz'
__decodemap = c de
for i in range (len (__ base32)):
 __decodemap [__ base32 [i]] = i
of i
def encode (latitude, longitude, precision = 6):
 "" "
 Encode a position given in float arguments latitude, longitude to
 a geohash which will have the character count precision.
 "" "
 lat_interval, lon_interval = (-90.0, 90.0), (-180.0, 180.0)
 geohash = []
 bits = [16, 8, 4, 2, 1]
 bit = 0
 ch = 0
 even = True
 while len (geohash) <precision:
 if even:
 mid = (lon_interval [0] + lon_interval [1]) / 2
 if longitude> mid:
 ch | = bits [bit]
 lon_interval = (mid, lon_interval [1])
 else:
 lon_interval = (lon_interval [0], mid)
 else:
 mid = (lat_interval [0] + lat_interval [1]) / 2
 if latitude> mid:
 ch | = bits [bit]
 lat_interval = (mid, lat_interval [1])
 else:
 lat_interval = (lat_interval [0], mid)
 even = not even
 if bit <4:
 bit + = 1
 else:
 geohash + = __base32 [ch]
 bit = 0
 ch = 0
 return '' .join (geohash)
for line in sys.stdin:
 tpep_pickup_datetime, tpep_dropoff_datetime, passenger_count,
trip_distance, total_amount, lat, lon, code = line.strip (). split ('\ t')
 lat = float (lat)
 lon = float (lon)
 code = encode (lat, lon)
```

```
print (','. join ([str (tpep_pickup_datetime), str (tpep_dropoff_datetime),
str (passenger_count), str (trip_distance), str (total_amount), str (lat),
str (lon), str (code)]) )
```

Step 1: Start VM and log in to HDP Sandbox.

ssh root @ localhost -p2222

Step 2: Download Taxi Trips raw file.

wget https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2015-12.csv

Step 3: We run the following commands from the terminal.

sudo su - hdfs
hdfs dfs -mkdir / user / root
hdfs dfs -mkdir / user / nyc_taxi
hdfs dfs -chown root: hdfs / user / root
hdfs dfs -chown root: hdfs / user / nyc_taxi
exit
*hadoop fs -put yellow_tripdata _ *. csv / user / nyc_taxi /*

Step 4: We create the table to read the data.

sudo hive

hive>
CREATE EXTERNAL TABLE yellow_trips (
vendorid string,
tpep_pickup_datetime string,
tpep_dropoff_datetime string,
passenger_count string,
trip_distance string,
pickup_longitude double,
pickup_latitude double,
ratecodeid string,
store_and_fwd_flag string,
dropoff_longitude double,
dropoff_latitude double,
payment_type string,
fare_amount string,
extra string,
mta_tax string,
tip_amount string,
tolls_amount string,
improvement_surcharge string,
total_amount string)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','

```
STORED AS TEXTFILE
location '/ user / nyc_taxi /'
tblproperties ("skip.header.line.count" = "1");
```

Step 5: Copy the python file.

```
scp -P 2222 ~ / Downloads / geohash_udf.py root @ localhost : / root /
```

Step 6: Login to the Hive shell.

```
sudo hive
hive> add file /root/geohash_udf.py;
hive> set hive.execution.engine = thesis;
hive>
SELECT
TRANSFORM (tpep_pickup_datesime,
 tpep_dropoff_datesime,
 passenger_count,
 trip_distance,
 total_amount,
 pickup_latitude,
 pickup_longitude,
 code) USING 'python geohash_udf.py' AS tpep_pickup_datetime,
 tpep_dropoff_datesime,
 passenger_count,
 trip_distance,
 total_amount,
 pickup_latitude,
 pickup_longitude,
 code
Frome
 (SELECT tpep_pickup_datesime,
 tpep_dropoff_datesime,
 passenger_count,
 trip_distance,
  total_amount,
 pickup_latitude,
 pickup_longitude,
 'txt' AS code
 FROM yellow_trips limit 10) a;
```

Step 7: We create the table to place the processed data.

```
create table yellow_trips_processed (
   tpep_pickup_datetime string,
   tpep_dropoff_datetime string,
   passenger_count string,
```

```
    trip_distance string,
    total_amount string,
    pickup_latitude string,
    pickup_longitude string,
    geohash string
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;
```

Step 8: We process the data and fill in the table above.

```
    hive> set hive.execution.engine = thesis;
hive> insert overwrite table yellow_trips_processed
    select
    TRANSFORM (
    tpep_pickup_datetime, tpep_dropoff_datetime, passenger_count, trip_distance, total_amount,
    pickup_latitude, pickup_longitude, code)
    using 'python geohash_udf.py' as tpep_pickup_datetime, tpep_dropoff_datetime,
    passenger_count, trip_distance, total_amount, pickup_latitude, pickup_longitude, code
    from (
    select tpep_pickup_datetime, tpep_dropoff_datetime, passenger_count, trip_distance,
    total_amount, pickup_latitude, pickup_longitude, 'txt' as code from yellow_trips
    ) a;
```

Step 9: Midtown data analysis.

Midtown Lat-Long: 40.758896, -73.985130

Midtown Geohash: dr5ru7

8. conclusions

Big Data improves and speeds up data retrieval especially when used by large businesses which receive large amounts of data all the time. Nowadays business development depends a lot on data collection and processing to determine how this business will go. Businesses receive large amounts of data which over time are growing exponentially and Big Data is one of the only ways for businesses to ensure that any data a customer seeks to provide is stored and then processed. Even if it is not the only solution big data offers a faster and more efficient way of securing data and processing them.

Through this project we learned how to use and take advantage of Big Data, explained how to use Hadoop software and two software built on hadoop which are Hive and Pig. We have described how they are built and the architecture of each of them, the different ways in which these software store data to improve and speed up performance, and what benefits they have over other ways of

storing data like sql. Recently to better understand the use and benefits of using software like Hive or Pig we have demonstrated some examples. Using practical cases and solving real life problems for a particular Big Data case study via Hadoop.

Reference

[Angeles, R. (2016). STEADYSERV BEER: IOT-ENABLED PRODUCT MONITORING USING RFID. IADIS International Journal on Computer Science & Information Systems, 11 (2).]

[Chen, H., Chiang, RH, & Storey, VC (2012). Business intelligence and analytics: From big data to big impact. MIS quarterly, 36 (4), 1165-1188.]

[Fredriksson, C. (2015, November). Knowledge management with Big Data Creating new possibilities for organizations. In The XXIVth Nordic Local Government Research Conference (NORKOM).]

[MacGillivray, C., Turner, V., & Lund, D. (2013). Worldwide Internet of Things (IoT) 2013–2020 Forecast: Billions of Things. Trillions of Dollars, Gartnet Market Analysis.]

[Tan, PN, Steinbach, M., & Kumar, V. (2013). Data mining cluster analysis: basic concepts and algorithms. Introduction to data mining.]

[Troester, M. (2012). Big data meets big data analytics: Three key technologies for extracting real-time business value from the big data that threatens to overwhelm traditional computing architectures. SAS Institute. SAS Institute Inc. White Paper.]