Chapter 1

Introduction

Data has been one of the important things from the start of this century. As the world is growing exponentially in every field even the data is growing tremendously. Data generated in the digital world is log data generated from lakhs of websites, data generated from tweets and many other different sources.

1.1 Problem Statement

For any company, customer satisfaction is the most important factor for growth, so company's use customer surveys and track their click records in their website to understand the taste of every customer and to recommend good products to the customer. But processing such amounts of big data is not easy as data generated is in GBs per second. Many different technologies are developed to collect data and process it without crashing the existing system. Most of the technologies used are developed by Apache Foundation and they are Open Source Software which anyone can use. There are also other paid services used by companies for better management of data by using services provided by companies like AWS, Cloudera and IBM.

1.2 Objective of the project

Objective of this project is to process the log data generated from different websites and to keep track of customer experience by monitoring the click stream and finally to sending to a common storage so anyone can access it depending on their requirement without any loss or delay of data.

Chapter 2

Literature Survey

2.1 Apache Kafka

Figure 2.1: Apache Kafka

Apache Kafka is an open-source stream-processing software platform developed by the Apache Software Foundation, written in Scala and Java. The project aims to provide a unified, high-throughput, low-latency platform for handling real-time data feeds. Kafka can connect to external systems (for data import/export) via Kafka Connect and provides Kafka Streams, a Java stream processing library. Kafka uses a binary TCP-based protocol that is optimized for efficiency and relies on a "message set" abstraction that naturally groups messages together to reduce the overhead of the network round trip. This leads to larger network packets, larger sequential disk operations, contiguous memory blocks which allows Kafka to turn a burst stream of random message writes into linear writes.

2.1.1 Kafka Architecture

Kafka stores key-value messages that come from arbitrarily many processes called producers. The data can be partitioned into different "partitions" within different "topics". Within a partition, messages are strictly ordered by their offsets (the position of a message within a partition), and indexed and stored together with a timestamp. Other processes called "consumers" can read messages from partitions. For stream processing, Kafka offers the Streams API that allows writing Java applications that consume data from Kafka and write results back to Kafka. Apache Kafka also works with external stream processing sys-

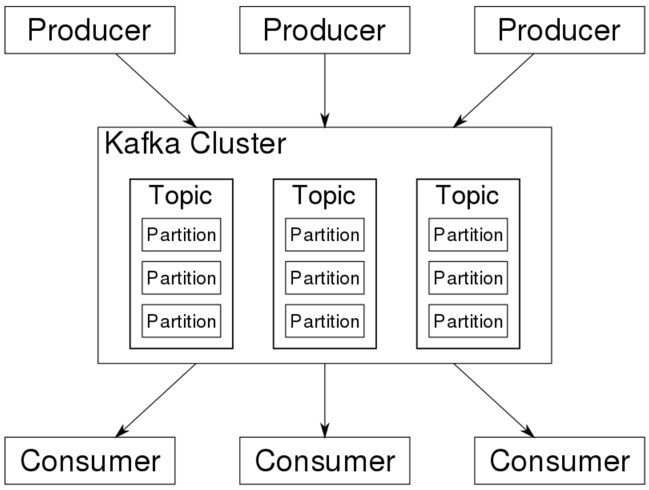


Figure 2.2: Kafka Architecture

tems such as Apache Apex, Apache Flink, Apache Spark, Apache Storm and Apache NiFi.

2.2 Apache Spark

Figure 2.3: Apache Spark

Apache Spark is an open-source distributed general-purpose cluster-computing framework. Spark provides an interface for programming entire clusters with implicit data parallelism and fault tolerance. Apache Spark has its architectural foundation in the Resilient Distributed Dataset (RDD), a read-only multiset of data items distributed over a cluster of machines, that is maintained in a fault-tolerant way.[2] The Dataframe API was released as an abstraction on top of the RDD, followed by the Dataset API.

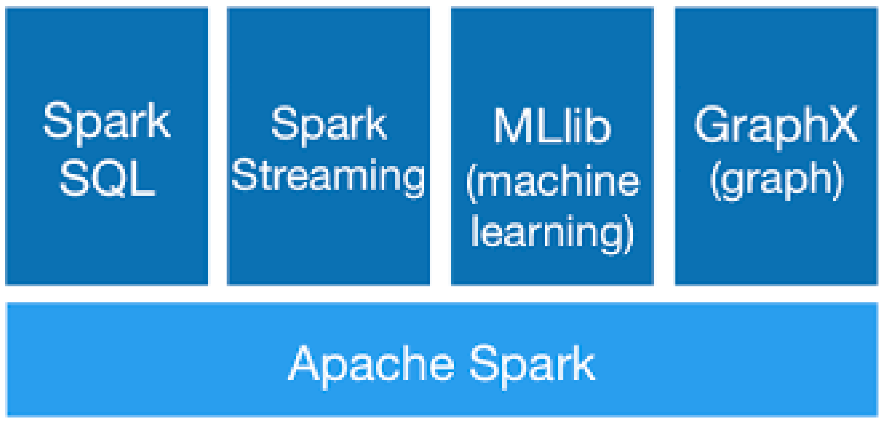


Figure 2.4: Spark API Structure

2.2.1 Spark API

Spark Core

Spark Core is the underlying general execution engine for the Spark platform that all other functionality is built on top of. It provides in-memory computing capabilities to deliver speed, a generalized execution model to support a wide variety of applications, and Java, Scala, and Python APIs for ease of development.

Spark SQL

Many data scientists, analysts, and general business intelligence users rely on interactive SQL queries for exploring data. Spark SQL is a Spark module for structured data processing. It provides a programming abstraction called DataFrames and can also act as distributed SQL query engine. It enables unmodified Hadoop Hive queries to run up to 100x faster on existing deployments and data. It also provides powerful integration with the rest of the Spark ecosystem (e.g., integrating SQL query processing with machine learning).

Spark Streaming

Many applications need the ability to process and analyze not only batch data, but also streams of new data in real-time. Running on top of Spark, Spark Streaming enables powerful interactive and analytical applications across both streaming and historical data, while inheriting Spark's ease of use and fault tolerance characteristics. It readily integrates with a wide variety of popular data sources, including HDFS, Flume, Kafka, and Twitter.

MLib

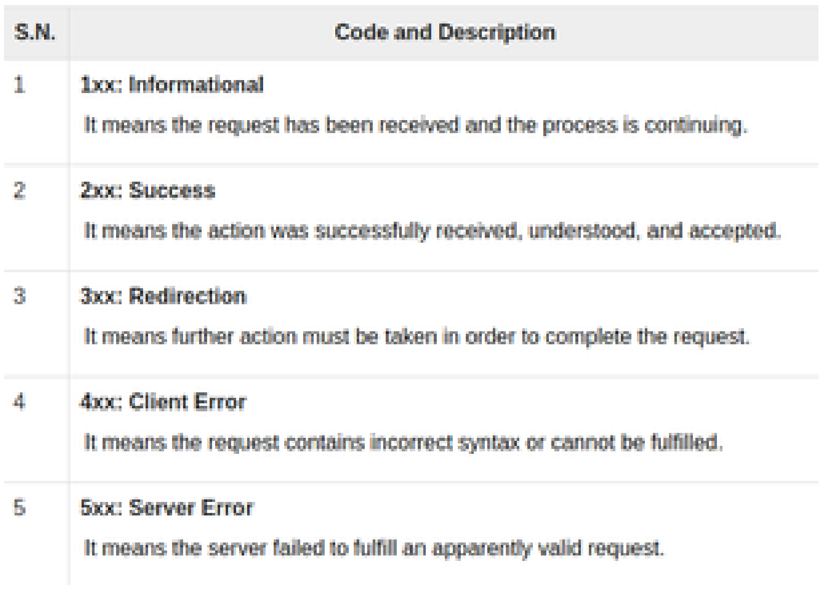
Machine learning has quickly emerged as a critical piece in mining Big Data for actionable insights. Built on top of Spark, MLlib is a scalable machine learning library that delivers both high-quality algorithms (e.g., multiple iterations to increase accuracy) and blazing speed (up to 100x faster than MapReduce). The library is usable in Java, Scala, and Python as part of Spark applications, so that you can include it in complete workflows.

GraphX

GraphX is a graph computation engine built on top of Spark that enables users to interactively build, transform and reason about graph structured data at scale. It comes complete with a library of common algorithms.

2.3 HTTP Response status code

The Status-Code element in a server response, is a 3-digit integer where the first digit of the Status-Code defines the class of response and the last two digits do not have any categorization role. There are 5 values for the first digit:

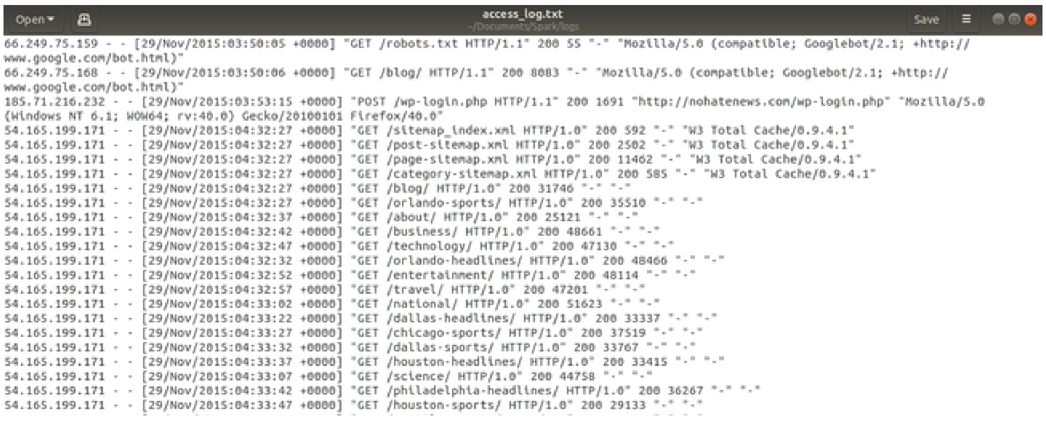


S.N.	Code and Description
1	**1xx: Informational** It means the request has been received and the process is continuing.
2	**2xx: Success** It means the action was successfully received, understood, and accepted.
3	**3xx: Redirection** It means further action must be taken in order to complete the request.
4	**4xx: Client Error** It means the request contains incorrect syntax or cannot be fulfilled.
5	**5xx: Server Error** It means the server failed to fulfill an apparently valid request.

Figure 2.5: HTTP status codes

2.4 Log Files

In computing, a log file is a file that records either events that occur in an operating system or other software runs or messages between different users of a communication software. Logging is the act of keeping a log. In the simplest case, messages are written to a single log file.



```
66.249.75.159 - - [29/Nov/2015:03:50:05 +0000] "GET /robots.txt HTTP/1.1" 200 55 "-" "Mozilla/5.0 (compatible; Googlebot/2.1; +http://
www.google.com/bot.html)"
66.249.75.168 - - [29/Nov/2015:03:50:06 +0000] "GET /blog/ HTTP/1.1" 200 8083 "-" "Mozilla/5.0 (compatible; Googlebot/2.1; +http://
www.google.com/bot.html)"
185.71.216.232 - - [29/Nov/2015:03:53:15 +0000] "POST /wp-login.php HTTP/1.1" 200 1691 "http://nohatenews.com/wp-login.php" "Mozilla/5.0
(Windows NT 6.1; WOW64; rv:40.0) Gecko/20100101 Firefox/40.0"
54.165.199.171 - - [29/Nov/2015:04:32:27 +0000] "GET /sitemap_index.xml HTTP/1.0" 200 592 "-" "W3 Total Cache/0.9.4.1"
54.165.199.171 - - [29/Nov/2015:04:32:27 +0000] "GET /post-sitemap.xml HTTP/1.0" 200 2502 "-" "W3 Total Cache/0.9.4.1"
54.165.199.171 - - [29/Nov/2015:04:32:27 +0000] "GET /page-sitemap.xml HTTP/1.0" 200 11462 "-" "W3 Total Cache/0.9.4.1"
54.165.199.171 - - [29/Nov/2015:04:32:27 +0000] "GET /category-sitemap.xml HTTP/1.0" 200 585 "-" "W3 Total Cache/0.9.4.1"
54.165.199.171 - - [29/Nov/2015:04:32:27 +0000] "GET /blog/ HTTP/1.0" 200 31746 "-" "-"
54.165.199.171 - - [29/Nov/2015:04:32:27 +0000] "GET /orlando-sports/ HTTP/1.0" 200 35510 "-" "-"
54.165.199.171 - - [29/Nov/2015:04:32:37 +0000] "GET /about/ HTTP/1.0" 200 25121 "-" "-"
54.165.199.171 - - [29/Nov/2015:04:32:42 +0000] "GET /business/ HTTP/1.0" 200 48661 "-" "-"
54.165.199.171 - - [29/Nov/2015:04:32:47 +0000] "GET /technology/ HTTP/1.0" 200 47130 "-" "-"
54.165.199.171 - - [29/Nov/2015:04:32:32 +0000] "GET /orlando-headlines/ HTTP/1.0" 200 48466 "-" "-"
54.165.199.171 - - [29/Nov/2015:04:32:52 +0000] "GET /entertainment/ HTTP/1.0" 200 48114 "-" "-"
54.165.199.171 - - [29/Nov/2015:04:32:57 +0000] "GET /travel/ HTTP/1.0" 200 47201 "-" "-"
54.165.199.171 - - [29/Nov/2015:04:33:02 +0000] "GET /national/ HTTP/1.0" 200 51623 "-" "-"
54.165.199.171 - - [29/Nov/2015:04:33:22 +0000] "GET /dallas-headlines/ HTTP/1.0" 200 33337 "-" "-"
54.165.199.171 - - [29/Nov/2015:04:33:27 +0000] "GET /chicago-sports/ HTTP/1.0" 200 37519 "-" "-"
54.165.199.171 - - [29/Nov/2015:04:33:32 +0000] "GET /dallas-sports/ HTTP/1.0" 200 33767 "-" "-"
54.165.199.171 - - [29/Nov/2015:04:33:37 +0000] "GET /houston-headlines/ HTTP/1.0" 200 33415 "-" "-"
54.165.199.171 - - [29/Nov/2015:04:33:07 +0000] "GET /science/ HTTP/1.0" 200 44758 "-" "-"
54.165.199.171 - - [29/Nov/2015:04:33:42 +0000] "GET /philadelphia-headlines/ HTTP/1.0" 200 36267 "-" "-"
54.165.199.171 - - [29/Nov/2015:04:33:47 +0000] "GET /houston-sports/ HTTP/1.0" 200 29133 "-" "-"
```

Figure 2.6: Log File Data

Chapter 3

Implementation

3.1 Methodology

The process used for this is ETL. ETL is a type of data integration that refers to the three steps (extract, transform, load) used to blend data from multiple sources. It's often used to build a data warehouse. During this process, data is taken (extracted) from a source system, converted (transformed) into a format that can be analyzed, and stored (loaded) into a data warehouse or other system. Extract, load, transform (ELT) is an alternate but related approach designed to push processing down to the database for improved performance.

3.1.1 Extraction

In this step, data is extracted from the source system into the staging area. Transformations if any are done in the staging area so that performance of source system is not degraded. Also, if corrupted data is copied directly from the source into the Data warehouse database, rollback will be a challenge. Staging area gives an opportunity to validate extracted data before it moves into the Data warehouse.
Data warehouse needs to integrate systems that have different DBMS, Hardware, Operating Systems and Communication Protocols. Sources could include legacy applications like Mainframes, customized applications, Point of contact devices like ATM, Call switches, text files, spreadsheets, ERP, data from vendors, partners amongst others. Hence one needs a logical data map before data is extracted and loaded physically. This data map describes the relationship between sources and target data.
For Extraction Technologies used are kafka and spark streaming.

3.1.2 Transformation

Data extracted from the source server is raw and not usable in its original form. Therefore it needs to be cleansed, mapped and transformed. In fact, this is the key step where the ETL process adds value and changes data such that insightful BI reports can be generated.

In this step, you apply a set of functions on extracted data. Data that does not require any transformation is called as direct move or pass through data.
In the transformation step, you can perform customized operations on data. For instance, if the user wants sum-of-sales revenue which is not in the database. Or if the first name and the last name in a table is in different columns. It is possible to concatenate them before loading.
For Transformation Technology used is Spark Core and Spark SQL.

3.1.3 Loading

Loading data into the target data warehouse database is the last step of the ETL process. In a typical Data warehouse, a huge volume of data needs to be loaded in a relatively short period (nights). Hence, the load process should be optimized for performance. In case of load failure, recover mechanisms should be configured to restart from the point of failure without data integrity loss. Data Warehouse admins need to monitor, resume, cancel loads as per prevailing server performance.
For loading of data again the technology used is Kafka.

3.2 Pipeline

The ETL Pipeline used for this project is this,

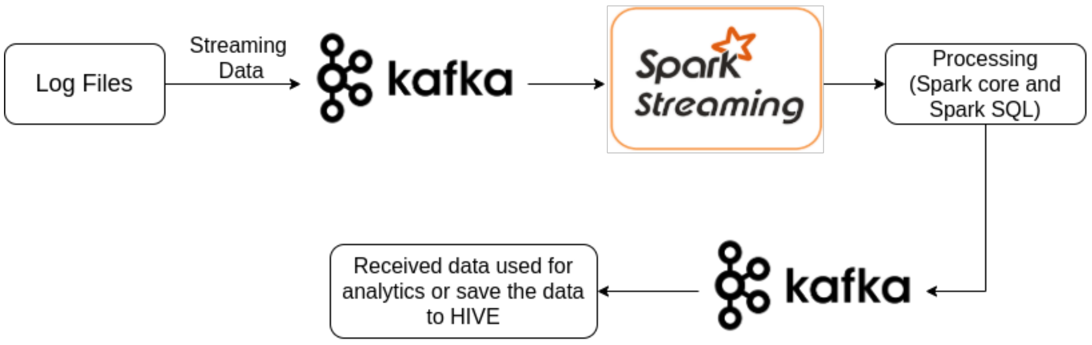

Log Files
Streaming Data
kafka
Spark Streaming
Processing (Spark core and Spark SQL)
kafka
Received data used for analytics or save the data to HIVE

Figure 3.1: Pipeline

Chapter 4

Procedure

Step 1: Log file to Kafka:

Log data is sent from log files to a kafka topic testing in real time using,

```
bin/kafka-console-producer.sh --broker-list localhost:9092 --topic \\
testing < access_logs.txt
```

Step 2: Kafka to Spark Application:

A Spark application pulls the data from kafka topic using spark streaming and process it using Spark SQL when an event occurs. The processed data is sent back to another kafka topic so anyone in the organization can access the data.

Step 3: Processing of data (Source Code):

```
    import java.sql.Timestamp
import org.apache.spark.sql._
import org.apache.spark.sql.functions._
import java.util.regex.Pattern
import java.util.regex.Matcher
import java.text.SimpleDateFormat
import java.util.Locale


object structuredStreaming {

  case class LogEntry(ip:String, client:String, user:String, \\
  dateTime:Timestamp, request:String, status:String, bytes:String, \\
  referer:String, agent:String)

  val logPattern: Pattern = apacheLogPattern()
  val datePattern: Pattern = Pattern.compile("\\[(.*?) .+]")


  def apacheLogPattern():Pattern = {
    val ddd = "\\d{1,3}"
    val ip = s"($ddd\\.$ddd\\.$ddd\\.$ddd)?"
    val client = "(\\S+)"
    val user = "(\\S+)"
    val dateTime = "(\\[.+?\\])"
    val request = "\"(.*?)\""
    val status = "(\\d{3})"
    val bytes = "(\\S+)"
    val referer = "\"(.*?)\""
    val agent = "\"(.*?)\""
    val regex = s"$ip $client $user $dateTime $request $status $bytes \\
    $referer $agent"
    Pattern.compile(regex)
  }
  def parseDateField(field: String): Option[Timestamp] = {

    val dateMatcher = datePattern.matcher(field)
    if (dateMatcher.find) {
      val dateString = dateMatcher.group(1)
      val dateFormat = new SimpleDateFormat("dd/MMM/yyyy:HH:mm:ss", \\
      Locale.ENGLISH)
      val date = (dateFormat.parse(dateString))
      val timestamp = new java.sql.Timestamp(date.getTime);
      return Option(timestamp)
    } else {
      None
    }
  }
```

```
def parseLog(x:Row) : Option[LogEntry] = {

  val matcher:Matcher = logPattern.matcher(x.getString(0));
  if (matcher.matches()) {
    val timeString = matcher.group(4)
    return Some(LogEntry(
      matcher.group(1),
      matcher.group(2),
      matcher.group(3),
      parseDateField(matcher.group(4)).getOrElse(null),
      matcher.group(5),
      matcher.group(6),
      matcher.group(7),
      matcher.group(8),
      matcher.group(9)
    ))
  } else {
    return None
  }
}

def main(args: Array[String]) {

  val spark = SparkSession
    .builder()
    .appName("StructuredStreaming")
    .master("local[*]")
    .config("spark.sql.streaming.checkpointLocation",\\
    "/home/UDHAV.MAHATA/Documents/Checkpoints")
    .getOrCreate()

  setupLogging()
```

```
val df = spark
  .readStream
  .format("kafka")
  .option("kafka.bootstrap.servers", "localhost:9092")
  .option("subscribe", "testing")
  .load()
val rawData = df.selectExpr("CAST(value as STRING)")

import spark.implicits._

val structuredData = rawData.flatMap(parseLog).select("status",
"dateTime")
val windowed = structuredData.withWatermark("dateTime", "1 hour")
  .groupBy($"status",window($"dateTime", "1 hour"))\\
  .count()
val finalOp1 = windowed.selectExpr("CAST(window AS STRING)",\\
"CAST(status AS STRING)","CAST(count AS STRING)")
finalOp1.createOrReplaceTempView("people")
val joint = finalOp1.sqlContext.sql("SELECT CONCAT(window,'||',
status,'||',count) AS status FROM people")
val finalOp = joint.withColumnRenamed("status","value")



val query = finalOp.select("value")
  .writeStream
  .format("kafka")
  .option("kafka.bootstrap.servers", "localhost:9092")
  .option("topic", "sink2")
  .start()
query.awaitTermination()
```

```
    spark.stop()
  }


}
```

Step 4: Kafka to Console: A kafka consumer of the same cluster can be used to retrieve the processed data using,

```
    bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 \\
    --topic sink2 --from-beginning
```

Chapter 5

Results and Discussions

Lets consider a case with the following details,

Input Kafka topic: testing

Output Kafka topic: sink2

At first to send data to a kafka topic, kafka must be running. As kafka depends on zookeeper,

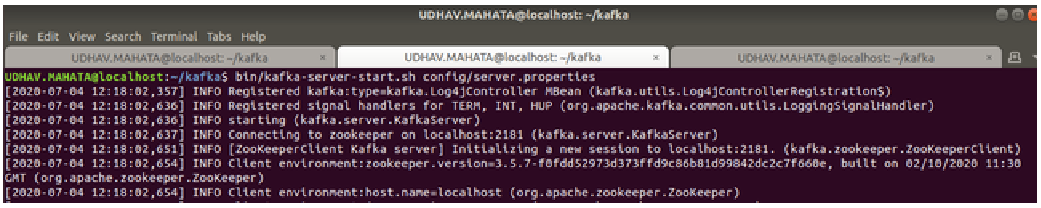
To start the zookeeper following statement must be entered on the terminal.

```
$ bin/zookeeper-server-start.sh config/zookeeper.properties
```

Then to start the start the kafka server following statement must be entered on the terminal

```
$ bin/kafka-server-start.sh config/server.properties
```

This image shows that the Kafka server is started



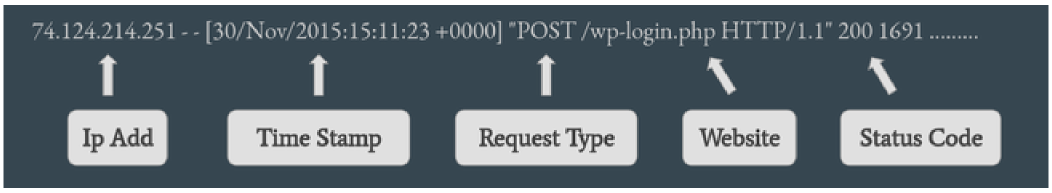
UDHAV.MAHATA@localhost: ~/kafka
File Edit View Search Terminal Tabs Help
UDHAV.MAHATA@localhost: ~/kafka
UDHAV.MAHATA@localhost: ~/kafka
UDHAV.MAHATA@localhost: ~/kafka

```
UDHAV.MAHATA@localhost:~/kafka$ bin/kafka-server-start.sh config/server.properties
[2020-07-04 12:18:02,357] INFO Registered kafka:type=kafka.Log4jController MBean (kafka.utils.Log4jControllerRegistration$)
[2020-07-04 12:18:02,636] INFO Registered signal handlers for TERM, INT, HUP (org.apache.kafka.common.utils.LoggingSignalHandler)
[2020-07-04 12:18:02,636] INFO starting (kafka.server.KafkaServer)
[2020-07-04 12:18:02,637] INFO Connecting to zookeeper on localhost:2181 (kafka.server.KafkaServer)
[2020-07-04 12:18:02,651] INFO [ZooKeeperClient Kafka server] Initializing a new session to localhost:2181. (kafka.zookeeper.ZooKeeperClient)
[2020-07-04 12:18:02,654] INFO Client environment:zookeeper.version=3.5.7-f0fdd52973d373ffd9c86b81d99842dc2c7f660e, built on 02/10/2020 11:30
GMT (org.apache.zookeeper.ZooKeeper)
[2020-07-04 12:18:02,654] INFO Client environment:host.name=localhost (org.apache.zookeeper.ZooKeeper)
```

Figure 5.1: Screenshot of Kafka Server

The log data to be sent to the kafka topic one line at a time.

Log data contains,



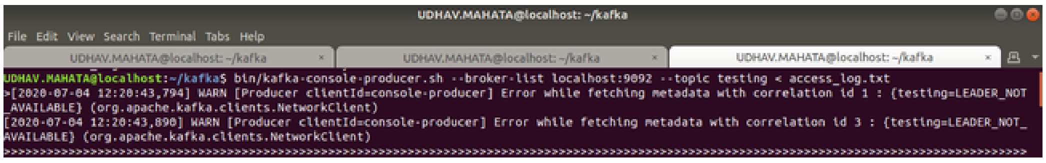
74.124.214.251 - - [30/Nov/2015:15:11:23 +0000] "POST /wp-login.php HTTP/1.1" 200 1691 ....
Ip Add
Time Stamp
Request Type
Website
Status Code

Figure 5.2: Log Data Explanation

The data is sent to the log file using,

```
$ bin/kafka-console-producer.sh --broker-list localhost:9092
--topic testing < access_logs.txt
```

This image shows the log file data sent to a Kafka topic


UDHAV.MAHATA@localhost: ~/kafka
File Edit View Search Terminal Tabs Help
UDHAV.MAHATA@localhost: ~/kafka
UDHAV.MAHATA@localhost: ~/kafka
UDHAV.MAHATA@localhost: ~/kafka

```
UDHAV.MAHATA@localhost:~/kafka$ bin/kafka-console-producer.sh --broker-list localhost:9092 --topic testing < access_log.txt
>[2020-07-04 12:20:43,794] WARN [Producer clientId=console-producer] Error while fetching metadata with correlation id 1 : {testing=LEADER_NOT
_AVAILABLE} (org.apache.kafka.clients.NetworkClient)
[2020-07-04 12:20:43,898] WARN [Producer clientId=console-producer] Error while fetching metadata with correlation id 3 : {testing=LEADER_NOT_
AVAILABLE} (org.apache.kafka.clients.NetworkClient)
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

Figure 5.3: Log data to Kafka

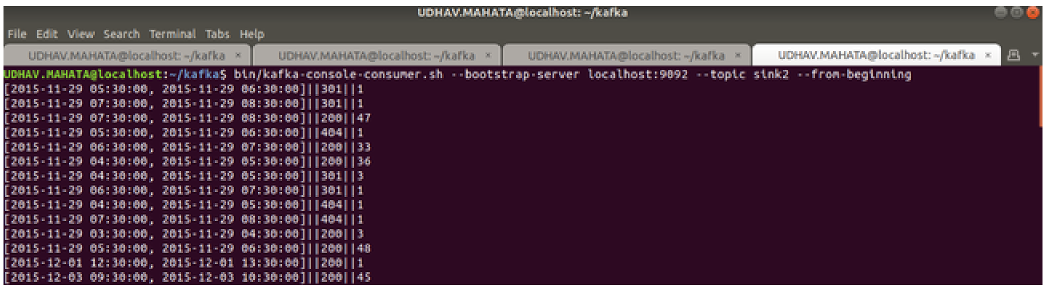
Then the spark application acts as receiver and pulls data from Kafka. Each line of data coming from the kafka topic **testing** is considered as row of a table and SQL queries are run on the data for processing.

Finally the spark application acts as producer and the final processed data is sent to a new kafka topic **sink2**.

The final output sent back to the kafka topic **sink2** can be viewed using the following statement,

```
$ bin/kafka-console-consumer.sh --bootstrap-server localhost:9092
--topic sink2 --from-beginning
```

This image shows the output of the processed data.

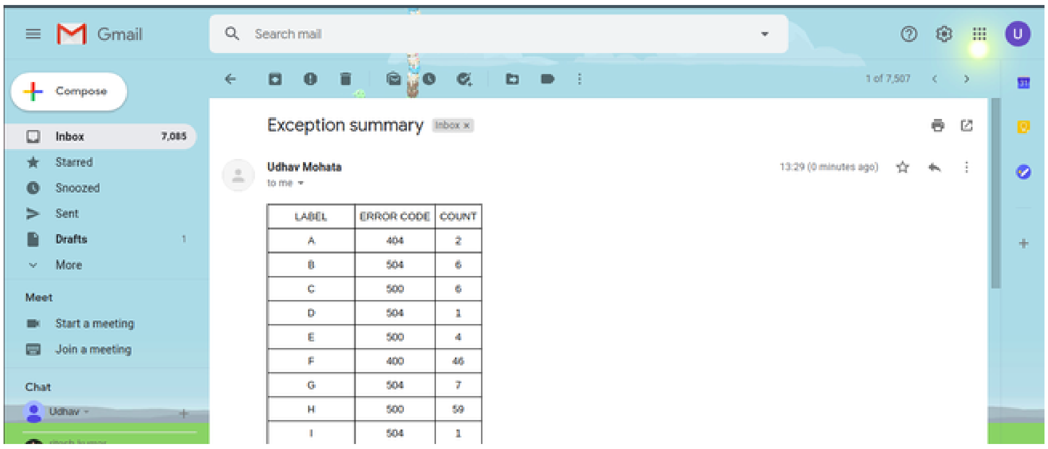

UDHAV.MAHATA@localhost: ~/kafka
File Edit View Search Terminal Tabs Help
UDHAV.MAHATA@localhost: ~/kafka
UDHAV.MAHATA@localhost: ~/kafka
UDHAV.MAHATA@localhost: ~/kafka
UDHAV.MAHATA@localhost: ~/kafka

```
UDHAV.MAHATA@localhost:~/kafka$ bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic sink2 --from-beginning
[2015-11-29 05:30:00, 2015-11-29 06:30:00]||301||1
[2015-11-29 07:30:00, 2015-11-29 08:30:00]||301||1
[2015-11-29 07:30:00, 2015-11-29 08:30:00]||200||47
[2015-11-29 05:30:00, 2015-11-29 06:30:00]||404||1
[2015-11-29 06:30:00, 2015-11-29 07:30:00]||200||33
[2015-11-29 04:30:00, 2015-11-29 05:30:00]||200||36
[2015-11-29 04:30:00, 2015-11-29 05:30:00]||301||3
[2015-11-29 06:30:00, 2015-11-29 07:30:00]||301||1
[2015-11-29 04:30:00, 2015-11-29 05:30:00]||404||1
[2015-11-29 07:30:00, 2015-11-29 08:30:00]||404||1
[2015-11-29 03:30:00, 2015-11-29 04:30:00]||200||3
[2015-11-29 05:30:00, 2015-11-29 06:30:00]||200||48
[2015-12-01 12:30:00, 2015-12-01 13:30:00]||200||1
[2015-12-03 09:30:00, 2015-12-03 10:30:00]||200||45
```

Figure 5.4: Processed Data

The final output can be sent via email in table format instead of viewing it on console.

This image shows the output is sent automatically via email.



Exception summary Inbox

Udhav Mohata
to me
13:29 (0 minutes ago)

LABEL	ERROR CODE	COUNT
A	404	2
B	504	6
C	500	6
D	504	1
E	500	4
F	400	46
G	504	7
H	500	59
I	504	1

Figure 5.5: Output on email

Observation

- The above output shows that website A had 2 "404" ie. 2 errors made by the user while login in to the page.
- And website B and D had 6 and 1 "504" errors respectively ie. error occurred because the server was down or the server crashed.
- By looking at the above errors developers can work on in and resolve the issue.

Chapter 6

Conclusion

Data being the most important asset for any company, it is very important for a company to manage its own and also customer data. As the data is growing exponentially so big data technology like Spark, Kafka is to be used to extract, transfer and load/save the data efficiently and with high fault tolerance. Another advantage of using these technologies is that managing such volume of data is possible with minimum cost and the quality of data never reduces.

6.1 Scope for future work

- It can connect to any type and any number of sources.
- It can be used to read details from different sensors and process it in parallel.
- The processed data can be sent to any output, like:
    - Console
    - Text Message or Email.
    - Webpage