Distributed Stream Processing of Twitter Data using Apache Spark

Thesis submitted in partial fulfilment of the requirements for the

award of degree of

Master of Engineering

In

Computer Science and Engineering

Submitted By

Shruti Arora

(801632045)

Under the supervision of:
Dr. Rinkle Rani
Associate Professor

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY
PATIALA – 147004

JUNE 2018

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, **"Distributed Stream Processing of Twitter Data using Apache Spark"**, in partial fulfillment of the requirements for the award of degree of Master of Engineering in *Computer Science and Engineering* submitted in Computer Science and Engineering Department of Thapar Institute of Engineering and Technology, Patiala, is an authentic record of my own work carried out under the supervision of ***Dr. Rinkle Rani*** and refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

Signature:

(**Shruti Arora**)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

Rinkle 21/06/18

(**Dr. Rinkle Rani**)
Associate Professor
Computer Science and Engineering Department

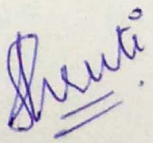
ACKNOWLEDGEMENT

The successful completion of any task would be incomplete without acknowledging the people who made it possible and whose constant guidance and encouragement secured the success.

First of all I wish to acknowledge the benevolence of omnipotent God who gave me strength and courage to overcome all obstacles and showed me the silver lining in the dark clouds with the profound sense of gratitude and heartiest regard. I express my sincere feelings of indebtedness to my guide **Dr. Rinkle Rani** for their positive attitude, excellent guidance, constant encouragement, keen interest, invaluable co-operation, generous attitude and above all their blessings. She has been a source of inspiration for me.

I am grateful to **Dr. Maninder Singh**, Head of Department and **Dr. Ashutosh Mishra**, P.G. Coordinator, Computer Science and Engineering Department, Thapar Institute of Engineering and Technology for the motivation and inspiration for the completion of this thesis. I will be failing in my duty if I don't express my gratitude to **Dr. S.S. Bhatia**, Senior Professor and Dean of Academics Affairs in the institute for making provisions of infrastructure such as Library facilities, Computer Lab equipped with internet facility immensely useful for the learners to equip themselves with latest in the field.

Last but not the least I would like to express my heartfelt thanks to my parents and my friends who with their thought provoking views, veracity and whole hearted cooperation helped in doing this thesis.



Shruti Arora

(801632045)

ABSTRACT

Data is continuously being generated from sources such as machines, network traffic, sensor networks, etc. Twitter is an online social networking service with more than 300 million users, generating a huge amount of information every day. Twitter's most important characteristic is its ability for users to tweet about events, situations, feelings, opinions, or even something totally new, in real time. Currently there are different workflows offering realtime data analysis for Twitter, presenting general processing over streaming data. This study will attempt to develop an analytical framework with the ability of in-memory processing to extract and analyze structured and unstructured Twitter data. The proposed framework includes data ingestion and stream processing and data visualization components with the Apache Kafka and Apache Flume messaging system that is used to perform data ingestion task. Furthermore, Spark makes it possible to perform sophisticated data processing and machine learning algorithms in real time. We have conducted a case study on tweets and analysis on the time and origin of the tweets. We also worked on study of SparkML component to study the K-Means Clustering algorithm.

TABLE OF CONTENTS

LIST OF FIGURES

Figure No. **Description** **Page No.**

LIST OF TABLES

Table No. **Description** **Page No.**

Chapter 1 INTRODUCTION

At present Big Data is not only trending expression in a specific context but it is all around. With a colossal measure of data produced each day and all around, it turns into an unquestionable requirement for any association and industry to manage, as available resources i.e. both hardware and software can't handle the huge measure of various sorts of information that is made at such a rapid speed.

Sensors, log information of machines, information stockpiles, open web, online networking, business applications, media, documents, and various different kinds of advancements are making and catching information ceaselessly in substantial amounts, offering incredible chances to control and utilize it [7]. In any case, we confront new specialized difficulties concerning overseeing, arranging, preparing, and breaking down this colossal measure of information and in how to empower businesses and organizations to control learning to enhance the procedure of basic analysis, decision making and accomplishing higher efficiency.

The interest for stream handling is expanding a great deal nowadays. The reason is that often processing enormous volumes of information isn't sufficient. Information must be handled quickly and should be dissected. The speed at which information is created, ingested, prepared, and broken down is expanding at a staggeringly quick pace.

The businesses request information processing and analysis in near real-time and continuously. In traditional database systems, for example, Apache Hadoop was the solution for processing of big data but as applications and information grew and analysis in real-time was demanded by various industries then the demand for other techniques [9] grew where the batch processing based Hadoop [14] wasn't appropriate for these utilization cases. Subsequently, various open source ventures have been begun over the most recent couple of years to manage the streams of data. All were intended to process a ceaseless grouping of records beginning from in excess of one source.

As opposed to the traditionally designed database models where sheer size data is first stored and recorded and after that along these lines processed by the queries, stream processing takes the inbound information while it is in flight, as it streams through the

server. Stream processing likewise associates with outer information sources, empowering applications to incorporate chosen data into the application stream, or to refresh an external database with handled data.

1.1 Characteristics of Big Data

The major characterisitics [9] of Big Data are described as 5V's in Figure 1.1:

1. Volume
2. Velocity
3. Variety
4. Veracity
5. Value

Volume: The term 'Big Data' itself is identified with a size which is gigantic. Size of data assumes exceptionally pivotal part in deciding an incentive out of information. Likewise, regardless of whether a specific information can really be considered as a Big Data or not, is endless supply of information.

Velocity: Data comes from variety of sources in volumes and in varying rates.

Variety: Data in the form of emails, photos, videos, monitoring devices, PDFs, audio, etc. is also being considered in the analysis applications. This variety of unstructured data poses certain issues for storage, mining and analysing data.

Veracity: expresses the chaos or dependability of the information. With numerous types of big data, quality and precision are less controllable, for instance Twitter posts with hashtags, contractions, mistakes and everyday discourse. Huge information and analytics innovation now enables us to work with these sorts of information. The volumes regularly compensate for the absence of value or exactness.

Value: The term is coined for expressing the worth of the data that is being extracted for use out of enormous data. The most imperative piece of setting out on a major information activity is to comprehend the expenses and advantages of gathering and dissecting the information to guarantee that at last the information that is harvested can be adapted.

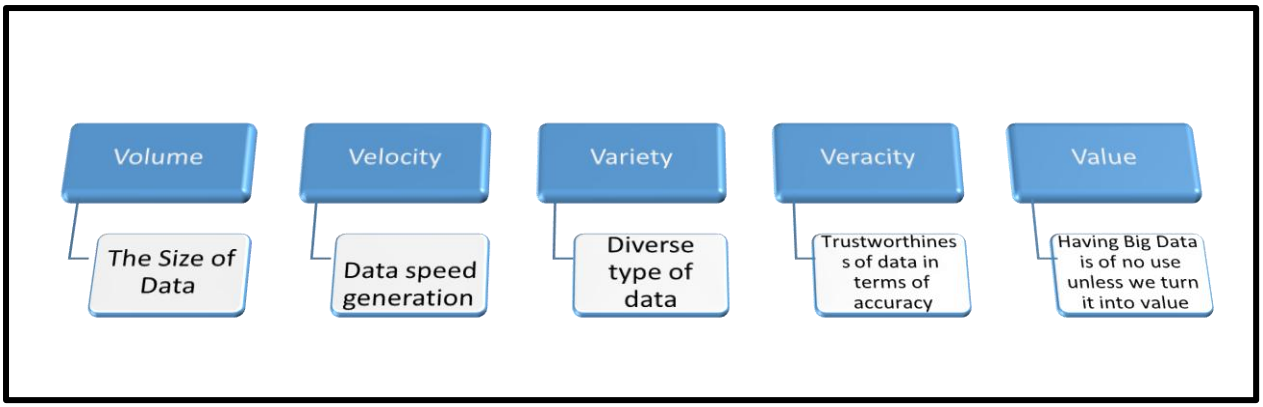


Figure 1.1: 5V Characteristics of Big Data

1.2 Data Streams

A stream is defined as a possibly unbounded sequence of data items or records that may or may not be related to, or correlated with each other. Each data is generally timestamped and in some cases geo-tagged.Streams pose very difficult challenges for conventional data management architectures which are built primarily on the concept of persistence and static data collections.

Some applications with 5V's characteristics presents the need to process data as it is generated, or in other words, as it streams. These types of applications are known as Streaming Data Processing Applications. This terminology refers to a constant stream of data flowing from a source for example data from a sensory machine or data from social media. An example application would be making data-driven marketing decisions in real-time or near real-time through the use of data from real-time sales trends, social media analysis, and sales distributions. Another major example for streaming data processing is monitoring of industrial or farming machinery, detection of potential system failures.In fact, any sensor network or internet of things environment controlled by another entity or set of entities fall under this category.

1.3 Challenges of Streaming Data

- Processing massive amounts of streaming events and performing various operations such as aggregate, filter, automate, rule, estimate, respond, monitor and caution about the changing trends.
- Continuous responsiveness to changing market situations.
- Scalability and performance [1] as voluminous data fluctuates in size and complexity.

- Data Analytics is very challenging in the field where discovery and monitoring, continuous query processing, automated alerts and reactions.
- Community (education/discussion, component/connector exchange, certification/ training).
- Push-based visualization.

1.4 Processing of streaming data

1.4.1 Batch Processing

Traditionally data was processed in batches or chunks on the basis of data arrival or in the context of similarity after fixed intervals of time and the term coined was Batch Processing. For example, processing all the records that have been generated by a major firm in a stipulated period of time. The data may contain millions of records for a day that can be stored as a file or record. The particular file will undergo processing at the end of the day for various analyses that firm wants to do and thus would turn out to be time inefficient approach. Hadoop MapReduce [14] is the best structure that is built for Batch Processing [4].

1.4.2 Stream Processing

With the rapid development of technology and applications the data is expanding every microsecond and henceforth some applications require real-time trend analysis and amongst such trend analysis applications intrusion detection, stock market prediction and earthquake alert system [13] are extensively critical. Such applications require real-time stream processing, storage and dashboard updations. Spark is the best structure built-on Hadoop MapReduce for Stream Processing. Other platforms such as Storm [2], Azure Stream Analytics, AWS are also available but Spark proves to be the best Stream Processing Platform.

Figure 1.2. explains the Batch Processing v/s Stream Processing for Big Data with the help of Earthquake Alert System which is the most critical application which requires rapid processing and real-time monitoring of Big Data so that many such natural calamities can be avoided.

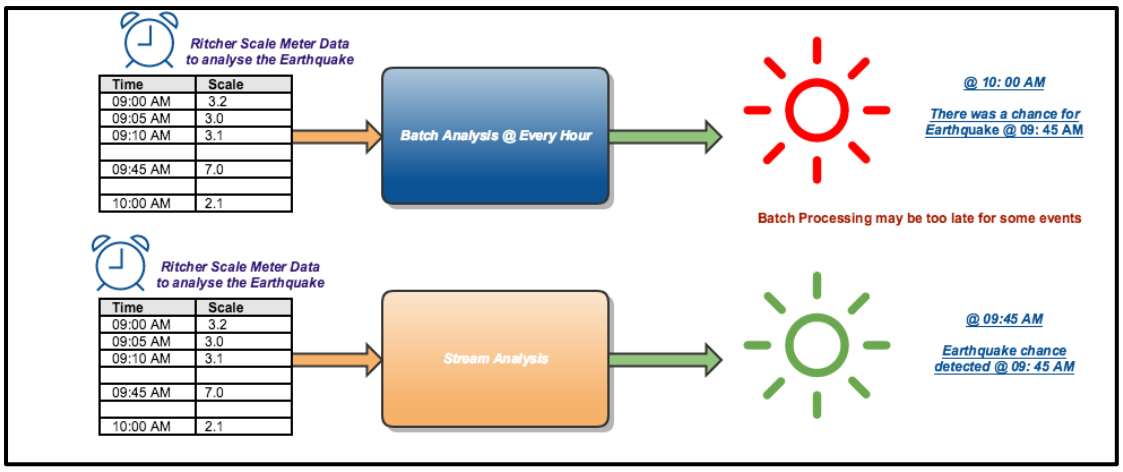


Figure 1.2: Batch Processing v/s Stream Processing for Big Data with the help of Earthquake Alert System [13]

1.5 Real-Time Stream Processing Platforms

The immense measures of data require more powerful softwares shown in Figure 1.3 for processing, best took care of by big data processing platforms. The best preffered stream processing structures which are reasonable for meeting a wide range of necessities of organizations are described below:

1.5.1 Open-Source Platforms

1. Apache Spark Streaming
2. Apache Flink
3. Apache Storm
4. Apache Samza
5. Apache Kafka Streams

Figure 1.3: Open-Source Stream Processing Platforms [12]

1.5.2 Commercial Platforms

1. Microsoft Azure Stream Analytics (Architecture in Figure 1.4)
2. IBM Streams
3. SAP Event Stream Processor
4. Oracle Stream Analytics
5. SAS Event Stream Processing

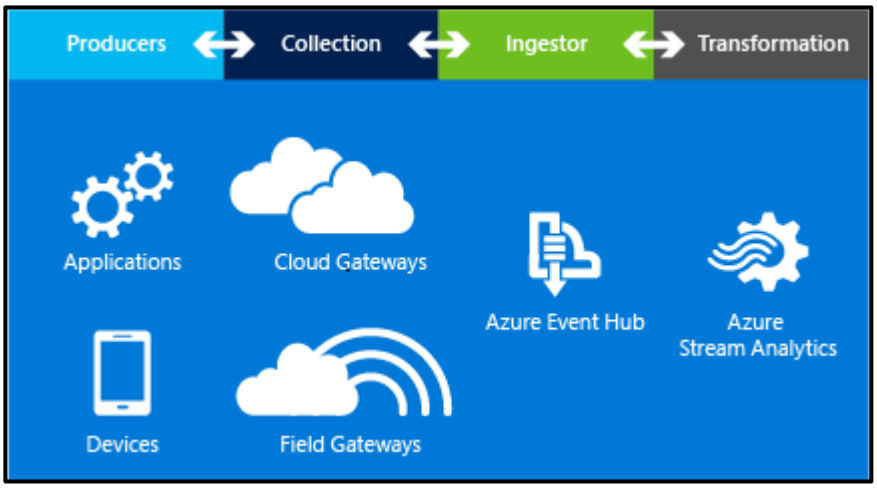


Figure 1.4: Commercial Platform for Stream Analytics- Azure [14]

1.6 Stream Storage Platforms

The heterogeneity of data brings challenges for big data relating to complexity and volume. A current review says that 80% of the information made on the planet are unstructured. One challenge is the manner by which these unstructured information can be organized, before we strive to capture and comprehend the most vital data. Another challenge is the manner by which we can store it.

The databases which are utilized to store big data [3] are the NoSQL databases. NoSQL databases don't utilize SQL to perform operations and transformations on the data.These sort of databases are not principally based on the idea of tables that is they can likewise store unstructured information.

Traditionally, RDMS was the only Platform for storage of data in rows and columns and was based on the ACID properties but with the rapid growth of unstructured data NoSQL databases were invented which are based on BASE properties.

BaSE Properties of NoSQL Databases are described as:

Basic Availability: The database seems to work more often than not. Every request will get a response whether it as a success or failure.

Soft state: The state of the system could change after some time, also when input is not provided and the changes were going on due to 'eventual consistency'.

Eventual consistency: Stores display consistency at some later point (e.g., sluggishly at read time).

The top tools/platforms available for distributed storage and query analysis of Big Data are described in Table 1.1:

Table 1.1: Distributed Storage Platforms and their features

Databases	API	Protocol	Query Method	Replication	Written in Language	Misc
Hadoop/HBase	Java	Any write call	MapReduce Java	HDFS Replication	Java	NA
Cassandra	CQL and Thrift	CQL Binary Protocol	CQL	Peer-to-peer	Java	Built-in data compression
Hypertable	Thrift	Thrift	HQL	HDFS Replication	Java,PHP,Perl,Python, Ruby	High performance C++ implementation of Google's Bigtable.
Apache Flink	Java,Scala	Two-phase commit protocol	Expressive data flows	State Replication	Java	Good integration with Hadoop stack (HDFS, YARN)
IBM Informix	REST	TCP/IP	SQL,MongoDB	Master-Slave,Peer-to-peer	C	ACID, built-in data compression

1.7 Streaming Data Visualization

Visualisation of streams of data represents various difficulties in the diversified space. In the period of enormous data, we require incremental representation strategies that will permit the experts to investigate data rapidly and enable them to make critical choices on time. Stream processing platforms like Apache Spark, Apache Storm [12], Apache Samza, Apache Kafka, and Amazon Kinesis were worked to give the appropriated stage whereupon streaming applications can be fabricated. These platforms have the best processing and storage architectures but these lack in user-friendly interfaces. Streaming Data require distribution handling UIs because the need for updated dashboards in critical applications should be fulfilled as early as possible That is the place items from sellers like Zoomdata, Alooma, Datawatch, and Splunk are planning to have any kind of effect. The advent of these technologies further enhance the business analytics decision making.

- **Zoomdata**: Zoomdata is not limited to the capability to do visual analysis of data streams. The Zoomdata Server is largely based on Spark provided data ingestion from Hadoop, NoSQL, cloud apps, and traditional data warehouses, and eases the availability in the visual data analysis and query interface. The ease of "push-down" data processing ensures better utilization of distributed computing resources. The technique used to process billions of rows of data in real time in Zoomdata is "data sharpening"
- **Alooma**: It is another technique that is making miracles in the stream processing visualisation. It has integerated solutions for open source platforms like Apache Kafka, Apache Storm, Redis, and Zookeeper to provide clients the capability to query and visualize streaming data.
- **Power BI:** Power BI is a compilation of connectors, apps, and software services that work together to amplify unrelated sources of data into coherent and visually immersive. Whether the data is a collection of cloud-based and on-premises hybrid data warehouses or a simple Excel spreadsheet, Power BI is used for connection of data sources and visualization. It is an online SaaS (Software as a Service)

Amongst the mentioned few streaming Data Visualization tools,an integration of Microsoft Azure i.e. PowerBI is the most powerful tool the offers various services mentioned in the Figure 1.5.

Metrics	Microsoft Power BI	Tableau Desktop	QlikView
Free Version Available	✔	✘	✔
Mobile Versions	✔	✔	✔
Point in time Analytics	✔	✔	✔
Real Time Analytics	✔	✔	✔
Predictive Analytics	✔	✔	✘
Data Prep Tools	✔	✔	✔
Tools to Blend/ Join/ Integrate Data	✔	✔	✔
Semantic Querying/ Natural Language	✔	✘	✘
Social Media Analytics	✔	✔	✔
Visualizations Feature	✔	✔	✔
Sharing/ Collaboration Tool	✔	✔	✔

Figure 1.5: Streaming data visualisation tools and their features

Chapter 2 LITERATURE SURVEY

2.1 Distributed Stream Processing

Mitch and Hari in [19] presented the architecture of current database management Platforms as pull-based model for accessing the data and stream-based data management as push-based model. The authors assumed a push based model of information as a client (the active party) presenting an inquiry to the server (the passive party) and an answer is returned. Conversely, in stream-based applications information is pushed to a framework that must assess questions accordingly to recognized occasions. Question answers are then pushed to a holding up client or application. Along these lines, the stream-based display reverses the conventional database management model by expecting clients to be latent and the database management system to be dynamic.

The various challenges faced by the distributed stream computation were considered by the author and a framework called Aurora which is a centralized stream processing system.Aurora is designed assuming the domain in which a single administrative domain is present.

Aurora system assumes the data coming from a diverse sources such as computer programs that produce data at regular or irregular intervals or sensor data. A data stream is actually a potentially unbounded collection of tuples generated by a data source. The relational database model has tuples to represent the rows or records, similarly, stream tuples are the ones that are generated in real-time and are typically not available after a particular interval of time. Aurora processes tuples as specified bythe application administrator. Aurora is built on the fundamentals of a data-flow system and uses the box and arrow paradigm. An extension of Aurora is Medusa which is built for service delivery among autonomous participants.Brief architecture of Aurora is displayed in Figure 2.1

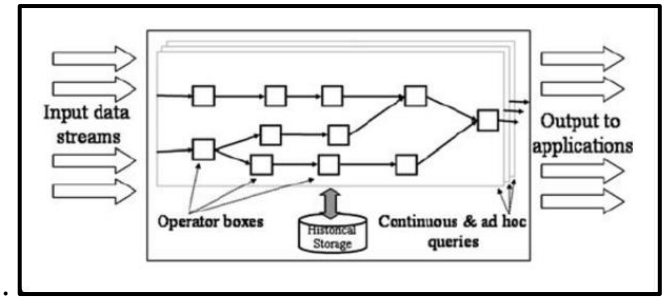


Figure 2.1: Architecture of Aurora [19]

2.2 Machine Learning with Stream Processing Platforms

Machine Learning is the method of teaching computers to make and improve predictions or behaviours based on some data. For example Gmail: It learns and classifies the emails as spam. More the machine trained with data more the accuracy of its learning and performing. Spark was developed for the pupose of efficient iterative computation and it's early releases were capsulated with dummy machine learning algorithms. The parent version of MLlib was developed at UC Berkeley by 11 contributors, and provided a limited set of standard machine learning methods.

Meng et al. reviewed the MLlib library of open source distributed stream processing platform Apache Spark. MLlib has a support for standard learning algorithms for common operations such as regression, classification, clustering [17], collaborative filtering, and dimensionality reduction. An important advantage of using Spark over other techniques for training machines is its speed and scalability. It is written in Scala and uses native c++ based linear algebra libraries on each node. It also has API support for Java, Scala and Python. It also provides a variety of underlying statistics, optimization primitives and linear algebra.

Boehm et al. [20] presented latest overview of machine learning, required extensions of engine to exploit Spark to the core, and optimized solutions to unique implementation challenges on Spark. Some of the major challenges under study are memory handling [36] and lazy evaluation. Spark related key optimizations are automatic integration of RDD caching/checkpointing and repartitioning, as well as partitioning-preserving operations to minimize data scans trials and shuffles. The experiments by author show that the hybrid execution plans are crucial for performance.

2.3 Study of architectures of Stream Processing Platforms

2.3.1 Apache Storm

Apache Storm is an open source after being acquired by Twitter. Originally created by Nathan Marz and team at BackType. The initial release was on 17 September 2011. It is distributed stream processing computation platform written in the Clojure programming language. A Storm application is designed as a topology in the shape of a directed acyclic graph with spouts and bolts acting as the graph vertices. It uses spouts and bolts to define data sources and transformations to allow batch, distributed processing of streaming data. Edges on the graph are named streams and direct data from one node to another. Together, the topology acts as a data transformation pipeline. The general topology structure resembles a MapReduce job, with the main difference being that data is processed in real time as opposed to in individual batches. Additionally, Storm topologies run indefinitely until killed.

Mattheis et al. in [7] describe a system for online map matching using the state of the art algorithm based on a Hidden Markov Model and open source software, Apache Storm and Apache Cassandra database, as well as OpenStreetMap data.

Stojanovic et al. in [21] worked to for the processing of Big Trajectory Systems using twitter feeds for obtaining the count of vehicles at each street segment and detected the congested street segments in a real-time. The authors correlated user mobility information to people's Twitter activities in order to provide geo-tagged tweets [11] generated on crowded street segments. The traffic status (dynamic travel time, slowdown, congestion, etc.) and events (accidents, roadwork, etc.) were identified that possibly trigger tweets that potentially offer more information related to traffic situation. For obtaining the results other real-time streaming data such as sensors integrated in smartphones, accelerometer, as well as in-vehicle sensors accessible through OBD II interface can be used.

The comparison of Storm with Hadoop is illustrated in Table 2.1.

Table 2.1: Comparison of Storm with Hadoop

Storm	***Hadoop***
Distributed real-time processing of enormous data streaming at high-velocity.	Distributed batch processing of enormous data at high-velocity data.
Data is mostly dynamic and continuously streamed**.**	Data is mostly static and stored in persistent storage**.**
Relatively slow.	Relatively fast.
Platform built of sprouts and bolts**.**	Platform built of HDFS and MapReduce**.**
Scalable and fault-tolerant**.**	Scalable and fault-tolerant**.**
Implemented in Clojure.	Implemented in Java.
Simple and can be used with any programming language.	Complex and can be used with any programming language.
Easy to set up and operate**.**	Easy to set up but difficult to operate**.**

2.3.2 Apache Kafka

Kafka is publish/subscribe messaging system. A cluster can be set up entirely for Kafka servers and their entire job is to just store all incoming messages from publishers which might be a bunch of web servers or sensors for some period of time and as it comes in it will store them up and publish them to anyone who wants to consume them. The messages are associated with a term called topic which represents a specific stream illustrated in Figure 2.2. For example, a topic of web logs from a given application or a topic of sensor data from a given system. Consumers genrally subscribe to one or more topics and they receive the data as it is published. A consumer if goes offline can catch up the data from some point in the past. The important characteristic about Kafka is that it has ability to manage multiple consumers that might be at different points in the same stream and it can do that very efficiently.

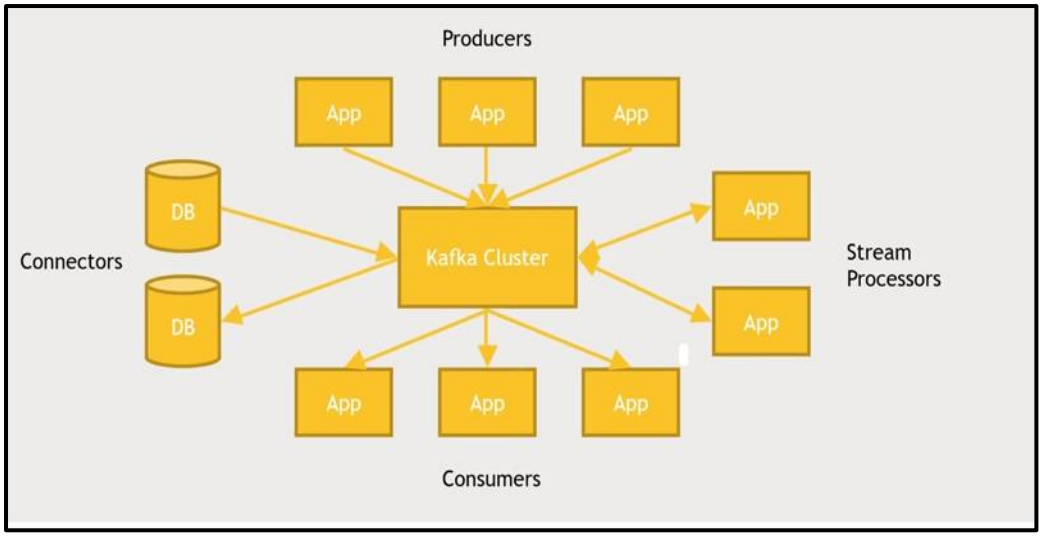


Figure 2.2: An architecture of Kafka [26]

As mentioned Kafka can be spread out on a cluster of itself which further eradicates any single point of failure where as multiple servers can also be considered each running multiple processes and which distribute the storage of the data on topics and processing of all publishers and subscribers connected to Kafka.

Kreps et al. in [22] presented Kafka messaging system for log processing in real time at LinkedIn Corp. Numerous specialized log aggregators have been deveoped over the last few years. One such is Scribe developed by Facebook.Scribe has an architecture where each frontend machine can send log data to a set of Scribe machines over sockets from where the aggregation of log entries are made and periodically uploaded into HDFS [23] or an NFS device. Similarly, Yahoo's data highway project has a resembling dataflow. Several machines aggregate the events from the users and divide them into minute files which are then uploaded to HDFS. Flume which is developed by Cloudera is a relatively new log aggregator. It has a flexible support for streaming data. However, most of these systems are developed for the offline consumption of log data. Additionally, most of them use a "push" model in which the broker is involved in order to forward data to consumers. Kafka enables the synchronous message retrieval by consumer at the maximum rate it can validate and also avoid being flooded by messages pushed faster than it can manage.

Comparitive analysis of Storm and Kafka is illustrated in Table 2.2

Table 2.2: Comparison of Apache Storm and Kafka

Apache Storm	Apache Kafka
Invented by Twitter	Invented by LinkedIn
Real Time Message Processing System	Distributed Messaging System
Data source can be Kafka or any other database system.	Data source is Facebook, Twitter etc.
It's basic use is stream processing	It is basically used as Message Broker
No storage.Data gets transferred from input stream to output stream.	Data is stored in File system such as EXT4 or XFS.
Processing type is Micro-Batch processing	Processing type is Small-Batch processing
It is independent of any external application.	It is zookeeper dependent.
It has a latency of milli-seconds	Latency is dependent on data source but is generally 1-2 seconds
It has support for all languages.	It works best with Java but has a support for all languages.

2.3.3 Apache Flink

Apache Flink's development started in technical university in Berlin in 2009 under the project called Stratosphere. Afterwards, it was incubated to Apache in April, 2014. It became the top level project in December, 2014. The name Flink was used for the platform because in German it means swift or agile.

It is the powerful open source engine which can be regarded as Next Generaton Big Data Tool, also known as 4G of Big Data. It is a single unified platform with lightening fast speed tool for sophisticated analytics of :

- Batch Processing
- Interactive Processing
- Real-time (streaming) processing
- Graph processing
- Iterative processing
- In-memory processing [15]

Flink is a true streaming framework. It does not cut the stream into micro-batches like Spark. It processes the data as soon as it arrives. Flink's core [32] is a streaming dataflow engine that facilitates data dispension, transmission and fault tolerance for computations that are distributed on various clusters. It is basically a general purpose framework which targets to unify different data loads. It has the capability of processing events at a consistently high rates with low latency as low as a single-digit in milliseconds.

Carbone et al. [24] in their research reviewed that the Flink Ecosystem consists of Kernel which has a component called Runtime used for Distributed Streaming Dataflow. On the layer above, the specialized components called Dataset API used for Batch Processing and Datastream API used for Stream Processing, Flink ML for machine learning libraries, Gelly for Graph processing.

Flink can be deployed on local machine i.e in Single JVM. The deployment in cluster mode can be done in standalone mode. The Resource manager of standalone mode is shipped with flink. The deployment can be done on cloud as well like Google compute engine or Amazon's Elastic Compute Cloud(EC2) [25].

Flink does not have its storage component. In other words, it is just a processing engine. It is dependent on third party storage system. Flink is capable of reading the data from various storage systems like local FS or HDFS(Hadoop File System) or S3 where the data is present in the form of files. It can also read the data from NoSQL databases like MongoDB, HBase or even from relational databases. It collects live streams from Kafka, Flume etc.

Flink architecture is described in the Figure 2.3

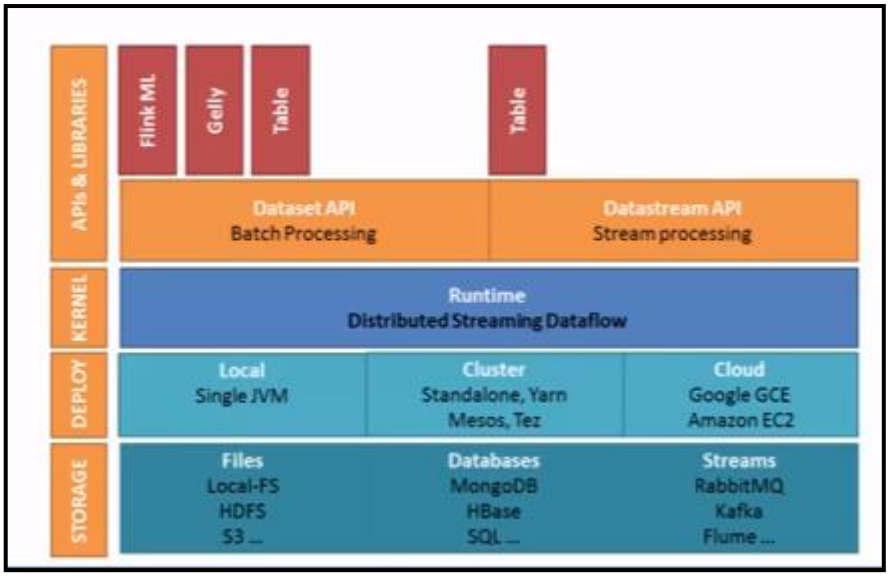


Figure 2.3: An architecture of Flink [24]

2.3.4 Apache Samza

Apache Samza is live platform for data processing and framework built purposely to perform asynchronous computation for processing the stream created by the Apache Software Foundation in Java as well as Scala. It is an open source tool that uses Kakfa for message passing integrated with Apache Hadoop YARN in order to achieve fault tolerance, security, processor segregation and management of resources.

Samza has the following key properties:

- **Simple API**: Unlike most messaging framework which are low-level APIs, Samza gives an extremely basic call-back-based "process message" API practically identical to Hadoop's MapReduce.
- **Durability**: Apache Samza utilises Kafka streams to assure that message processing is in the sequence in which they were written to a partition, and hence none of the messages are worn out.
- **Managed state**: Samza organizes checkpointing as well as recovers the state of stream processor.It restores its state to a snapshot which is consistent as soon as it is restarted. Samza is developed for managing huge amounts of state (several GBs / partition).

- **Fault tolerance**: Migration of tasks from one machine to another is carried out if one of system fails in the whole cluster which is taken care by Samza by transparently working with YARN.
- **Scalability**: At every level, Samza is partitioned and distributed. It has integration of Kafka for providing ordered, fault-tolerant and partitioned streams. YARN provides a distributed environment for Samza containers to execute in.
- **Processor isolation**: Samza works with Apache YARN to support security model of Hadoop, and resource isolation through Linux CGroups.

Kleppmann et al. [26] defined a hybrid framework of Kafka and Samza and presented the state management problem for implementing stream processing by creating small number of general purpose abstractions. Durable state is implemented by Samza through the abstraction of KeyValueStore. An interface of Java call StreamTask is provided by Samza which is implemented by application code. Every instance of StreamTask has a different store that it can write and read as required. The low latency, high-throughput key-value [8] store embedded in RocksDB4 is used by Samza access to data on local disk. To ensure the durability of the embedded store in the face of node and disk failures, every write to the store is also sent to a dedicated topic-partition in Kafka. This changelog topic acts as a replication log which is durable. The log compaction mode in Kafka protects infinite growth of the changelog topic: if the same key is repeatedly overwritten, Kafka eventually has an ability of garbage-collection of overwritten values, and retains the latest value for any given key indefinitely.

R.Ranjan in [18] reviewed that Apache Samza has parallel-distributed architecture. It has a support for Java Virtual Machine(JVM) languages.

2.3.5 Apache Spark

Spark is an iterative distributed stream processing platform designed to process the data through functional transformation on distribution collections of Big Data. Spark is not another tool but it is a new concept. Uber, Amazon, Yahoo, Baidu, Pinterest, Spotify, Alibaba, and Shopify Verizon are some of the top industries where Spark is already being used in production.

It introduced a new programming and fundamental abstraction. It is better than map-reduce as it is an abstraction which wraps Map Reduce into single concept. It performs efficiently in solving interactive data mining problems. Distributed computing is almost in every field such as machine learning. Hadoop was designed for the transformation on data and detection and further moving on with another data. It ignores the historical data as it is not designed for the lookback computation.

Core Issues in Traditional Approaches:

- The way of solving interactive and iterative algorithms is in-efficient and these in-efficiencies is a result of not keep an enormous data in-memory.
- Keeping data in memory can improve efficiency for iterative algorithms as well as in interactive data mining.
- The idea of SPARK, especially RDD originated from this single thought i.e. figure out a way to keep the data in memory and when the data is so big figure out a way to keep data in distributed memory.

2.3.5.1 Distributed Shared Memory in Apache Spark

Hadoop has a good programming abstraction for accessing a clusters [16] computational resource but have a poor abstractions for leveraging distributed memory. Current distributed algorithms needs access to intermediate results of computation. They have to reuse intermediate results across multiple computation. Having distributed shared memory for intermediate results have its own issues as DSM are designed for fine grained updates to shared state which becomes bottleneck for algorithm that needs bulk update between computations.

Spark application has 2 parts:

1. Driver: main method
2. Executor: other tasks

Application in which main method is running is called driver application and the application derived from driver application is run on executor memory. For a single node application both the driver and executor memory will be allocated on the same node. If an application is run on cluster then the driver memory will be allocated on one system and the executor memory on other system.

2.3.5.2 Spark Ecosystem

Apache Spark ecosystem shown in Figure 2.4 is developed on top of an extensible API's supportive core execution engine in different languages. Spark framework is built on Scala, so programming in Scala for Spark can provide access to some of the greatest and latest features that might not be available in other supported programming spark languages. Spark is comparatively slower when used with programming language Python although it has a support for it. Spark also has a support of R language which is basically used for statistical operations. SparkR is another language component of Spark that enables usage of machine learning libraries designed for Spark. Data analysis libraries like Pandas and Sci-kit available in Python can also be used in Spark but is relatively slower than Scala.

1. **Spark Core**: Core is the establishment for parallel and distributed handling of huge datasets. Spark Core is a component that is responsible for all the essential I/O functionalities, monitoring and scheduling the tasks sent on clusters of spark, job dispatching, networking with various storage frameworks, efficient memory management and fault recovery.
2. **Spark SQL**: SQL developers write the query in SQL format which is converted into spark format and hence output is achieved for the unstructured data. It is quite analogous to hive but is faster in comparison to hive.
3. **Spark Streaming**: It enables all the analytical and interactive applications for the live streaming data.
4. **MLlib**: It is the library mostly for machine learning and infact interestingly it is completely replacing Mahout. All the core developers of Mahout have moved towards MLlib because of efficient and faster performance.
5. **GraphX**: To retrieve graphical structure [16] of the data to achieve optimal paths For example in Google Maps.
6. **Spark R**: It is basically the package provided for R which is an open tool which is mostly used by analyst and nowadays, spark committee mostly handles such analysis tasks.

Zhang et al. explained the in-memory cluster computing feature of spark [27]. The authors expressed the distributed memory management platform for spark. Cluster Computing is completely combination of the infrastructure as well as the software

platform. As software platform is responsible for distributing tasks among multiple nodes.

Zaharia et al. [17] proposed improvised programming model called discretized streams (D-Streams), that offers an abnormal state practical programming API, solid consistency, and productive fault recovery. D-Streams support another recovery component that enhances proficiency over the customary replication and upstream reinforcement arrangements in streaming databases: parallel recovery of lost state over the cluster. The authors prototyped D-Streams in an augmentation to the Spark cluster computing system called Spark Streaming, which lets clients consistently intermix streaming, batch and intuitive queries.

Cluster Manager in spark is used primarily for the distribution of resources across applications. It executes as an external service for fetching resources on the cluster. Spark has a support of pluggable cluster managers namely

- Standalone Cluster Manager
- Hadoop Yarn
- Mesos

Kozyrakis et al. [29] performed comparison between the three types of Cluster Managers is illustrated in Table 2.3

Table 2.3: Comparison of Cluster Managers- Standalone, Yarn and Mesos

Feature	Standalone	Yarn	Mesos
Recovery	It has a Zookeeper support for automatic recovery of Master node.	Manual recovery is guaranteed using command line utility.	It has a support of Zookeeper for automatic recovery of master
Security	The user needs to configure each nodes with the shared secret. The protocol required for encryption is SSL.	It contains security for, servicelevel authorization and authentication for Web consoles and data confidentiality.	This includes the slaves registering with the master, frameworks submitted to the cluster, and operators using endpoints such as HTTP endpoints.
User Interface	It has Web UI to view cluster and job statistics as well as detailed log output for each job.	It has a Web UI for the ResourceManager and the NodeManager	It facilitates various metrics for the master and slave nodes which are accessible via a URL.

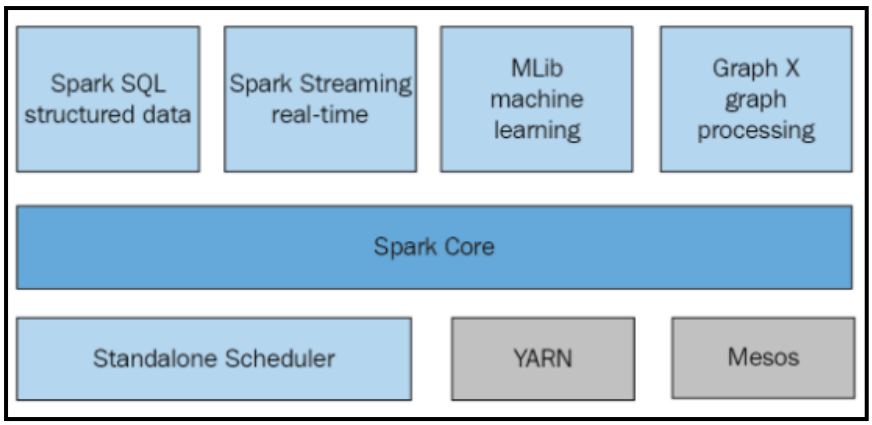


Figure 2.4: Components of Spark [17]

2.3.5.3 Resilient Distributed Datasets

Resilient distributed datasets [31] are known as the core piece od Spark. This is kind of a building block for all Spark and Spark Streaming programs.

Resilient means that it can be distributed and run on a cluster that is actually resilient to failure. If a program is run on a good cluster manager and one of the nodes goes down for some reason, then the RDDs can still survive. There is probably the back up of the data on some other machine that it can be recovered from. Distributed means that it can actually distribute the processing of RDD. Dataset is the massive dataset that can never fit on one machine.

When any Spark program is started, an object called SparkContext need to be set up. The SparkContext object is what that creates RDD objects.Similarly, in SparkStreaming a variant of SparkContext called StreamingContext is setup, which is a special Spark Context object that can actually deal with real-time data.

2.4 Twitter Stream Analytics with Spark

Spark has a package called Twitter.utils, which contains all the built-in functions to stream data from Twitter. Using a subset of the endless twitter's stream looks as the perfect choice since it has everything we need, an endless and continuous datasource (streams) ready to be explored. The parameters that Streaming context take are; the application configuration and the streaming time. As Spark streams data in micro batches, it is required to set some time so that for every set time (time_set), be it seconds or milliseconds, it will stream the data. It is flexible to know what is trending on Twitter at any precise moment. It is just matter of counting the appearances of each tag on the stream.

The identification of trending topics on Twitter helps in targeting and increase audience.Consequently, there is a possibility of accessing the different data sets using a single set of tools such as SQL. Solving real life problems in a easy way is what most people actually want and Spark once again proves to achieve this.

Table 2.4: Literature Survey

S.No	Author(s)	Title	Year	Technique(s) Used
1.	M. Kholghi and M. Keyvanpour [1]	An analytical framework for data stream mining techniques based on challenges and requirements	2015	Storm as distributed computing technique.
2.	D. Surekha, G. Swamy and S. VenkatramaphaniKumar [2]	Real time streaming data storage and processing using storm and analytics with Hive	2016	Hybrid of Storm with Hive Analytics and MongoDB for storage
3.	D. Jayanthi and G. Sumathi [3]	A Framework for Real-time Streaming Analytics using Machine Learning Approach	2016	Hybrid framework of Apache Spark with result presentation on SpagoBI
4.	J. Samosir, M. Indrawan and P. Haghighi [7]	An architecture for context-aware adaptive data stream mining	2012	Evaluation elements for processing of data streams
5.	Gaber, M. Krishnaswamy and S. Zaslavsky[5]	An Evaluation of Data Stream Processing Systems for Data Driven Applications	2016	Amazon EC2 with Apache Flume and Kafka

Chapter 3 RESEARCH PROBLEM

3.1 Problem Statement

Nowadays, another class of data-comprehensive applications has turned out to be widely perceived: applications in which the data is modeled best not as relentless relations but instead as momentary data streams. Instances of such applications incorporate real-time network monitoring, financial data monitoring, intrusion detection systems, telecommunications, information management, web clickstream analysis, manufacturing, sensor systems, and many more. Individual data items in the data stream model may be relational tuples, e.g., network measurements, web page visits, call records, sensor readings, and so on. However, their uninterrupted emergence in various, high-speed, time-differing, possibly unpredictable and absolute streams happen to result in some fundamentally advanced research problems. In all of the applications refered to above, it isn't attainable to just load the arriving information into a conventional database management system (DBMS) and work on it there. Conventional DBMS's are not intended for fast and consistent stacking of individual data items, and they don't straightforwardly support the constant queries that are common to streaming data applications. Moreover, it is perceived that both adaptivity and approximation are key components in performing other processing (e.g., data analysis and mining) and executing queries over swift data streams, while conventional DBMS's focus mostly against the goal of accurate answers computed by steady query plans. The open challenges cover the the full topic of knowledge discovery and include such issues as: dealing with legacy systems, protecting data privacy, analysis of complex data, handling incomplete and delayed information, and evaluation of stream mining algorithms. The resulting analysis is illustrated by practical applications such as Twitter Stream Analysis, acquiring top trending hashtags, clustering tweets using K-Means Clustering algorithm and provides general suggestions concerning lines of future research in streaming data processing. In this research, a general-purpose platform for ingesting, processing, analysis, storage and visualization of streaming data along with machine learning in real-time is studied. However, it has also been attempted to provide a broad overview of the area, along with its related and current work.

3.2 Research Gaps

There are several challenges when processing of streaming data comes in focus. As the data is captured, processed in real-time there are challenges [5] that are categorized as follows:

1. **Business and Data Understanding:** The broad purpose of handling data streams involves making business decisions in real-time. Analysis and Visualisation of data can be achieved using various available platforms as discussed earlier. The research still lacks protecting privacy and confidentiality of data that can be replicated on open-source platforms while being processed.
2. **Data Preparation:** Data stream management is another broad category because it involves big data processing, transformation, query analysis and storage problems. Although various rapid processing and data availability platforms such as Apache Spark, Flink are made available still such systems need to be tested under various conditions for some critical applications such as Fraud Detection, Intrusion detection in web applications and many more.
3. **Modelling Stream Queries:** There has been a massive amount of work done data streams using SQL-like sematics to provide the definition of stream queries.Babock et al. [28] used consistent SQL for the purpose of streaming the queries and elaborated the concept of sliding windows [27] .However, there are challenges in modelling the stream queries such as data dependency and correlation among streams in heterogeneous data sources.
4. **Evaluation of Stream Mining Algorithms:** The main issue in methods of evaluations while learning from dynamic and ever-changing data streams consists of supervising the evolution of the learning process. Fault tolerance, resource management and other such factors should be considered while focusing on performance criteria. Memory is one of the most crucial restriction as learning algorithms execute in static memory. Therefore the need for the evaluation arises because of usage of memory over time, and the impact in accuracy when using the available memory.

3.3 Research Objectives

The accompanying objectives have been planned towards the previously mentioned research gaps:

1. To study various techniques and tools available for streaming data analysis.

2. To analyze twitter streams 10x faster on disk and 100x faster in memory using open-source distributed stream processing platform Apache Spark using iterative approach for the computation of data available in continuous streams.
3. Stream processing application development, using interactive operators, shell and writing optimized code in few lines using Apache Spark with Scala development support.
4. Providing deployment flexibility with various options to deploy applications on various cluster managers such as Mesos, YARN, Standalone, and Local.
5. To provide the storage flexibility for the real-time data streams using integration of HDFS, Amazon S3, Cassandra, etc.
6. To provide Unified Stack i.e. building applications combining different processing models such as Batch, Streaming and Interactive Analytics.

3.4 Research Methodology

In this research work, Scala and Spark shell scripting will be used. Scala is a very popular choice for basic programming and Spark Streaming because Spark itself is written in Scala and it is a lot easier to use than Java. It runs on top of Java Virtual Machine(JVM) and hence can get access to Java classes. It is built on the concepts of functional programming.

The modules for analyzing the twitter streams for various applications such as popular hashtags, clustering similar tweets, saving the tweets in local filesystem are implemented using scala scripting language in Scala IDE extension of Eclipse for the ease of execution. Live tweets are streamed by setting up a Twitter developer account to get authorization access tokens to programmatically receive live twitter updates as they happen. This is achieved by logging in to https://apps.twitter.com. After logging in to account set up an application by filling up the required information like name, description, website etc.

Once an application is setup the keys and access tokens will be generated. There are four types of keys and access tokens which are required for the authorization purpose

to use TwitterAPI. These are Consumer Key(API Key), Consumer Secret(API Secret), Access Token and Access Token Secret. Once the credentials are set up they need to be used in the code to get access to live twitter feeds. Various modules are built once this step is successfully completed. The twitter streams are stored on local file system to carry out the further analysis and also this can be integrated with NoSQL database Cassandra. The machine learning library of Apache Spark known as SparkML is used for implementing Collaborative Filtering technique for the application of Movie Recommendation System to study various operations and use cases of machine learning in distributed environment.

This is achieved by setting up spark-shell on Cloudera Virtual Box environment to test the functionality of various operations.To achieve this various extensions of Spark such as Simple Build Tool(SBT) is used which is an open-source tool for Java and Scala projects, similar to Java's Ant and Maven. Its primary highlights are: Native support for compiling Scala code and coordinating with numerous Scala test systems. The other extensions explored for achieving this is spark-submit shell script which facilitaes to manage Spark applications. The visualization of results is achieved on spark-shell in Cloudera and Scala IDE integration of Eclipse.

Chapter 4 SETTING UP SPARK FOR TWITTER

Apache Spark is an open source, rapid and multi-purpose system for cluster computing. It is primarily built at the University of California, Berkeley's AMPLab and the codebase of Spark was later contributed to the Apache Software Foundation, where it is still maintained. Spark has introduced an coalition for programming whole clusters with inbuilt information parallelism and fault-tolerance. An extension of the core Spark API is Spark Streaming [33] which facilitates adaptable, highthroughput, fault-tolerant processing of data streams in real-time. Data can be fetched using diverse sources like Flume, Kinesis, TCP sockets, or Kafka, and can be managed using composite algorithms stated with high-level functions like map, reduce, window and join. Conclusively, processed data can be saved to file systems, databases, and further pushed to live dashboards. In fact, Spark's other components such as SparkML which consists of machine learning and GraphX consisting of graph processing algorithms can be applied on streams of data. Additionally, it facilitates a distinguished abstraction called discretized stream or DStream, that states a continuously moving stream of data. DStreams can be generated either from input data streams from multiple sources which are Flume, Kafka, and Kinesis, or by implementing foremost operations on other DStreams. Internally, a sequence of RDDs is named as DStreams.

4.1 Installing Spark

- As Spark is built using Scala programming language and also Scala runs on the top of JVM for the compilation and execution purpose, therefore in order to get started with Spark, Java installation is the primary step. Java Development Kit(JDK) can be installed from oracle.com followed by setting up of couple of environment variables in Windows operating system.
- The secondary step is to install Spark which can be downloaded from spark.apache.org shown in Figure 4.1. The Spark release 2.0 or latest is downloaded so that all the latest libraries are installed and configured.The package type to be installed should be Pre-built for Hadoop 2.7 or later

Figure 4.1: Spark Download Website

- In order to run Hadoop cluster on Windows there has to be pseudo Hadoop distribution file present known as winutils.exe which can be downloaded from any external source of internet which makes sure to Spark that Hadoop is present.
- The environment variables are set up after downloading Spark, Java and Hadoop to make sure the application developed finds the path to the executable files.
- Next comes the installation of Scala-IDE which can be downloaded from scala-ide.org which gives eclipse.exe file which has scala inbuilt in it. The spark installation can be verified by writing command spark-shell in command prompt run as administrator as shown in Figure 4.2.

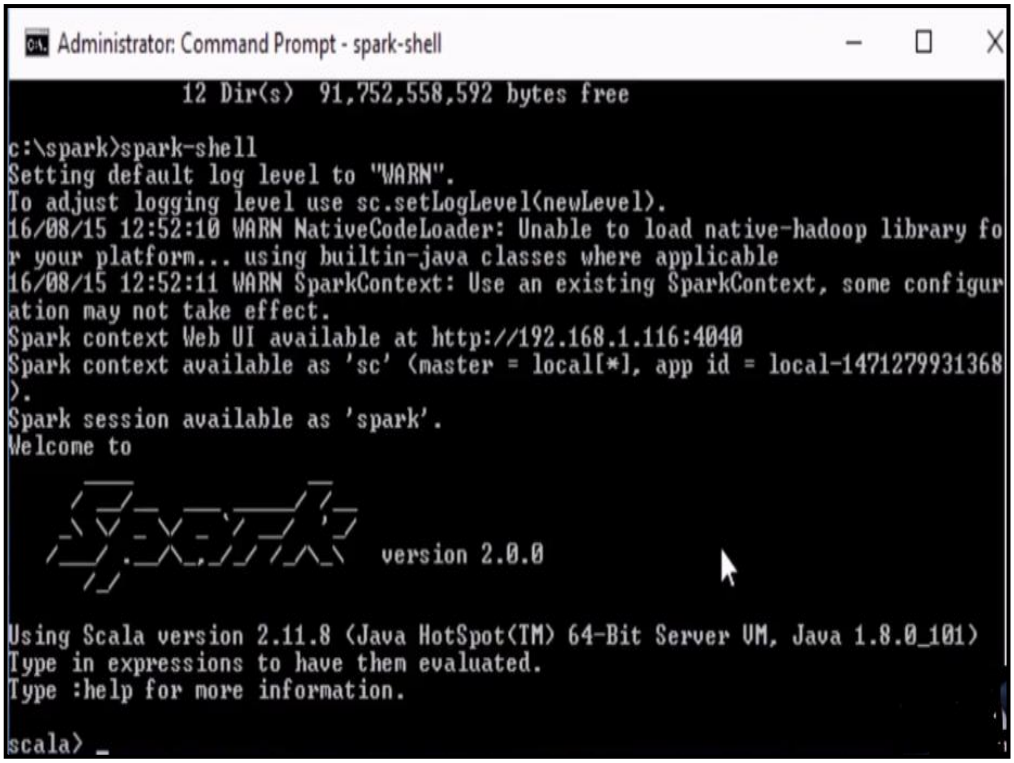


Figure 4.2: Spark-Shell

4.2 Stream Live Tweets with Spark Streaming

- Initially create/ sign in to twitter developer account and create an application with any suitable name providing description and website for the application. The values in these fields can be anything which are just casually filled in. The OAuth twitter key and access tokens are general after the creation of the application which are unique and are used in the spark streaming application for fetching the tweets using Twitter API.
- After writing the code for printing the live streams of tweets there need to be some configuration settings done which are to add external jars. The external jars are present in folder spark where all the configuration, libraries and jar files are present. Select them all and add them in the project.
- The important jars that are required for specifically Twitter applications are twitter4j-core-4.0.4.jar, twitter4j-stream-4.0.4.jar and dstream-twitter_2.11-0.1.0-SNAPSHOT.jar .
- After all the configuration settings, an application module PrintTweets.scala is run locally using all CPU cores and one-second batches of data which is illustrated in Figure 4.3.

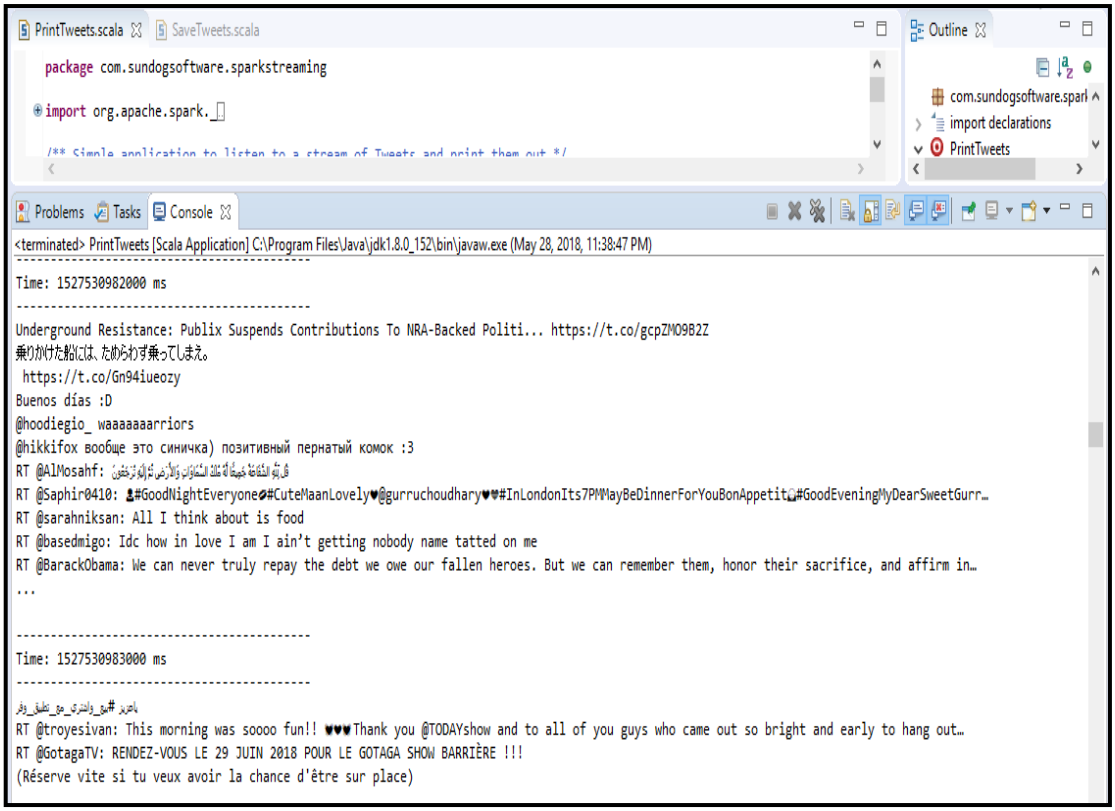


Figure 4.3: Module for printing Tweets using Scala IDE

The basic step is the setting up Twitter API credentials and then saving them in twitter.txt which is used by another module named as Utilities.scala. The streaming context is setup which creates objects that are used in Spark Streaming. The twitter stream is created using function TwitterUtils.createStream(). After this, as each batch comes out tweets are printed and the first ten results from each batch are displayed for the visualization to be user friendly.

4.3 Saving Tweets to Disk

This section covers saving the tweets to persistent storage instead of printing them out in the output area of Scala-IDE. Here the storage is local file system but it can be extended to dumping out data to database or distributed file system or whatever is the output target. It is desirable to get number of tweets < 1000 as we don't want to fill up the hard disk if kept running forever accidentally. Therefore, the restriction is imposed to on the number of tweets.

Each individual RDD is accessed in DStream as it comes in. To achieve this, foreachRDD function is used which allows extraction of both individual RDD from

the DStream and a time stamp. The timestamp is used here for saving the tweets with file name as time stamp. To achieve the above mentioned couple of steps are:

- Repartioning of RDD into a single RDD which actually consolidates the RDD down into single partition. It might be distributed across the cluster but here the access to everything at once is required and that is specially crucial when dumping the data streams into database is required because if one creates a database connection to output the results and the data is actually distributed on multiple machines then the database connection may not be valid on all those different machines. Therefore, firstly it is required to consolidate everything back together into a single partition.
- The cache is called in the application because whenever there is more than one action on RDD it is desirable to cache it. The reason being that every time Spark sees an action it will compute the directed acyclic graph to figure out an optimal way to get the data which is desirable.
- The module is saved with the name SaveTweets.scala and the result obtained in obtained in batches of 1 second in the console illustrated in Figure 4.4 and the tweets saved on local disk are illustrated in Figure 4.5

  .

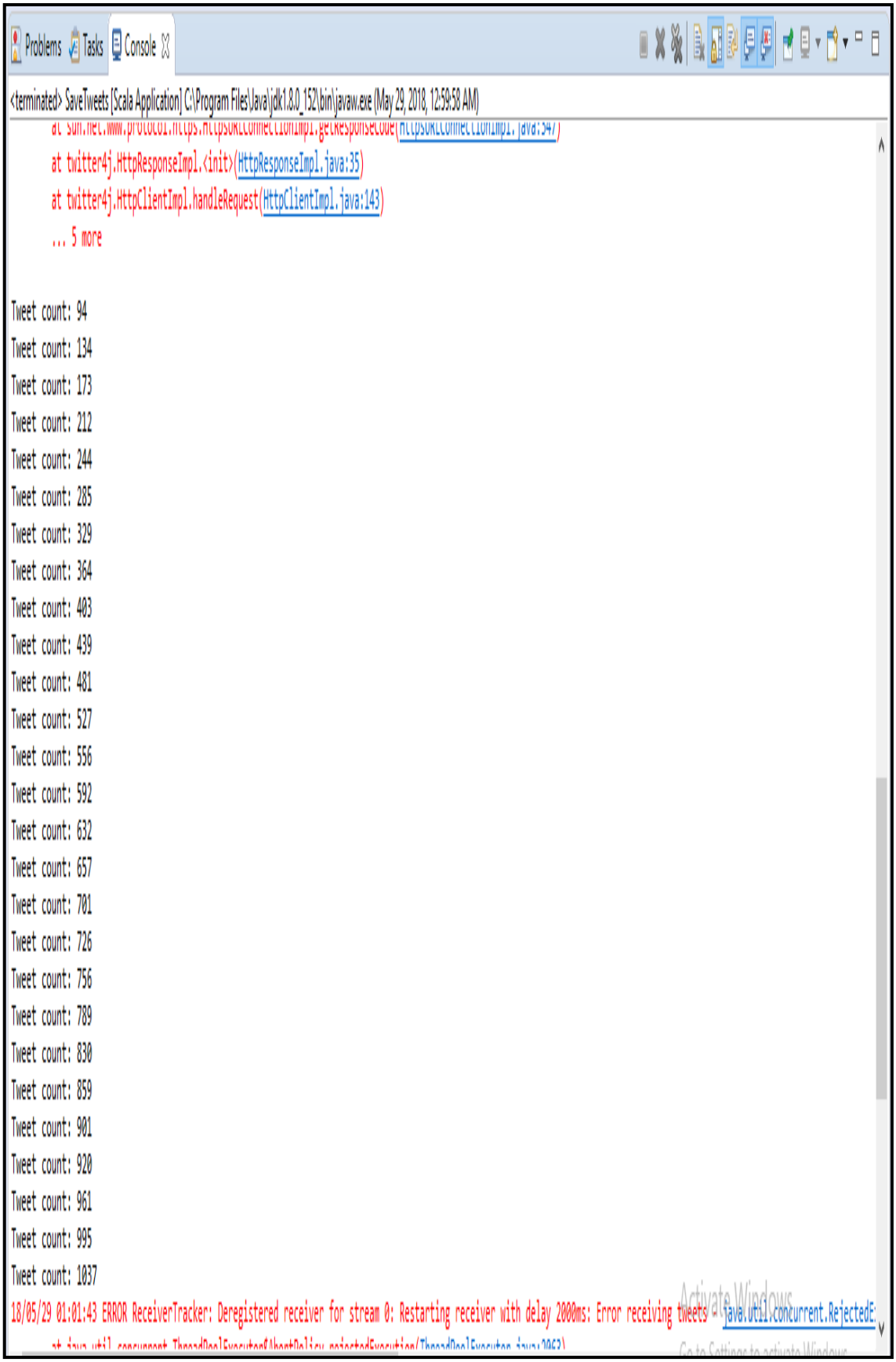


Figure 4.4: SaveTweets module counting tweets per batch

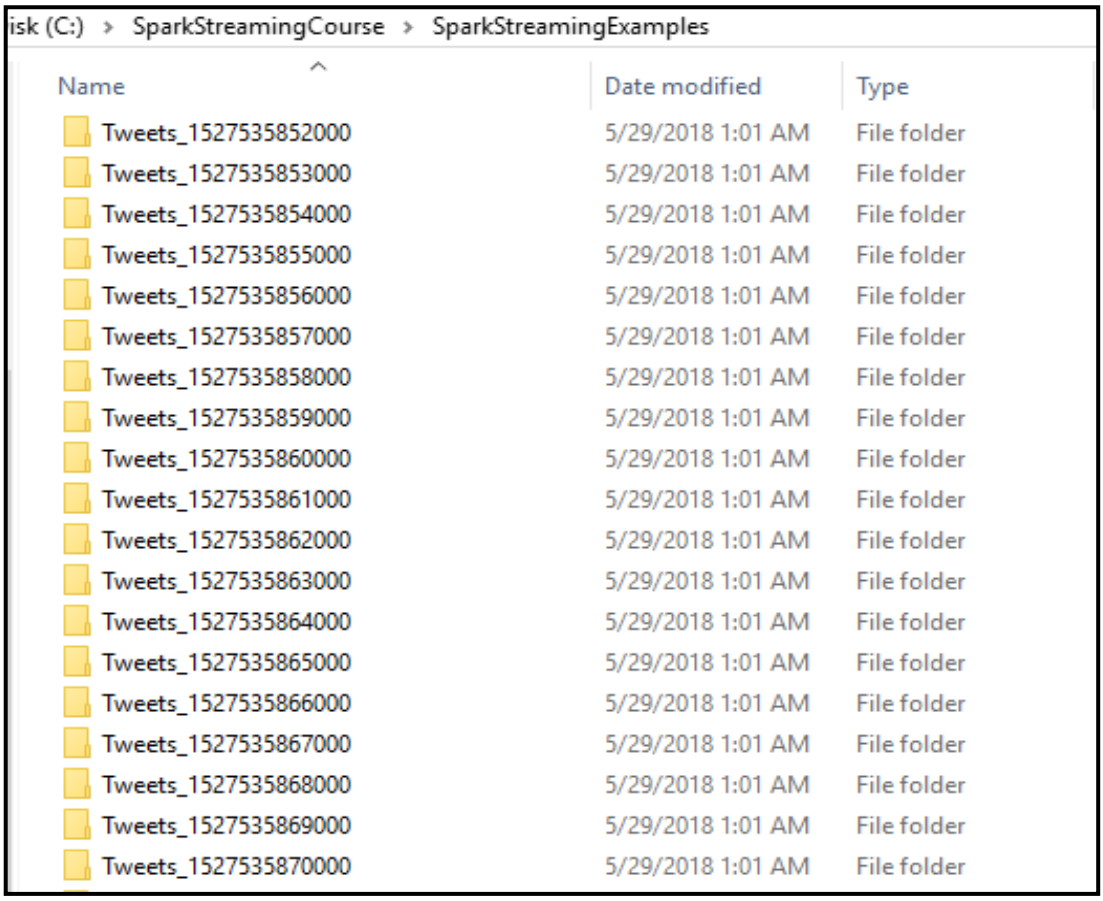


Figure 4.5: Tweets saved on Local Directory

- In the local disk the directories are created with the name as timestamp and inside each directory a file with the name part-00000 as shown in Figure 4.6 is created in which the tweets are present illustrated in Figure 4.7.

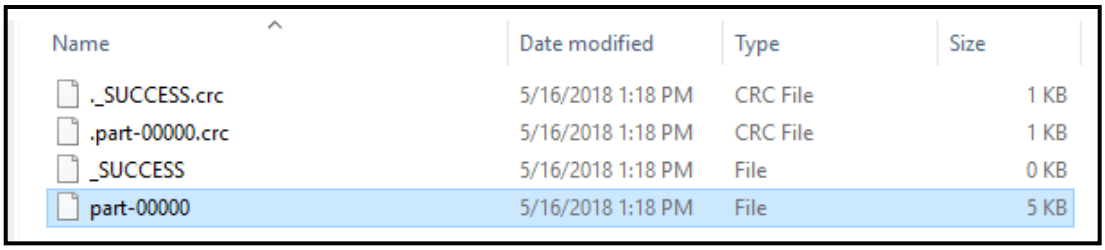


Figure 4.6: Partition part-00000 created where tweets are captured

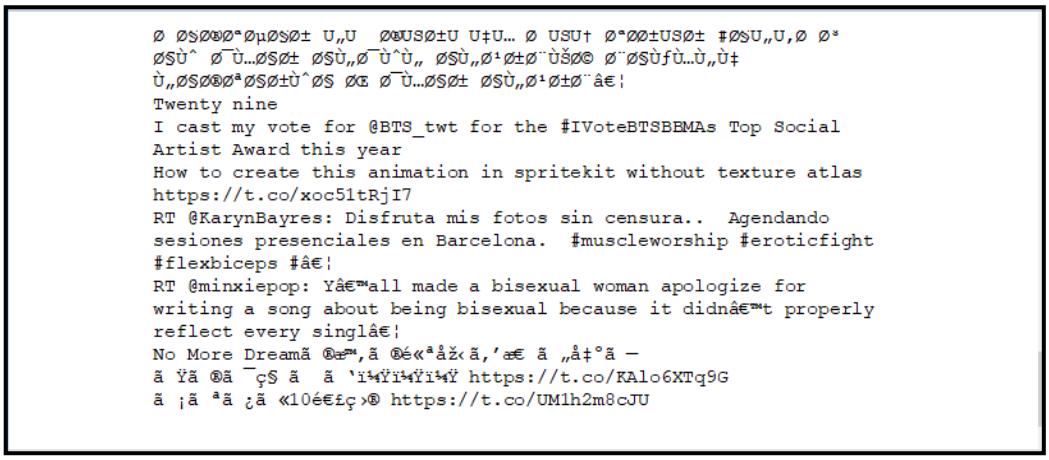


Figure 4.7:Tweets contained in part-0000 file

4.4 Tracking the average tweet length

One of the further set of manipulations that can be performed on the twitter streams are tracking the average tweet length. The requirement for achieving this manipulation is accomplished after establishing the connection with database. The another case can be by computing the aggregate data of the tweets as they come in. This is accomplished using a thread-safe counter within the driver script so that the track of things can be kept safely as the processing happens. Another pre-requisite to achieve this is more complex DStream transformations.

Steps involved in achieving this are:

1. Set up the Twitter credentials using utilities function that was used earlier to print the tweets and verify the OAuth credentials of Twitter.
2. Set up a StreamingContext to run it locally using all the cores of CPU and the batch size is set as 1 second.
3. Extract the text only from each tweet that comes in that is stored in DStream.
4. Extract the length of all the tweets which is computed over time.
5. A new DStream is created which is named lengths which maps the status Dstream using function status.map(status=> status.length()) which transforms the status string which comes in from the status DStream into a length integer. This is just the length of each status tweet and the number of characters.

6. The resultant new DStream which is obtained after the transformation of status DStream from its actual tweets is just the length of each tweet.
7. The two things that are tracked are the total number of tweets and the total number of characters. As new data comes in on lengths DStream it is aggregated with the previous data and each RDD is accessed individually.
8. In this module, time stamp is ignored and just the RDD is accessed and the information contained in them is counted.
9. The resultant console output is illustrated in Figure 4.8 and AverageLengthTweet Application on Spark UI in Figure 4.9

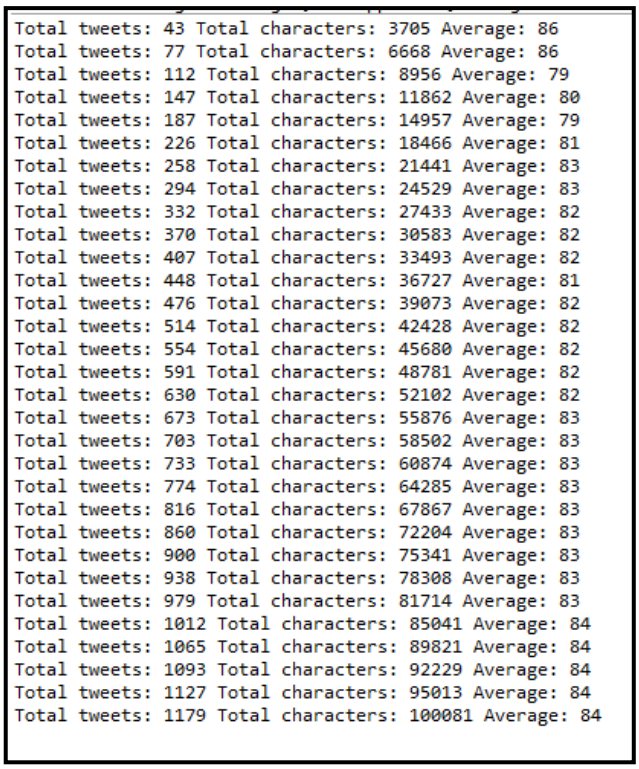


Figure 4.8: Average Tweet Length Module

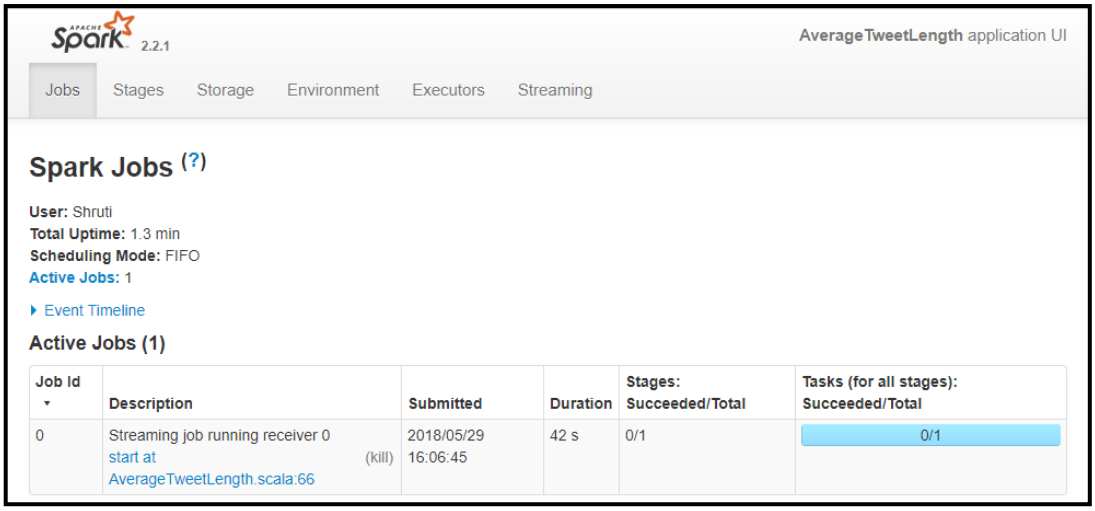


Figure 4.9: Average Tweet Length Application on Spark UI

4.5 Finding out the most popular Hashtag

In this module, the most popular hashtag is tracked from a stream of tweets.It is done over a sliding window of 5-minutes. Here the use of filter operation is introduced on DStream. FlapMap is used instead of Map operation. The steps required to achieve this are:

1. Get a Twitter Stream and extract just the messages themselves.
2. Split up the tweets into individual words.by using FlatMap operation. Map has a one to one relationship where every row of one DStream in transformed into another row of another DStream. But with FlatMap we can actually have the different number of output results than the input. Since an entry for every word in a row is required, there will be many more entries in the resulting DStream.
3. Eliminate everything that is not hashtag by using filter() function as

   val hashtags=tweetwords.filter(word=>word.startsWith('#'))
4. Convert individual RDDs of hashtags to key/value pairs where instead of the single hashtag element in each row, there is bound to be a tuple which is a key and a value where key is the hashtag and the value is going to be the number one. A reduce() operation is used next to count up everything. It will combine together all the values for a given key and use whatever operation is needed to combine them together.

   val hashtagKeyValues=hastags.map(hashtag=>(hashtag,1))

5. The thing that is required is not only to reduce the strings but also reduce it over a sliding window of time. In this thesis there is a look back over the past five minutes otherwise the stuff will pile forever. For this purpose, reduceByKeyAndWindow function is used on hashtagKeyValues RDD. This helps in providing a function not only in how to combine together individual values in a given key but also as an optimization [6] we can also provide it with a function for removing something from it which is a more efficient way to do this.
6. Finally, the requirement is to sort and output the results illustrated in Figure 4.10.

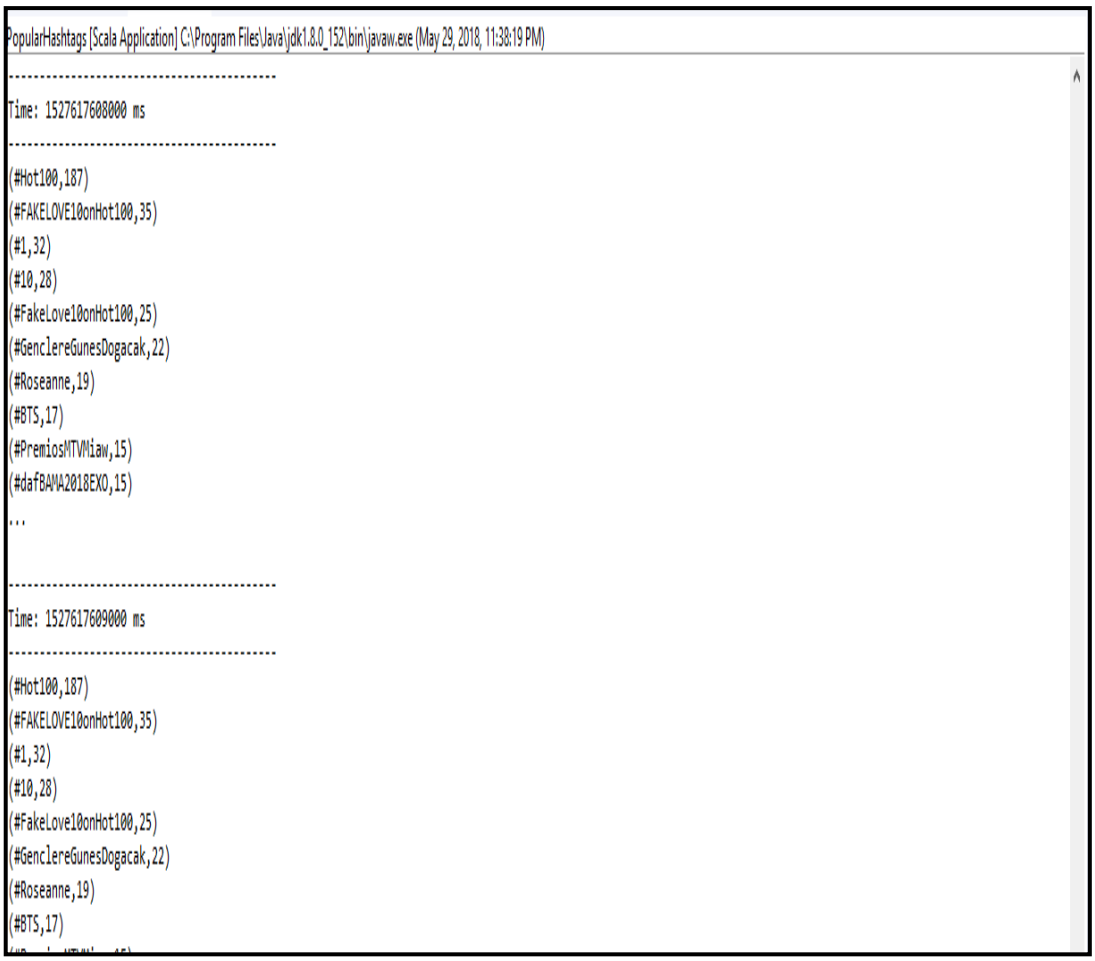


Figure 4.10: Popular Hashtags Module Output

Chapter 5 INTEGRATION OF SPARK WITH OTHER SYSTEMS

5.1 Integration with Apache Kafka

This section covers integrating spark streaming with some real world systems that might be required to consume data from. In the real-world, it is seen that probably listening to data sources that are coming from some other system and one such popular data source is called Apache Kakfa. Kafka is a high-throughput distributed messaging system which is very scalable, durable, distributed and robust. It is a publish/ subscribe messaging system where it can publish out messages received from a bunch of different message producers and consumers of those messages can subscribe to given topics to actually listen to those messages. For example: Broadcasting weblogs that are dropped into Apache access log directories. It is reliable way to send data over a cluster of computers and Kafka acts as a broker between Producers and Consumers. As of Spark 1.3, Spark Streaming can connect directly to Kafka. Earlier, it used to be that it had to connect to the Zookeeper hosts that actually sat on the top of Kafka and there were lot of room for things to go wrong and for messages to get lost. But today, Spark Streaming can connect directly to the Kafka server. It is much more reliable mechanism and there is just one less person in the middle. Spark-Streaming-Kafka package is not built-in and it is needed to be downloaded from Maven. It is to be copied to Spark Lib folder and then further needed to be imported as an additional external jar.

This is achieved for looking at the top URLs from the apache access log which is a very important click-stream application. Related work is done by listening to TCP port for the streaming data. In this thesis, the log data is accessed through Kafka. The steps involved to achieve this are:

1. Import the two classes org.apache.spark.streaming.kafka._ and kafka.serializer.StringDecoder.
2. Create a StreamingContext with 1 second batch size
   val ssc= new StreamingContext(“local[*]”, “KafkaExample”, Seconds(1))
3. Construct a regex to extract fields from raw Apache log lines

val pattern= apacheLogPattern()

4. Set up a map of Kafka broker hosts and where they can be found i.e. hostname:port for Kafka brokers and not Zookeeper shown in Figure 5.1.
   val kafkaParams= Map("metadata.broker.list"-> "localhost:9092")
5. Create the list of topics that is required to be listened.
   val topics= List("testLogs").toSet
6. Create a Kafka stream which will contain(topic, message) pairs. The map(_._2) is tacked at the end in order to only get the messages which contain individual lines of data.
7. Extract the request field from each log line.
8. Extract the URL from the request.
9. Reduce by URL over a 5-minute window sliding every second.
10. Sort and print the results shown in Figure 5.2.

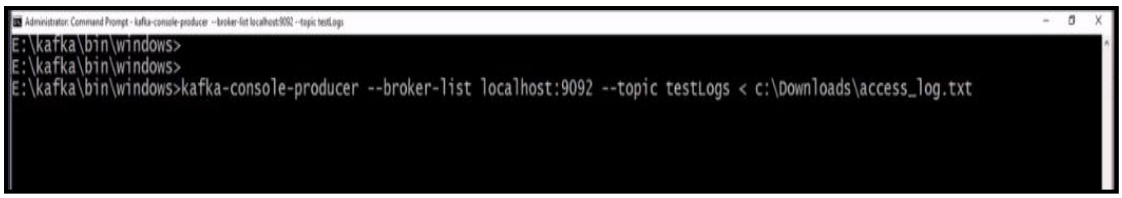


Figure 5.1: Kafka server initialisation

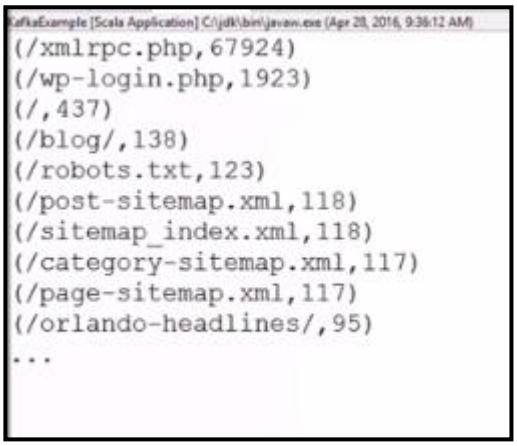


Figure 5.2: Clickstream Analysis Module for Kafka integration

5.2 Integrating with Apache Flume

Apache Flume can be thought of as similar to Kafka but it is very specifically tailored for log data. So for web logs, it is very tightly coupled with Hadoop ecosystem. So it builds itself as a distributed, reliable and available service just like Kafka for efficiently collecting, aggregating and moving large amounts of log data. It's high level architecture is little bit like a bunch of web servers in a cloud which

provide a source to the Flume which has a channel which basically feeds information to Sinks which listen to it. For example, there might be a sink that is connected to HDFS which gets a bunch of log data pushed to it from the channel and then the sink stores it to the distributed file system. There are a couple of ways of setting up Flume [35] with Spark Streaming which are :

1. **Push based approach:** In this approach, Spark receiver is set up as Spark Streaming receiver as another sink. So, the channel for a Flume keeps pushing data and Spark Streaming can consume it.
2. **Pull based approach:** In this approach, instead of waiting for Flume to push data to Spark Streaming, one can install a special custom sink into Flume that actually pulls data from flume and if Spark Streaming cluster goes down the special sink can be asked for resending of information. This approach is more robust for dealing with failures on Spark Streaming cluster.

5.3 Integrating with Apache Cassandra

Cassandra is a very popular output source choice for integration of Spark Streaming. Cassandra is basically a NoSQL database [34] which is based on key-value data. It is a distributed, fast and reliable database. The tables are created with specific queries in mind to keep things really fast. If a data is sent for a given key to more than one host then there is a condition for redundancy to be there which Cassandra takes care of. So, in Spark Streaming one can transform raw input into key value RDD where the first element of the tuple is the key and the second element is the value. This becomes very efficient with Cassandra because if there is a key-value RDD which Spark can easily understand. This can be achieved by the following:

- Getting the spark-Cassandra-connector package from DataStacks which is a .jar file which is further added to the project and then Spark is to be configured with it.
- Set spark.cassandra.connection.host in SparkConf to address of Cassandra host.
- Use rdd.saveToCassandra to store tuples into named columns in a specific Cassandra table.
- Import Cassandra package by using:

import com.datastax.spark.connector._

- Set up the Cassandra host address as:

  conf.set("spark.cassandra.connection.host", "127.0.0.1")

- Set up a master as local using command:

  Conf.setMaster("local[*]")

- Create the context with a 10 second batch size

  val ssc= new StreamingContext(conf, seconds(10))

- Create a socket stream to read log data published via netcat on port 9999 locally.

  val lines= ssc.socketTextStream("127.0.0.1", 9999,
  StorageLevel.MEMORY_AND_DISK_SER)

- Extract the (IP, URL, status, useragent) tuples that match the schema in Cassandra.
- Extraction of information from real-life access log and store it in real Cassandra database using command:

  rdd.saveToCassandra("shruti", "LogTest", SomeColumns("IP", "URL", "Status", "UserAgent")

- To run the script firstly the server needs to be turned on which is done using the command in Figure 5.3 which broadcasts the access logs on port 9999.

Figure 5.3: Broadcasting Access Log using ncat

- The results output of the script in the Cassandra console are shown in Figure 5.4

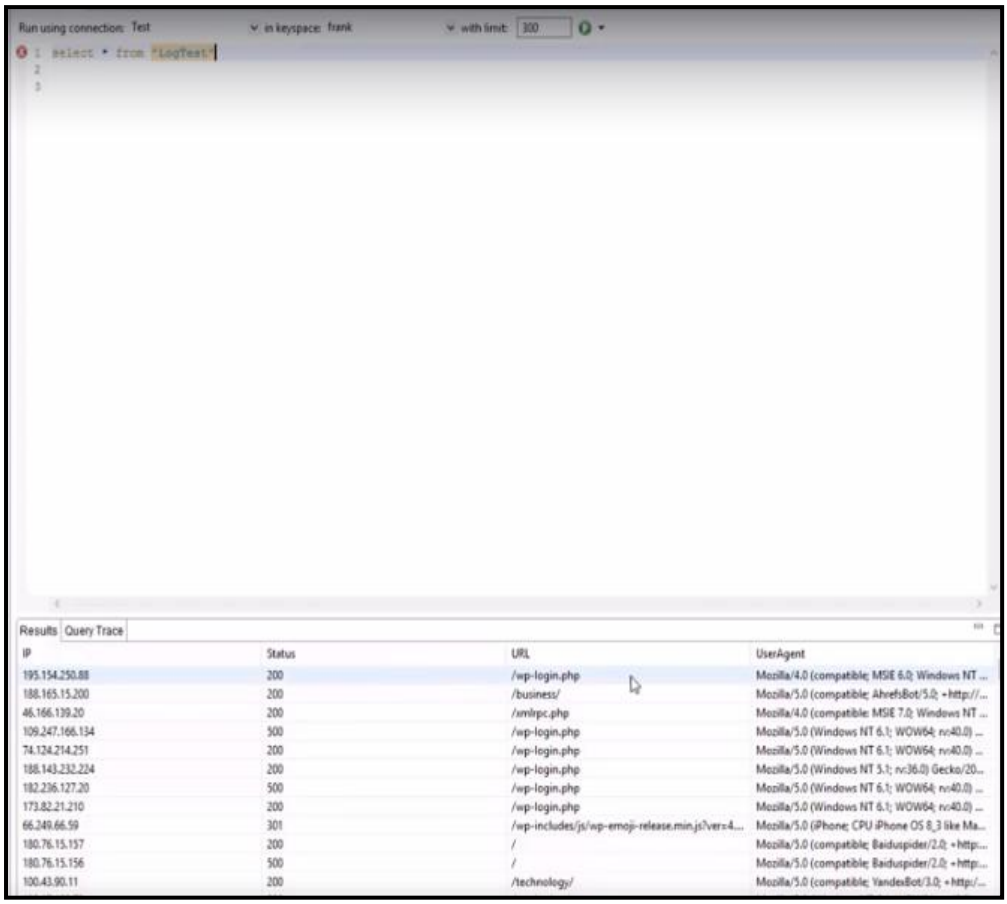


Figure 5.4: Cassandra Database integration with Apache Spark

Chapter 6 CLUSTERING OF STREAMING DATA

This section covers the concept of combining Streaming with Machine Learning using the MLLib component of Spark. This section is primarily focused on an important algorithm used for Clustering which is K-Means Clustering. The potential of actually updating a machine learning model in real-time as new data is received is quite interesting. It explains how the machine learning model behaves in real-time just like a human brain does given more and more input.

6.1 K-Means Clustering Algorithm

The clustering algorithm [30] attempts to split data into K groups that are closest to K –centroids. For example, there is some data that is broken into X-axis and Y-axis. With K-Means clustering it can be assumed that there are 3 distinct neighbourhoods Figure 6.1 and the value of K=3 in this example and it is desirable to cluster the millionaires living in the area into these clusters in the most optimal way. So that whenever a new millionaire appears he/she would be clustered appropriately. This algorithms works on the principle of unsupervised learning which uses the positions of each data point.

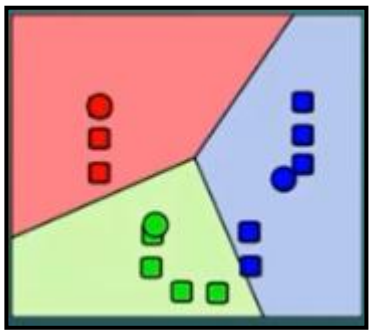


Figure 6.1:Clustering Phenomenon

6.1.1 Working of K-Means clustering

Let X = {x1,x2,x3,……..,xn} be the set of data points and V = {v1,v2,…….,vc} be the set of centers.

1) Randomly select ‘c’ cluster centers.

2) Calculate the distance between each data point and cluster centers.

3) Assign the data point to the cluster center whose distance from the cluster center is minimum of all the cluster centers..

4) Recalculate the new cluster center using equation 1:

$$v_i = (1/c_i) \sum_{j=1}^{c_i} x_i$$

Equation 1: Calculating Cluster Centroid

where, 'ci' represents the number of data points in ith cluster.

5) Recalculate the distance between each data point and new obtained cluster centers.

6) If no data point was reassigned then stop, otherwise repeat from step 3).

In order to predict the cluster for new points, just find the centroid they are closest to shown in Figure 6.2.

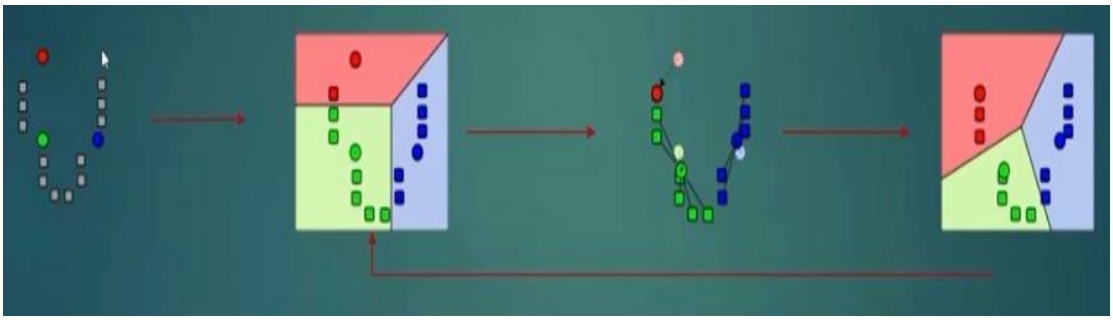


Figure 6.2: Working of K- Means Clustering Algorithm

6.1.2 Creating a Streaming K-Means Clustering Model with Spark MLLib

MLLib has a general concept of a model where K-Means algorithm is a machine learning model [37] and models need to be trained and tested for new data points that come in.

To Train the model, keep feeding it a new data as it comes in.It then updates the centroids and clusters and the data is fed as Vector objects which are just the raw information and are called feature data. model.trainOn() is the function used in Spark MLLib.

To actually get the prediction from the model that what cluster should a new data point belong to, testing is required. It is done by feeding it a labeled point objects for which the predictions are required. Labeled points consists of a label(the cluster that

is assigned) and the features (the data on which label assignments are based on). So, this contains both the feature information that contains the data on which the prediction is based and a label. To achieve this model.predictOnValues() is used.

The code snippet for building K-Means Clustering for Streaming Data is shown below:

```
val model= new StreamingKMeans()
.setK(5)
.setDecayFactor(1.0)
.setRandomCenters(2, 0.0)
```

- Set the value of K(number of clusters)
- A decay factor(so that old data doesn't pile up forever)
- Random Centres means how many dimensions that are worked upon.

There are 3 files in this experiment with the name kmeans-test.txt, kmeans-train.txt and StreamingKMeans.scala

The kmeans-train.txt Figure 6.3 file is streamed continuosly on a socket which are generating randomly the clusters mapping incomes to ages. The first column is the individual's income followed by their age in years. In MLLib terminology, each row is a vector that contains two features (income and age).

The kmeans-test.txt file Figure 6.3 also contains the same kind of data but just that the ClusterID is assigned while generating the data.

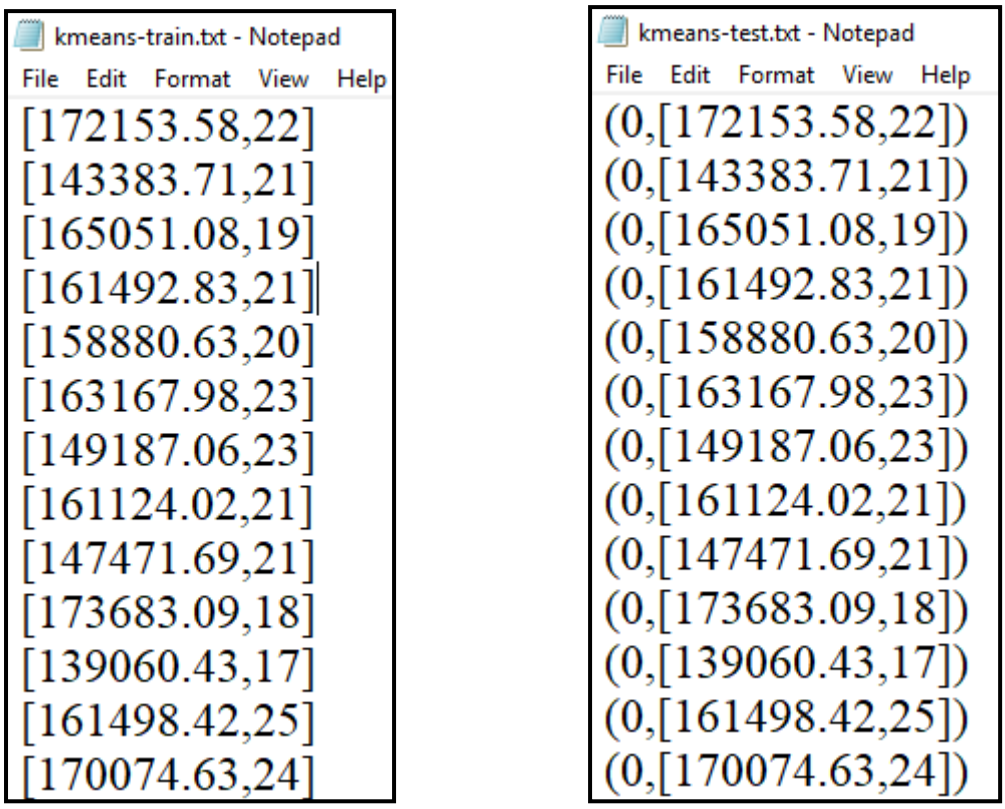


Figure 6.3: K-Means Training and Testing Dataset

The kmeans-training data is sent through server port number 9999 using the below command shown in Figure 6.4.

```
c:\Downloads>ncat -kl 9999 < kmeans-train.txt
```

Figure 6.4: Command to broadcast kmeans-training dataset

The kmeans-training data is sent through server port number 9999 using the below command shown in Figure 6.5

```
c:\Downloads>ncat -kl 7777 < kmeans-test.txt
```

Figure 6.5: Command to broadcast kmeans-testing dataset

The StreamingKMeans.scala script is executed and the output received in the console is shown in Figure 6.6

Figure 6.6: K-Means Clustering Output

Chapter 7 RESULTS AND DISCUSSIONS

7.1 Similarities and Differences between Apache Stream Engines

Table 7.1: Important properties of Apache Stream Engines

	Spark	Storm	Flink	Kafka	Samza
Category	Event Stream Processing	Event Stream Processing/ Complex Event Processing	Event Stream Processing/Complex Event Processing	Event Stream Processing	Event Stream Processing
Streaming API	HDFS, DStream	Spout, Tuple	Data Stream	KafkaStream	Message
Cluster Management	Mesos, Yarn and Standalone	Mesos, Yarn	Yarn, Tez and Standalone	Any	Yarn
Process Model	Micro-batch	Event	Event, Micro-batch	Event	Event
Latency	Medium	Very low	Low	Low	Low
Throughput	High	Low	High	Medium	High
Fault-Tolerance	Yes	Yes	Yes	Yes	Yes
Language Support	Scala, Python, Java	Any JVM	Scala, Python, Java	Java	Scala/Java
Delivery Guarantee	Exactly Once	Atleast Once	Exactly Once	Atleast Once	Atleast Once
Windowing	Temporal	Temporal Count-based	Temporal Count-based	Temporal	Temporal

7.2 Streaming Statistics for PrintTweets Application

The application for printing the tweets in real time using Apache Spark was executed and the result obtained on Scala IDE console is illustrated in Figure 7.1.

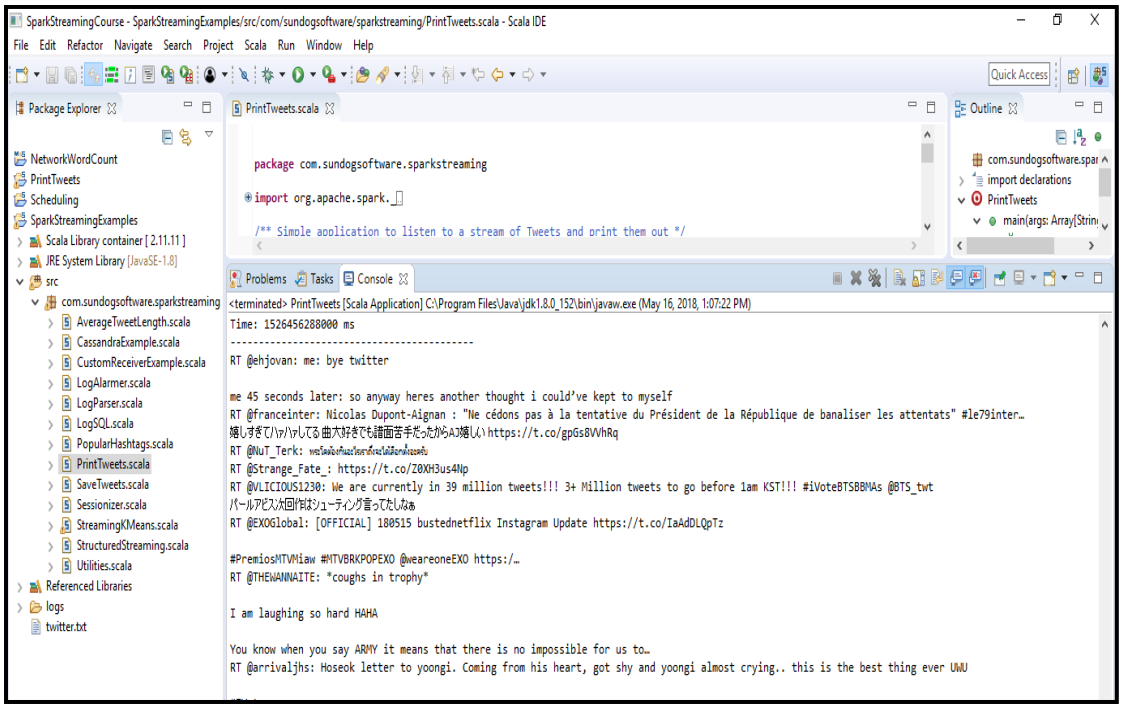


Figure 7.1: Real-Time Tweets in Spark

An application for printing the tweets was executed on Spark Cluster and the it was run in batches of 1 second for 58 seconds 445 ms. During this execution, 58 batches were completed and 2076 records were processed. The scheduling delay during this run is 0 ms and the time taken to process the application was 34 ms. The streaming statistics for this application are illustrated in Figure 7.2

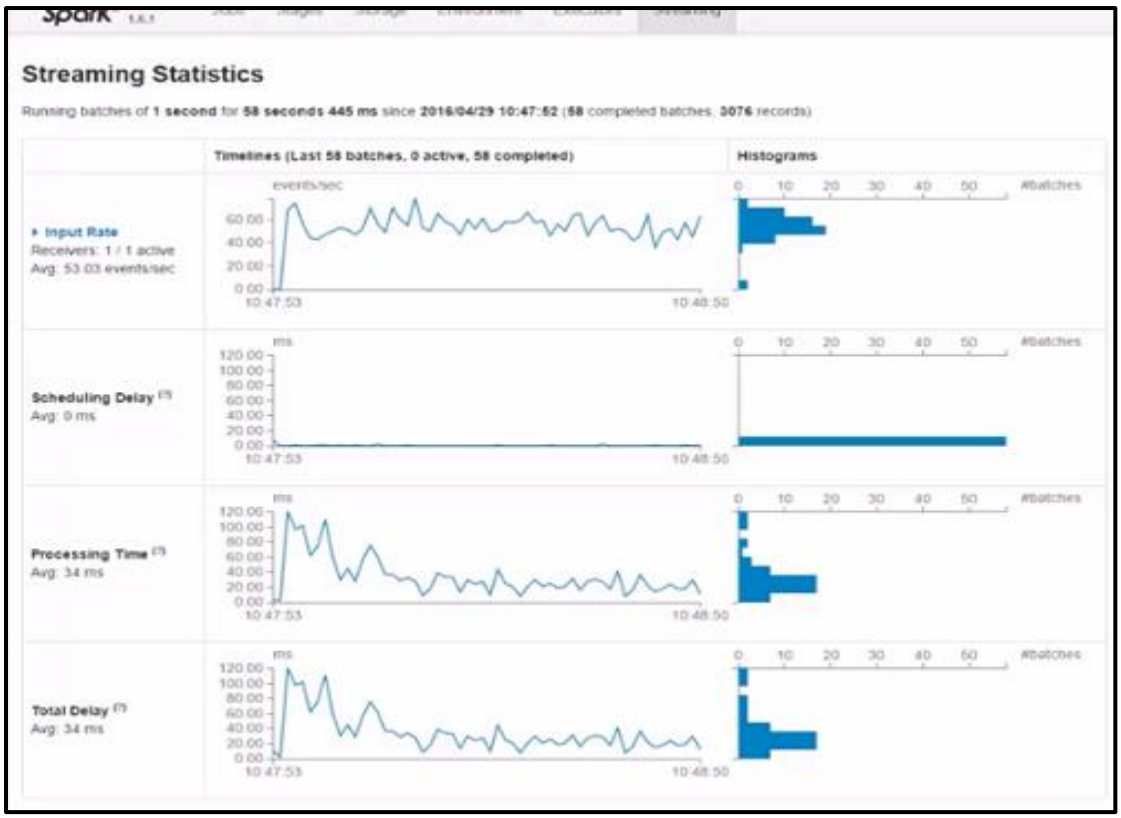


Figure 7.2: Streaming Statistics of PrintTweets Application

7.3 Streaming Statistics for tracking the Average Tweet Length

The streaming statistics are captured from the Spark Server UI when the spark application is run on Scala IDE. The statistics consists of Input Rate, Scheduling Delay, Processing Time and Total Delay in Figure 7.3. The status of the micro-batches can also be captured. The application to track the average length of a tweet runs in batches of 1 second and experimentally the application was executed for 1 minute 18 seconds. It completed 78 batches and 2544 records. The application encountered average scheduling delay of 7ms and took 64 ms processing time. It can be concluded from statistics that Apache Spark has shown high speed stream processing in the distributed environment.

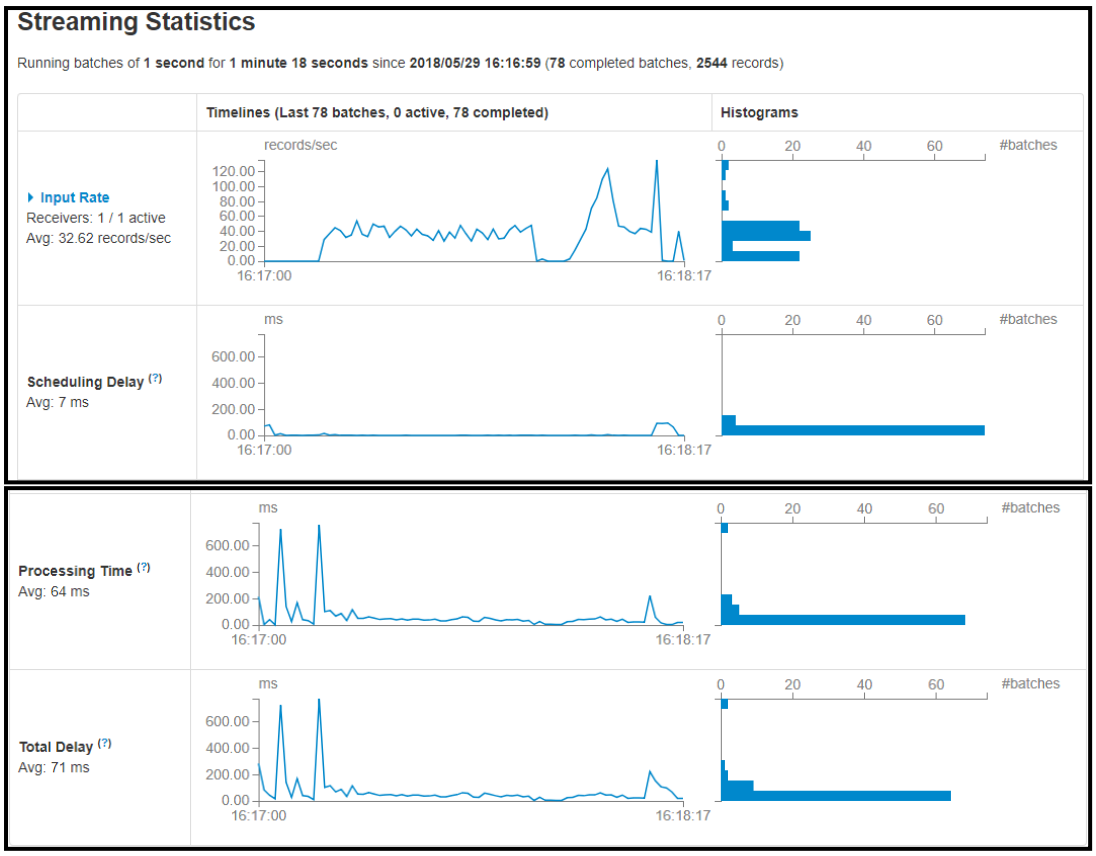
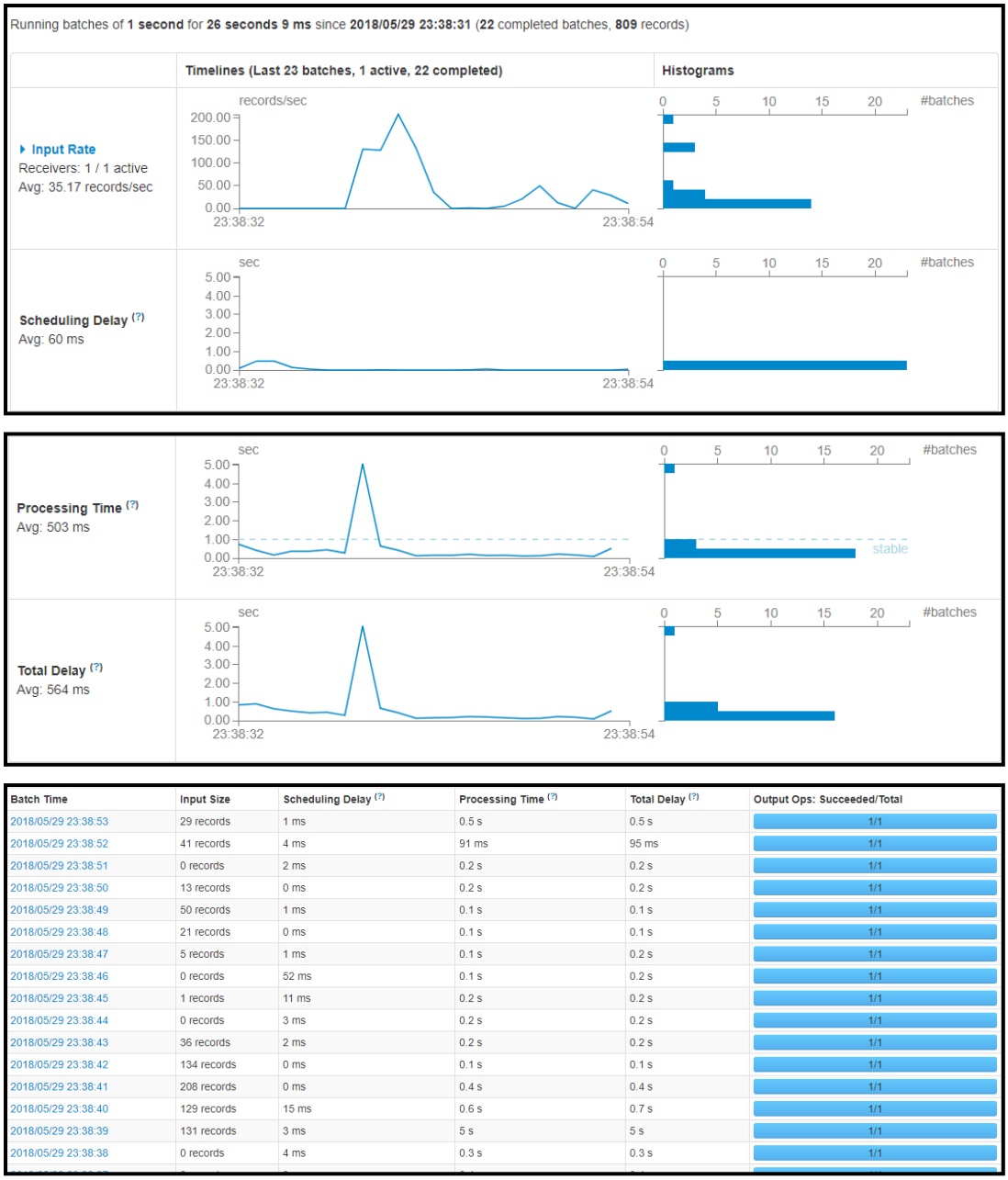


Figure 7.3: Streaming Statistics for Average Tweet Length

7.4 Streaming Statistics for the application of Finding out Most Popular Hashtags

The application was run in the batches of 1 second and was executed for 26 seconds 9 ms. Within this time frame it completed 22 batches and processed 809 records. The streaming statistics shown in Figure 7.4 shows the average scheduling delay of Avg. 60 milliseconds, the average processing time shown is 503 ms and the status of the micro batch processing is shown in Figure 7.4.



Batch Time	Input Size	Scheduling Delay (?)	Processing Time (?)	Total Delay (?)	Output Ops: Succeeded/Total
2018/05/29 23:38:53	29 records	1 ms	0.5 s	0.5 s	1/1
2018/05/29 23:38:52	41 records	4 ms	91 ms	95 ms	1/1
2018/05/29 23:38:51	0 records	2 ms	0.2 s	0.2 s	1/1
2018/05/29 23:38:50	13 records	0 ms	0.2 s	0.2 s	1/1
2018/05/29 23:38:49	50 records	1 ms	0.1 s	0.1 s	1/1
2018/05/29 23:38:48	21 records	0 ms	0.1 s	0.1 s	1/1
2018/05/29 23:38:47	5 records	1 ms	0.1 s	0.2 s	1/1
2018/05/29 23:38:46	0 records	52 ms	0.1 s	0.2 s	1/1
2018/05/29 23:38:45	1 records	11 ms	0.2 s	0.2 s	1/1
2018/05/29 23:38:44	0 records	3 ms	0.2 s	0.2 s	1/1
2018/05/29 23:38:43	36 records	2 ms	0.2 s	0.2 s	1/1
2018/05/29 23:38:42	134 records	0 ms	0.1 s	0.1 s	1/1
2018/05/29 23:38:41	208 records	0 ms	0.4 s	0.4 s	1/1
2018/05/29 23:38:40	129 records	15 ms	0.6 s	0.7 s	1/1
2018/05/29 23:38:39	131 records	3 ms	5 s	5 s	1/1
2018/05/29 23:38:38	0 records	4 ms	0.3 s	0.3 s	1/1

Figure 7.4: Streaming Statistics of Popular Hashtags Module

Chapter 8 CONCLUSION AND FUTURE SCOPE

8.1 Conclusion

In the span of mounting technology, industries are producing increasingly large volumes of rapid velocity data in very diverse formats with accelerating speeds of business. Hence, the streaming data managing frameworks such as Apache Spark, Apache Storm, Samza, Flink and Kafka will be dominated in the business fields, for the purpose of processing and handling real-time data streams and information, permitting the ability to analyze risks before its occurence. From Healthcare to Finance, from Social Media [10] to the Internet of Things, the requirement to bring about the superfluous size of data sets, is clear. The widespread big data manipulated frameworks like Hadoop are not appropriate for such use cases. In conclusion, in order to cope up with the technological competition, the enterprises must make use of state of the art technical instruments for the analysis of streaming data, for supporting much rapid decision making. In order to change the market position of company, there are plenty of benefits of streaming integration and intelligence in context to the right choice of streaming tools help in achieving it.

In this work, the study of Spark Streaming framework is carried out. The idea is to study Twitter Analytics for ingesting, processing, storage and visualization of Tweets in real-time and perform various other transformations on them such as tracking the average tweet length, detecting the most popular hashtag, etc. There were various experiments conducted on web logs to study the SparkML component of Apache Spark. The experiments include integration of Spark with Kafka, Flume and distributed storage framework known as Apache Cassandra for proving that this implementation can work in near real time. Moreover, the work is expanded to be executable in a distributed fashion on the K-Means Clustering algorithm being able to utilize the power of the Spark clusters. Furthermore, the experimental results prove the effectiveness and efficiency of the proposed approaches.

8.2 Future Scope

This work could be extended and improved following ways:

- There performance improvement can be to perform the transformations done

on Twitter using Scala IDE which can be done on Cloudera which has a HDFS built-in and Spark Libraries pre-installed which requires hardware cost as Cloudera requires the atleast 6GB RAM to function smoothly on personal system.

- The cluster mode step in our implementation which is set to local mode can be parallelized to be run on Mesos Cluster Manager for an increase in performance in big data clusters.
- Spark Streaming framework as of now is in a transforming state. The RDD abstraction is slowly getting deprecated and better ones emerge that are fully implemented in Spark Core and Spark SQL like the DataFrames and DataSets. So, when the Spark Streaming framework decides to make the change, we can migrate this work there.
- This study can be investigated more and more toward sentiment analytics or conducting intangible poll in election time. Another very interesting area of study can be medical and healthcare issues. This may help to find out about the spread of a disease even before reported samples of it become of the interest of the officials.

REFERENCES

[1] M. Kholghi, "An analytical framework for data stream mining techniques based on challenges and requirements", *International Journal of Computer Networks and Communications Security,* vol. 3, no. 5, pp. 156-201, 2015.

[2] D. Surekha, G. Swamy and S. Venkatramaphanikumar, "Real time streaming data storage and processing using storm and analytics with Hive", *in Proceedings of International Conference on Informatics, Electronics and Vision (ICIEV),* pp. 1120-1400, 2016.

[3] D. Jayanthi and G. Sumathi, "A Framework for Real-time Streaming Analytics using Machine Learning Approach", *in Proceedings of The International Conference on Computational Science,* pp. 126-201, 2016.

[4] S. Shahrivari, and S. Jalili, "Beyond Batch Processing: Towards Real-Time and Streaming Big Data", *in Proceedings of International Conference on Advanced Communication Control and Computing Technologies (ICACCCT),* pp. 1201-1280, 2014.

[5] M. Gaber, A. Zaslavsky and S. Krishnaswamy, "Mining Data Stream: A Review on complete reference about streaming issues", pp. 121-128, 2017.

[6] H. Yang and S. Fong, "Incremental optimization mechanism for constructing a decision tree in data stream mining", *International Journal of Engineering Science and Technology (IJEST),* vol. 13, no. 14, pp. 123-200.

[7] M. A. Beyer and D. Laney, "The Importance of Big Data: A Definition", 2012.

[8] D. Dai, X, Li, C. Wang, M. Sun and X. Zhou, "Sedna: A Memory Based Key-Value Storage System for Real-time Processing in Cloud", *in Proceedings of International Conference on Informatics, Electronics and Vision (ICIEV*), pp. 48-56, 2012.

[9] B. Yadranjiaghdam, N. Pool, N. Tabrizi, "A Survey on Real-time Big Data Analytics: Applications and Tools," *in Proceedings of International Conference on Computational Science and Computational Intelligence*, 2016.

[10] J. Zaldumbide, R. O. Sinnott, "Identification and Validation of Real-Time Health Events through Social Media," *in Proceedings of IEEE International Conference on Data Science and Data Intensive Systems*, pp. 9 – 16, 2015.

[11] M. Wachowicz, M.D. Artega, S. Cha, and Y. Bourgeois, "Developing a streaming data processing workflow for querying space–time activities from

geotagged tweets" *International Journal of Computers, Environment and Urban Systems*, 2015.

[12] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, JM. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham, N. Bhagat, "Storm@ twitter", *In the Proceedings of the ACM SIGMOD International Conference on Management of data*, pp. 147-156, 2014.

[13] T. Sakaki, M. Okazaki, Y. Matsuo, "Earthquake shakes Twitter users: real-time event detection by social sensors", *In the Proceedings of the 19th international conference on World wide web*, pp. 851-860, 2010.

[14] M. M. Rathore, A. Ahmad, A. Paul, and A. Daniel. "Hadoop based real-time Big Data architecture for remote sensing earth observatory system", *In the Proceedings of 6th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pp. 1–7, 2015.

[15] H. Duan, Y. Peng, G. Min, X. Xiang, W. Zhan, and H. Zou, "Distributed in-memory vocabulary tree for real- time retrieval of Big Data images", *In the Proceedings of International Conference on Ad Hoc Networks*, pp. 137–148, 2015.

[16] J. Wei, K. Chen, Y. Zhou, Q. Zhou, J. He. "Benchmarking of Distributed Computing Engines Spark and GraphLab for Big Data Analytics.", *In the Proceedings of IEEE Second International Conference on Big Data Computing Service and Applications*, 2016.

[17] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets." , *In the Proceedings of IEEE Second International Conference on HotCloud*, vol. 10, pp. 10–12, 2010.

[18] R. Ranjan, "Streaming Big Data Processing in Datacenter Clouds," *In the Proceedings of IEEE Cloud Computing*, pp. 78-83, May 2014.

[19] Cherniack, Mitch, et al. "Scalable Distributed Stream Processing.", *In the Proceedings of Conference on Innovative Data Systems Research* (CIDR). pp. 2021-2025, 2003.

[20] Boehm, Matthias, et al. "SystemML: Declarative machine learning on spark." , *In the Proceedings of the Very Large Data Base Endowment* v, pp.1425-1436, 2016.

[21] Sakaki, Takeshi, Makoto Okazaki, and Yutaka Matsuo. "Earthquake shakes Twitter users: real-time event detection by social sensors." , *In the Proceedings of the 19th international conference on World wide web (ACM)*, 2010.

[22] Kreps, Jay, N. Narkhede, and J. Rao. "Kafka: A distributed messaging system for

log processing." *In Proceedings of the NetDB*, pp. 1-7, 2011.

[23] Shahrivari, Saeed. "Beyond batch processing: towards real-time and streaming big data." , *International Journal of Engineering Science and Technology (IJEST)*, vol. 3, no. 4 , pp. 117-129, 2014.

[24] Carbone, Paris, et al. "Apache flink: Stream and batch processing in a single engine.", *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, pp. 4, 2015.

[25] Yi, Sangho, D. Kondo, and A. Andrzejak. "Reducing costs of spot instances via checkpointing in the amazon elastic compute cloud." , *In the proceedings of IEEE 3rd International Conference Cloud Computing (CLOUD)*, pp. 236-243, 2010.

[26] Kleppmann, Martin, and Jay Kreps "Kafka, Samza and the Unix Philosophy of Distributed Data.", *In the proceedings of IEEE Data Engineering Bulletin*, vol. 38, pp. 4-14, 2015.

[27] Jiang, Tao, et al. "Understanding the behavior of in-memory computing workloads." , *In the proceedings of IEEE International Symposium on Workload Characterization (IISWC)*, 2014.

[28] Golab, Lukasz, and M. Tamer Ozsu. "Processing Sliding Window Multi-Joins in Continuous Queries over Data Streams." *In the Proceedings of VLDB Conference*, pp. 500-511, 2003.

[29] Shanahan, James G., and Laing Dai. "Large scale distributed data science using apache spark.", *In the Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 2323-2324, 2015.

[30] O'callaghan, Liadan, et al. "Streaming-data algorithms for high-quality clustering." , *In the Proceedings of IEEE 18th International Conference on Data Engineering*, pp. 685-694, 2002.

[31] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S.Venkataraman,D. Liu and D. Xin, “Mllib: Machine learning in apache spark.”, The Journal of Machine Learning Research, vol. 17, no. 1, pp. 1235-1241, 2016.

[32] Mika, Peter. "Flink: Semantic web technology for the extraction and analysis of social networks." , *International Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 3.2, pp. 211-223, 2005.

[33] Armbrust, Michael, et al. "Spark sql: Relational data processing in spark.", *In the Proceedings of ACM SIGMOD International Conference on Management of Data*. pp. 1383-1395, 2015.

[34] Wang, Guoxi, and J. Tang. "The nosql principles and basic application of

Cassandra model." *In the Proceedings of IEEE Computer Science & Service System (CSSS)*, pp. 1332-1335, 2012.

[35] Maarala, Altti Ilari, et al. "Low latency analytics for streaming traffic data with Apache Spark.", *In the Proceedings of IEEE International Conference on Big Data*, pp. 2855-2858, 2015.

[36] Gu, Lei, and Huan Li. "Memory or time: Performance evaluation for iterative operation on hadoop and spark." , *In the Proceedings of IEEE International Conference on High Performance Computing and Communications and Embedded and Ubiquitous Computing (HPCC_EUC)*, pp. 721-727, 2013.

[37] Lin, Chieh-Yen, et al. "*Large-scale logistic regression and linear support vector machines using spark.*", *In the Proceedings of IEEE International Conference on Big Data*, pp. 519-528, 2014.

LIST OF PUBLICATIONS

[1] S. Arora and R. Rani, "A Streamlined Approach for Real-Time Data Analytics", *In the Proceedings of IEEE Second International Conference on Inventive Communication and Computational Technologies (ICICCT),* March 13, 2018 held at Coimbatore, Tamil Nadu.

Distributed Stream Processing of Twitter Data using Apache Spark

ORIGINALITY REPORT

9%	4%	4%	4%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

MATCH ALL SOURCES (ONLY SELECTED SOURCE PRINTED)

< 1%

★ Submitted to Delhi Technological University

Student Paper

Exclude quotes	On	Exclude matches	< 5 words
Exclude bibliography	On		