Improvising Processing of Huge Real Time Data Combining Kafka and Spark Streaming

M.Sc. Research Project
M.Sc. In Cloud Computing

Jeevantika Lingalwar
Student ID: 17156424

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Jeevantika Sanjay Lingalwar
Student ID:	17156424
Programme:	M.Sc. in Cloud Computing
Year:	2018-19
Module:	MSc Research Project
Supervisor:	Vikas Sahni
Submission Due Date:	18th April 2019
Project Title:	Improvising Processing of Huge real Time Data combining Apache Kafka and Spark Streaming
Word Count:	8480
Page Count:	20

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	17th April 2019

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	

Assignments that are submitted to the Program Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Improvising Processing of Huge Real Time Data Combining Apache Kafka and Spark Streaming

Jeevantika Lingalwar
17156424

Abstract

Cloud Computing era brings lots of new innovations in computing technologies which includes processing of data, storage, network access, internet security, web data etc. Knowledge Discovery plays a key role in inventions and abstractions of new knowledge. World Wide Web also called as WWW has now became a hub of these new inventions. Due to the increase in data WWW became overloaded because of which extracting of data and processing the same got complex. Data must be processed quickly, with the goal that a firm can respond to changing business conditions proactively. Stream processing is the ongoing processing of information simultaneously with its creation. It is a perfect platform for processing information streams such as log data, sensor information etc. There are number of alternatives additionally to do constant preparing over information like Spark, Kafka Stream, Storm etc. In this paper the proposed framework for processing huge real-time data is Apache Spark Streaming combined with Apache Kafka. The execution time is recorded after each experiment to get a clear view about the processing speed. Based on the experiments performed it has been concluded that Spark alone takes less time to processes huge amount of data whereas this framework i.e. Kafka combined with Spark execution time depends on the amount of dataset taken.

Keywords: Cloud Computing, MapReduce, Hadoop, Apache Kafka, Apache Spark, Java

1 Introduction

Cloud Computing provides a platform for sharing data related to Internet, Software Information and resources and to be provided as a mean of computing equipment. Parallel programing model is the key technology introduced by cloud computing. The cloud computing technology is a part of vitalization technology which is generally used in abstraction and realization of data. The advancement in the development of data mining is referred to as web data mining. web data mining is used to extract useful information from web data, this is a process model which is an incomplete large amount of data which is understandable. These data set's characteristics is novel and effective.

This era is a world of computing and computers. Everything here is dependent on new and intelligent technologies. The computing world is changing from parallel computing to distributed computing to grid and now it has over taken by cloud computing. As the growth in Internet is increasing, the data is also increasing rapidly. Every day and every second huge amount of data is generated which comes from millions of sources whose numbers is increasing constantly. For instance, playing out a purchase where it appears as though we're purchasing only a certain thing- might produce several solicitations that would send and create information. Saving a record in the cloud doesn't mean putting away it on one server, it implies duplicating it over numerous areas for adaption to non-critical failure and accessibility. Additionally, it implies putting away logs and point

by point data about each miniaturized scale venture of the procedure, to have the capacity to recuperate things on the off chance that they return out badly. We log huge amounts of information. We reserve things for quicker access. We imitate information and setup reinforcements.
We spare information for future investigation. All these real-time structures mean specialized prerequisites to structure an information handling framework: where and how the data is created, what is the recurrence of changes and updates in the data, how fast we need to react to the change (Cha and Wachowicz, 2019). A strong framework is produced with the help pf Hadoop and MapReduce for predictive analytics implementation. There is need to build a solution that will get information from an assortment of sources, perform explicit calculation and examination on information on the fly. The implementation sequence is divided into two parts: Data Ingestion and decoupling layer between wellsprings of information and goal of information, Data Processing and response (Georgios Gousio, 2018). This proposed framework in this paper is a solution to the above challenges and issues. Processing of data is done using Apache Kafka combining it with apache Spark Streaming. The result is analysed based on the execution time of each experiment. The motivation behind this research is to provide a new approach for processing huge real time data. We compare the execution time of our approach with the existing one which makes this topic worth to work on.

2 Related Work

Since a long time, numerous scientists are learning about the new ways and methods of extraction and handling of enormous measure of information. Different calculations and systems have been proposed to illuminate this issue. Hadoop MapReduce assumed a vital job for all the proposed methodologies. The following is a concise synopsis of some past writing papers which are identified with the theme. The information gives a concise finish of the work done by different analysts additionally a short synopsis of the different calculations and methodologies that are being utilized by them.

2.1 Developing a Real-time Data Analytics Framework for Twitter Streaming Data

Twitter being a social networking communication service with in excess of 300 million clients, producing a large amount of data consistently every day. The most critical and important trademark of Twitter is its capacity for managing the clients to tweet about every day events, occasions, emotions and circumstances, suppositions, or not withstanding something which is new progressively. Currently there are various distinctive work processes offering continuous information examination for Twitter, displaying general preparing over gushing information. The examination will endeavour to build up an expository system with the capacity of in-memory preparing to remove and investigate organized and unstructured Twitter information. The Researchers in this paper have proposed a system that will incorporated information ingestion, stream handling, and information perception parts with the Apache Kafka information framework that is utilized to perform information ingestion task.

Moreover, Apache Spark makes it conceivable to perform modern information preparing and AI calculations continuously. The researchers have directed a contextual analysis on tweets about the quake in Japan and the responses of individuals around the world with the investigation on the time and birthplace of the tweets. Analysing information continuously requires data ingestion and preparing of the surge of information before the information stock piling step (Cha and Wachowicz, 2019). A portion of the utilizations of the continuous information examination are reconnaissance, conditions, social insurance, business knowledge, marketing, representation, cybersecurity and social media. This investigation exhibits a continuous information examination structure for breaking down Twitter information. The fundamental distinction between this investigation and different explores is that the proposed structure cannot just play out the fundamental preparing undertakings, be that as it may, makes a framework for performing more refined and muddles examination on the gushing data.

There are various studies carried out on real-time data analysis directed in the field of continuous information investigation. Each one of them makes a few commitments to a classification in day to day life and employment diverse approaches. A portion of these zones of utilization have a high rate of affectability to respond to real information. The securities exchange is a substantial case of zones, which dependably have had a substantial dependence upon quick and exact examination. Internet based life information then again is generally the kind of information in view of sentiments and emotions. Regardless of this, there are still loads of valuable shrouded data, which might be separated from this kind of information. Dissecting posts on locales, for example, Facebook and Twitter may demonstrate vert valuable for illustration ends and making forecasts about exercises that happen on explicit regions of the worked at specific occasions (A. Bifet, 2018).

The motivation behind this investigation is to build up a structure for investigating Twitter information continuously. This structure has a few qualities which recognize it from conventional information examination approaches. The primary thought here is that there is a requirement for strategies to investigate a huge number of tweets coming each second, in a short measure of time. Additionally, the system ought to be autonomous of imported information volume; this is critical on the grounds that the volume of tweets is developing at a discernible rate. Figure below demonstrates a schematic of an adaptable 330 stream handling. The idea here is to gather occasion streams by various hubs and let numerous handling hubs to break down information in parallel. In this way, the test here is the means by which to oversee spilling information and how to examine it over the groups. Information handling work process often associates figuring assets to mechanize a succession of errands by preparing huge volumes of information, where diverse assets are associated for computerizing diverse assignments. On account of spilling information preparing, an adaptable and conveyed stage is required for consolidating expansive volumes of notable what's more, gushing information in the meantime. The system comprises of three areas: I) data ingestion; ii) data processing and iii) data visualization. The information ingestion area, interfaces straightforwardly to Twitter gushing API and in an adaptable way import information to the system. The information preparing segment with the capacity of gushing preparing over group accesses dispersed imported information, dissects information in memory, and performs handling undertakings on information, lastly sends the outcomes to be observed.

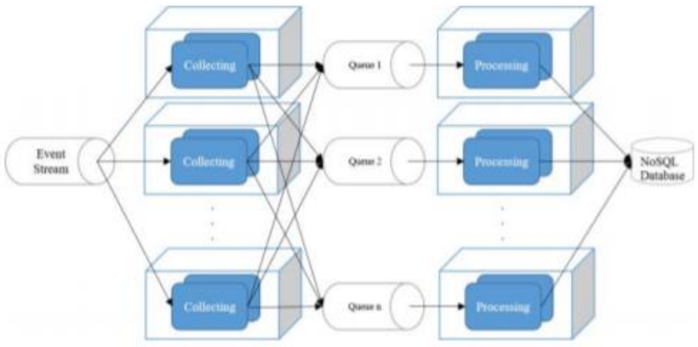


Figure 1: scalable processing structure (Babak Yadranjiaghdam, 2017).

Finally, the researchers have concluded by proposing a structure for Realtime examination of Twitter information. This system is intended to gather, channel, and investigate floods of information and gives us an understanding to what is well known amid a time and condition. The system comprises of three primary advances; information ingestion, stream preparing, and information representation. Information ingestion is performed by Kafka, an incredible message handling framework to import tweets, and to circulate it based on Topics that it characterizes, and to make it accessible over customers' hubs to be prepared by investigative devices. Apache Spark is utilized to get to these customers straightforwardly and examine information by Spark Streaming. The contextual analysis in this examination means to demonstrate the quality what's more, the significance of constant information investigation on social media gushing data. For example, assessment investigation of the tweets from the nation over can be done in presidential discussions. Doing as such, each discussion and the responses of the web-based life clients to it, can be utilized to train calculations and in up and coming discussions the fabricated model can play out a precise spilling handling on the web.

2.2 Literature review Performance evaluation of cloud-based log file analysis with Apache Hadoop and Apache Spark

Cloud based log file are generated in many formats by number of devices and in number of formats. When the analysis is carried properly each analysis leads to a useful information regarding each aspects of every system. Cloud computing is the most usable and suitable for such analysis as it can manage huge and high production rate and the huge size and variety in log files. The production rate of these log files can reach to number of TeraBytes or Petabytes. Due to these huge number of datasets reliable database solutions cannot be used for the analysis which is replaced by virtual database which is combined with distributed parallel processing and are considered more appropriate. As per the research done it has been assuming that log analysis is a use case which is of big importance and hence many of the important challenges that are related to cloud-based log file analysis are the big data challenges.

These analysis of log files later come on with other different challenges. Many of these systems are heterogenous and distributed so the logs from various number of other components must be related to each other. In addition, there may be some missing data or duplicate data that can make these log analyses more complicated or sometimes impossible. Hadoop is a framework that is been used by cloud computing in many companies for log analysis. The original design of Hadoop was mainly done for batch processing which provided scalability and fault tolerance, but it was not that useful for fast performance. It helps in running 1000s of nodes in petabytes of data. Apache Hadoop manages huge amount of log by breaking the files into blocks and give them to the Hadoop cluster nodes. It then breaks those jobs into number of small tasks. Apache Hadoop follows a processing model which reads and writes data from the disk. This process affects the performance and makes it non workable and unsuitable for real-time applications. The next approach is using Apache Spark which is more effective and suitable in which is uses more efficient main memory and minimizes the transfer of data from disk and to disk. Spark also provides new set of high-level tools which are used SQL queries, machine learning and graph processing (Mavridis, 2019).

For the performance evaluation of cloud-based log analysis the researchers in this paper have used two cloud computational frameworks, Apache Hadoop and Apache spark. The components of Apache Spark ecosystem used are Spark Core: a general execution engine, Spark SQL: Spark module for structured data processing, Spark Streaming: Enables real time data processing, MLib; Scalable machine learning library, GraphX: Graph computation Engine (Zaharia et al., 2012). Spark's Fault tolerance and parallel data structure i.e. Resilient Distributed Dataset. This is automatically distributed by Spark into the cluster and paralle operations are performed on them. For this analysis the researchers have carried out number of experiments which will evaluate the performance of Hadoop and spark and to do so they have used IAAS to create a private cloud infrastructure and some realistic log analysis operations with real log files. For the experimental purpose virtualized computing resources of Okeanos is been used (Koukis et al., 2013). It is an open source IaaS, which has been developed by Greek technology Network (Koukis & Louridas, 2012). With the help of Okeanos research six virtual machines have been created which are connected to a private virtual local network. In this there is one master node and 5 slave nodes, where the master node carries 8 virtual cores, disk space of 40GB and memory of 8GB. The 5 slave nodes are not that powerful as each of them has 4 virtual cores having a disk space of 40GB and memory of 6GB.

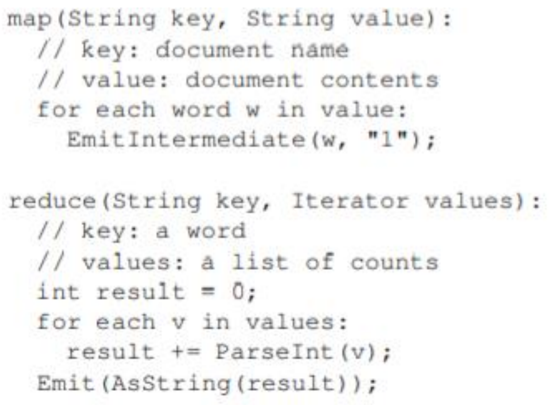
In order to evaluate these log files and investigate with Apache spark and Apache Hadoop the researchers have developed realistic applications in all the frameworks. The applications are developed using Java language with the help of Eclipse IDE and Apache Maven. The log files are stored in HDFS which is accessible by both Apache Hadoop and Apache Spark. In this paper researchers have focused on three main performances that are indicators, execution time, and the number of resources utilized with scalability. They have also compared the power consumption and cost of spark and Hadoop which is based on resource utilization and time measurements in which it is executed. Finally, the researchers have concluded that the analysis of cloud-based log files have been studied with the help of Apache Spark and Apache Hadoop comparing the cost, execution time and resource utilization with the memory usage. Several experiments have been conducted with number of different slave nodes, having different size of input file and the type of application which will test the best performance of spark as compared to Hadoop. The total execution time of Hadoop and Spark log analysis application has been recorded and the factors affecting the same. [Mavridis and Karatza, 2019]

2.3 Literature review on Simplified Data processing on Large clusters: MapReduce

MapReduce is a programming model that is utilized for preparing vast number of datasets which is into number of genuine undertakings. The procedure is conveyed regarding Map and Reduce two unique capacities likewise called as the mapper and reducer work and the runtime condition or the framework parallelizes into groups which is of extensive scales which handles disappointments in machine and between machine interchanges which will utilize different circles and system. MapReduce capacities and employments are executed in the bunches of Google each day which process in excess of 20 petabytes of information every day (Mitali Srivastava, 2017). Pretty much every information is huge and the preparing or the calculation should be dispersed to a huge number of machines to complete the activity in a productive measure of time (Gousios, 2018). Consequently, the issue of parallelizing the calculation and to deal with the disappointments with gigantic arrangement of complex code is required to tackle this issue. Thus, to conquer the previously mentioned issues analysts have planned another deliberation to diminish the multifaceted nature of preparing calculation employments yet it conceals the adaptation to non-critical failure and entangled parallelization subtleties.

The reflection approach depends on Map and Reduce work. the guide work is utilized for all sources of info bringing about a key-esteem sets and later these sets are exchanged to the lessen capacity to produce the yield result by assessing these key-esteem sets (Dean and Ghemawat, 2008). They thought utilizing client determined guide and diminish work alongside useful model permits performing parallelization of expansive handling of calculations and furthermore for adaptation to internal failure utilizing re-execution as the essential required mechanism.(Srivastava et al., 2017) To actualize this methodology utilize straightforward however incredible interface has been made which will open up programmed appropriation of extensive scale calculations with parallelization which will accomplish vast calculation bunches with superior of the equivalent. The fundamental programming model incorporates a lot of sources of info and later creates a yield of key-esteem sets. The Map work takes the info pair which is given by the client and creates set of other key-esteem pair. MapReduce work together procedure the qualities with the middle of the road key 1 and pass it to the diminish work for further preparing. lessen work likewise acknowledges a key-esteem pair which is given by client with middle of the road key 1 and sets some an incentive for a similar key. later these qualities are converged to get a little arrangement of significant worth which is in zero or one (Armbrust, M., 2015). In this paper the scientists have proposed different MapReduce execution techniques which expresses that the correct decision of the strategy is finished by the earth. They clarified this with the assistance of a precedent which says, a usage might be pertinent for a little memory machine which is shared, another for substantial processors or multiprocessors additionally with significantly bigger organized machines.

The most exact clients of MapReduce are a finished improve of the ordering arrangement of the creation which produces the information structure which is utilized by Google web search tool. A contribution of expansive arrangements of reports is taken by information which is assembled by this slithering framework that is put away as GFS document. Utilizing the specially appointed conveyed MapReduce has ended up being helpful in each field. The code managing adaptation to internal failure, parallelization and appropriation are covered up inside the MapReduce library which makes the ordering basic and straightforward.



```
map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");

reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```

Figure 2: Examples of map and reduce function (Anubhav Tarar, 2018)

MapReduce libraries perform calm and in a legitimate manner so the reasonable calculations which are not related anyplace can be kept independently as opposed to blending them up keeping away from additional heap on information.

At long last, the analysts have finished up the system data transfer capacity is a constrained asset. Along these lines, tremendous number of advancements in their framework are modified to diminish the measure of information which must be sent everywhere throughout the system, this enables them to peruse information from the circles which are neighborhood and composing a duplicate to a similar nearby plate to spare system transfer speed. Next, they see that a similar execution can be actualized to facilitate the effect of the machine which are running moderate and to deal with disappointments of machine and loss of information.

3 Research Methodology

To achieve the result, four important process need to be performed before any implementation or development which uses Spark streaming and Kafka implementation and they are Transformation, Cleaning, writing the data received from various sources and Cleaning.

3.1 Objectives

The main objective of this approach is to keep up with the ever-increasing stream and amount of data that every data the developers are challenged with processing. From the network of data ingestion to processing real time data, this method will make it easier to keep up with the distributed streaming workflow. The main motive here is to reduce the complexity of processing huge real time data which makes it fast, reliable, scalable and pliable data workflow using two frameworks like Apache Kafka and Spark streaming. This system will be used to process real-time data from number of real time sources with some machine learning task and will efficiently experiment with real-time data processing in the real time world.

3.2 Proposed Algorithm workflow and framework

Apache Spark Streaming Pipeline

In the paper, some standard workflow has been implemented before processing the dataset which consists of 4 main stages which consists of Data Transformation, Data Cleaning, Data Validation and Data writing. For this purpose, various platform can be used like Java, Python and Spark Streaming for some of the workflow. In this paper python, Java and spark is used for the workflow.

1. **Data Transformation:**

The process is the first step and the main point for streaming an application. The data transformation can be easy, simple and sometimes complex depending on the required changes in the data that is processed between the source data and the target data. In the Spark Streaming the data transformation is performed using a built-in function which is filter, map, RDDforeach and many more. Following is an iteration that have been used for the transformation in Spark streaming.

2. **Data Cleaning:**
Cleaning is the second most important stage before processing the data. You don't want any Null values to be disturbing the process for that purpose a clean data is very important. To remove the garbage data, a separate streaming application is executed for cleaning the data and for this purpose some custom built-in libraries in python is used. Following is the iteration used for data cleaning.

3. **Data Validation:**
Data validation is basically validation your data against a certain requirement whether the data produced matches the required fields, string, length etc. For data validation there is a separate spark streaming script written which is as below.

4. **Data writing:**
The final step is writing the data. After the data is processed through above method it is then passed to writing application which will write the data to the HBase for further processing of data. Kafka manages all the communication between all the applications using various topics and various applications.

The job streaming is done as below, and the pseudo code is presented for better understanding of the process (Databricks, 2019).

Step 1: **Initializing the Spark Context**

```
spark_context = SparkContext(appName="Application Transformation")
```

This is the main step in spark which will allow to connect to spark cluster and create Resilient Distributed Datasets. The streaming application name is appName

Step 2: Initialize Spark Streaming context

```
streaming_spark_context = StreamingContext(spark_context, 5)
```

This is used to create Dstream from another various source. Here the values are taken from Kafka where value 5 indicates the batch interval. Spark streaming make sure that streaming data is divided not batches. Each batch is converted into RDD and these RDD are called as Dstream.

Step 3: Get data from Kafka topics

```
direct_stream = KafkaUtils.createDirectStream(streaming_spark_context, [list of topics comma separated], {"metadata.broker.list": kafka_host})
```

This is used to connect Kafka broker to fetch the messages from Kafka. Here no need of creating multiple input Kafka streams and integrating with DirectStream.

Step 4: Transformation

```
direct_stream.map(custom_defined_functions)
```

Here we have used map methods for the transformation. Note that you can used other transformation method as well as mentioned in this section i.e. filter, foreachRDD etc.

Step 5: Pipeline Management

```
streaming_spark_context.start()

streaming_spark_context.awaitTermination()
```

Performance Tunning:
To track and improve the performance of the pipeline the following configurations can be used.

1. spark.executor.memory: is use to define the memory that is being given per use per executor which shows up in the same format as JVM strings(memory strings).
2. Spark.executor.instances: This gives the number of executors also creates a conflict having spark.dynamicAllocation.enable.
3. Spark.streaming.reciever.maxRate: It gives the number(maximum) of records every sec a receiver will receive.
4. Spark.streaming.kafka.maxRatePerPartition: This gives the number(maximum) of records that will be read from each Kafka stream when using very new Kafka Directstream API. (Opcito Technologies, 2019)

Apache Kafka
Kafka is a publish-subscribe messaging system which is known for its real-time data streams ingestion and providing them to the downstream consumers which is managed in parallel and fault-tolerant specification. The Kafka data are stored in the topics which are split into partitions which are the sequence of records that are ordered and can be accessed as structured commit log. Producers writes these logs and the consumers read them as per their own convenience (Databricks, 2019).

Kafka Topic

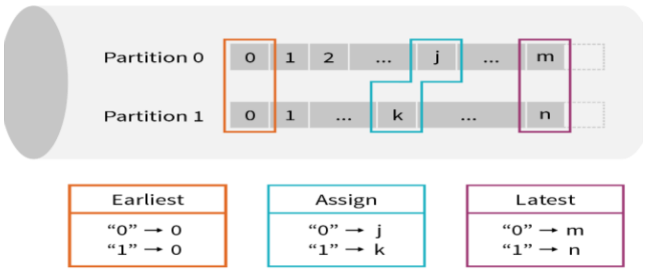


Figure 6: Kafka Topic Partitions (Databricks, 2019)

Kafka subject is endless flow in which the records are taken back for a certain amount of time. The countless nature of this flow it is said that there is a need to first decide which data is to be studied and the place of time it is to be started, when starting a new query. As shown in the figure the three main choice of partition:

- Earliest: In this the older data is deleted which is outdated and the data is been read form the beginning.
- Latest: The data that has been arrived after starting the query is started is being processed.
- Per-partition assign: A unique offset is suggested to start from each partition, which permits fine-grained manipulate over where processing is supposed to start.

Apache Kafka structured streaming:

The structured streaming presents a unified batch and streaming API that allows us to see the posted data to Kafka which is as DataFrame. The data when processing in streaming fashion, the same API is used, and the identical information consistency is retrieved which is promised in batch processing.

Kafka Topics records reading:
First, the location of the Kafka cluster and the topic that is to be read is specified. Spark is the best framework which allows to read every single topic, set of topics, pattern of topic which in regex or any partitions that is a part of set of topics.

4 Design Specification

Workflow: Processing Huge real Time Data

The processing of huge real time data is done using Apache Kafka and Apache Spark and the execution time is recorded to compare the difference between both the framework's execution. The following are the steps used to process the huge amount of data.

For coding Apache NetBeans have been used which is an integrated development environment (IDE) for Java. To run and compile Linux applications on windows operating system Cygwin is used. Cygwin is an open source platform that allows Linux application to run on windows. For a user to use windows specific service on a different platform winutils is important. It is a collection of useful TcI commands which takes part of Win32 API. For processing the data, Apache Spark and Apache Kafka framework have been used. Apache Spark is a consolidated analytics engine for large-scale or huge amount of data processing. On the other hand, Apache Kafka is a distributed data streaming platform. Apache Kafka is used to process streams of records as they occur. It is also used for building real-time streaming applications that react to streams of data. In this project Apache Kafka will be used to process huge amount real -time data and compare with the processing time of Apache Spark.

The following are the steps involved for the processing of data:

Steps No	Steps
Step 1	For this project we need Windows 7 32-bit OS or more update version
Step 2	The tools required for this project are Cygwin, Winutils, Apache Spark, Apache Kafka, NetBeans.
Step 3	Download and Install NetBeans IDE 8.0.2 Java SE edition.
Step 4	Download and Install Cygwin.
Step 5	Download and Install Apache kafka and Apache Spark.
Step 6	Download and Install Zookeeper-3.4.14.
Step 7	Create a new Java project naming it ImprovisingProcessingFramework having three classes including one main class, MainFrame and Receiver. The MainFrame consists of Data Ingestion, Processing data using Spark and Kafka view. This is will give the complete view Loading the dataset and ingesting it into Spark and Kafka.
Step 8	: Create a new project naming it ApacheSpark. This project will contain Java Source Package which will have three java classes one Main class, one Spark Frame, and Spark Receiver. The Dependencies folder will contain all the Spark and Java Dependencies required to process the data. The MainFrame consists of Datasets that is ingested from Processing Framework MainFrame which needs to be processed using Spark.
Step 9	Create another new project naming it ApacheKafka. This project folder will contain Java Source Package which will contain three java classes, Main class, KafkaFrame class, KafkaReceiver Class. The Dependencies folder contains all the spark, Kafka

	and Java dependencies that is required for processing the data using Kafka. The Kafka MainFrame consists of dataset that is ingested from Processing Framework MainFrame.
Step 10	Click on the button Data Processing using Apache Spark in the Spark Main Frame. The process is started and when finished it will show a popup saying Data processing completed. Click on Ok.
Step 11	Click on the button Data Processing using Apache Kafka in the Kafka Main Frame. The process is started and when finished it will show a popup saying Data processing completed. Click on Ok.
Step 12	Go to the Improvising Processing Framework MainFrame and check the result of both Apache Spark and Apache Kafka where the processed data s recorded.
Step 13	Click on Comparison Tab where you can see the processing time comparison between apache Spark and apache kafka.
Step 14	Perform the above experiment 5 times and record the time of each execution to compare the exact difference

Table 1: Steps for Processing Huge Real Time Data

Flow Diagram

The following is the flow diagram of the complete design specs.

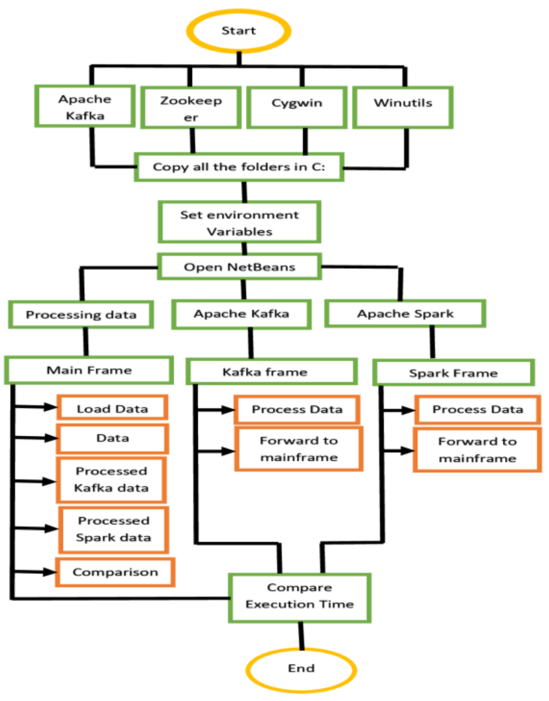


Figure 10: Design Specification flow

5 Implementation

In this section, the implementation of the project has been discussed in detail. The processing of huge real-time data is done using Kafka Frame, Spark Frame and Processing Data main frame that will ingest the dataset in Kafka and Spark frame for processing. The Frames are created using Java libraries. The detail implementation is explained as below,

5.1 Dataset – Market Basket Analysis

As per the definition, MapReduce is a programming model used for distributed programming based on Java. The Map function takes a set of data and converts it into another, and the reducer break these individual elements into smaller set tuple called as key/value pairs.

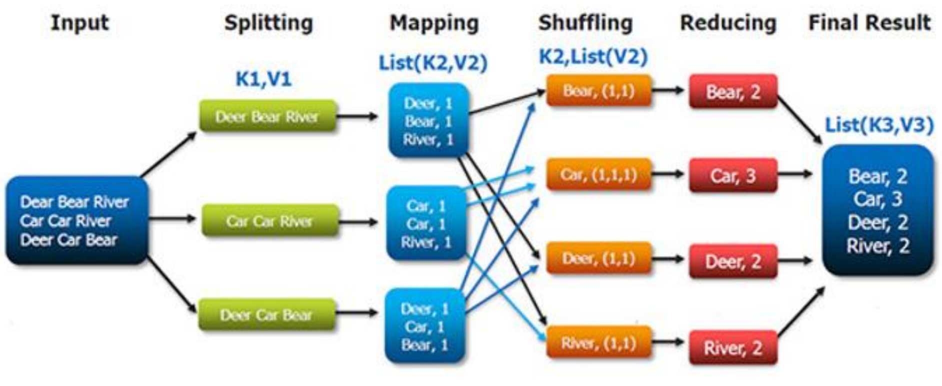


Figure 11: MapReduce Workflow (Talend Real-Time Open Source Data Integration Software, 2019)

Based on the above MapReduce workflow, Market basket analysis dataset is taken for this project. In each transaction many items are purchased by customers.
For Example,

Transaction	**Items**
t1	{T-shirt, Trousers, Belt}
t2	{T-shirt, Jacket}
t3	{Jacket, Gloves}
t4	{T-shirt, Trousers, Jacket}
t5	{T-shirt, Trousers, Sneakers, Jacket, Belt}
t6	{Trousers, Sneakers, Belt}
t7	{Trousers, Belt, Sneakers}

5.2 Project Execution

Processing Real-time Data Framework

The Processing Data framework consists of three Java classes

1. Main Class
2. MainFrame Class
3. MainFrameReceiver Class

The Main Class consists of function that is needed to call the processing data main frame class and the main frame receiver class that will receive the data which is used for processing the data. The main () method is a starting point for the Java application. The Main Frame is the UI for presenting the processing of real-time data. For reading and writing the file Java FileInputStream library is used. To build the UI java free UI Application frame library is used. To connect the application to any available port on local host DatagramSocket is used. The UI consist of loading the huge real-time data, ingesting the data to Apache Spark Frame and Apache Kafka Frame, comparing the executing time for Apache Spark an Apache Kafka. The MainFrameReceiver includes the function that will return the processed data in the Spark processed result frame and Kafka processed result frame which is used for comparison of execution time. Below is the class diagram for this framework which gives a brief detail.

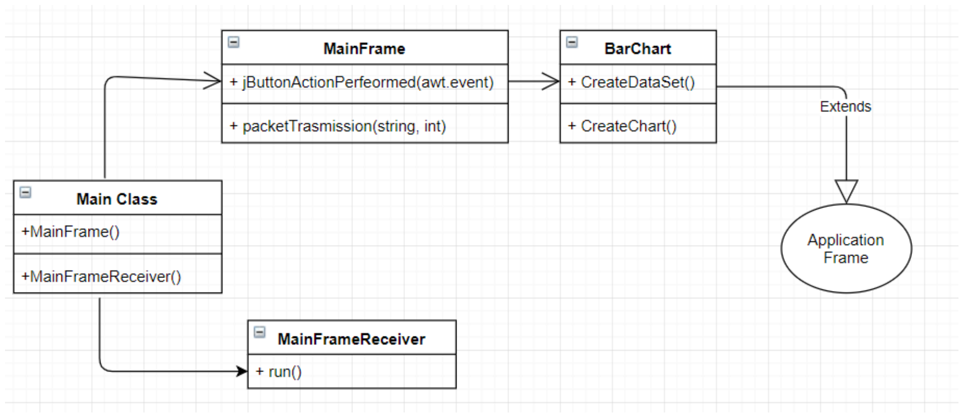


Figure 12: Class Diagram for Processing Data Framework

Apache Spark framework

The Apache Spark framework consists of three Java classes

1. Main Class
2. SparkFrame Class
3. SparkReceiver Class

The Main class consists of function that is used to call the Spark Main Frame class and the Spark Main Frame Receiver class which will receive the data used for processing. The SparkFrame consists of the function that will contain the data that was ingested from the processing data main frame. The org.apache.spark.api.java.JavaRDD is a local handle class for Resilient Distributed Dataset. It is used to write function that is required to transform the information and the data to collect it locally. The org.apache.spark.api.java.JavaPairRDD is like JavaRDD which is also a local handle for a distributed collection of data [Dei.unipd.it, 2019]. Here maptoPair () method is used to count the frequency of each word and reduceByKey () method counts the repetitions of word in the text file. JavaSparkContext is Java friendly and a part of SparkContext which returns back JavaRDDs and implemented with collections in Java instead of the one in scala.
Below is the class diagram for this framework that gives a brief detail about it.

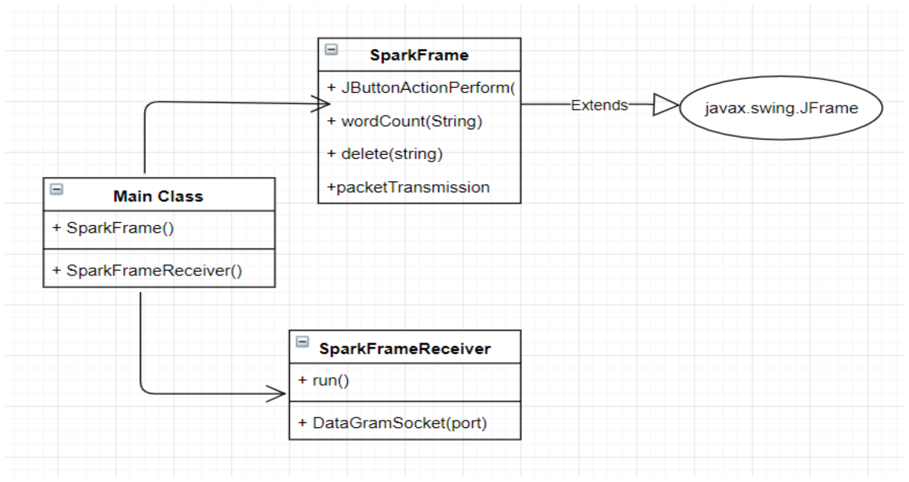


Figure 13: Class Diagram for Spark Framework

Apache Kafka Framework

The Apache Kafka framework consists of three Java classes

1. Main Class

2. KafkaFrame Class
3. KafkaReceiver Class

The Main class consists of function that is used to call the Kafka Main Frame class and the Kafka Main Frame Receiver class which will receive the data used for processing. The kafkaFrame consists of the function that will contain the data that was ingested from the processing data main frame. JavaDStream is an interface to DStream, the essential abstraction in Apache Spark Streaming which speaks with the constant stream of data. DStreams can either be made from live information for e.g. Kafka, TCP sockets etc. The JavaPairDStream is an interface of DStream which consists of key-value pairs which generates more methods like join and reduceByKey. The Kafka Frame UI consists of two buttons, Processing data using Kafka and forwarding that data to result in the main frame. On click of button 1, a function is called that will process the data in Kafka and on the click of second button the processed data is sent to the main frame. Below is the class diagram for this framework that gives a brief detail about it.

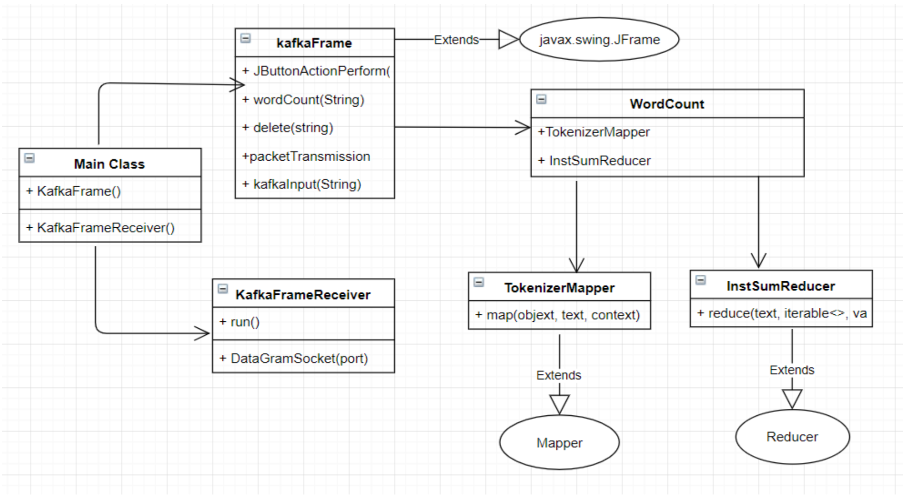


Figure 14: class Diagram for Kafka Framework

5.3 Project Setup

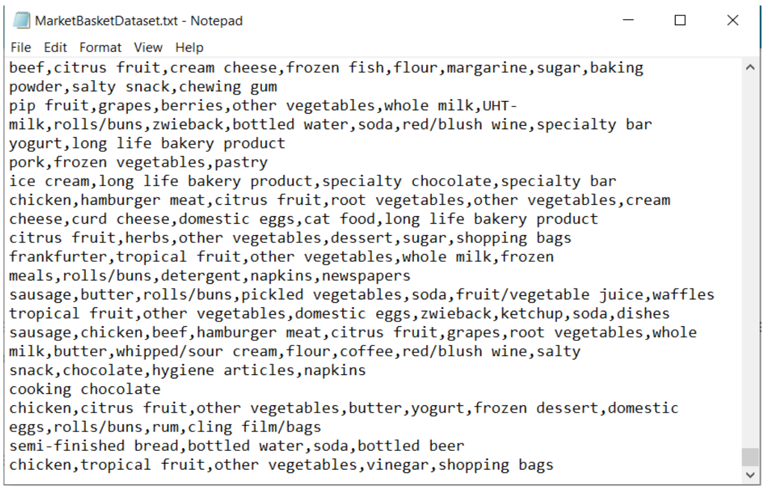
System Requirement: Windows 64-bit with access to 32-bit application run

For the implementation purpose NetBeans IDE 8.0.2 and Java code deployment has been used.

The Market Basket dataset is taken from Kaggle which is used for processing using two frameworks and execution time is recorded for the both and compared in end Specifications and evaluations are explained in the section below. Figure shows datasets details that is used for processing in Kafka and Spark.

6 Evaluation

This section gives the details of the evaluation scenarios that is implemented and processed.



```
beef,citrus fruit,cream cheese,frozen fish,flour,margarine,sugar,baking
powder,salty snack,chewing gum
pip fruit,grapes,berries,other vegetables,whole milk,UHT-
milk,rolls/buns,zwieback,bottled water,soda,red/blush wine,specialty bar
yogurt,long life bakery product
pork,frozen vegetables,pastry
ice cream,long life bakery product,specialty chocolate,specialty bar
chicken,hamburger meat,citrus fruit,root vegetables,other vegetables,cream
cheese,curd cheese,domestic eggs,cat food,long life bakery product
citrus fruit,herbs,other vegetables,dessert,sugar,shopping bags
frankfurter,tropical fruit,other vegetables,whole milk,frozen
meals,rolls/buns,detergent,napkins,newspapers
sausage,butter,rolls/buns,pickled vegetables,soda,fruit/vegetable juice,waffles
tropical fruit,other vegetables,domestic eggs,zwieback,ketchup,soda,dishes
sausage,chicken,beef,hamburger meat,citrus fruit,grapes,root vegetables,whole
milk,butter,whipped/sour cream,flour,coffee,red/blush wine,salty
snack,chocolate,hygiene articles,napkins
cooking chocolate
chicken,citrus fruit,other vegetables,butter,yogurt,frozen dessert,domestic
eggs,rolls/buns,rum,cling film/bags
semi-finished bread,bottled water,soda,bottled beer
chicken,tropical fruit,other vegetables,vinegar,shopping bags
```

Figure 15: Dataset Market Basket Analysis

Processing the Data using Apache Kafka

Apache Kafka being an open source streaming system is used for building real time application and streaming huge amount of data. To process the data, in the project Kafka topics are created. Kafka runs as a cluster on more than one server called as brokers. Kafka returns stream of records.

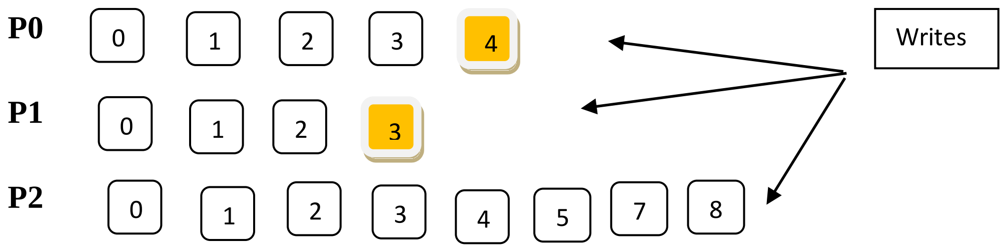


Figure 16: Kafka Topics

Each of these partitions are ordered and are sequence of record that is updated in the log. The Apache Kafka producers and Consumers play a very important role. Producer write the data in the topic. Consumers receives a message as soon as they subscribe to it. Consumer can be a part of a group or can be independent. Not only huge amount of data in ingested by Kafka but also works well with small data providing flexibility and scalability which makes is fast. For processing this huge real-time data there is a need for a framework of stream processing. There are number of streaming platforms like Kafka, Spark, Storm etc. In this paper Kafka and spark has been used. Kafka Stream is fast and lightweight that works well if the data ingestion is proper.

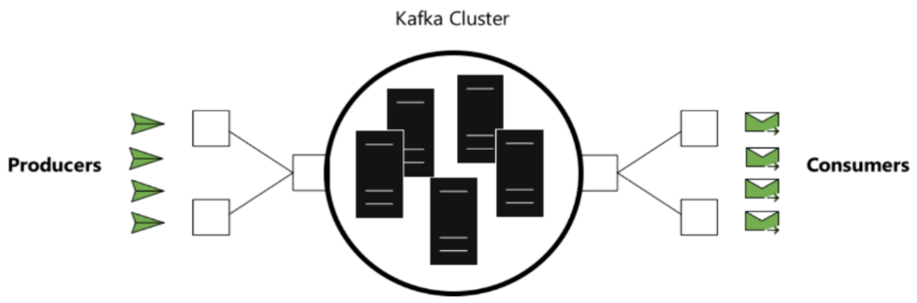


Figure 17: Kafka producers and Consumers (Lenadroid.github.io, 2019)

Processing Data using Apache Spark

Apache Spark being an open source framework for large scale computing environment. In this project, Apache spark is used for processing huge-real time data. Although spark is same as MapReduce, but it is more fast and powerful. For the implementation latest release of spark is used which supports continuous processing and execution as well as micro-batch processing. Apache Spark consists of workers which are the physical nodes. There is a driver that looks after all the workers. The execution of the tasks is managed by the executor where the tasks are the small unit of execution which are independent.

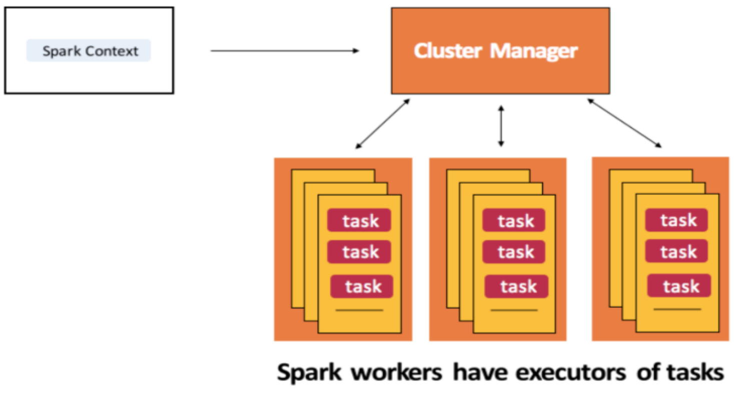


Figure 18: Spark working structure (Lenadroid.github.io, 2019)

As discussed above Apache Kafka is scalable when it comes to ingesting the streams of data produced by consumers. Whereas, Apache Spark is great for processing huge real-time or large amount of data. When combining both Apache Kafka and Apache Spark the processing of real-time streaming data gets easier and faster then before. There are number of experiments performed based on the above and following are the results generated for Kafka and Spark as per the execution time.

6.1 Experiment 1

The below figure shows the output for experiment 1 performed. The result generated shows the time comparison in seconds for the processing huge-real-time data using Kafka and spark. The Data is ingested via main frame into Spark frame and Kafka frame. The processed result is forwarded to the main frame result section and the comparison graph is generated as below.

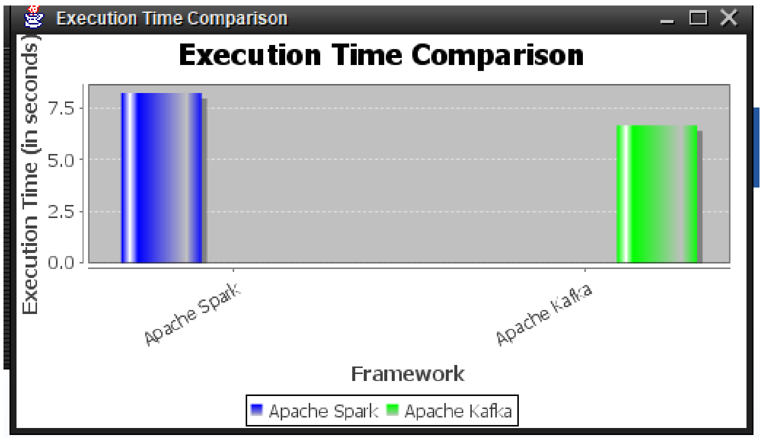


Figure 19: Observation 1 Execution time apache spark and Apache Kafka

The above observation clearly specifies that the execution Time for Apache Kafka and Apache Spark differs.
Execution Time Recorded for Kafka: 6.501 sec
Execution Time Recorded for Spark: 8.109 sec
These observations were noted 5 times and final readings have been taken.
According to the above observation it can be said that processing huge real time data using Kafka and streaming platform takes less time than processing data using Spark alone.

6.2 Experiment 2

The below figure shows the output for experiment 2 performed.

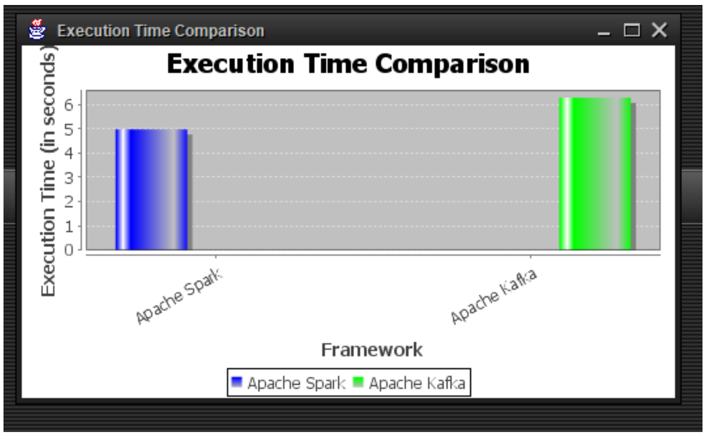


Figure 20: Observation 2 Execution time apache spark and Apache Kafka

The above observation clearly specifies that the execution Time for Apache Kafka and Apache Spark differs.
Execution Time Recorded for Kafka: 6.111 sec
Execution Time Recorded for Spark: 4.960 sec
These observations were noted 5 times and final readings have been taken.
According to the above observation it can be said that processing huge real time data using Kafka and streaming platform takes more time than processing data using Spark alone.

6.3 Experiment 3

The below figure shows the output for experiment 3 performed.

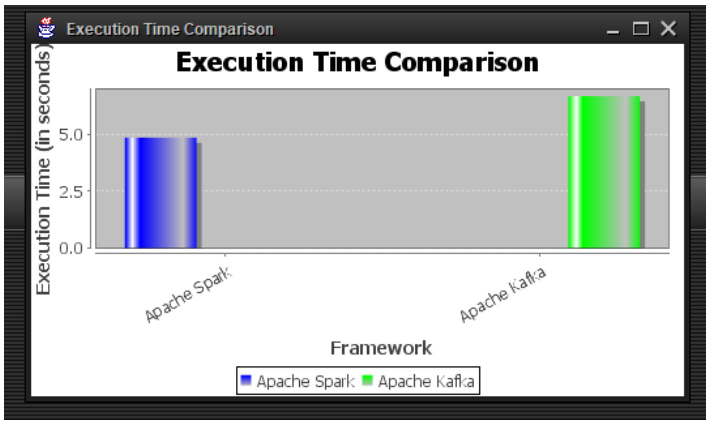


Figure 21: Observation 3 Execution time apache spark and Apache Kafka

The above observation clearly specifies that the execution Time for Apache Kafka and Apache Spark differs.

Execution Time Recorded for Kafka: 5.897 sec
Execution Time Recorded for Spark: 4.990 sec
These observations were noted 5 times and final readings have been taken.
According to the above observation it can be said that processing huge real time data using Kafka and streaming platform takes less time than processing data using Spark alone.

6.4 Experiment 4

The below figure shows the output for experiment 4 performed.

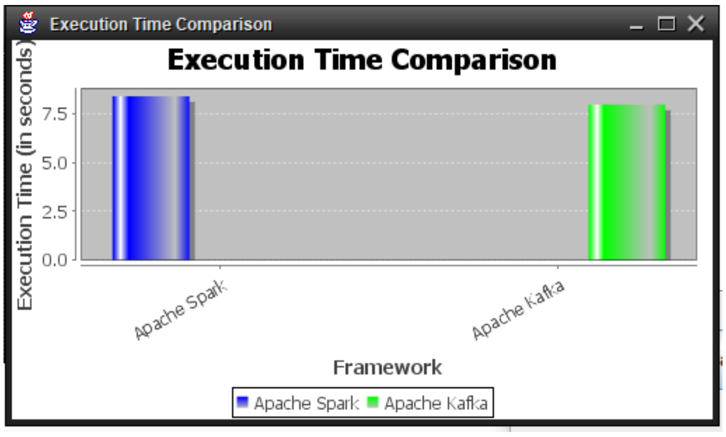


Figure 22: Observation 4 Execution time apache spark and Apache Kafka

The above observation clearly specifies that the execution Time for Apache Kafka and Apache Spark differs.
Execution Time Recorded for Kafka: 7.516 sec
Execution Time Recorded for Spark: 7.789 sec
These observations were noted 5 times and final readings have been taken.
According to the above observation it can be said that processing huge real time data using Kafka and streaming platform takes less time than processing data using Spark alone.

6.5 Experiment 5

The below figure shows the output for experiment 5 performed.

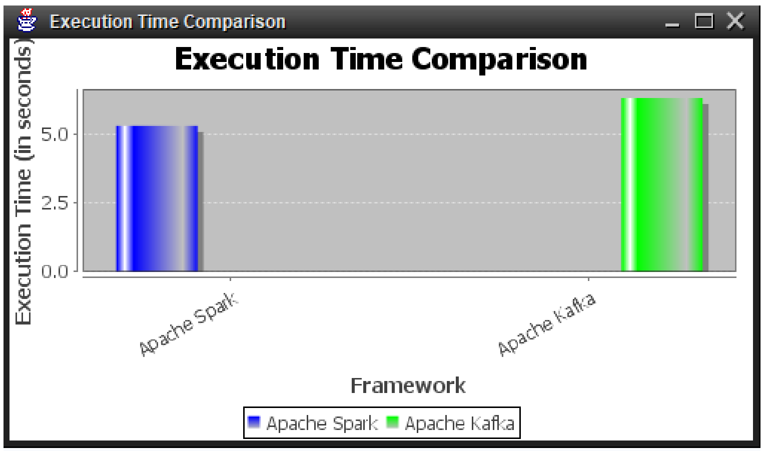


Figure 23: Observation 5 Execution time apache spark and Apache Kafka

The above observation clearly specifies that the execution Time for Apache Kafka and Apache Spark differs.

Execution Time Recorded for Kafka: 7.102 sec
Execution Time Recorded for Spark: 5.112 sec
These observations were noted 5 times and final readings have been taken.
According to the above observation it can be said that processing huge real time data using Kafka and streaming platform takes more time than processing data using Spark alone.

6.6 Discussion

There are 5 experiments performed with the same dataset and all observations having different execution times. As all observation's execution time differs from each other, the aggregate has been taken to get the exact result of the experiment. The result varies as per the performance. It has been noted Apache Spark take less time for processing real-time data whereas Kafka takes a bit long. But when combined both the execution time again varies depending on the amount of data. According to the analysis done and result generated it can be suggested that result various as per the amount of data taken. Apache Kafka is best worked in Kafka > Kafka context whereas Spark Streaming is best used in Kafka > Database kind of context.

7 Conclusion

As the demand of stream processing is increasing every day lots of new technologies and frameworks are emerging. The reason behind this increasing demand is the huge amount of data generating each day. The implementation of this project has highlighted some pros and cons of using Apache Kafka and Apache Spark for processing huge real-time data. The streaming process, the execution time, complexity level according to data set has been noted. From processing data using Spark and Kafka alone to combining both, the result has been discussed in detail in above section. Apache Spark is a general structure for huge scale information handling that bolters bunces of various programming dialects and ideas for e.g. MapReduce, in-memory preparing, stream handling, chart handling or AI. Spark streaming gives and abnormal state reflection called DStream, which speaks to a nonstop stream of data. DStreams cab be made either from data streams from other sources like Kafka, Flume etc. The DStream is referred to as a grouping RDDs. Kafka Streams is a customer library for preparing and breaking down information put away in Kafka and either compose the subsequent information back to Kafka or send the last yield to an outer framework.
Kafka Streams straightforwardly addresses a ton of the troublesome issues in stream preparing: Occasion at once preparing (not micro batch) with millisecond idleness. Stateful preparing including conveyed joins and collections. A helpful DSL. Windowing without-of-request information utilizing a Dataflow-like model. Appropriated preparing and adaptation to internal failure with quick failover. No-vacation moving arrangements. According to the observations retrieved from the implementation it can be said that additional plugins can also be utilized in Kafka because the data that is once sent is retained and can be used many times. Apache Spark can be utilized with Kafka to stream the information yet if you are sending a Spark group for the sole reason for this new application, that is unquestionably a major intricacy hit. So, to reduce the complexity, we can utilize undeniable stream handling structure and after that Kafka streams comes into picture with the accompanying objective.

7.1 Future Work

For those who like to utilize cloud conditions for enormous information handling, this may intrigue. Azure currently has an option in contrast to running Kafka on HDInsight. You can leave your current Kafka applications as is and use Event Hubs as a backend through Kafka API. Occasion Hubs is an administration for gushing information on Azure, thoughtfully fundamentally the same as Kafka. As such, Event Hubs for Kafka biological systems gives a Kafka endpoint that can be utilized by your current Kafka based applications as an option in contrast to running your own Kafka bunch. It bolsters Apache Kafka 1.0 and more up to date customer forms, and works with existing Kafka applications, including Mirror Maker - you should simply change the association string and begin gushing occasions from your applications that utilization the Kafka convention into Event Hubs. Practically, obviously, Event Hubs and Kafka are two distinct things. Be that as it may, this element can be valuable on the off chance that you as of now have administrations written to work with Kafka, and you'd like to not deal with any framework and attempt Event Hubs as a backend without changing your code.

Acknowledgements

I would especially like to thank my supervisor Vikas Sahni for cooperating with me and guiding me throughout the research and implementation process of this module. His consistent motivation has inspired me to achieve the desired outputs and complete my dissertation work efficiently. Finally, I would like to give a big thanks to my family and friends for supporting and helping me throughout.

References

URL- https://github.com/Jeevantika/ImproviseProcess

Koukis, E. and Louridas, P., 2012, December. ~ okeanos IaaS. In *EGI Community Forum 2012/EMI Second Technical Conference* (Vol. 162, p. 007). SISSA Medialab.

Armbrust, M., Xin, R.S., Lian, C., Huai, Y., Liu, D., Bradley, J.K., Meng, X., Kaftan, T., Franklin, M.J., Ghodsi, A. and Zaharia, M., 2015, May. Spark sql: Relational data processing in spark. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data* (pp. 1383-1394). ACM.

Koukis, E. and Louridas, P., 2012, December. ~ okeanos IaaS. In *EGI Community Forum 2012/EMI Second Technical Conference* (Vol. 162, p. 007) SISSA Medialab.

Mavridis, I. and Karatza, H. (2019). Performance evaluation of cloud-based log file analysis with Apache Hadoop and Apache Spark.
Opcito Technologies. (2019). Building a real-time data pipeline using Spark Streaming and Kafka - Opcito Technologies. [online] Available at: https://www.opcito.com/blogs/building-a-real-time-data-pipeline-using-spark-streaming-and-kafka/ [Accessed 4 Apr. 2019].

Databricks. (2019). *Processing Data in Apache Kafka with Structured Streaming in Apache Spark 2.2*. [online] Available at: https://databricks.com/blog/2017/04/26/processing-data-in-apache-kafka-with-structured-streaming-in-apache-spark-2-2.html [Accessed 5 Apr. 2019].

Jacinto Arias, Jose A Gamez, and Jose M Puerta. Learning distributed discrete bayesian network classifiers under mapreduce with apache spark. Knowledge-Based Systems, 117:16–26, 2017.

Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. Communications of the ACM, 51(1):107–113, 2008.

Georgios Gousios. Big data software analytics with apache spark. In Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings, pages 542–543. ACM, 2018.

S. Cha and M. Wachowicz. Developing a real-time data analytics framework using Hadoop. 2017 IEEE International Congress on Big Data, pages 657–660, June 2017

A. Bifet, “Mining Big Data in real time,” Informatica, 37(1), 2018, Pages 15 - 20.

Babak Yadranjiaghdam, Seyedfaraz Yasrobi, and Nasseh Tabrizi. In *Big Data (BigData Congress), 2017 IEEE International Congress on*, pages 329–336. IEEE, 2017.

URL- https://www.kaggle.com/xvivancos/market-basket-analysis

Talend Real-Time Open Source Data Integration Software. (2019). *MapReduce 101: What It Is & How to Get Started - Talend*. [online] Available at: https://www.talend.com/resources/what-is-mapreduce/ [Accessed 14 Apr. 2019].

Dei.unipd.it. (2019). *Most useful Spark methods — Big Data course documentation*. [online] Available at: http://www.dei.unipd.it/~capri/BDC/HOMEWORKS/api.html [Accessed 14 Apr. 2019].

Díaz, M., Martín, C. and Rubio, B. (2019). *State-of-the-art, challenges, and open issues in the integration of Internet of things and cloud computing*. (Díaz, Martín and Rubio, 2019)

Lenadroid.github.io. (2019). *Processing streams of data with Apache Kafka and Spark: ingestion, processing, reaction, examples*. [online] Available at: https://lenadroid.github.io/posts/distributed-data-streaming-action.html [Accessed 15 Apr. 2019].

dzone.com. (2019). *Spark Streaming vs. Kafka Streaming - DZone Big Data*. [online] Available at: https://dzone.com/articles/spark-streaming-vs-kafka-stream-1 [Accessed 15 Apr. 2019].

Ken Hess. Hadoop vs. spark: The new age of big data, Feb 2016. URL
.
K Jayamalini and M Ponnavaikko. Research on web data mining concepts, techniques and applications. In *Algorithms, Methodology, Models and Applications in Emerging Technologies (ICAMMAET), 2017 International Conference on*, pages 1–5. IEEE, 2017.

Kyong-Ha Lee, Yoon-Joon Lee, Hyunsik Choi, Yon Dohn Chung, and Bongki Moon. Parallel data processing with mapreduce: a survey. *AcM sIGMoD Record*, 40(4): 11–20, 2012.

Wang Lei and Liu Chong. Implementation and application of web data mining based on cloud computing. In *2015 International Conference on Intelligent Transportation, Big Data & Smart City (ICITBS)*, pages 470–473. IEEE, 2015.

Anand Singh Rajawat and Akhilesh R Upadhya. Big web data mining for predicting usage behaviour using fusion map reduce model. 2018.

R Shyam, Bharathi Ganesh HB, Sachin Kumar, Prabaharan Poornachandran, and KP Soman. Apache spark a big data analytics platform for smart grid. *Procedia Technology*, 21:171–178, 2015.

Mitali Srivastava, RAKHI Garg, and PK Mishra. A mapreduce-based parallel data cleaning algorithm in web usage mining. *International Journal of Computer Science and Applications*, 14(2), 2017.

Anubhav Tarar. How does spark use mapreduce? - dzone big data, Jan 2018. URL https://dzone.com/articles/how-does-spark-use-mapreduce.
Wenzheng Zhu and Changhoon Lee. A new approach to web data mining based on cloud computing. *Journal of Computing Science and Engineering*, 8(4):181–186, 2014.

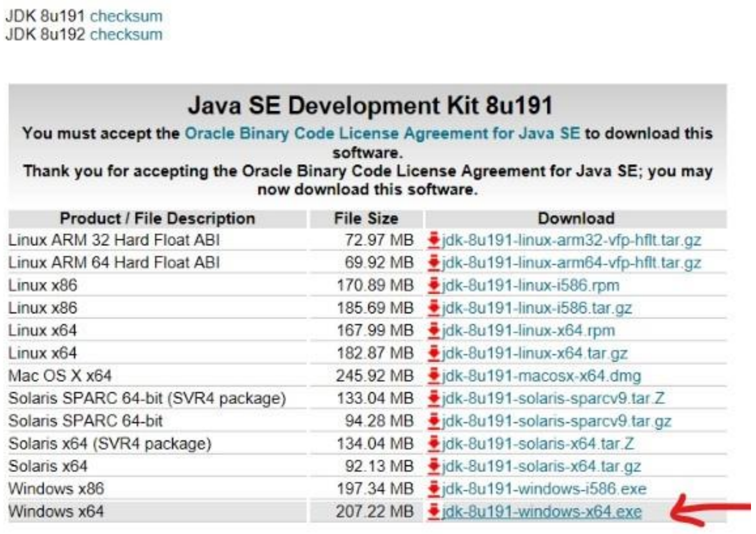
8 Appendix - Configuration Manual

Configuration manual gives a detail knowledge about software used, how to install and Implement. Following sections give a brief information about executing the project and the observations. For evaluation, downloading and installing the software is needed.

1. Downloading Software

1.1 Java Development Kit

The Java SE Development Kit is a development environment that is used for building applications, applets, and other components which makes use of Java Programming Language. The Java Development Kit also includes some tools that are used for development and testing of programs which are written in Java Programming Language and that runs on the Java platform. The JDK can be downloaded from the link provided in references (*Java SE Development Kit 8 - Downloads*; n.d.) Choose the setup as per the configuration of the systems where the software to be installed.



JDK 8u191 checksum
JDK 8u192 checksum

Java SE Development Kit 8u191

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

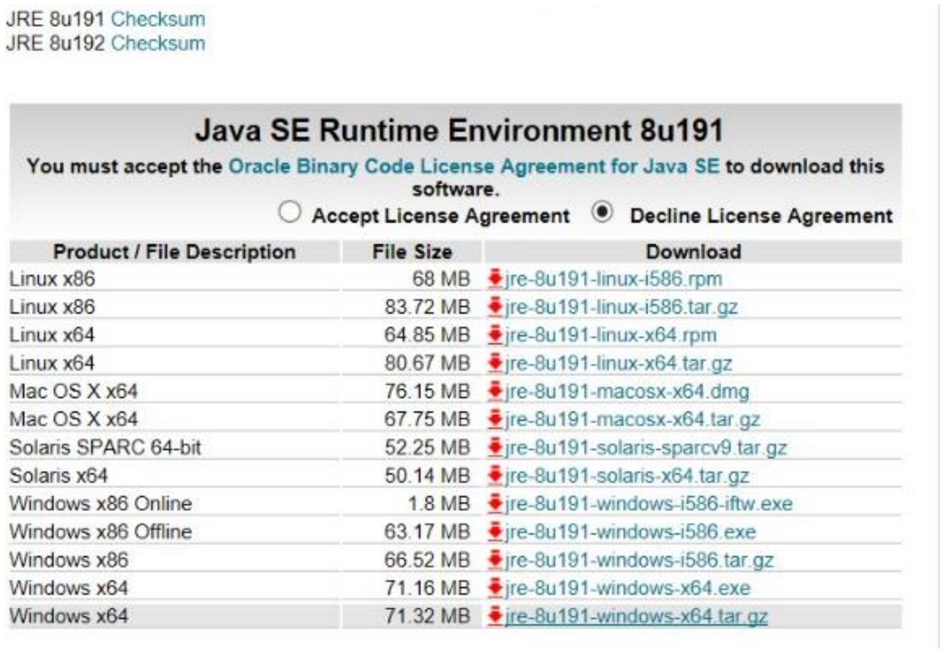
Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.

Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	72.97 MB	jdk-8u191-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 Hard Float ABI	69.92 MB	jdk-8u191-linux-arm64-vfp-hflt.tar.gz
Linux x86	170.89 MB	jdk-8u191-linux-i586.rpm
Linux x86	185.69 MB	jdk-8u191-linux-i586.tar.gz
Linux x64	167.99 MB	jdk-8u191-linux-x64.rpm
Linux x64	182.87 MB	jdk-8u191-linux-x64.tar.gz
Mac OS X x64	245.92 MB	jdk-8u191-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	133.04 MB	jdk-8u191-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	94.28 MB	jdk-8u191-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	134.04 MB	jdk-8u191-solaris-x64.tar.Z
Solaris x64	92.13 MB	jdk-8u191-solaris-x64.tar.gz
Windows x86	197.34 MB	jdk-8u191-windows-i586.exe
Windows x64	207.22 MB	jdk-8u191-windows-x64.exe

Figure 1: Java SE Development Kit

1.2 Java SE Runtime Environment 8

When there is a need to run a Java program, but there is no development required for this purpose we need to download Java Runtime environment which is also Called as JRE. If there is a need to develop applications for Java, Java Development Kit needs to be downloaded also called as JDK. This JDK includes JRE due to which you there is no need to download them separately. The JRE can be downloaded with the link provided in the references.



JRE 8u191 Checksum
JRE 8u192 Checksum

Java SE Runtime Environment 8u191

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

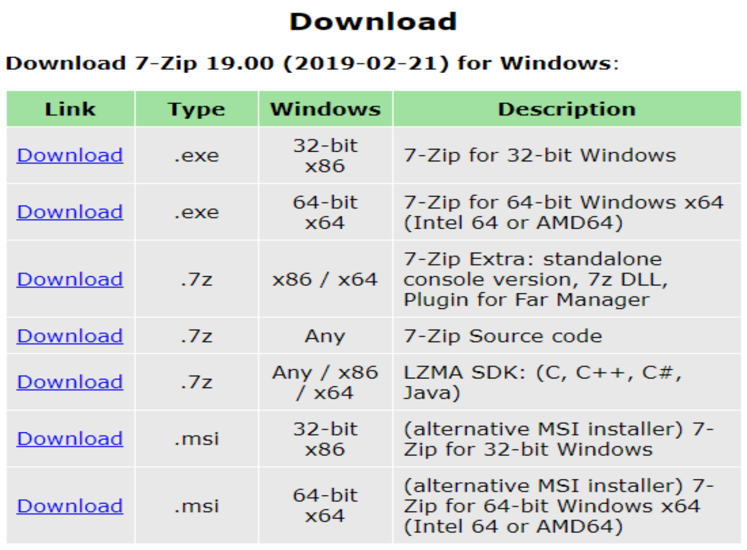
○ Accept License Agreement [● Decline License Agreement]

Product / File Description	File Size	Download
Linux x86	68 MB	jre-8u191-linux-i586.rpm
Linux x86	83.72 MB	jre-8u191-linux-i586.tar.gz
Linux x64	64.85 MB	jre-8u191-linux-x64.rpm
Linux x64	80.67 MB	jre-8u191-linux-x64.tar.gz
Mac OS X x64	76.15 MB	jre-8u191-macosx-x64.dmg
Mac OS X x64	67.75 MB	jre-8u191-macosx-x64.tar.gz
Solaris SPARC 64-bit	52.25 MB	jre-8u191-solaris-sparcv9.tar.gz
Solaris x64	50.14 MB	jre-8u191-solaris-x64.tar.gz
Windows x86 Online	1.8 MB	jre-8u191-windows-i586-iftw.exe
Windows x86 Offline	63.17 MB	jre-8u191-windows-i586.exe
Windows x86	66.52 MB	jre-8u191-windows-i586.tar.gz
Windows x64	71.16 MB	jre-8u191-windows-x64.exe
Windows x64	71.32 MB	jre-8u191-windows-x64.tar.gz

Figure 2: Java SE Runtime Environment 8

1.3 7-ZIP Download

7-Zip is a file archiver having a high compression ratio. It is a free software which is open source. Most part o the code comes under GNU LGPL license, whereas the remaining code comes under the BSD clause 3. 7-Zip can be used on any computer including the commercial organization's computers as well. It's completely free. The link to download 7-Zip can be found in the references (*7-zip.org*).



Download

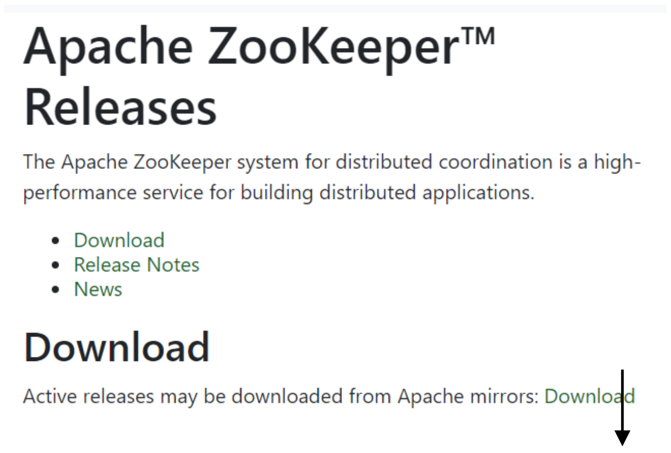
Download 7-Zip 19.00 (2019-02-21) for Windows:

Link	Type	Windows	Description
Download	.exe	32-bit x86	7-Zip for 32-bit Windows
Download	.exe	64-bit x64	7-Zip for 64-bit Windows x64 (Intel 64 or AMD64)
Download	.7z	x86 / x64	7-Zip Extra: standalone console version, 7z DLL, Plugin for Far Manager
Download	.7z	Any	7-Zip Source code
Download	.7z	Any / x86 / x64	LZMA SDK: (C, C++, C#, Java)
Download	.msi	32-bit x86	(alternative MSI installer) 7-Zip for 32-bit Windows
Download	.msi	64-bit x64	(alternative MSI installer) 7-Zip for 64-bit Windows x64 (Intel 64 or AMD64)

Figure 3: Download 7-Zip

1.4 Download Apache Zookeeper New Releases.

Apache Zookeeper is used to develop open-source and highly maintained server which provides access to highly reliable distributed coordination. Zookeeper is a centralized medium which is used for maintaining the configuration information, providing distributed sync and group service. These services are used by distributed applications in other forms. The link to download apache Zookeeper is provided in the references (*http://zookeeper.apache.org/releases.html*).



Apache ZooKeeper™ Releases

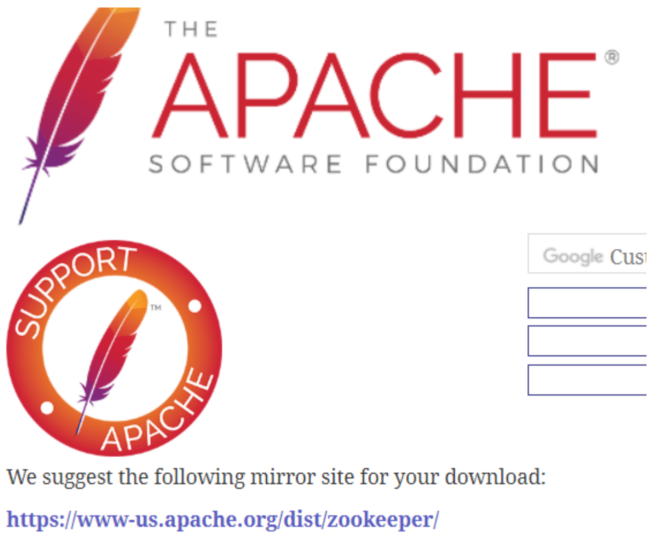
The Apache ZooKeeper system for distributed coordination is a high-performance service for building distributed applications.

- Download
- Release Notes
- News

Download

Active releases may be downloaded from Apache mirrors: Download

Figure 4: Step 1: Apache Zookeeper Download



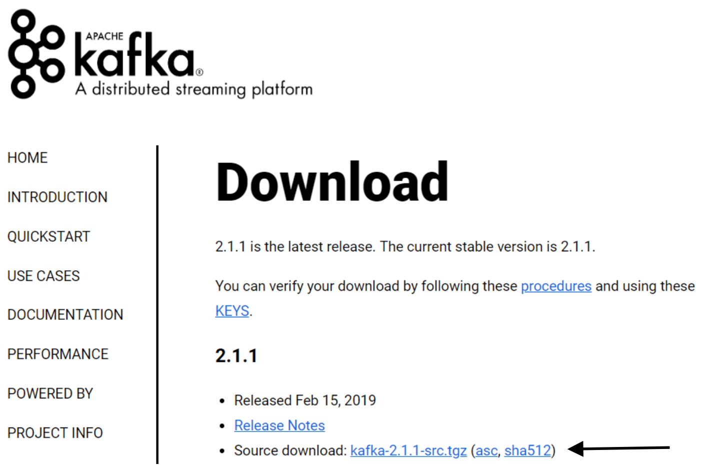
We suggest the following mirror site for your download:

https://www-us.apache.org/dist/zookeeper/

Figure 5: Step 2 Download newest release

1.5 Download Apache Kafka

Apache Kafka is a distributed streaming platform. It is used for building real-time streaming data pipelines that has data in between applications and systems. Kafka is also used for building real-time streaming applications which change itself to the streams of the data. The link to download Apache Kafka is provided in the references (http://kafka.apache.org/downloads).


APACHE kafka A distributed streaming platform

HOME
INTRODUCTION
QUICKSTART
USE CASES
DOCUMENTATION
PERFORMANCE
POWERED BY
PROJECT INFO

Download

2.1.1 is the latest release. The current stable version is 2.1.1.

You can verify your download by following these procedures and using these KEYS.

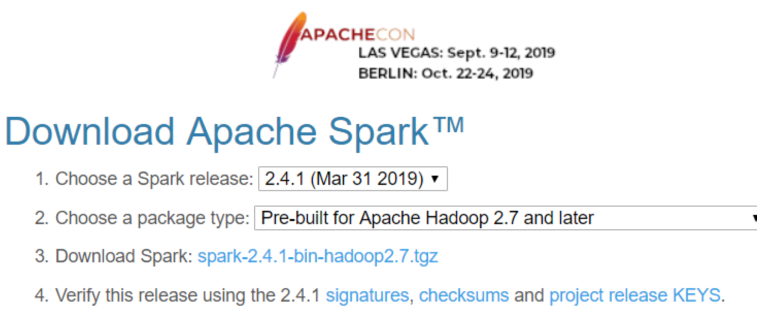
2.1.1

- Released Feb 15, 2019
- Release Notes
- Source download: kafka-2.1.1-src.tgz (asc, sha512)

Figure 6: Download Apache Kafka

1.6 Download Apache Spark

Apache Spark is an exceptionally fast analytics engine that is used for Machine Learining, AI and Big Data.Apache Spark is 100% open source, facilitated at the vendor autonomous Apache Software Foundation. Fast, adaptable, and developer friendly, Apache Spark is the main stage for vast scale SQL, data and stream processing, stream handling, and AI.


APACHECON
LAS VEGAS: Sept. 9-12, 2019
BERLIN: Oct. 22-24, 2019

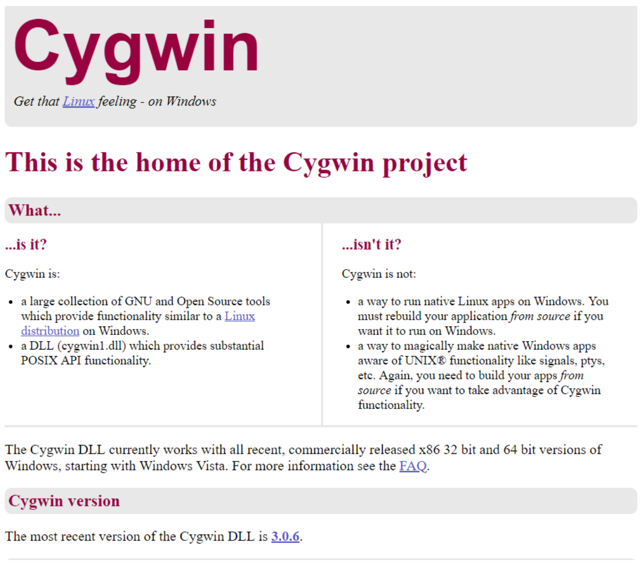
Download Apache Spark™

1. Choose a Spark release: 2.4.1 (Mar 31 2019)
2. Choose a package type: Pre-built for Apache Hadoop 2.7 and later
3. Download Spark: spark-2.4.1-bin-hadoop2.7.tgz
4. Verify this release using the 2.4.1 signatures, checksums and project release KEYS.

Figure 7: Download Apache Spark

1.7 Download Cygwin

Cygwin is a vast accumulation of GNU and Open Source apparatuses which give usefulness like a Linux appropriation on Windows. A DLL (cygwin1.dll) which gives generous POSIX API usefulness.


Cygwin

Get that Linux feeling - on Windows

This is the home of the Cygwin project

What...

...is it?

Cygwin is:

- a large collection of GNU and Open Source tools which provide functionality similar to a Linux distribution on Windows.
- a DLL (cygwin1.dll) which provides substantial POSIX API functionality.

...isn't it?

Cygwin is not:

- a way to run native Linux apps on Windows. You must rebuild your application *from source* if you want it to run on Windows.
- a way to magically make native Windows apps aware of UNIX® functionality like signals, ptys, etc. Again, you need to build your apps *from source* if you want to take advantage of Cygwin functionality.

The Cygwin DLL currently works with all recent, commercially released x86 32 bit and 64 bit versions of Windows, starting with Windows Vista. For more information see the FAQ.

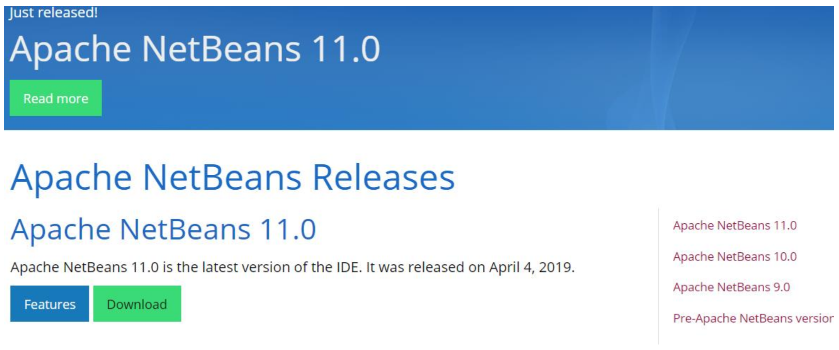
Cygwin version

The most recent version of the Cygwin DLL is 3.0.6.

Figure 8: Download Cygwin

1.8 Download Apache NetBeans

Apache NetBeans is significantly more than a editor. It features source code semantically, lets you effectively refactor code, with a scope of convenient and incredible assets. Apache NetBeans gives editors, wizards, and formats to enable you to make applications in Java, PHP and numerous different languages.


Just released!

Apache NetBeans 11.0

Read more

Apache NetBeans Releases

Apache NetBeans 11.0

Apache NetBeans 11.0 is the latest version of the IDE. It was released on April 4, 2019.

Features Download

Apache NetBeans 11.0

Apache NetBeans 10.0

Apache NetBeans 9.0

Pre-Apache NetBeans versior

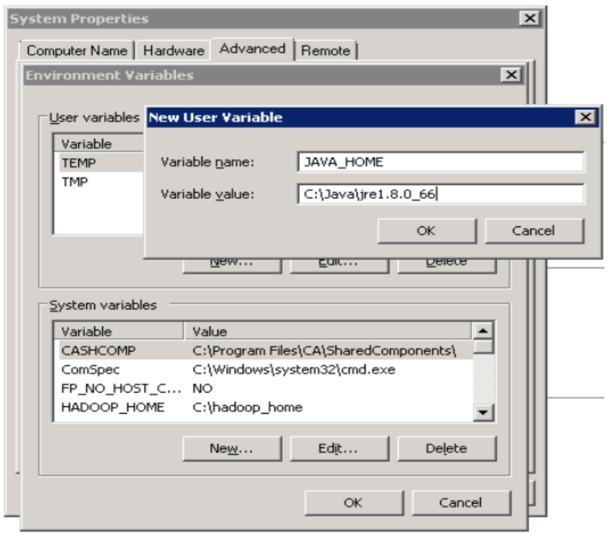
Figure 9: Download Apache Netbeans.

2 Installation

In this section step by step installation procedures are explained for each software.

2.1 Java Development Kit Setup

1. Start the JRE Installation setup and click the” Change destination folder” and install.
2. Change the path of the directory without any spaces and click next.
3. Next, open the system environment variables by clicking on control panel > system > Advanced System settings > Environment variables.
4. Click New User Variable button in the section of user variables and type JAVA_HOME in the section of variable name and the path where you have your JRE installed in the variable value.

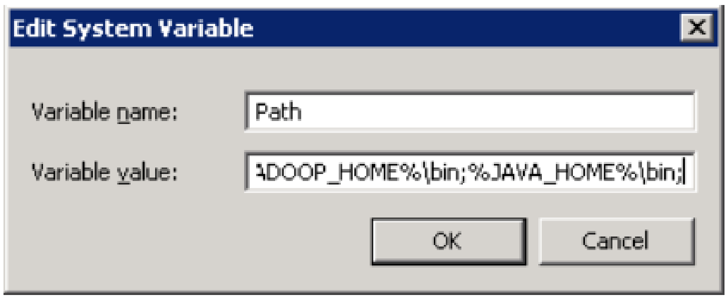


Variable name: JAVA_HOME
Variable value: C:\Java\jre1.8.0_66

Variable	Value
CASHCOMP	C:\Program Files\CA\SharedComponents\
ComSpec	C:\Windows\system32\cmd.exe
FP_NO_HOST_C...	NO
HADOOP_HOME	C:\hadoop_home

Figure 10: Set Path for JDk- 1

5. Click OK. Now search for the variable in the section name “System Variable” in the section called “Environment Variables” and open it.
6. Edit the path and type the following text “; %JAVA_HOME%\bin” and the very end where already entered text ends as shown in the image below



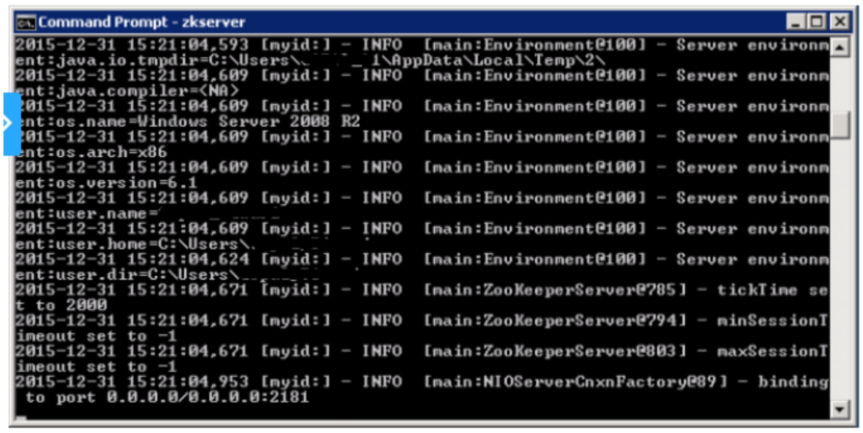
Variable name: Path
Variable value: ADOOP_HOME%\bin;%JAVA_HOME%\bin;

Figure 11: Set Path for JDK- 2

7. Check if the installation has been done properly and to do so, open command prompt and type “java -version” if everything is installed correctly you will see a the current installed version of your java.

2.2 Apache Zookeeper Installation

1. Go to the directory where you have downloaded Zookeeper and go to its config.
2. Once the Config file is found rename your “zoo_sample.cfg” to “zoo.cfg”.
3. Edit the zoo.cgf in notepad++ or any other text editor.
4. Search for dataDir=/tmp/zookeeper and replace it with: \zookeeper- (your version) \data.
5. Set the path in system environment variable similarly done for Java.
6. Add ZOOKEEPER_HOME = C:\zookeeper-3.4.7 to the system variable.
7. Run the zookeeper using the command prompt by typing zkserver.
8. If everything is installed properly you will see the output as the image below.

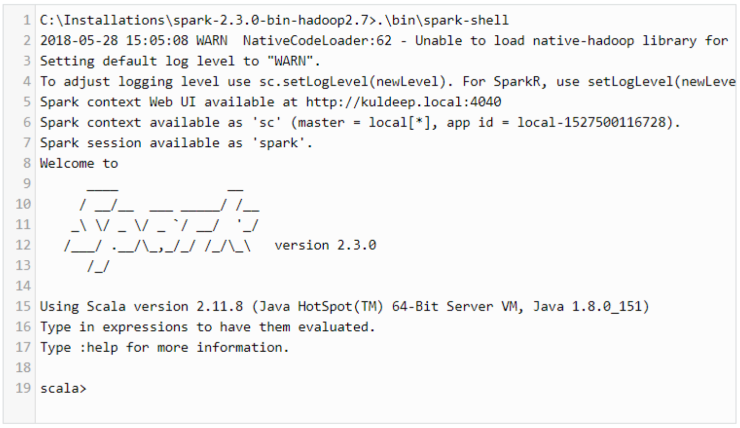


```
2015-12-31 15:21:04,593 [myid:] - INFO  [main:Environment@100] - Server environm
ent:java.io.tmpdir=C:\Users\[redacted]1\AppData\Local\Temp\2\
2015-12-31 15:21:04,609 [myid:] - INFO  [main:Environment@100] - Server environm
ent:java.compiler=<NA>
2015-12-31 15:21:04,609 [myid:] - INFO  [main:Environment@100] - Server environm
ent:os.name=Windows Server 2008 R2
2015-12-31 15:21:04,609 [myid:] - INFO  [main:Environment@100] - Server environm
ent:os.arch=x86
2015-12-31 15:21:04,609 [myid:] - INFO  [main:Environment@100] - Server environm
ent:os.version=6.1
2015-12-31 15:21:04,609 [myid:] - INFO  [main:Environment@100] - Server environm
ent:user.name=[redacted]
2015-12-31 15:21:04,609 [myid:] - INFO  [main:Environment@100] - Server environm
ent:user.home=C:\Users\[redacted]
2015-12-31 15:21:04,624 [myid:] - INFO  [main:Environment@100] - Server environm
ent:user.dir=C:\Users\[redacted]
2015-12-31 15:21:04,671 [myid:] - INFO  [main:ZooKeeperServer@785] - tickTime se
t to 2000
2015-12-31 15:21:04,671 [myid:] - INFO  [main:ZooKeeperServer@794] - minSessionT
imeout set to -1
2015-12-31 15:21:04,671 [myid:] - INFO  [main:ZooKeeperServer@803] - maxSessionT
imeout set to -1
2015-12-31 15:21:04,953 [myid:] - INFO  [main:NIOServerCnxnFactory@89] - binding
 to port 0.0.0.0/0.0.0.0:2181
```

Figure 12: Zookeeper running on cmd.

2.3 Apache Spark Installation

1. Download Spark from https://spark.apache.org/downloads.html and choose "Pre-built for Apache Hadoop 2.7 or more later revision."
2. Unpack downloaded spark tgz file in a directory.
3. Download Hadoop 2.7's winutils.exe and put it in the location C:\Installations\Hadoop\bin
4. Now set HADOOP_HOME = C:\loaction to your Hadoop file
5. After Installation start windows shell ignore the warnings and move

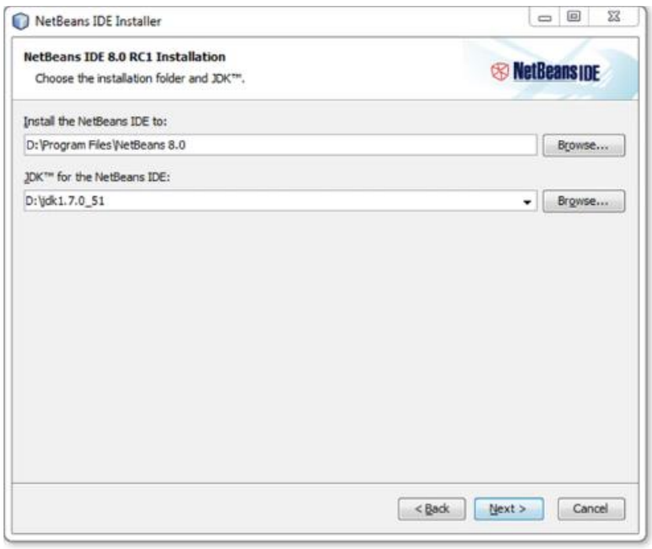


```
C:\Installations\spark-2.3.0-bin-hadoop2.7>.\bin\spark-shell
2018-05-28 15:05:08 WARN  NativeCodeLoader:62 - Unable to load native-hadoop library for
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLeve
Spark context Web UI available at http://kuldeep.local:4040
Spark context available as 'sc' (master = local[*], app id = local-1527500116728).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.3.0
      /_/

Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_151)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

Figure 13: Install Apache Spark

2.4 Install Apache NetBeans

1. Click on .exe installer file and run
2. If you downloaded the **All** or **Java EE** bundle, follow the following steps. Click Customize, make your choice and click ok.
3. Click Next
4. Accept the License
5. Chose the destination to install your IDE or select the default. Chose default JDK installation or chose your own.
6. Update your JDK to the newer version.
7. Click begin to start the installation
8. Click Finish


NetBeans IDE Installer

NetBeans IDE 8.0 RC1 Installation

Choose the installation folder and JDK™.

Install the NetBeans IDE to:

D:\Program Files\NetBeans 8.0

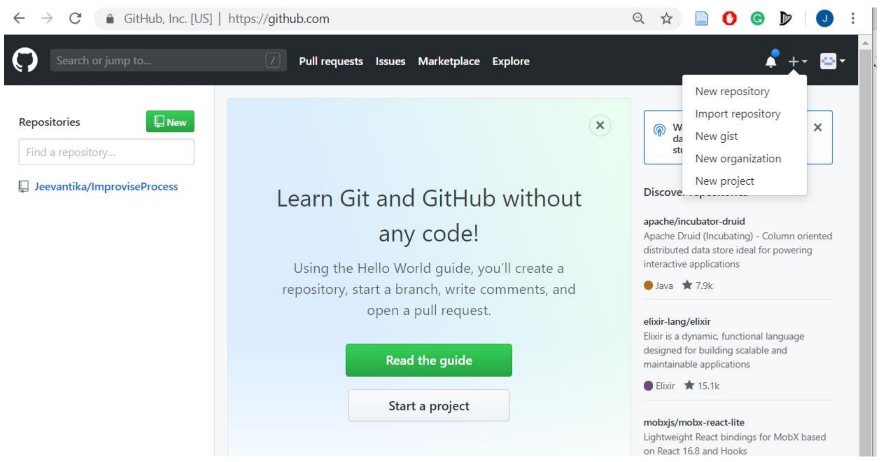
JDK™ for the NetBeans IDE:

D:\jdk1.7.0_51

Figure 14: Apache NetBeans Install

2.5 GitHub Configuration

GitHub is an online hosting service for adaptation control utilizing Git. It is for the most part utilized for computational code. It offers most of the Distributed version control and source code of Git including its own characteristics.


GitHub, Inc. [US] | https://github.com

Learn Git and GitHub without any code!

Using the Hello World guide, you'll create a repository, start a branch, write comments, and open a pull request.

apache/incubator-druid
Apache Druid (Incubating) - Column oriented distributed data store ideal for powering interactive applications
Java ★ 7.9k

elixir-lang/elixir
Elixir is a dynamic, functional language designed for building scalable and maintainable applications
Elixir ★ 15.1k

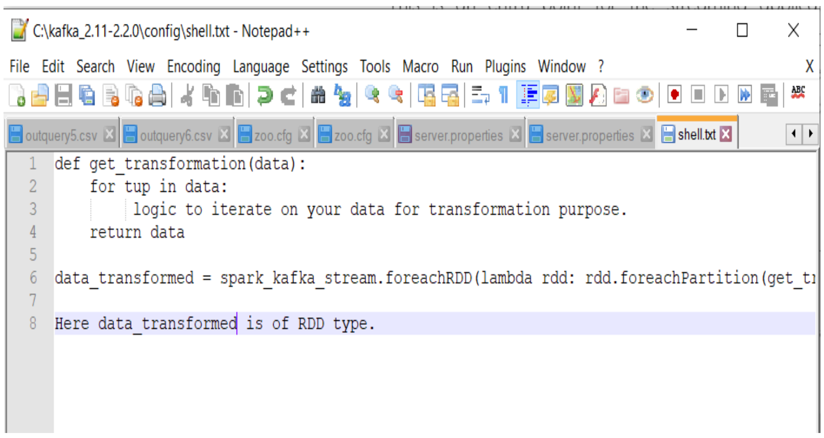
mobxjs/mobx-react-lite
Lightweight React bindings for MobX based on React 16.8 and Hooks

Figure 15: Creating new Git Repository

3 Implementation

This section describes about implementation, importing procedures for any project and executing the codes to get the expected output. Figures below shows the clear explanation of creating, implementation and executing a project with their respective outputs.

3.1 Transforming Data

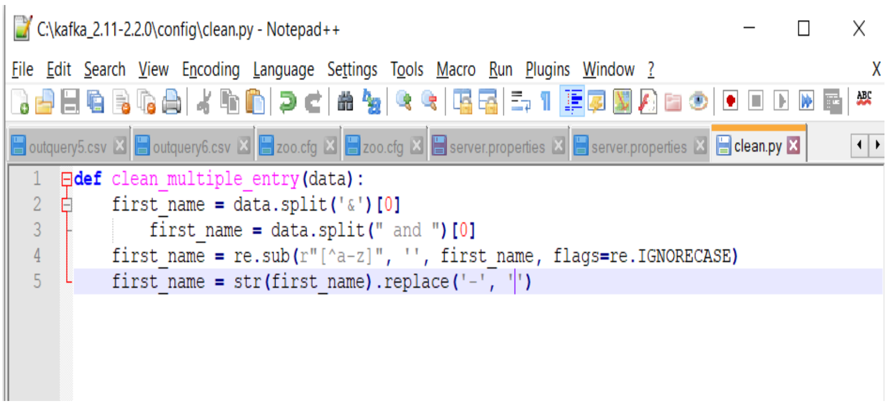


C:\kafka_2.11-2.2.0\config\shell.txt - Notepad++

```
def get_transformation(data):
    for tup in data:
        logic to iterate on your data for transformation purpose.
    return data

data_transformed = spark_kafka_stream.foreachRDD(lambda rdd: rdd.foreachPartition(get_tı

Here data_transformed is of RDD type.
```

Figure 16: Spark Streaming script for Data Transformation

3.2 Data Cleaning

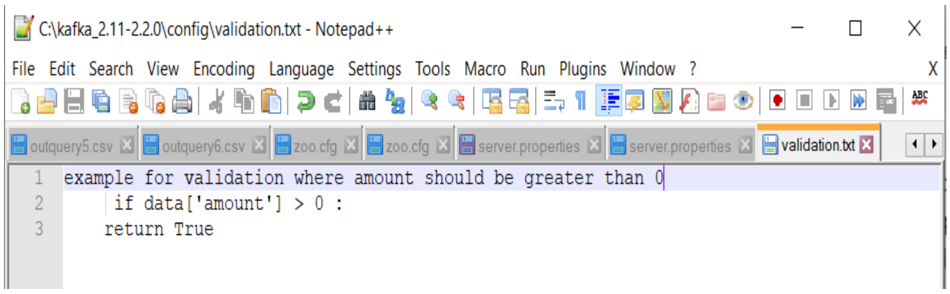


C:\kafka_2.11-2.2.0\config\clean.py - Notepad++

```
def clean_multiple_entry(data):
    first_name = data.split('&')[0]
        first_name = data.split(" and ")[0]
    first_name = re.sub(r"[^a-z]", '', first_name, flags=re.IGNORECASE)
    first_name = str(first_name).replace('-', '')
```

Figure 17: Spark Streaming python script for cleaning Data.

3.3 Data Validation



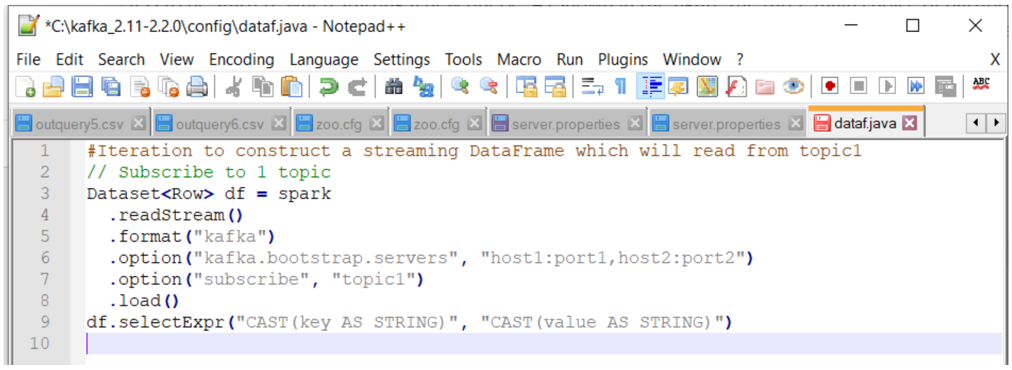
C:\kafka_2.11-2.2.0\config\validation.txt - Notepad++

```
example for validation where amount should be greater than 0
    if data['amount'] > 0 :
   return True
```

Figure 18: Spark Streaming script for data validation

3.4 Kafka Structured Streaming

1. pseudo code java for reading data from Kafka (single topic)

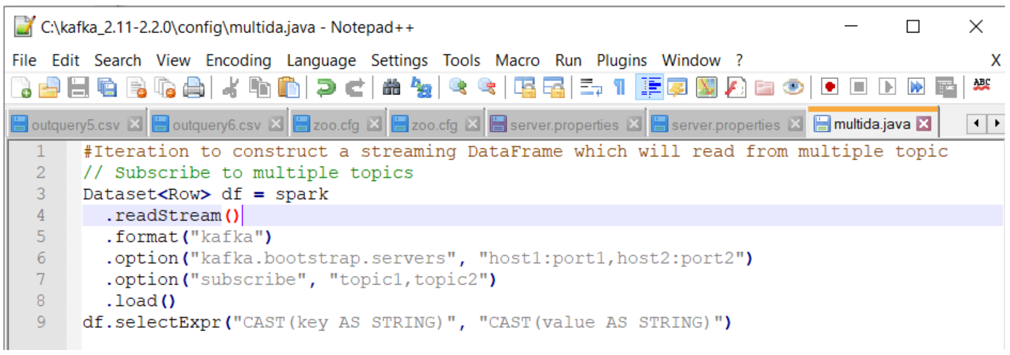


*C:\kafka_2.11-2.2.0\config\dataf.java - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
outquery5.csv outquery6.csv zoo.cfg zoo.cfg server.properties server.properties dataf.java

```
#Iteration to construct a streaming DataFrame which will read from topic1
// Subscribe to 1 topic
Dataset<Row> df = spark
  .readStream()
  .format("kafka")
  .option("kafka.bootstrap.servers", "host1:port1,host2:port2")
  .option("subscribe", "topic1")
  .load()
df.selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)")
```

Figure 19: Subscribe to 1 topic

2. Pseudo code in java for multiple topics

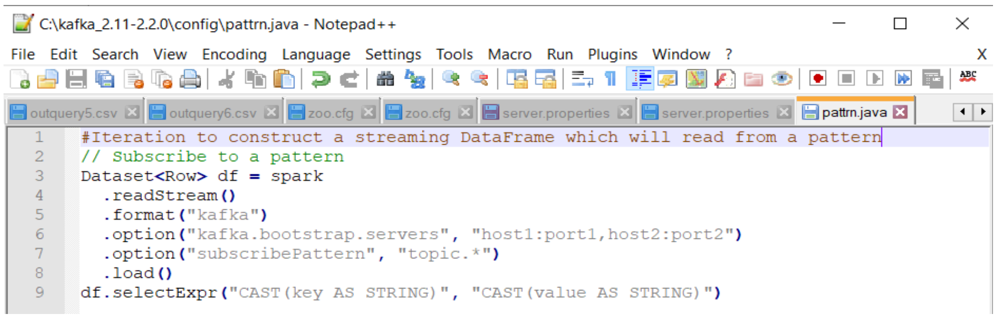


C:\kafka_2.11-2.2.0\config\multida.java - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
outquery5.csv outquery6.csv zoo.cfg zoo.cfg server.properties server.properties multida.java

```
#Iteration to construct a streaming DataFrame which will read from multiple topic
// Subscribe to multiple topics
Dataset<Row> df = spark
  .readStream()
  .format("kafka")
  .option("kafka.bootstrap.servers", "host1:port1,host2:port2")
  .option("subscribe", "topic1,topic2")
  .load()
df.selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)")
```

Figure 20: Subscribe to multiple topics

3. Pseudo code in Java for a pattern



C:\kafka_2.11-2.2.0\config\pattrn.java - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
outquery5.csv outquery6.csv zoo.cfg zoo.cfg server.properties server.properties pattrn.java

```
#Iteration to construct a streaming DataFrame which will read from a pattern
// Subscribe to a pattern
Dataset<Row> df = spark
  .readStream()
  .format("kafka")
  .option("kafka.bootstrap.servers", "host1:port1,host2:port2")
  .option("subscribePattern", "topic.*")
  .load()
df.selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)")
```

Figure 21: Subscribe to a pattern

3.5 Creating a New Java Project in NetBeans

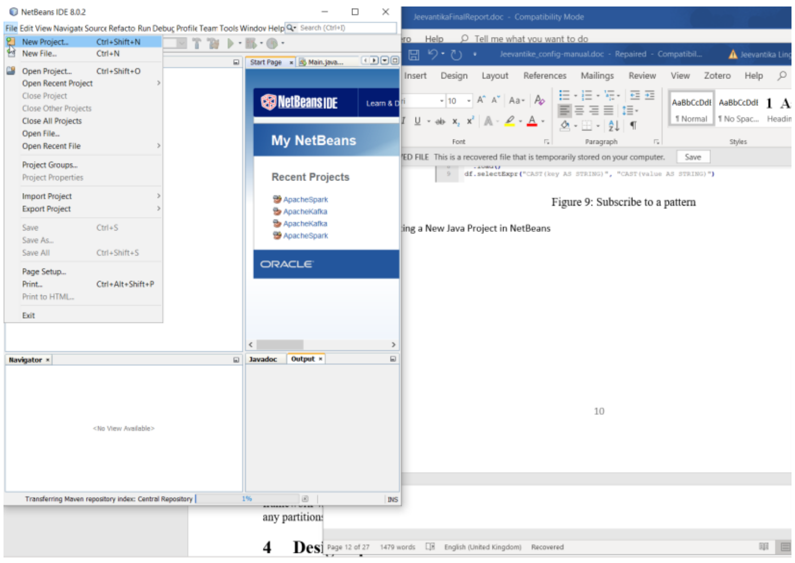


Figure 22: Create New Project

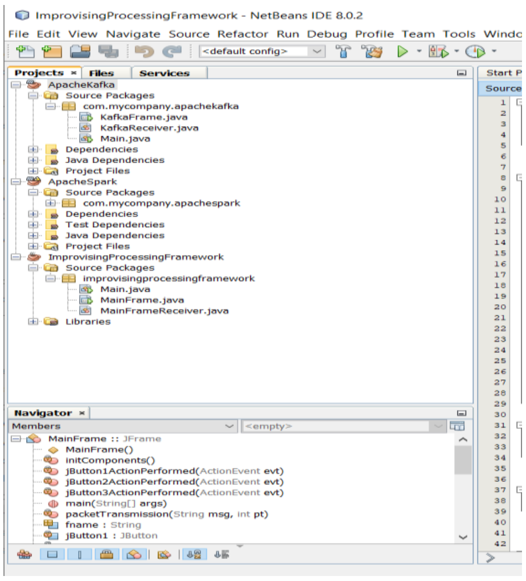
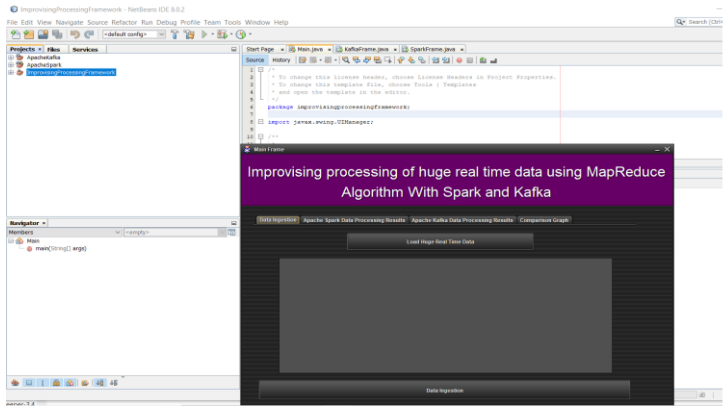


Figure 23: Required Main classes

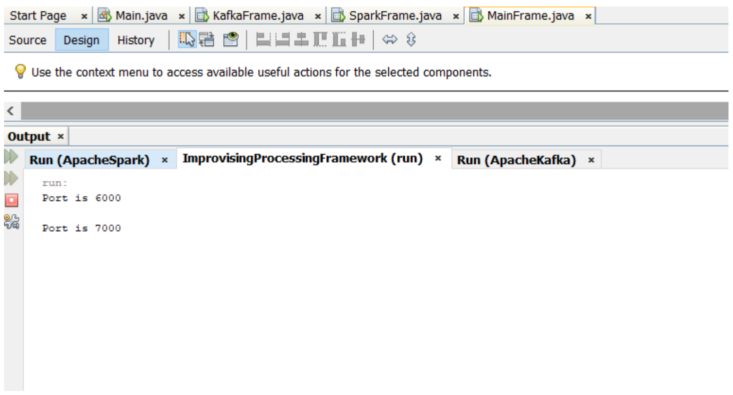
3.6 Executing the created project Apache ImprovingProcessing Framework



```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package improvisingprocessingframework;

import javax.swing.UIManager;

/**
```

Improvising processing of huge real time data using MapReduce Algorithm With Spark and Kafka

Figure 24: Improvising Processing Framework

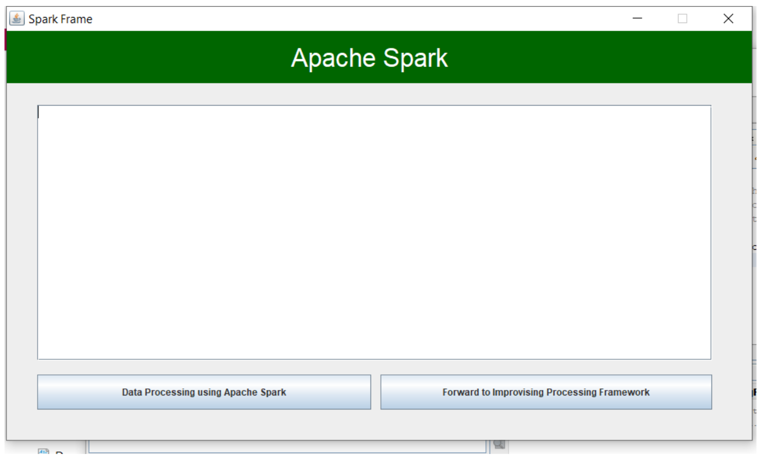


Use the context menu to access available useful actions for the selected components.

```
run:
Port is 6000

Port is 7000
```

Figure 25: Executing Port

3.7 Executing Apache Spark Framework



Spark Frame
Apache Spark
Data Processing using Apache Spark
Forward to Improvising Processing Framework

Figure 26: Apache Spark Frame

3.8 Executing Apache Kafka Framework

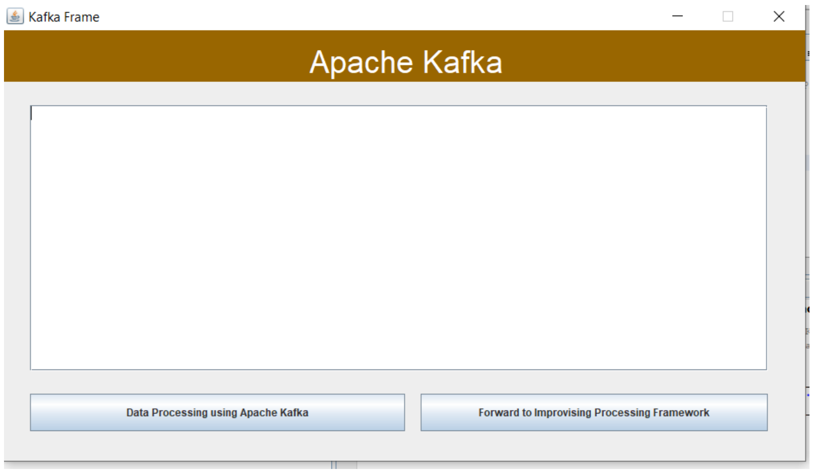
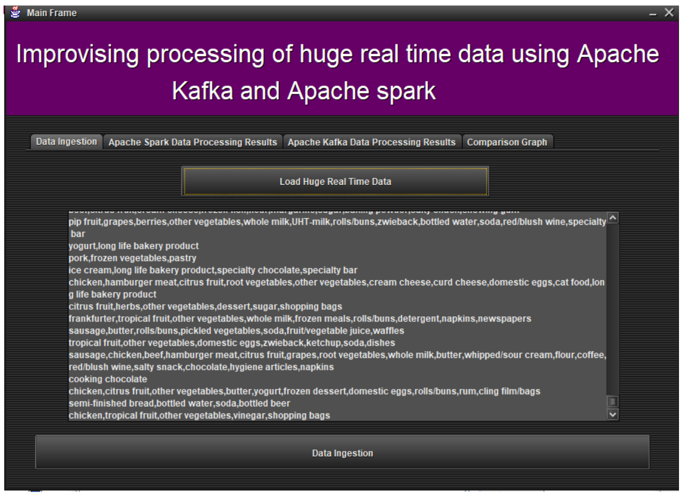

Kafka Frame
Apache Kafka
Data Processing using Apache Kafka
Forward to Improvising Processing Framework

Figure 27: Apache Kafka Frame

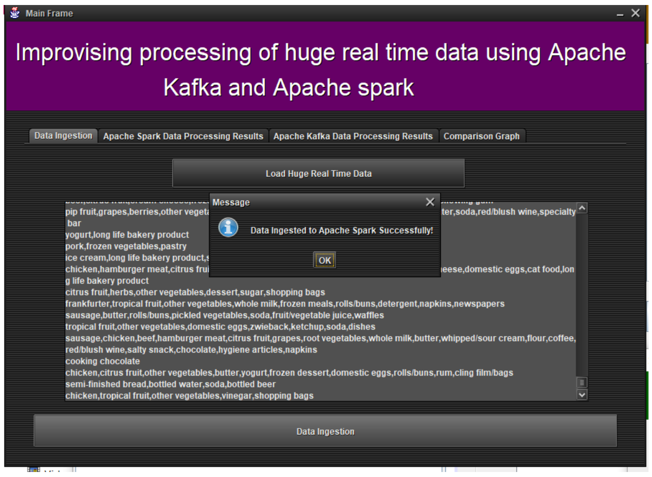
3.9 Loading Data in Main Frame


Main Frame
Improvising processing of huge real time data using Apache Kafka and Apache spark
Data Ingestion
Apache Spark Data Processing Results
Apache Kafka Data Processing Results
Comparison Graph
Load Huge Real Time Data
Data Ingestion

```
pip fruit,grapes,berries,other vegetables,whole milk,UHT-milk,rolls/buns,zwieback,bottled water,soda,red/blush wine,specialty
 bar
yogurt,long life bakery product
pork,frozen vegetables,pastry
ice cream,long life bakery product,specialty chocolate,specialty bar
chicken,hamburger meat,citrus fruit,root vegetables,other vegetables,cream cheese,curd cheese,domestic eggs,cat food,lon
g life bakery product
citrus fruit,herbs,other vegetables,dessert,sugar,shopping bags
frankfurter,tropical fruit,other vegetables,whole milk,frozen meals,rolls/buns,detergent,napkins,newspapers
sausage,butter,rolls/buns,pickled vegetables,soda,fruit/vegetable juice,waffles
tropical fruit,other vegetables,domestic eggs,zwieback,ketchup,soda,dishes
sausage,chicken,beef,hamburger meat,citrus fruit,grapes,root vegetables,whole milk,butter,whipped/sour cream,flour,coffee,
red/blush wine,salty snack,chocolate,hygiene articles,napkins
cooking chocolate
chicken,citrus fruit,other vegetables,butter,yogurt,frozen dessert,domestic eggs,rolls/buns,rum,cling film/bags
semi-finished bread,bottled water,soda,bottled beer
chicken,tropical fruit,other vegetables,vinegar,shopping bags
```

Figure 28: Data Loaded

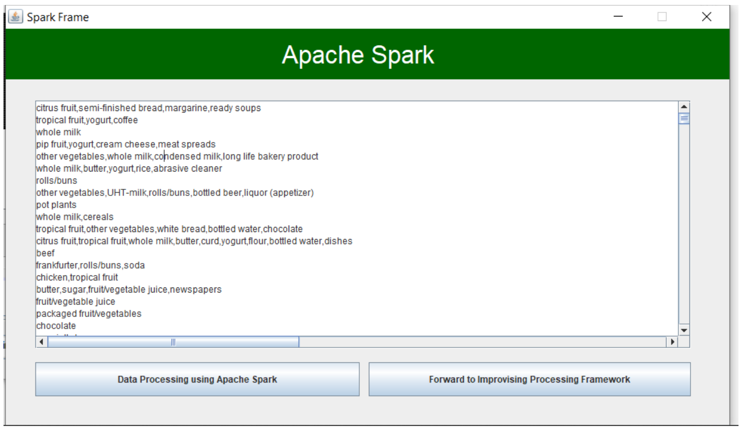
3.10 Data Ingestion


Main Frame
Improvising processing of huge real time data using Apache Kafka and Apache spark
Data Ingestion
Apache Spark Data Processing Results
Apache Kafka Data Processing Results
Comparison Graph
Load Huge Real Time Data
Message
OK
Data Ingestion

```
Data Ingested to Apache Spark Successfully!
```

```
pip fruit,grapes,berries,other veget                                      ter,soda,red/blush wine,specialty
 bar
yogurt,long life bakery product
pork,frozen vegetables,pastry
ice cream,long life bakery product,s
chicken,hamburger meat,citrus frui                                        eese,domestic eggs,cat food,lon
g life bakery product
citrus fruit,herbs,other vegetables,dessert,sugar,shopping bags
frankfurter,tropical fruit,other vegetables,whole milk,frozen meals,rolls/buns,detergent,napkins,newspapers
sausage,butter,rolls/buns,pickled vegetables,soda,fruit/vegetable juice,waffles
tropical fruit,other vegetables,domestic eggs,zwieback,ketchup,soda,dishes
sausage,chicken,beef,hamburger meat,citrus fruit,grapes,root vegetables,whole milk,butter,whipped/sour cream,flour,coffee,
red/blush wine,salty snack,chocolate,hygiene articles,napkins
cooking chocolate
chicken,citrus fruit,other vegetables,butter,yogurt,frozen dessert,domestic eggs,rolls/buns,rum,cling film/bags
semi-finished bread,bottled water,soda,bottled beer
chicken,tropical fruit,other vegetables,vinegar,shopping bags
```

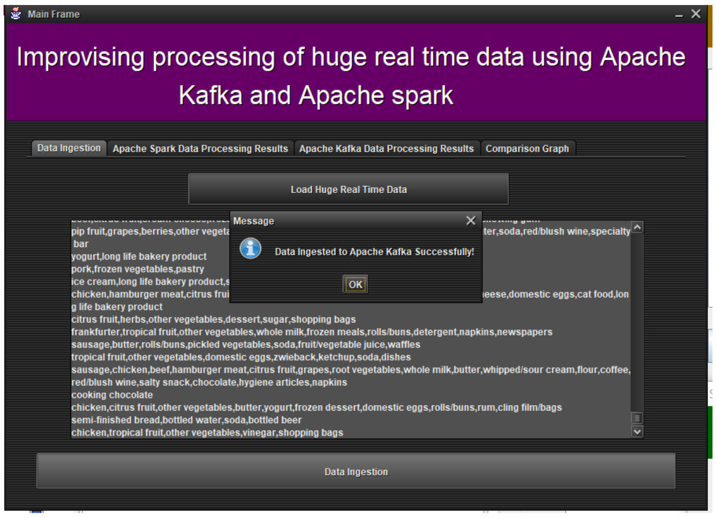
Figure 29: Data Ingested to Spark



```
citrus fruit,semi-finished bread,margarine,ready soups
tropical fruit,yogurt,coffee
whole milk
pip fruit,yogurt,cream cheese,meat spreads
other vegetables,whole milk,condensed milk,long life bakery product
whole milk,butter,yogurt,rice,abrasive cleaner
rolls/buns
other vegetables,UHT-milk,rolls/buns,bottled beer,liquor (appetizer)
pot plants
whole milk,cereals
tropical fruit,other vegetables,white bread,bottled water,chocolate
citrus fruit,tropical fruit,whole milk,butter,curd,yogurt,flour,bottled water,dishes
beef
frankfurter,rolls/buns,soda
chicken,tropical fruit
butter,sugar,fruit/vegetable juice,newspapers
fruit/vegetable juice
packaged fruit/vegetables
chocolate
```

Data Processing using Apache Spark
Forward to Improvising Processing Framework

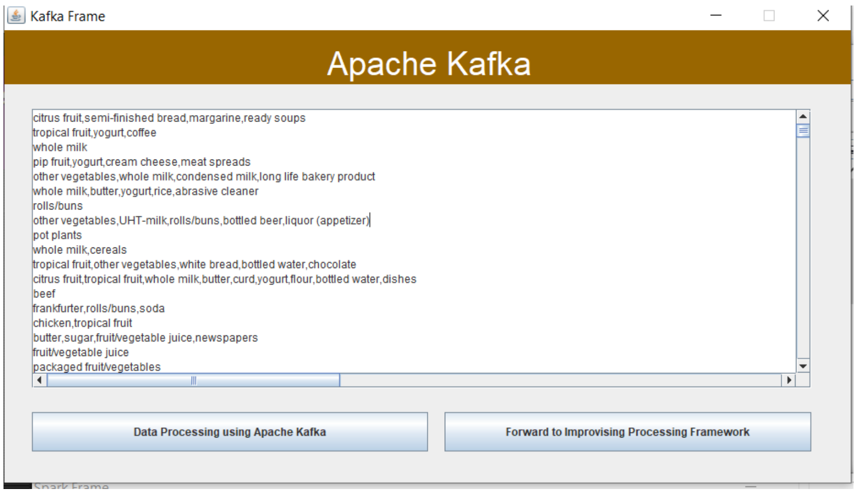
Figure 30: Spark Frame with data


Main Frame
Improvising processing of huge real time data using Apache Kafka and Apache spark
Data Ingestion
Apache Spark Data Processing Results
Apache Kafka Data Processing Results
Comparison Graph
Load Huge Real Time Data
Message
OK
Data Ingestion

Data Ingested to Apache Kafka Successfully!

```
pip fruit,grapes,berries,other vegeta                                    ter,soda,red/blush wine,specialty
 bar
yogurt,long life bakery product
pork,frozen vegetables,pastry
ice cream,long life bakery product,s
chicken,hamburger meat,citrus frui                                       eese,domestic eggs,cat food,lon
g life bakery product
citrus fruit,herbs,other vegetables,dessert,sugar,shopping bags
frankfurter,tropical fruit,other vegetables,whole milk,frozen meals,rolls/buns,detergent,napkins,newspapers
sausage,butter,rolls/buns,pickled vegetables,soda,fruit/vegetable juice,waffles
tropical fruit,other vegetables,domestic eggs,zwieback,ketchup,soda,dishes
sausage,chicken,beef,hamburger meat,citrus fruit,grapes,root vegetables,whole milk,butter,whipped/sour cream,flour,coffee,
red/blush wine,salty snack,chocolate,hygiene articles,napkins
cooking chocolate
chicken,citrus fruit,other vegetables,butter,yogurt,frozen dessert,domestic eggs,rolls/buns,rum,cling film/bags
semi-finished bread,bottled water,soda,bottled beer
chicken,tropical fruit,other vegetables,vinegar,shopping bags
```

Figure 31: Data Ingested to Apache Kafka

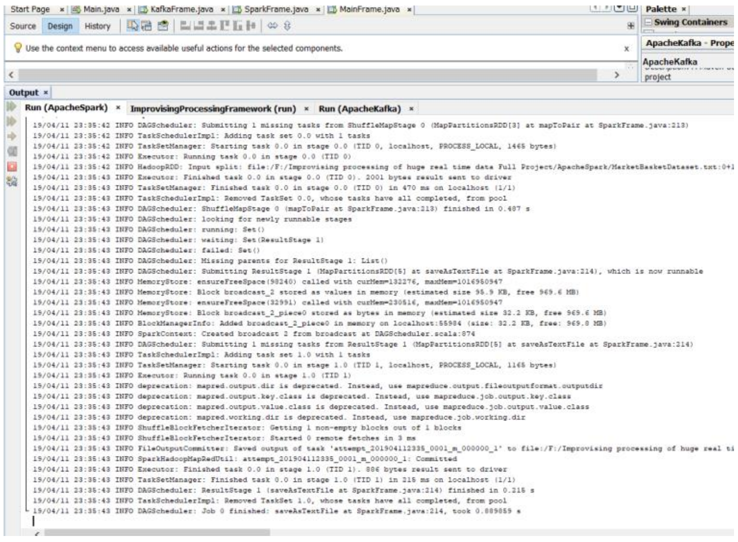


```
citrus fruit,semi-finished bread,margarine,ready soups
tropical fruit,yogurt,coffee
whole milk
pip fruit,yogurt,cream cheese,meat spreads
other vegetables,whole milk,condensed milk,long life bakery product
whole milk,butter,yogurt,rice,abrasive cleaner
rolls/buns
other vegetables,UHT-milk,rolls/buns,bottled beer,liquor (appetizer)
pot plants
whole milk,cereals
tropical fruit,other vegetables,white bread,bottled water,chocolate
citrus fruit,tropical fruit,whole milk,butter,curd,yogurt,flour,bottled water,dishes
beef
frankfurter,rolls/buns,soda
chicken,tropical fruit
butter,sugar,fruit/vegetable juice,newspapers
fruit/vegetable juice
packaged fruit/vegetables
```

Data Processing using Apache Kafka
Forward to Improvising Processing Framework

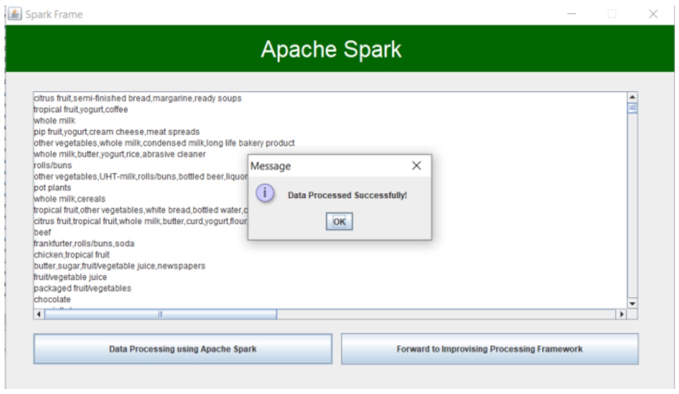
Figure 32: Spark Frame with data

3.11 Data Processing Using Spark


Start Page | Main.java | KafkaFrame.java | SparkFrame.java | MainFrame.java | Palette
Source | Design | History
Use the context menu to access available useful actions for the selected components.
Swing Containers
ApacheKafka - Prope
ApacheKafka
project
Output
Run (ApacheSpark) | ImprovisingProcessingFramework (run) | Run (ApacheKafka)

```
19/04/11 23:35:42 INFO DAGScheduler: Submitting 1 missing tasks from ShuffleMapStage 0 (MapPartitionsRDD[3] at mapToPair at SparkFrame.java:213)
19/04/11 23:35:42 INFO TaskSchedulerImpl: Adding task set 0.0 with 1 tasks
19/04/11 23:35:42 INFO TaskSetManager: Starting task 0.0 in stage 0.0 (TID 0, localhost, PROCESS_LOCAL, 1465 bytes)
19/04/11 23:35:42 INFO Executor: Running task 0.0 in stage 0.0 (TID 0)
19/04/11 23:35:42 INFO HadoopRDD: Input split: file:/F:/Improvising processing of huge real time data Full Project/ApacheSpark/MarketBasketDataset.txt:0+1
19/04/11 23:35:43 INFO Executor: Finished task 0.0 in stage 0.0 (TID 0). 2001 bytes result sent to driver
19/04/11 23:35:43 INFO TaskSetManager: Finished task 0.0 in stage 0.0 (TID 0) in 470 ms on localhost (1/1)
19/04/11 23:35:43 INFO TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have all completed, from pool
19/04/11 23:35:43 INFO DAGScheduler: ShuffleMapStage 0 (mapToPair at SparkFrame.java:213) finished in 0.487 s
19/04/11 23:35:43 INFO DAGScheduler: looking for newly runnable stages
19/04/11 23:35:43 INFO DAGScheduler: running: Set()
19/04/11 23:35:43 INFO DAGScheduler: waiting: Set(ResultStage 1)
19/04/11 23:35:43 INFO DAGScheduler: failed: Set()
19/04/11 23:35:43 INFO DAGScheduler: Missing parents for ResultStage 1: List()
19/04/11 23:35:43 INFO DAGScheduler: Submitting ResultStage 1 (MapPartitionsRDD[5] at saveAsTextFile at SparkFrame.java:214), which is now runnable
19/04/11 23:35:43 INFO MemoryStore: ensureFreeSpace(98240) called with curMem=132276, maxMem=1016950947
19/04/11 23:35:43 INFO MemoryStore: Block broadcast_2 stored as values in memory (estimated size 95.9 KB, free 969.6 MB)
19/04/11 23:35:43 INFO MemoryStore: ensureFreeSpace(32991) called with curMem=230516, maxMem=1016950947
19/04/11 23:35:43 INFO MemoryStore: Block broadcast_2_piece0 stored as bytes in memory (estimated size 32.2 KB, free 969.6 MB)
19/04/11 23:35:43 INFO BlockManagerInfo: Added broadcast_2_piece0 in memory on localhost:55984 (size: 32.2 KB, free: 969.8 MB)
19/04/11 23:35:43 INFO SparkContext: Created broadcast 2 from broadcast at DAGScheduler.scala:874
19/04/11 23:35:43 INFO DAGScheduler: Submitting 1 missing tasks from ResultStage 1 (MapPartitionsRDD[5] at saveAsTextFile at SparkFrame.java:214)
19/04/11 23:35:43 INFO TaskSchedulerImpl: Adding task set 1.0 with 1 tasks
19/04/11 23:35:43 INFO TaskSetManager: Starting task 0.0 in stage 1.0 (TID 1, localhost, PROCESS_LOCAL, 1165 bytes)
19/04/11 23:35:43 INFO Executor: Running task 0.0 in stage 1.0 (TID 1)
19/04/11 23:35:43 INFO deprecation: mapred.output.dir is deprecated. Instead, use mapreduce.output.fileoutputformat.outputdir
19/04/11 23:35:43 INFO deprecation: mapred.output.key.class is deprecated. Instead, use mapreduce.job.output.key.class
19/04/11 23:35:43 INFO deprecation: mapred.output.value.class is deprecated. Instead, use mapreduce.job.output.value.class
19/04/11 23:35:43 INFO deprecation: mapred.working.dir is deprecated. Instead, use mapreduce.job.working.dir
19/04/11 23:35:43 INFO ShuffleBlockFetcherIterator: Getting 1 non-empty blocks out of 1 blocks
19/04/11 23:35:43 INFO ShuffleBlockFetcherIterator: Started 0 remote fetches in 3 ms
19/04/11 23:35:43 INFO FileOutputCommitter: Saved output of task 'attempt_201904112335_0001_m_000000_1' to file:/F:/Improvising processing of huge real ti
19/04/11 23:35:43 INFO SparkHadoopMapRedUtil: attempt_201904112335_0001_m_000000_1: Committed
19/04/11 23:35:43 INFO Executor: Finished task 0.0 in stage 1.0 (TID 1). 886 bytes result sent to driver
19/04/11 23:35:43 INFO TaskSetManager: Finished task 0.0 in stage 1.0 (TID 1) in 215 ms on localhost (1/1)
19/04/11 23:35:43 INFO DAGScheduler: ResultStage 1 (saveAsTextFile at SparkFrame.java:214) finished in 0.215 s
19/04/11 23:35:43 INFO TaskSchedulerImpl: Removed TaskSet 1.0, whose tasks have all completed, from pool
19/04/11 23:35:43 INFO DAGScheduler: Job 0 finished: saveAsTextFile at SparkFrame.java:214, took 0.889859 s
```

Figure 33: Processing Data- Spark

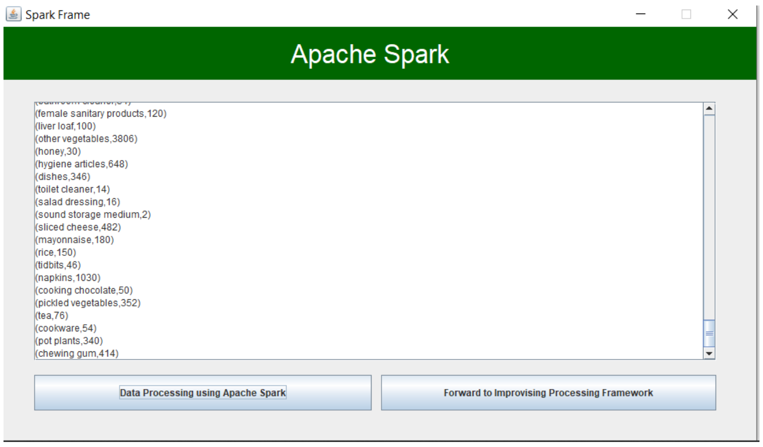

Spark Frame
Apache Spark

```
citrus fruit,semi-finished bread,margarine,ready soups
tropical fruit,yogurt,coffee
whole milk
pip fruit,yogurt,cream cheese,meat spreads
other vegetables,whole milk,condensed milk,long life bakery product
whole milk,butter,yogurt,rice,abrasive cleaner
rolls/buns
other vegetables,UHT-milk,rolls/buns,bottled beer,liquor
pot plants
whole milk,cereals
tropical fruit,other vegetables,white bread,bottled water,c
citrus fruit,tropical fruit,whole milk,butter,curd,yogurt,flour
beef
frankfurter,rolls/buns,soda
chicken,tropical fruit
butter,sugar,fruit/vegetable juice,newspapers
fruit/vegetable juice
packaged fruit/vegetables
chocolate
```

Message
Data Processed Successfully!
OK

Data Processing using Apache Spark | Forward to Improvising Processing Framework

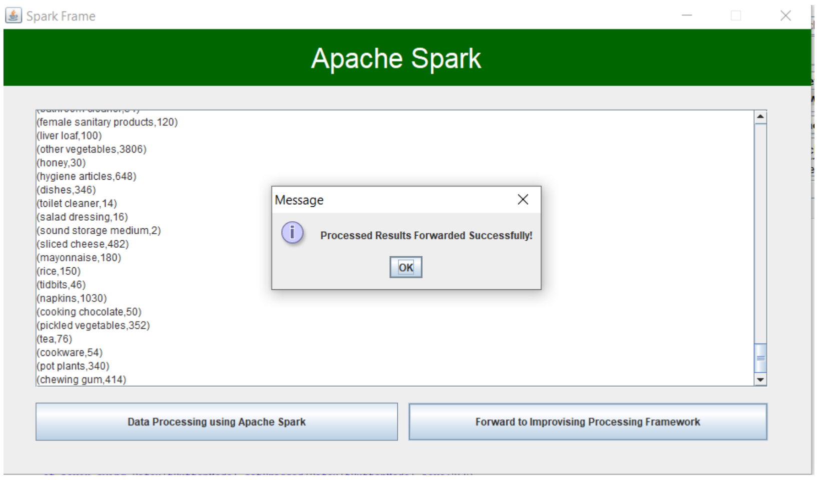
Figure 34: Successfully processed Data- Spark


Spark Frame
Apache Spark

```
(female sanitary products,120)
(liver loaf,100)
(other vegetables,3806)
(honey,30)
(hygiene articles,648)
(dishes,346)
(toilet cleaner,14)
(salad dressing,16)
(sound storage medium,2)
(sliced cheese,482)
(mayonnaise,180)
(rice,150)
(tidbits,46)
(napkins,1030)
(cooking chocolate,50)
(pickled vegetables,352)
(tea,76)
(cookware,54)
(pot plants,340)
(chewing gum,414)
```

Data Processing using Apache Spark | Forward to Improvising Processing Framework

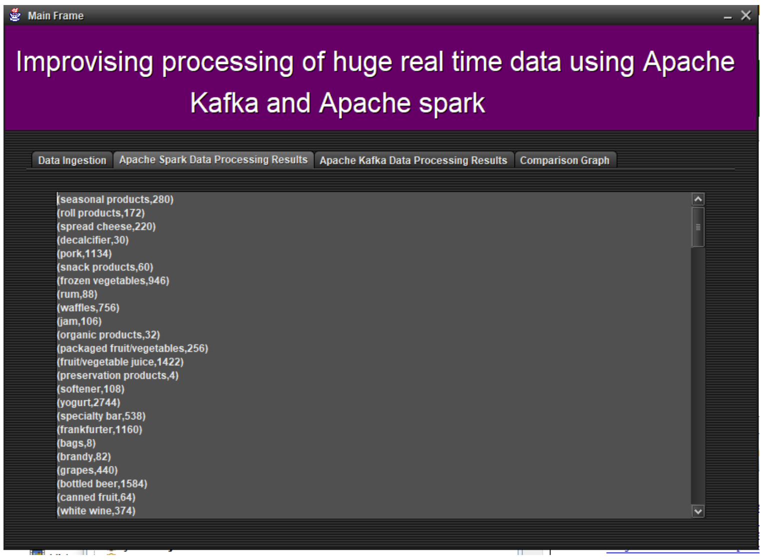
Figure 35: Processed Data in Spark


Spark Frame
Apache Spark
Message
OK
Data Processing using Apache Spark
Forward to Improvising Processing Framework

```
(female sanitary products,120)
(liver loaf,100)
(other vegetables,3806)
(honey,30)
(hygiene articles,648)
(dishes,346)
(toilet cleaner,14)
(salad dressing,16)
(sound storage medium,2)
(sliced cheese,482)
(mayonnaise,180)
(rice,150)
(tidbits,46)
(napkins,1030)
(cooking chocolate,50)
(pickled vegetables,352)
(tea,76)
(cookware,54)
(pot plants,340)
(chewing gum,414)
```

Processed Results Forwarded Successfully!

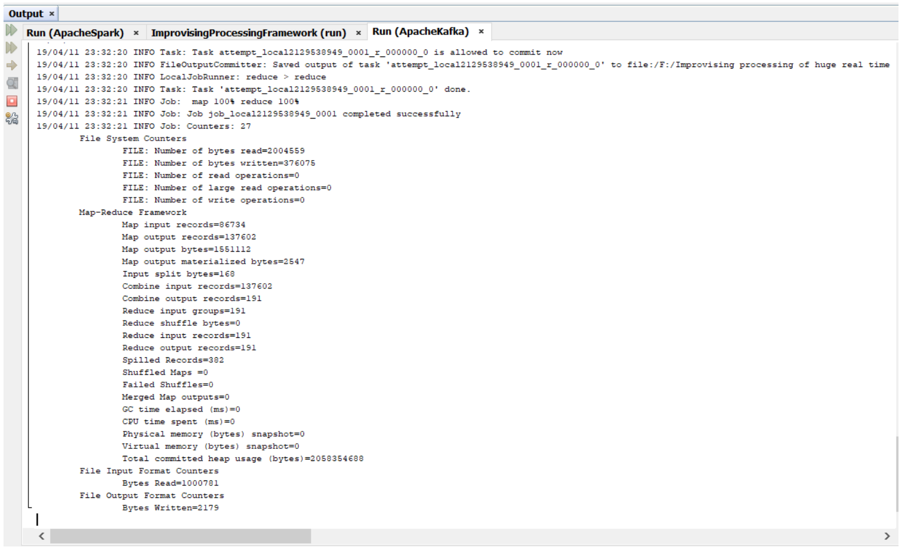
Figure 36: Forwarding Processed Data to MainFrame- Spark


Main Frame
Improvising processing of huge real time data using Apache Kafka and Apache spark
Data Ingestion
Apache Spark Data Processing Results
Apache Kafka Data Processing Results
Comparison Graph

```
(seasonal products,280)
(roll products,172)
(spread cheese,220)
(decalcifier,30)
(pork,1134)
(snack products,60)
(frozen vegetables,946)
(rum,88)
(waffles,756)
(jam,106)
(organic products,32)
(packaged fruit/vegetables,256)
(fruit/vegetable juice,1422)
(preservation products,4)
(softener,108)
(yogurt,2744)
(specialty bar,538)
(frankfurter,1160)
(bags,8)
(brandy,82)
(grapes,440)
(bottled beer,1584)
(canned fruit,64)
(white wine,374)
```

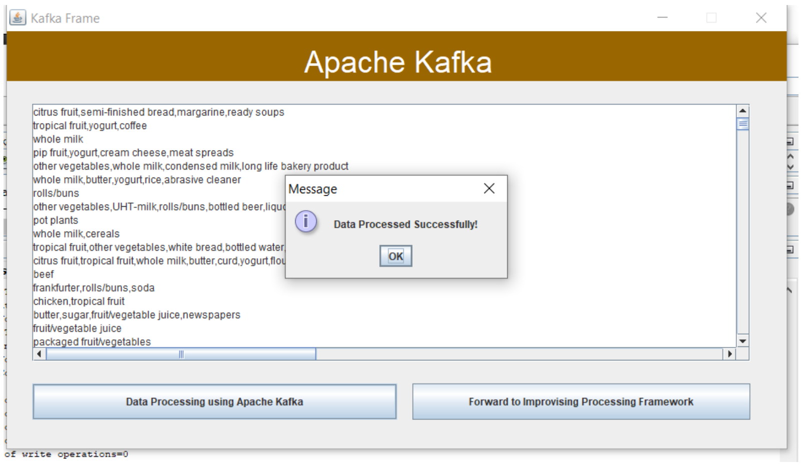
Figure 37: Spark Processed Data in Main Frame

3.12 Data Processing Using Apache Kafka



```
19/04/11 23:32:20 INFO Task: Task attempt_local2129538949_0001_r_000000_0 is allowed to commit now
19/04/11 23:32:20 INFO FileOutputCommitter: Saved output of task 'attempt_local2129538949_0001_r_000000_0' to file:/F:/Improvising processing of huge real time
19/04/11 23:32:20 INFO LocalJobRunner: reduce > reduce
19/04/11 23:32:20 INFO Task: Task 'attempt_local2129538949_0001_r_000000_0' done.
19/04/11 23:32:21 INFO Job:  map 100% reduce 100%
19/04/11 23:32:21 INFO Job: Job job_local2129538949_0001 completed successfully
19/04/11 23:32:21 INFO Job: Counters: 27
	File System Counters
		FILE: Number of bytes read=2004559
		FILE: Number of bytes written=376075
		FILE: Number of read operations=0
		FILE: Number of large read operations=0
		FILE: Number of write operations=0
	Map-Reduce Framework
		Map input records=86734
		Map output records=137602
		Map output bytes=1551112
		Map output materialized bytes=2547
		Input split bytes=168
		Combine input records=137602
		Combine output records=191
		Reduce input groups=191
		Reduce shuffle bytes=0
		Reduce input records=191
		Reduce output records=191
		Spilled Records=382
		Shuffled Maps =0
		Failed Shuffles=0
		Merged Map outputs=0
		GC time elapsed (ms)=0
		CPU time spent (ms)=0
		Physical memory (bytes) snapshot=0
		Virtual memory (bytes) snapshot=0
		Total committed heap usage (bytes)=2058354688
	File Input Format Counters
		Bytes Read=1000781
	File Output Format Counters
		Bytes Written=2179
```

Figure 38: Processing Data- Kafka



```
citrus fruit,semi-finished bread,margarine,ready soups
tropical fruit,yogurt,coffee
whole milk
pip fruit,yogurt,cream cheese,meat spreads
other vegetables,whole milk,condensed milk,long life bakery product
whole milk,butter,yogurt,rice,abrasive cleaner
rolls/buns
other vegetables,UHT-milk,rolls/buns,bottled beer,liqu
pot plants
whole milk,cereals
tropical fruit,other vegetables,white bread,bottled water,
citrus fruit,tropical fruit,whole milk,butter,curd,yogurt,flou
beef
frankfurter,rolls/buns,soda
chicken,tropical fruit
butter,sugar,fruit/vegetable juice,newspapers
fruit/vegetable juice
packaged fruit/vegetables
```

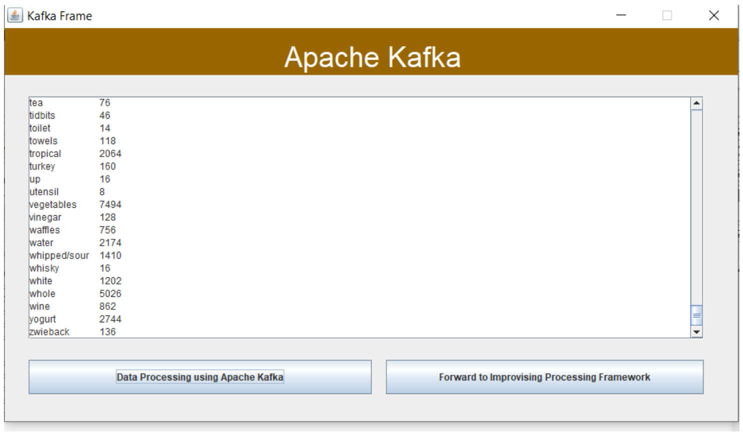
Message

Data Processed Successfully!

OK
Data Processing using Apache Kafka
Forward to Improvising Processing Framework

```
of write operations=0
```

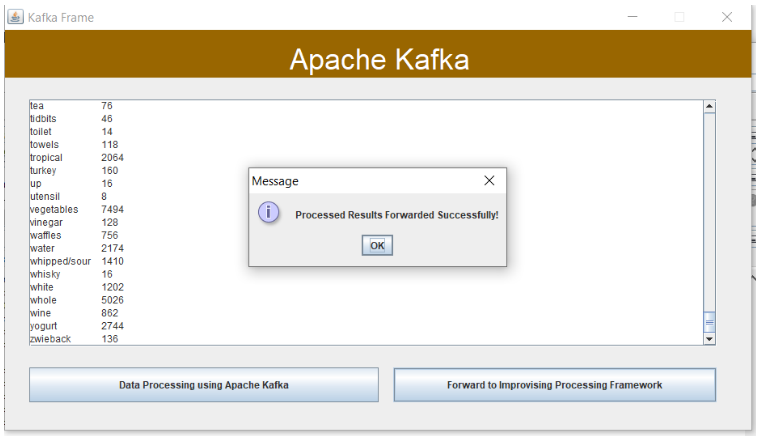
Figure 39: Successfully processed Data- Kafka



```
tea           76
tidbits       46
toilet        14
towels        118
tropical      2064
turkey        160
up            16
utensil       8
vegetables    7494
vinegar       128
waffles       756
water         2174
whipped/sour  1410
whisky        16
white         1202
whole         5026
wine          862
yogurt        2744
zwieback      136
```

Data Processing using Apache Kafka
Forward to Improvising Processing Framework

Figure 40: Processed Data in Spark



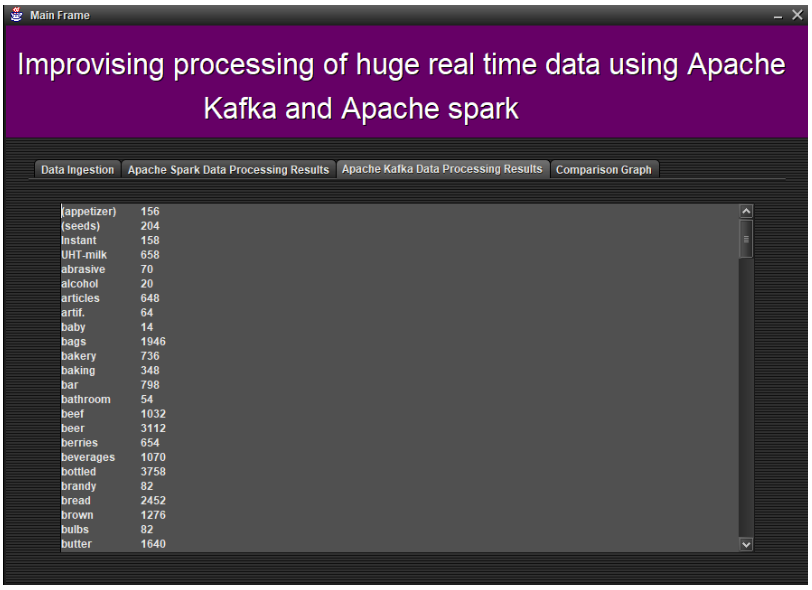
```
tea           76
tidbits       46
toilet        14
towels        118
tropical      2064
turkey        160
up            16
utensil       8
vegetables    7494
vinegar       128
waffles       756
water         2174
whipped/sour  1410
whisky        16
white         1202
whole         5026
wine          862
yogurt        2744
zwieback      136
```

Message

Processed Results Forwarded Successfully!

OK
Data Processing using Apache Kafka
Forward to Improvising Processing Framework

Figure 41: Forwarding Processed Data to MainFrame- Kafka


Main Frame
Improvising processing of huge real time data using Apache Kafka and Apache spark
Data Ingestion
Apache Spark Data Processing Results
Apache Kafka Data Processing Results
Comparison Graph

```
(appetizer)   156
(seeds)       204
Instant       158
UHT-milk      658
abrasive      70
alcohol       20
articles      648
artif.        64
baby          14
bags          1946
bakery        736
baking        348
bar           798
bathroom      54
beef          1032
beer          3112
berries       654
beverages     1070
bottled       3758
brandy        82
bread         2452
brown         1276
bulbs         82
butter        1640
```

Figure 42: Spark Processed Data in Main Frame

3.13 Comparing the Execution Time

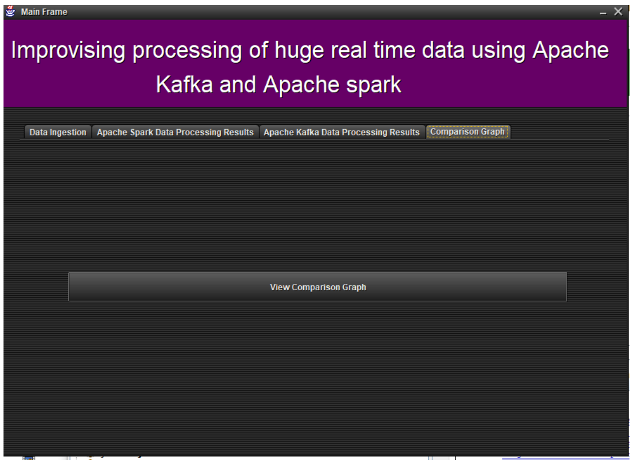

Main Frame
Improvising processing of huge real time data using Apache Kafka and Apache spark
Data Ingestion
Apache Spark Data Processing Results
Apache Kafka Data Processing Results
Comparison Graph
View Comparison Graph

Figure 43: Comparing Execution Time

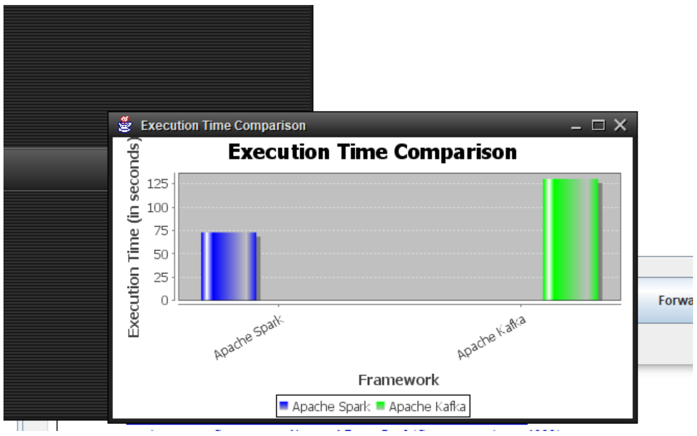
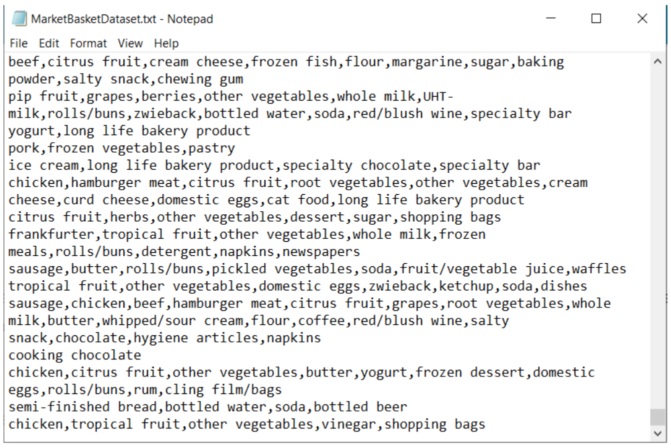


Figure 44: Comparison Graph

3.14 Dataset

Market basket analysis might tell a retailer that customers often purchase shampoo and conditioner together, so putting both items on promotion at the same time would not create a significant increase in revenue, while a promotion involving just one of the items would likely drive sales of the other. Market basket analysis may provide the retailer with information to understand the purchase behavior of a buyer.



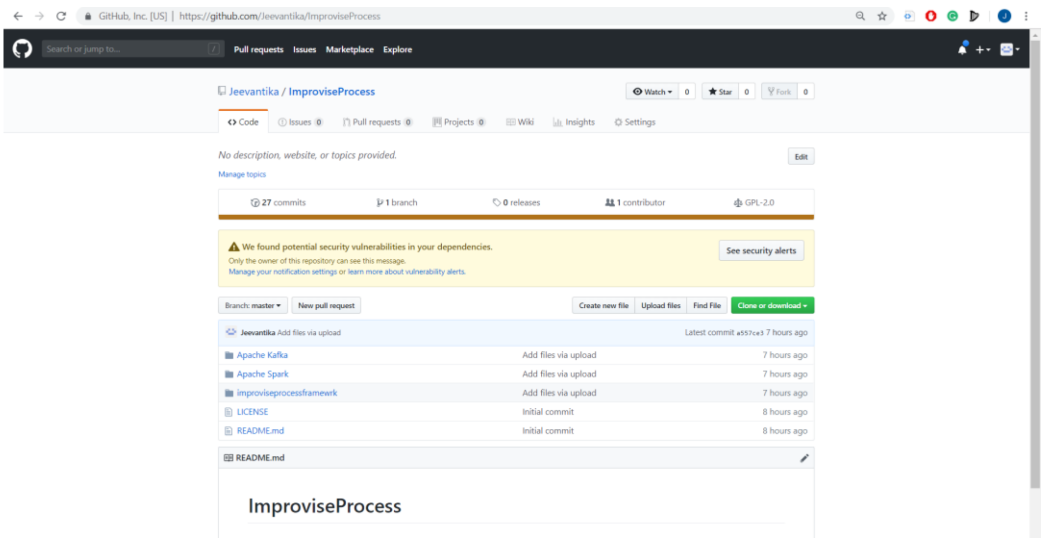
MarketBasketDataset.txt - Notepad

File Edit Format View Help

```
beef,citrus fruit,cream cheese,frozen fish,flour,margarine,sugar,baking
powder,salty snack,chewing gum
pip fruit,grapes,berries,other vegetables,whole milk,UHT-
milk,rolls/buns,zwieback,bottled water,soda,red/blush wine,specialty bar
yogurt,long life bakery product
pork,frozen vegetables,pastry
ice cream,long life bakery product,specialty chocolate,specialty bar
chicken,hamburger meat,citrus fruit,root vegetables,other vegetables,cream
cheese,curd cheese,domestic eggs,cat food,long life bakery product
citrus fruit,herbs,other vegetables,dessert,sugar,shopping bags
frankfurter,tropical fruit,other vegetables,whole milk,frozen
meals,rolls/buns,detergent,napkins,newspapers
sausage,butter,rolls/buns,pickled vegetables,soda,fruit/vegetable juice,waffles
tropical fruit,other vegetables,domestic eggs,zwieback,ketchup,soda,dishes
sausage,chicken,beef,hamburger meat,citrus fruit,grapes,root vegetables,whole
milk,butter,whipped/sour cream,flour,coffee,red/blush wine,salty
snack,chocolate,hygiene articles,napkins
cooking chocolate
chicken,citrus fruit,other vegetables,butter,yogurt,frozen dessert,domestic
eggs,rolls/buns,rum,cling film/bags
semi-finished bread,bottled water,soda,bottled beer
chicken,tropical fruit,other vegetables,vinegar,shopping bags
```

Figure 45: Dataset

3.15 GitHub



Jeevantika / ImproviseProcess

No description, website, or topics provided.

Manage topics

27 commits | 1 branch | 0 releases | 1 contributor | GPL-2.0

We found potential security vulnerabilities in your dependencies.
Only the owner of this repository can see this message.
Manage your notification settings or learn more about vulnerability alerts.

Jeevantika Add files via upload — Latest commit a557ce3 7 hours ago

Apache Kafka	Add files via upload	7 hours ago
Apache Spark	Add files via upload	7 hours ago
improviseprocessframewrk	Add files via upload	7 hours ago
LICENSE	Initial commit	8 hours ago
README.md	Initial commit	8 hours ago

README.md

ImproviseProcess

Figure 46: Project on GitHub

The Project is uploaded on GitHub
URL-https://github.com/Jeevantika/ImproviseProcess