ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 6, June 2016

Unified Big Data Lambda Architecture with Hadoop / Flume / Spark SQL, Streaming / Scala / Cassandra

B. Sunil Kumar[1], J. Mohan Kumar[2], P. Suman Rao[3]

[1,2,3]Assistant Professor, Dept. of CSE, Brindavan Institute of Technology & Science, Kurnool, Andhra Pradesh, India.

ABSTRACT: Big data is a term that describes the large volume of both structured and unstructured data that inundates a business on a day-to-day basis. Due to the fact that the database systems like RDBMS can process the unstructured data but RDBMS finds it challenging to handle such huge data volumes. To deal with the challenges Hadoop is used. It is software framework for distributed storage and distributed processing of very large data sets. But the disadvantages faced by Hadoop are Security Concerns and Vulnerable by Nature. To resolve those problems, Spark is introduced and used which is a cluster computing framework and performances up to 100 times faster for certain applications. As this paper is on both real time and interactive big data, to cope up them, we use Lambda architecture which is a data-processing architecture designed to handle massive quantities of data by taking advantages of both batch and stream processing layers. To pull the data from external sources and pass to Hadoop, tools like Flume is used. Scala is the programming language in which the programs are written because it is Scalable Language which in inspired by Java. Spark SQL and Spark Streaming are tools of Spark which are used for batch and stream data-processing. Cassandra is an open-source distributed database management system designed to handle and store large amounts of data.

KEYWORDS: Lambda Architecture; Spark SQL Streaming; unstructured data; distributed storage and processing; Hadoop; Flume.

I. INTRODUCTION

Big Data is the term for a collection of data sets so large and complex that it becomes difficult to process using on-hand database management tools or traditional data processing applications. The challenges include capture, cu ration, storage, search, sharing, transfer, analysis and visualization. Big Data usually includes data sets with sizes beyond the ability of commonly used software tools to capture, curate, manage, and process the data within a tolerable elapsed time. Analysis of data sets can find new correlations to "spot business trends, prevent diseases, and combat crime and so on." Scientists, business executives, practitioners of medicine, advertising and governments alike regularly meet difficulties with large data sets in areas including Internet search, finance and business informatics.

Data sets are growing rapidly in part because they are increasingly gathered by cheap and numerous information-sensing mobile devices, aerial (remote sensing), software logs, cameras, microphones, radio-frequency identification (RFID) readers and wireless sensor networks. The world's technological per-capita capacity to store information has roughly doubled every 40 months since the 1980s; as of 2012, every day 2.5exabytes (2.5×10^{18}) of data are created. One question for large enterprises is determining who should own big data initiatives that affect the entire organization. Relational database management systems and desktop statistics and visualization packages often have difficulty handling big data. The work instead requires "massively parallel software running on tens, hundreds, or even thousands of servers". What is considered "big data" varies depending on the capabilities of the users and their tools, and expanding capabilities make big data a moving target. "For some organizations, facing hundreds of gigabytes of data for the first time may trigger a need to reconsider data management options. For others, it may take tens or hundreds of terabytes before data size becomes a significant consideration."

Big data involves the data produced by different devices and applications. Given below are some of the fields that come under the umbrella of the big data.

- **Black box data:** it is a component of helicopter, airplanes, and jets, etc. it captures voices of the flight crew, recordings of microphones and earphones, and the performance information of the aircraft.

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 6, June 2016

- **Social Media Data:** social media such as Facebook and twitter hold information and the views posted by millions of the people across the globe.
- **Stock exchange data:** the stock exchange data holds information about the buy and sell decisions made on a share of different companies made by the customers.
- **Power grid data:** the power grid data holds information consumed by a particular node with respect to a base station.
- **Transport data:** transport data includes model, capacity, distance and availability of a vehicle.
- **Search engine data:** search engines retrieve lots of data from different databases.

Thus big data includes huge volume, high velocity, and extensible variety of data. The data in it will be of three types. The major challenges associated with big data are capturing data, storage, searching, sharing, transfer, analysis and presentation.

II. RELATED WORK

HADOOP – BIG DATA SOLUTIONS

Traditional Approach

In this approach, an enterprise will have a computer to store and process big data. Here data will be stored in an RDBMS like Oracle Database, MS SQL Server or DB2 and sophisticated software's can be written to interact with the database, process the required data and present it to the users for analysis purpose.

Limitation

This approach works well where we have less volume of data that can be accommodated by standard database servers, or up to the limit of the processor which is processing the data. But when it comes to dealing with huge amounts of data, it is really a tedious task to process such data through a traditional database server.

Google's Solution

Google solved this problem using an algorithm called Map Reduce. This algorithm divides the task into small parts and assigns these parts to many computers connected over the network, and collects the results to form the final result dataset.

HADOOP

Doug Cutting, Mike Cafarella and team took the solution provided by Google and started an open source project call Hadoop in 2005 and Daug named it after his son's toy elephant. Now apache Hadoop is a registered trademark of the apache software foundation.

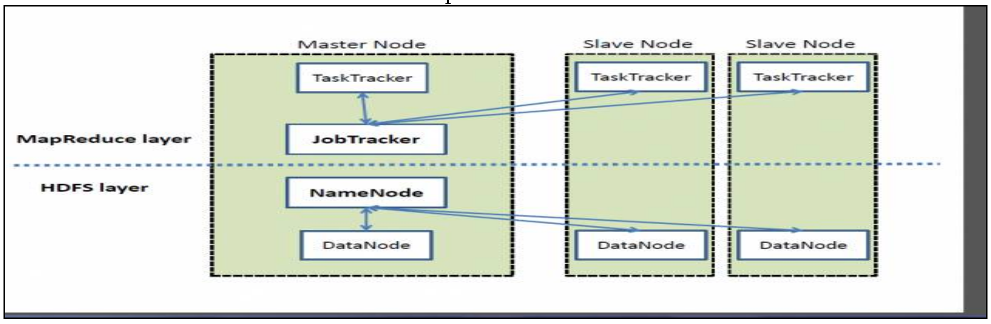


Fig: 2.1. Hadoop Architecture

Hadoop is an open-source software framework written in Java for distributed storage and distributed processing of very large data sets on computer clusters built from commodity hardware. All the modules in Hadoop are designed with a fundamental assumption that hardware failures are common and should be automatically handled by the framework.

Hadoop – HDFS Overview

Hadoop distributed file system was developed using distributed file system design. It is run on commodity hardware. Unlike other distributed systems, HDFS is highly fault tolerant and designed using low-cost hardware. HDFS holds very large amount of data and provides easier access. To store such huge data, the files are stored across multiple machines. These files are stored in redundant fashion to rescue the system from possible data losses in case of failure. HDFS also makes applications available to parallel processing.

Features of HDFS

- It is suitable for distributed storage and processing.
- Hadoop provides a command interface to interact with HDFS
- The built-in servers of name node and data node help users to easily check the status of cluster.
- Streaming access to file system data.
- HDFS provides file permissions and authentication.

HDFS Architecture

HDFS follows the master-slave architecture and it has the following elements.

➔ **NameNode**

The NameNode is commodity hardware that contains the GNU/Linux operating system and the NameNode software. It is a software that can be run on commodity hardware. The system having the Name Node act as the maser server and it does the following tasks:

- Manages the file system namespace.
- Regulates the client's access to the files.
- It also executes file system operations such as renaming, closing, and opening files and directories.

➔ **DataNode**

- The DataNode is a commodity hardware having the GNU/Linux operating system and DataNode software. For every node in cluster, there will be a data node. These nodes manage the data storage of their system.
- Data nodes perform read-write operations on the file systems, as per client request.
- They also perform operations such as block creation, deletion, and replication according to the instructions of the NameNode.

➔ **Block**

Generally, the user data is stored in the file of HDFS. The file in a file system will be divided into one or more segments and/or stored in individual data nodes. These files are called as blocks. In other words, the minimum amount of data that HDFS can read or write is called a block.

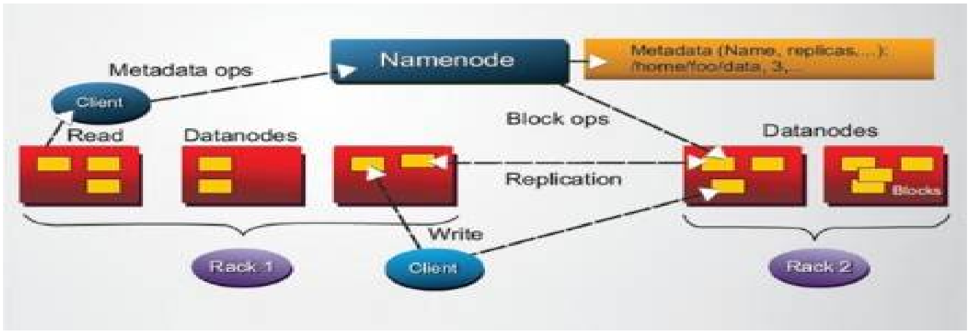


Fig: 2.2. HDFS Architecture

The default block size is 64MB, but it can be increased as per the need to change in HDFS configuration. In the Cloudera Hadoop the default is 128 MB. If block size was set to less than **64**, there would be a huge number of blocks throughout the cluster, which causes NameNode to manage an enormous amount of metadata.

➔ **Hadoop – MapReduce**

MapReduce is a framework using which we can write applications to process huge amounts of data, in parallel, on large clusters of commodity hardware in a reliable manner.

What is MapReduce?

MapReduce is a processing technique and a program model for distributed computing based on java. The MapReduce algorithm contains two important tasks, namely map and reduce.

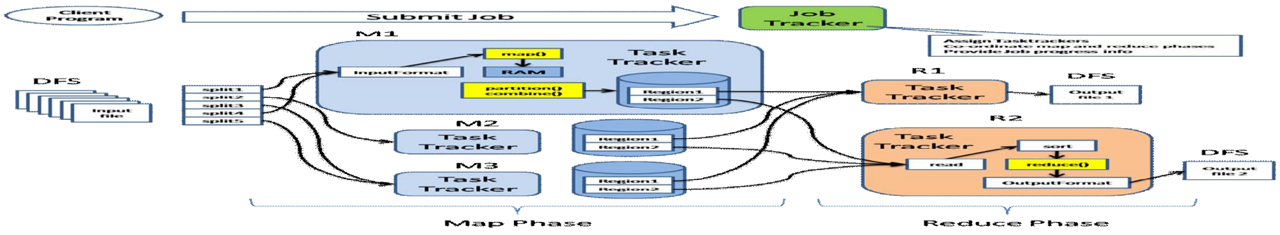


Fig: 2.3. MapReduce Architecture

Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

The major advantage of MapReduce is that it is easy to scale data processing over multiple computing nodes. Under the MapReduce mode, the data processing primitives are called mappers and reducers. Decomposing a data processing applications into mappers and reducers is sometimes nontrivial. But, once we write an application in the MapReduce form, scaling the application to run over hundred, thousands, or even tens of thousands of machines in a cluster is merely a configuration change. This simple scalability is what has attracted many programmers to use the MapReduce model. The Job Tracker is the service within Hadoop that farms out MapReduce tasks to specific nodes in the cluster, ideally the nodes that have the data, or at least are in the same rack. A Task Tracker is a node in the cluster that accepts tasks - Map, Reduce and Shuffle operations - from a Job Tracker.

HADOOP ECO-SYSTEM

- **Oozie**: Oozie is a workflow scheduler system to manage Apache Hadoop jobs. Oozie Workflow jobs are Directed Acyclical Graphs (DAGs) of actions. Oozie is a scalable, reliable and extensible system.
- **Hive**: Apache Hive is a Data Warehouse system which is built to work on Hadoop. It is used to querying and managing large datasets residing in distributed storage. It provides a mechanism to project structure onto the data in Hadoop and to query that data using SQL-like language called HiveQL(HQL).
- **Pig**: Apache Pig is a platform for analysing large data sets that consists of a high-level language for expressing data analysis programs, coupled with infrastructure for evaluating these programs. Pig's language layer currently consists of a textual language called Pig Latin.

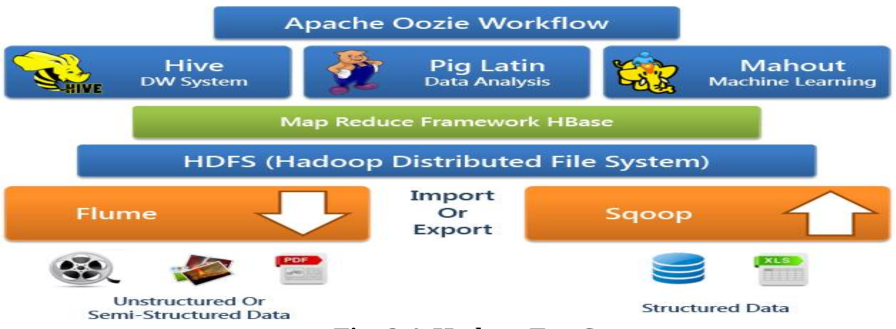


Fig: 2.4. Hadoop Eco-System

- **Mahout**: The Apache Mahout project's goal is to build an environment for quickly creating scalable performant machine learning applications. Mahout's mature Hadoop MapReduce algorithms.
- **Flume**: Flume is a distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of log data. It has a simple and flexible architecture based on streaming data flows. It uses a simple extensible data model that allows for online analytic application.
- **Sqoop**: Apache Sqoop is a tool designed for efficiently transferring bulk data between Apache Hadoop and structured data stores such as relational databases.

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 6, June 2016

LAMBDA ARCHITECTURE

Presented as a software design pattern, the lambda architecture unifies online and batch processing within a single framework. The pattern is suited to applications where there are time delays in data collection and availability through dashboards, requiring data validity for online processing as it arrives. The pattern also allows for batch processing for older data sets to find behavioural patterns as per user needs .

It is divided into three processing layers: The batch layer, Serving layer, and Speed layer. The results of the batch layer are called batch views and are stored on Amazon S3 as a tab-separated value file. The Serving layer indexes the batch views produced by the batch layer. Basically, the serving layer is a scalable database that swaps in new batch views as they become available. Due to the latency of the batch layer, the results from the serving layer are always out-of-date by a few hours. The speed layer compensates for the high latency of updates to the serving layer. It uses a fast, in-memory Spark engine to process data that has not been processed in the last batch of the batch layer. This layer produces the real-time views that are always up-to-date; it stores them in databases for both read and write operations. The speed layer is more complex than the batch layer due to the fact that the real-time views are continuously discarded as data makes its way through the batch and serving layers.

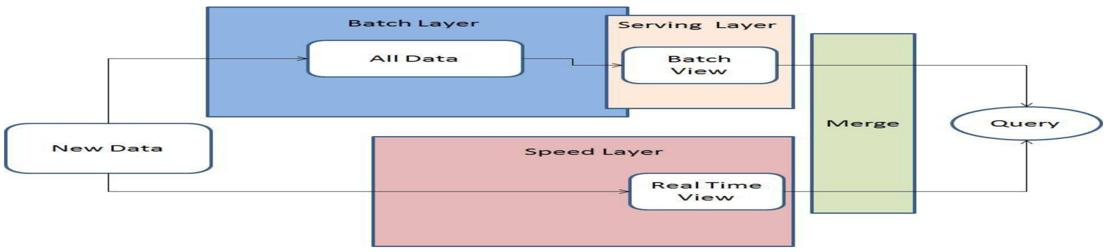


Fig: 2.5. Basic design of how the lambda architecture works.

. One well-developed approach to merging real-time data with historical data is to use Hadoop and Apache Storm together. Each engine produces a table in the serving database; applications can issue a query, which merges those results, as shown in the following figure.

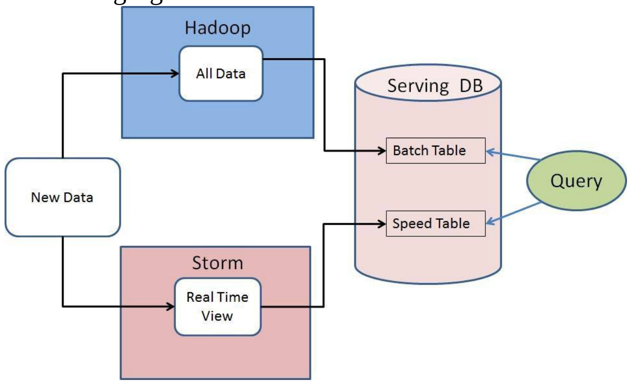


Fig: 2.6. Traditional Lambda architecture

The downside to traditional Lambda Architecture is that you must maintain the code required to produce the query result in two, complex, distributed systems.

CURRENT DATA PROCESSING SOLUTIONS

Data analytics are essential to plan and create decision support systems for optimising the underlying infrastructure. This involves not only processing of the online data, in search for certain events, but also the historical data sources which may be needed to find data patterns which influence decisions. Researchers have often merged techniques with other tools to develop field related solutions. Abouzied discussed HadoopDB, a hybrid of MapReduce and DBMS technologies, to allow scalability and performance of massive data processing. The authors present the application for a biological protein analysis or for business warehousing. Another example of merging was for image analysis in medical fields.

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 6, June 2016

TOOLS USED:

➔ **Flume**

Apache Flume is a distributed, reliable, and available system for efficiently collecting, aggregating and moving large amounts of log data from many different sources to a centralized data store.

The use of Apache Flume is not only restricted to log data aggregation. Since data sources are customizable, Flume can be used to transport massive quantities of event data including but not limited to network traffic data, social-media-generated data, email messages and pretty much any data source possible.

System Requirements

1. Java Runtime Environment - Java 1.6 or later (Java 1.7 Recommended)
2. Memory - Sufficient memory for configurations used by sources, channels or sinks
3. Disk Space - Sufficient disk space for configurations used by channels or sinks
4. Directory Permissions - Read/Write permissions for directories used by agent

Network Streams

Flume supports the following mechanisms to read data from popular log stream types, such as:

1. Avro 2. Thrift 3. Syslog 4. Netcat

A Flume event is defined as a unit of data flow having a byte payload and an optional set of string attributes. A Flume agent is a (JVM) process that hosts the components through which events flow from an external source to the next destination (hop).

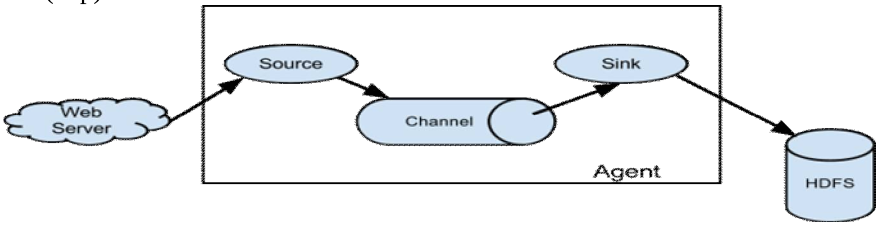


Fig: 2.8. Flume Architecture

A Flume source consumes events delivered to it by an external source like a web server. The external source sends events to Flume in a format that is recognized by the target Flume source. For example, an Avro Flume source can be used to receive Avro events from Avro clients or other Flume agents in the flow that send events from an Avro sink.

➔ **Spark**

Apache Spark is a lightning-fast cluster computing technology, designed for fast computation. It is based on Hadoop MapReduce and it extends the MapReduce model to efficiently use it for more types of computations, which includes interactive queries and stream processing. Spark uses Hadoop in two ways – one is storage and second is processing. Since Spark has its own cluster management computation, it uses Hadoop for storage purpose only. The main feature of Spark is its in-memory cluster computing that increases the processing speed of an application. Spark is designed to cover a wide range of workloads such as batch applications, iterative algorithms, interactive queries and streaming.

Spark Built on Hadoop

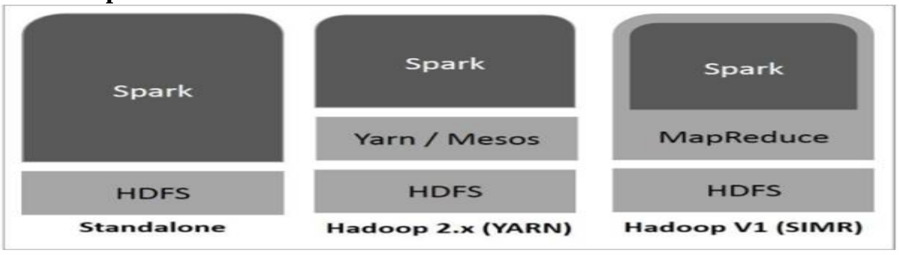


Fig: 2.10. Three ways of how Spark can be built with Hadoop components

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 6, June 2016

There are three ways of Spark deployment as explained below.

- **Standalone** − Spark Standalone deployment means Spark occupies the place on top of HDFS HadoopDistributedFileSystem. Here, Spark and MapReduce will run side by side to cover all spark jobs on cluster.
- **Hadoop Yarn** − Hadoop Yarn deployment means, simply, spark runs on Yarn without any pre-installation or root access required. It helps to integrate Spark into Hadoop ecosystem or Hadoop stack. It allows other components to run on top of stack.
- **Spark in MapReduce** SIMRSIMR − Spark in MapReduce is used to launch spark job in addition to standalone deployment. With SIMR, user can start Spark and uses its shell without any administrative access.

Spark Eco-System

The Spark project contains multiple closely-integrated components. At its core, Spark is a "computational engine" that is responsible for scheduling, distributing, and monitoring applications consisting of many computational tasks across many worker machines, or a *computing cluster*. Because the core engine of Spark is both fast and general purpose, it powers multiple higher-level components specialized for various workloads, such as SQL or machine learning. These components are designed to interoperate closely, letting you combine them like libraries in a software project.

Spark Core

Spark Core contains the basic functionality of Spark, including components for task scheduling, memory management, fault recovery, interacting with storage systems, and more. Spark Core is also home to the API that defines Resilient Distributed Datasets (RDDs), which are Spark's main programming abstraction. RDDs represent a collection of items distributed across many compute nodes that can be manipulated in parallel. Spark Core provides many APIs for building and manipulating these collections.

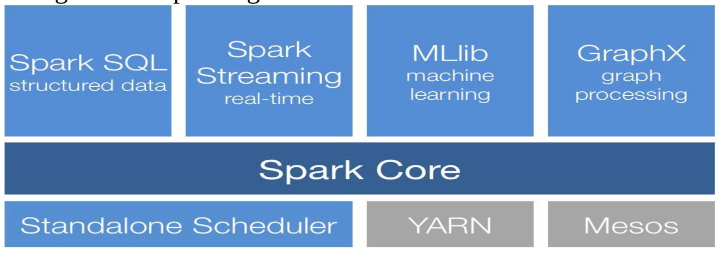


Fig: 2.11. The Spark Eco-System

Spark SQL

Spark SQL is a Spark module for structured data processing. Unlike the basic Spark RDD API, the interfaces provided by Spark SQL provide Spark with more information about the structure of both the data and the computation being performed. Internally, Spark SQL uses this extra information to perform extra optimizations.

Spark Streaming

Spark Streaming is a Spark component that enables processing live streams of data. Examples of data streams include log files generated by production web servers, or queues of messages containing status updates posted by users of a web service.

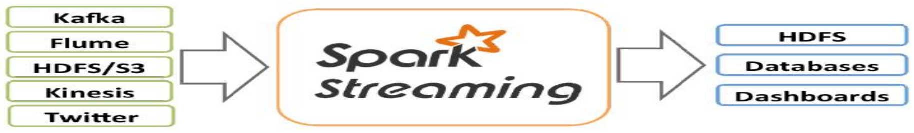


Fig: 2.12. Spark Streaming

Internally, it works as follows. Spark Streaming receives live input data streams and divides the data into batches, which are then processed by the Spark engine to generate the final stream of results in batches. Spark Streaming provides a high-level abstraction called discretized stream or DStream, which represents a continuous stream of data.

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 6, June 2016

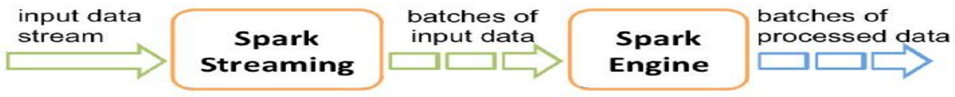


Fig: 2.13. Computation of Spark Streaming

DStreams can be created either from input data streams from sources such as Kafka, Flume, and Kinesis, or by applying high-level operations on other DStreams. Internally, a DStream is represented as a sequence of RDDs.

Scala

Scala, short for Scalable Language, is a hybrid functional programming language. It was created by Martin Odersky and it was first released in 2003. Scala smoothly integrates features of object-oriented and functional languages and Scala is compiled to run on the Java Virtual Machine. Many existing companies, who depend on Java for business critical applications, are turning to Scala to boost their development productivity, applications scalability and overall reliability. Scala is a pure object-oriented language in the sense that every value is an object. Types and behavior of objects are described by classes and traits which will be explained in subsequent chapters. Classes are extended by sub classing and a flexible mix in-based composition mechanism as a clean replacement for multiple inheritance. Scala is also a functional language in the sense that every function is a value and because every value is an object so ultimately every function is an object. Scala provides a lightweight syntax for defining anonymous functions, it supports higher-order functions, it allows functions to be nested, and supports currying. These concepts will be explained in subsequent chapters. Scala, unlike some of the other statically typed languages, does not expect you to provide redundant type information. You don't have to specify a type in most cases, and you certainly don't have to repeat it. Scala is compiled into Java Byte Code which is executed by the Java Virtual Machine (JVM). This means that Scala and Java have a common runtime platform. You can easily move from Java to Scala. The Scala compiler compiles your Scala code into Java Byte Code, which can then be executed by the scala command. The scala command is similar to the java command, in that it executes your compiled Scala code.

Programming Scala

1. **Method Parameters:**

   Method parameters are variables, which are used to pass the value inside a method when the method is called. Method parameters are only accessible from inside the method but the objects passed in may be accessible from the outside, if you have a reference to the object from outside the method. Method parameters are always mutable and defined by val keyword.
2. **Local Variables:**

   Local variables are variables declared inside a method. Local variables are only accessible from inside the method, but the objects you create may escape the method if you return them from the method. Local variables can be both mutable or immutable types and can be defined using either var or val
3. **Basic Data Structures**

   i) Lists ii) Sets iii) Tuple iv) Maps v) Option
4. **Functional Combinators**

   i) Map ii) Foreach iii) Zip iv) Partition v) Find vi) drop and dropWhile
   vii) foldRight and foldLeft viii) flatten ix) flatMap x)Generalized functional combinators xi) Map

Cassandra

Apache Cassandra is an open source, distributed and decentralized/distributed storage system (database), for managing very large amounts of structured data spread out across the world. It provides highly available service with no single point of failure.

Features of Cassandra:

Cassandra is highly scalable; it allows adding more hardware to accommodate more customers and more data as per requirement. It has no single point of failure and it is continuously available for business-critical applications that cannot afford a failure. It is linearly scalable, i.e., it increases your throughput as you increase the number of nodes in the cluster. Therefore, it main taints a quick response time. It Cassandra accommodates all possible data formats including: structured, semi-structured, and unstructured. It can dynamically accommodate changes to your data structures according to your need. It provides the flexibility to distribute data where you need by replicating data across

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 6, June 2016

multiple datacenters. It supports properties like Atomicity, Consistency, Isolation, and Durability (ACID). It was designed to run on cheap commodity hardware. It performs blazingly fast writes and can store hundreds of terabytes of data, without sacrificing the read efficiency.

Cassandra Query Language

Users can access Cassandra through its nodes using Cassandra Query Language(CQL). CQL treats the database (Key space) as a container of tables. Programmers use CQLSH: a prompt to work with CQL or separate application language drivers. Clients approach any of the nodes for their read-write operations. That node (coordinator) plays a proxy between the client and the nodes holding the data.

III.PROPOSED ALGORITHM

INTERACTIVE ANALYSIS

According to interactive analusis of Lambda Architecture, the data processing for interactive analysis starts from flume, which pulls the unstructured and semi-structured data from external sources and passes the data to Hadoop, which is used for storage purpose.

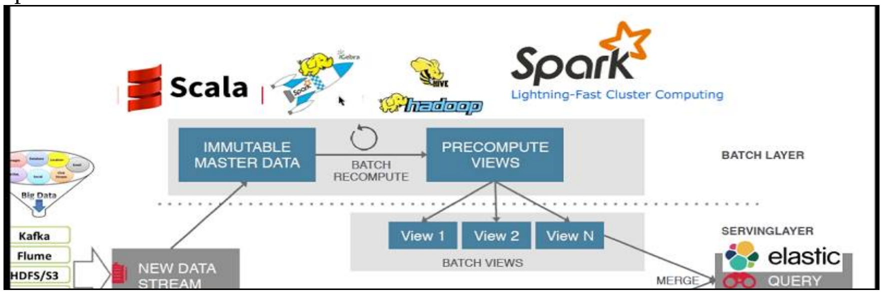


Fig: 3.1. Interactive Analysis of Lambda Architecture

Before the data is send to Hadoop for storage purpose, the data under goes several components of Flume which includes source, flume agent, channel, interceptor, sink and so on. Through sink, the data passes to Hadoop. The event manager separates the batch data or interactive data from complete data and then the batch data is passed to Hadoop. The data which is storage in Hadoop is also called as Immutable Master Data which either cannot be updated or deleted. If the storage reaches to max limit, the commodity hardware's are added depending upon the size of data. This data consists of relevant data, irrelevant data, inconsistent data, and incomplete data.

REAL-TIME ANALYSIS

According to real-time analysis of Lambda Architecture, Flume which is a tool used to pull the large amount of real-time and interactive data of unstructured and semi-structured is used at the beginning of real-time analysis. This tool pulls the data from external sources and the event manager separates the data into two which includes real-time and interactive data.

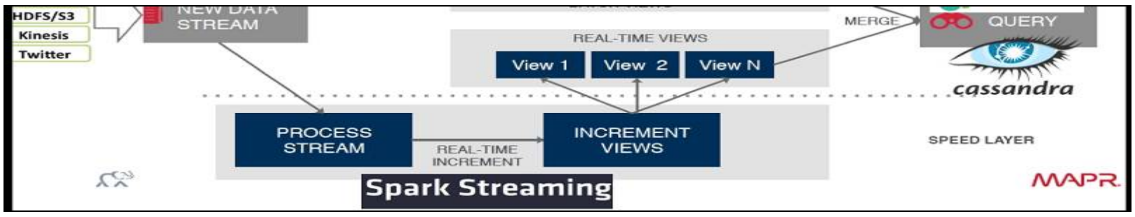


Fig: 3.2. Real-time Analysis of Lambda Architecture

As this is real-time analysis, the data cannot be stored first rather it should be processed by using some tool which can process real-time data.

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

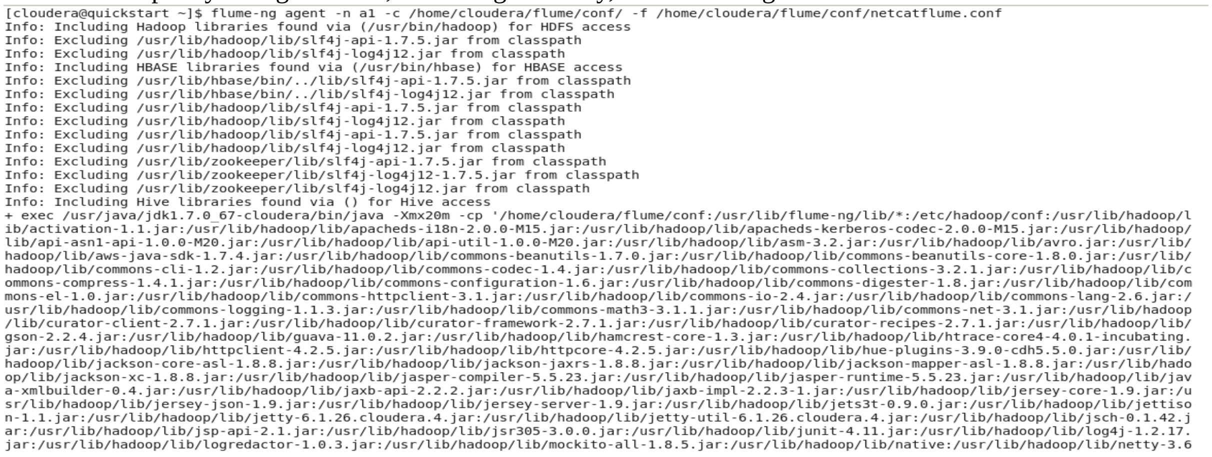
Vol. 4, Issue 6, June 2016

IV.PSEUDO CODE

4.1. FLUME

4.1.1. Starting an Agent

An agent is started using a shell script called flume-ng which is located in the bin directory of the Flume distribution. You need to specify the agent name, the config directory, and the config file on the command line:



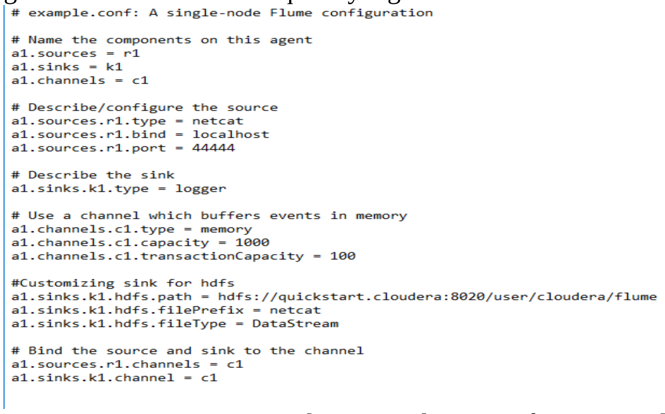
```
[cloudera@quickstart ~]$ flume-ng agent -n a1 -c /home/cloudera/flume/conf/ -f /home/cloudera/flume/conf/netcatflume.conf
Info: Including Hadoop libraries found via (/usr/bin/hadoop) for HDFS access
Info: Excluding /usr/lib/hadoop/lib/slf4j-api-1.7.5.jar from classpath
Info: Excluding /usr/lib/hadoop/lib/slf4j-log4j12.jar from classpath
Info: Including HBASE libraries found via (/usr/bin/hbase) for HBASE access
Info: Excluding /usr/lib/hbase/bin/../lib/slf4j-api-1.7.5.jar from classpath
Info: Excluding /usr/lib/hbase/bin/../lib/slf4j-log4j12.jar from classpath
Info: Excluding /usr/lib/hadoop/lib/slf4j-api-1.7.5.jar from classpath
Info: Excluding /usr/lib/hadoop/lib/slf4j-log4j12.jar from classpath
Info: Excluding /usr/lib/hadoop/lib/slf4j-api-1.7.5.jar from classpath
Info: Excluding /usr/lib/hadoop/lib/slf4j-log4j12.jar from classpath
Info: Excluding /usr/lib/zookeeper/lib/slf4j-api-1.7.5.jar from classpath
Info: Excluding /usr/lib/zookeeper/lib/slf4j-log4j12-1.7.5.jar from classpath
Info: Excluding /usr/lib/zookeeper/lib/slf4j-log4j12.jar from classpath
Info: Including Hive libraries found via () for Hive access
+ exec /usr/java/jdk1.7.0_67-cloudera/bin/java -Xmx20m -cp '/home/cloudera/flume/conf:/usr/lib/flume-ng/lib/*:/etc/hadoop/conf:/usr/lib/hadoop/l
ib/activation-1.1.jar:/usr/lib/hadoop/lib/apacheds-i18n-2.0.0-M15.jar:/usr/lib/hadoop/lib/apacheds-kerberos-codec-2.0.0-M15.jar:/usr/lib/hadoop/
lib/api-asn1-api-1.0.0-M20.jar:/usr/lib/hadoop/lib/api-util-1.0.0-M20.jar:/usr/lib/hadoop/lib/asm-3.2.jar:/usr/lib/hadoop/lib/avro.jar:/usr/lib/
hadoop/lib/aws-java-sdk-1.7.4.jar:/usr/lib/hadoop/lib/commons-beanutils-1.7.0.jar:/usr/lib/hadoop/lib/commons-beanutils-core-1.8.0.jar:/usr/lib/
hadoop/lib/commons-cli-1.2.jar:/usr/lib/hadoop/lib/commons-codec-1.4.jar:/usr/lib/hadoop/lib/commons-collections-3.2.1.jar:/usr/lib/hadoop/lib/c
ommons-compress-1.4.1.jar:/usr/lib/hadoop/lib/commons-configuration-1.6.jar:/usr/lib/hadoop/lib/commons-digester-1.8.jar:/usr/lib/hadoop/lib/com
mons-el-1.0.jar:/usr/lib/hadoop/lib/commons-httpclient-3.1.jar:/usr/lib/hadoop/lib/commons-io-2.4.jar:/usr/lib/hadoop/lib/commons-lang-2.6.jar:/
usr/lib/hadoop/lib/commons-logging-1.1.3.jar:/usr/lib/hadoop/lib/commons-math3-3.1.1.jar:/usr/lib/hadoop/lib/commons-net-3.1.jar:/usr/lib/hadoop
/lib/curator-client-2.7.1.jar:/usr/lib/hadoop/lib/curator-framework-2.7.1.jar:/usr/lib/hadoop/lib/curator-recipes-2.7.1.jar:/usr/lib/hadoop/lib/
gson-2.2.4.jar:/usr/lib/hadoop/lib/guava-11.0.2.jar:/usr/lib/hadoop/lib/hamcrest-core-1.3.jar:/usr/lib/hadoop/lib/htrace-core4-4.0.1-incubating.
jar:/usr/lib/hadoop/lib/httpclient-4.2.5.jar:/usr/lib/hadoop/lib/httpcore-4.2.5.jar:/usr/lib/hadoop/lib/hue-plugins-3.9.0-cdh5.5.0.jar:/usr/lib/
hadoop/lib/jackson-core-asl-1.8.8.jar:/usr/lib/hadoop/lib/jackson-jaxrs-1.8.8.jar:/usr/lib/hadoop/lib/jackson-mapper-asl-1.8.8.jar:/usr/lib/hado
op/lib/jackson-xc-1.8.8.jar:/usr/lib/hadoop/lib/jasper-compiler-5.5.23.jar:/usr/lib/hadoop/lib/jasper-runtime-5.5.23.jar:/usr/lib/hadoop/lib/jav
a-xmlbuilder-0.4.jar:/usr/lib/hadoop/lib/jaxb-api-2.2.2.jar:/usr/lib/hadoop/lib/jaxb-impl-2.2.3-1.jar:/usr/lib/hadoop/lib/jersey-core-1.9.jar:/u
sr/lib/hadoop/lib/jersey-json-1.9.jar:/usr/lib/hadoop/lib/jersey-server-1.9.jar:/usr/lib/hadoop/lib/jets3t-0.9.0.jar:/usr/lib/hadoop/lib/jettiso
n-1.1.jar:/usr/lib/hadoop/lib/jetty-6.1.26.cloudera.4.jar:/usr/lib/hadoop/lib/jetty-util-6.1.26.cloudera.4.jar:/usr/lib/hadoop/lib/jsch-0.1.42.j
ar:/usr/lib/hadoop/lib/jsp-api-2.1.jar:/usr/lib/hadoop/lib/jsr305-3.0.0.jar:/usr/lib/hadoop/lib/junit-4.11.jar:/usr/lib/hadoop/lib/log4j-1.2.17.
jar:/usr/lib/hadoop/lib/logredactor-1.0.3.jar:/usr/lib/hadoop/lib/mockito-all-1.8.5.jar:/usr/lib/hadoop/lib/native:/usr/lib/hadoop/lib/netty-3.6
```

Plate: 4.1. Starting an agent

Now the agent will start running source and sinks configured in the given properties file.

4.1.2. A Simple Example

Here, we give an example configuration file, describing a single-node Flume deployment. This configuration lets a user generate events and subsequently logs them to the console.



```
# example.conf: A single-node Flume configuration

# Name the components on this agent
a1.sources = r1
a1.sinks = k1
a1.channels = c1

# Describe/configure the source
a1.sources.r1.type = netcat
a1.sources.r1.bind = localhost
a1.sources.r1.port = 44444

# Describe the sink
a1.sinks.k1.type = logger

# Use a channel which buffers events in memory
a1.channels.c1.type = memory
a1.channels.c1.capacity = 1000
a1.channels.c1.transactionCapacity = 100

#Customizing sink for hdfs
a1.sinks.k1.hdfs.path = hdfs://quickstart.cloudera:8020/user/cloudera/flume
a1.sinks.k1.hdfs.filePrefix = netcat
a1.sinks.k1.hdfs.fileType = DataStream

# Bind the source and sink to the channel
a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1
```

Plate: 4.2. Flume Configuration File

Initializing StreamingContext

To initialize a Spark Streaming program, a StreamingContext object has to be created which is the main entry point of all Spark Streaming functionality.

A StreamingContext object can be created from a SparkConf object.

```
import org.apache.spark._
import org.apache.spark.streaming._
//val conf = new SparkConf().setAppName(appName).setMaster(master)
val ssc = new StreamingContext(conf, Seconds(1))
```

The appName parameter is a name for your application to show on the cluster UI. master is a Spark, Mesos or YARN cluster URL, or a special “local[*]” string to run in local mode. In practice, when running on a cluster, you will not want to hardcode master in the program, but rather launch the application with spark-submit and receive it there.

A StreamingContext object can also be created from an existing SparkContext object.

```
import org.apache.spark.streaming._
val sc = ... // existing SparkContext
```

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

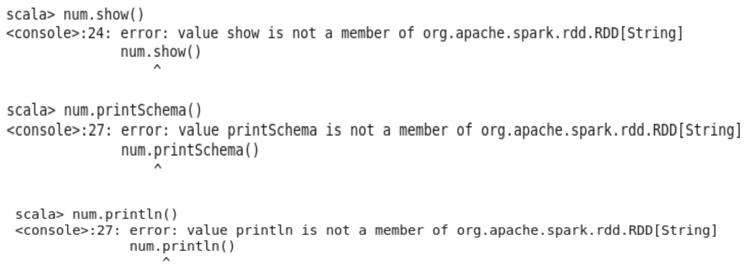
(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 6, June 2016

val ssc = new StreamingContext(sc, Seconds(1))
After a context is defined, you have to do the following.
Define the input sources by creating input DStreams.
Define the streaming computations by applying transformation and output operations to DStreams.
Start receiving data and processing it using streamingContext.start().
Wait for the processing to be stopped (manually or due to any error) using streamingContext.awaitTermination().
The processing can be manually stopped using streamingContext.stop().

V.SIMULATION RESULTS

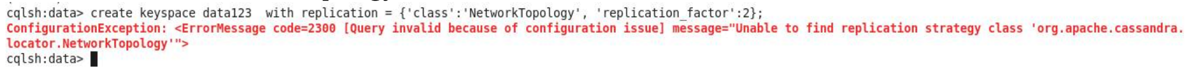
5.1. UNABLE TO PERFORM OPERATIONS IN SCALA



```
scala> num.show()
<console>:24: error: value show is not a member of org.apache.spark.rdd.RDD[String]
              num.show()
                  ^


scala> num.printSchema()
<console>:27: error: value printSchema is not a member of org.apache.spark.rdd.RDD[String]
              num.printSchema()
                  ^


 scala> num.println()
 <console>:27: error: value println is not a member of org.apache.spark.rdd.RDD[String]
               num.println()
                   ^
```

Plate: 5.1. Unable to Perform Operations

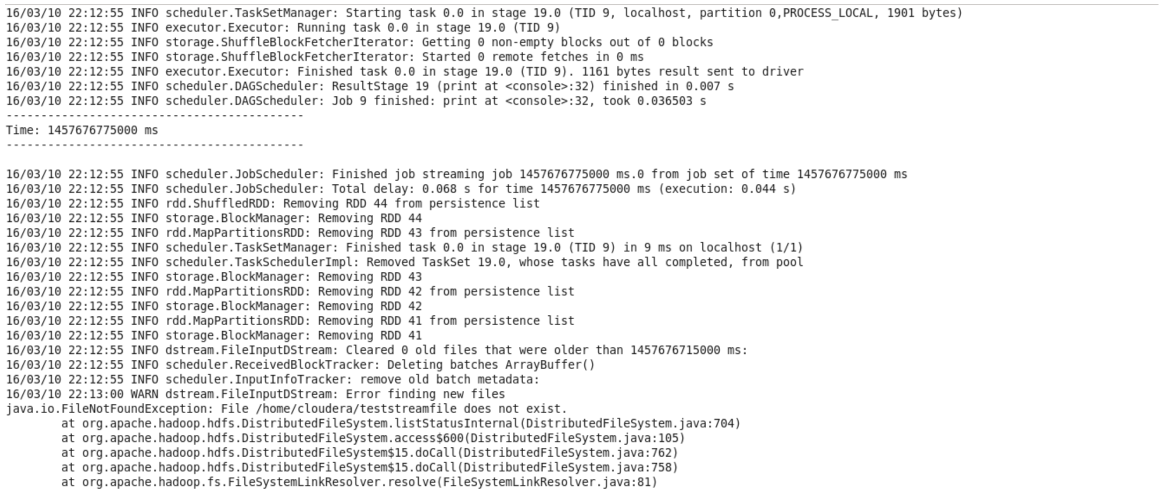
5.2. CAN'T USE"NetworkTopology" CLASS IN CASSANDRA



```
cqlsh:data> create keyspace data123  with replication = {'class':'NetworkTopology', 'replication_factor':2};
ConfigurationException: <ErrorMessage code=2300 [Query invalid because of configuration issue] message="Unable to find replication strategy class 'org.apache.cassandra.
locator.NetworkTopology'">
cqlsh:data> █
```

Plate: 5.2. Unable to use NetworkTopology Class

5.3. UNABLE TO DISPLAY DATA IN SPARK STREAMING

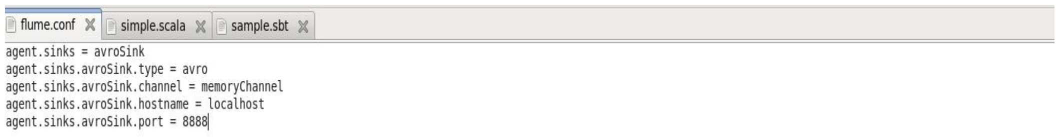


```
16/03/10 22:12:55 INFO scheduler.TaskSetManager: Starting task 0.0 in stage 19.0 (TID 9, localhost, partition 0,PROCESS_LOCAL, 1901 bytes)
16/03/10 22:12:55 INFO executor.Executor: Running task 0.0 in stage 19.0 (TID 9)
16/03/10 22:12:55 INFO storage.ShuffleBlockFetcherIterator: Getting 0 non-empty blocks out of 0 blocks
16/03/10 22:12:55 INFO storage.ShuffleBlockFetcherIterator: Started 0 remote fetches in 0 ms
16/03/10 22:12:55 INFO executor.Executor: Finished task 0.0 in stage 19.0 (TID 9). 1161 bytes result sent to driver
16/03/10 22:12:55 INFO scheduler.DAGScheduler: ResultStage 19 (print at <console>:32) finished in 0.007 s
16/03/10 22:12:55 INFO scheduler.DAGScheduler: Job 9 finished: print at <console>:32, took 0.036503 s
-------------------------------------------
Time: 1457676775000 ms
-------------------------------------------

16/03/10 22:12:55 INFO scheduler.JobScheduler: Finished job streaming job 1457676775000 ms.0 from job set of time 1457676775000 ms
16/03/10 22:12:55 INFO scheduler.JobScheduler: Total delay: 0.068 s for time 1457676775000 ms (execution: 0.044 s)
16/03/10 22:12:55 INFO rdd.ShuffledRDD: Removing RDD 44 from persistence list
16/03/10 22:12:55 INFO storage.BlockManager: Removing RDD 44
16/03/10 22:12:55 INFO rdd.MapPartitionsRDD: Removing RDD 43 from persistence list
16/03/10 22:12:55 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 19.0 (TID 9) in 9 ms on localhost (1/1)
16/03/10 22:12:55 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 19.0, whose tasks have all completed, from pool
16/03/10 22:12:55 INFO storage.BlockManager: Removing RDD 43
16/03/10 22:12:55 INFO rdd.MapPartitionsRDD: Removing RDD 42 from persistence list
16/03/10 22:12:55 INFO storage.BlockManager: Removing RDD 42
16/03/10 22:12:55 INFO rdd.MapPartitionsRDD: Removing RDD 41 from persistence list
16/03/10 22:12:55 INFO storage.BlockManager: Removing RDD 41
16/03/10 22:12:55 INFO dstream.FileInputDStream: Cleared 0 old files that were older than 1457676715000 ms:
16/03/10 22:12:55 INFO scheduler.ReceivedBlockTracker: Deleting batches ArrayBuffer()
16/03/10 22:12:55 INFO scheduler.InputInfoTracker: remove old batch metadata:
16/03/10 22:13:00 WARN dstream.FileInputDStream: Error finding new files
java.io.FileNotFoundException: File /home/cloudera/teststreamfile does not exist.
        at org.apache.hadoop.hdfs.DistributedFileSystem.listStatusInternal(DistributedFileSystem.java:704)
        at org.apache.hadoop.hdfs.DistributedFileSystem.access$600(DistributedFileSystem.java:105)
        at org.apache.hadoop.hdfs.DistributedFileSystem$15.doCall(DistributedFileSystem.java:762)
        at org.apache.hadoop.hdfs.DistributedFileSystem$15.doCall(DistributedFileSystem.java:758)
        at org.apache.hadoop.fs.FileSystemLinkResolver.resolve(FileSystemLinkResolver.java:81)
```

Plate: 5.3. Unable to display data in Spark Streaming

5.4. FLUME-SPARK STREAMING CONNECTION ERROR

1. Flume.conf File:



```
agent.sinks = avroSink
agent.sinks.avroSink.type = avro
agent.sinks.avroSink.channel = memoryChannel
agent.sinks.avroSink.hostname = localhost
agent.sinks.avroSink.port = 8888
```

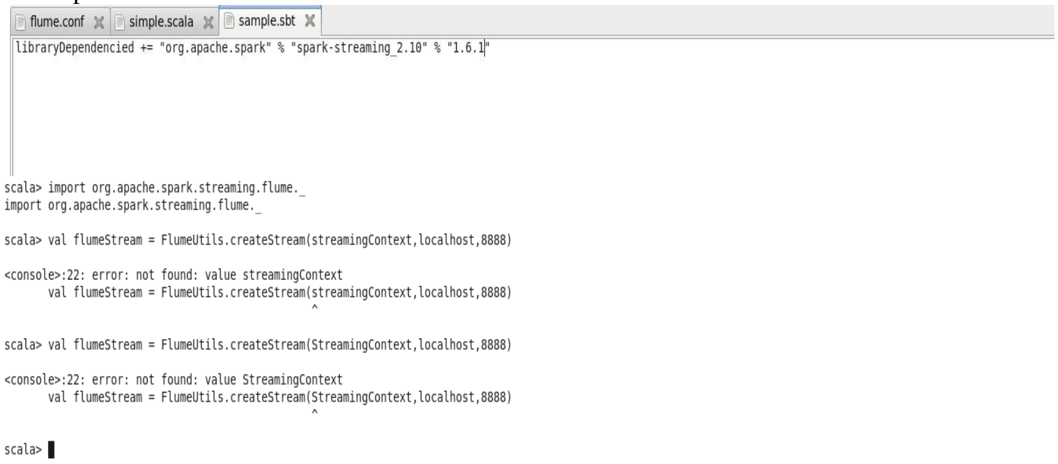
ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 6, June 2016

2.Sample.sbt File:

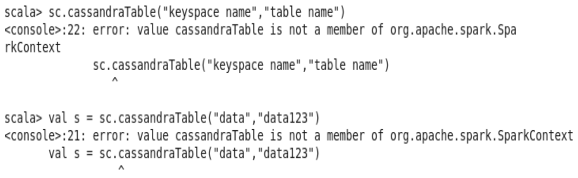


```
libraryDependencied += "org.apache.spark" % "spark-streaming_2.10" % "1.6.1"
```

```
scala> import org.apache.spark.streaming.flume._
import org.apache.spark.streaming.flume._

scala> val flumeStream = FlumeUtils.createStream(streamingContext,localhost,8888)

<console>:22: error: not found: value streamingContext
       val flumeStream = FlumeUtils.createStream(streamingContext,localhost,8888)
                                                 ^

scala> val flumeStream = FlumeUtils.createStream(StreamingContext,localhost,8888)

<console>:22: error: not found: value StreamingContext
       val flumeStream = FlumeUtils.createStream(StreamingContext,localhost,8888)
                                                 ^

scala> 
```

Plate: 5.4. Flume-Spark Streaming Connection error

5.5. SPARKSQL-CASSANDRA CONNECTION ERROR:



```
scala> sc.cassandraTable("keyspace name","table name")
<console>:22: error: value cassandraTable is not a member of org.apache.spark.Spa
rkContext
              sc.cassandraTable("keyspace name","table name")
                 ^


scala> val s = sc.cassandraTable("data","data123")
<console>:21: error: value cassandraTable is not a member of org.apache.spark.SparkContext
       val s = sc.cassandraTable("data","data123")
                  ^
```

Plate: 5.5. SparkSQL Cassandra Connection error

VI.CONCLUSION AND FUTURE WORK

As the data increases in a humongous fashion, normal databases like RDBMS struggle to handle and process the huge data. To handle such huge data, Big data came up with some tools and technologies which can do the processing of large datasets of different data formats. One such technology is Hadoop which is used to handle large datasets in distributed environment. With the help of Lambda Archiecture, the incoming data is divided into interactive data and real-time data. The batch data is first stored and then processed where as real-time data is first processed and then stored. For batch data, we used Hadoop for storage purpose, SparkSQL for processing the data and Cassandra for storing the processed data. For real-time data, we use Spark Streaming to process the data and Cassandra to storage the processed data. Here Flume is used to pull the data from external sources and the event manager divides the complete data into batch data and real-time data.

REFERENCES

1. Edward Capriolo,Dean Wampler, and Jason Ruthergl, “Programming Hive”, Tata McGrawHill, 1992 .
2. Tom White, “Hadoop The Definitive Guide”, 3rd Edition, O’Reilly Media,2012
3. Tom White, “Hadoop The Definitive Guide”, 4th Edition, O’Reilly Media,2012
4. Eben Hewitt, “Cassandra The Definitive Guide”, O’Reilly Media,2011
5. N. Gershenfeld, R. Krikorian, and D. Cohen, “The internet of things,” Scientific American, vol. 291, no. 4, p. 76, 2004.
6. V. Marx, “Biology: The big challenges of big data,” Nature, vol. 498, no. 7453, pp. 255–260, 2013.
7. D. E. O’Leary, “Artificial intelligence and big data,” IEEE Intelligent Systems, vol. 28, no. 2, pp. 0096–99, 2013.
8. J. Dean and S. Ghemawat, “Mapreduce: Simplified data processing on large clusters,” in OSDI\’04, 2005, pp. 137–150.
9. http://en.wikipedia.org/wiki/Apache Hadoop\
10. http://en.wikipedia.org/wiki/Apache Hive\
11. http://hadoop.apache.org
12. http://flume.apache.org/
13. http://www.tutorialspoint.com/cassandra/

14. http://spark.apache.org/
15. http://docs.scala-lang.org/

BIOGRAPHY

B. Sunil Kumar is an Assistant Professor in the Department of Computer science & Engineering, Brindavan Institute of Technology & Science, Kurnool, Andhra Pradesh, India. He received Master of Technology (M. Tech) degree in 2011 from JNTU, Hyderabad, AP, India. He is having approximately 10 years of teaching experience in and out this college. He is the Life Member of Indian Society of Technical Education (ISTE). He has got the International certifications in Networking and System administration from Red Hat Linux India and Novel Netware and awarded the CMS Network Specialist Certificate. His research interests are Computer Networks (Wireless Sensor Networks), Mobile Computing, Big Data, Artificial Intelligence, Design & Analysis of Algorithms, Data Structures and Computer Graphics.

J. Mohan Kumar is an Assistant Professor in the Department of Computer science & Engineering, Brindavan Institute of Technology & Science, Kurnool, Andhra Pradesh, India. He received Master of Technology (M. Tech) degree in 2010 from Bellary Institute of Technology, Karnataka, India. His research interests are Object oriented analysis and modelling, Big Data and Web Technologies.

P. Suman Rao is an Assistant Professor in the Department of Computer science & Engineering, Brindavan Institute of Technology & Science, Kurnool, Andhra Pradesh, India. He received Master of Technology (M. Tech) degree in 2010 from JNTU, Anantapur, AP, India. His research interests are Computer Networks, Compiler Design, Big Data, HCI, Algorithms, Ontologies, web 2.0 etc.