Big Data hadoop Development & Administration MATERIAL

By Mr. Gopal Krishna, 12+ Years Of Real Time Exp , 6+ Years On BIGDATA Projects
Working as Sr. Hadoop Technical Architect, CCA 175 - Spark and Hadoop Certified Consultant

Kelly Technologies

Flat No: 212, 2nd Floor, Annapurna Block, Aditya Enclave, Ameerpet, Hyderabad.
Ph No: 040 6462 6789, 0998 570 6789 info@kellytechno.com, www.kellytechno.com.

Bangalore: HAL Road, Marathahalli. 080 6012 6789, 0784 800 6789
Online: 001 973 780 6789.

PREFACE

1.HADOOP &HDFS.....17-75

1.1.What Is Big Data.....18

1.1.1.Some interesting facts of Big Data.....18

1.2.Uses of Big Data.....19

1.3. What is Hadoop.....20

1.4.Relation between Hadoop and bigdata.....20

1.5.Why to go ahead with Hadoop.....21

1.5.1.Some of the key features of Hadoop are

1.5.1.1 Flexible

1.5.1.2. Scalable

1.5.1.3. Building more efficient data economy

1.5.1.4.Robust Ecosystem

1.5.1.5. Hadoop is getting more "Real-Time"!

1.5.1.6. Cost Effective

1.5.1.7. Upcoming Technologies using Hadoop

1.5.1.8. Hadoop is getting cloudy!

1.6. Some of the use cases of Hadoop.....23

1.7.Scenarios to opt Hadoop technology in REAL TIME projects.....24

1.7.1. Migration

1.7.2. Integration

1.7.3. Scalability

1.7.4. Cloud Hosting

1.7.5. Specialized Data Analytics

1.8.Challenges of Big Data.....26

1.8.1.The Four -V's of Big Data

1.8.1.1.Volume

1.8.1.2.Variety

1.8.1.3.Velocity

1.8.1.4.Veracity

1.9.How does Hadoop address the big Data changes.....27

1.9.1.Hadoop is built to run on a cluster of machines

1.9.2.Hadoop clusters scale horizontally

1.9.3.Hadoop can handle unstructured/semi-structured data

1.9.4.Hadoop clusters provides storage and computing

1.9.5.Hadoop provides storage for big data at reasonable cost

1.9.6.Hadoop allows for the capture of new or more data

1.9.7.With Hadoop, you can store data longer

1.9.8.Hadoop provides scalable analytics
1.9.9.Hadoop provides rich analytics
1.10.Comparison with other technologies.....29
1.10.1.Extreme low cost per byte
1.10.2.Very high bandwidth to support MapReduce workload
1.10.3.Rock solid data reliability
1.11. How does the HDFS competition stack up? 30
1.11.1.System not designed for Hadoop's scale
1.11.2.System that don't use commodity hardware or open source software
1.11.3.Not designed for MapReduce's I/O patterns
1.11.4.Unproven technology
1.12.Hadoop vs Rdbms.....31
1.13.Hadoop vs Data Warehouse 32
1.14.Core Hadoop Components. 33
1.14.1. Hadoop Common
1.14.2. Hadoop Distributed File System (HDFS)
1.14.3.MapReduce- Distributed Data Processing Framework of Apache Hadoop
1.14.4.YARN
1.15. Some of the basic terminology in Hadoop.....38
1.15.1.What is a cluster environment?
1.15.2.Hadoop Cluster Node
1.15.3.Hadoop Cluster
1.16. HDFS (Hadoop Distributed File System) 40
1.17. Features of HDFS 41
1.17.1.Fault Tolerance
1.17.2.High Availability
1.17.3.Reliability
1.17.4.Replication
1.17.5.Scalability
1.17.6.Distributed Storage
1.18.Storage Aspects Of HDFS. 43
1.18.1.HDFS Block
1.18.2.How to configure the block size
1.19. Why is a Block in HDFS So Large? 45
1.20.HDFS Architecture 46
1.20.1.Name Node
1.20.2.Data Node
1.20.3.Secondary name node

1.20.4.JobTracker
1.20.5.TaskTracker
1.21. Replication in Hadoop.....50
1.21.1.Data Storage in Data Nodes
1.21.2.Replilcation Configuration
1.22. Core Design Principles of Hadoop.....51
1.23.Commads Guide.....52
1. General Commands
2. User Commands
3. Administration Commands
2.MAPREDUCE.....76-134
2.1.Map Reduce.....77
2.2.Daemons of Hadoop.....77
2.3.Hadoop 1.x Architecture.....78
2.4.Limitations of Hadoop 1.x Architecture.....78
2.5.MapReduce Phases.....79
2.6.Map Reduce Life Cycle.....84
2.7. The Parts of a Hadoop MapReduce Job.....88
2.8.MapReduce Programming Model.....89
2.9.MapReduce Framework.....89
2.10.MapReduce Processing.....89
2.11.MapReduce Implementation.....89
2.12.Basic MapReduce Program.....93
2.12.1.Driver
2.12.2.Mapper
2.12.3.Reducer
2.13.Sample MapReduce Program for dynamic wordcount.....97
2.14.Creation of jar file and Execution Process in Clustered Environment...100
2.14.1.Process of Creation of Jar file
2.14.2.Execution Process
2.15.Identitiy Mapper and Identitiy Reducer in MapReduce.....102
2.16.Input Format's in MapReduce.....102
2.16.1.File Input Format
2.16.2.Text Input Format
2.16.3.KeyValueTextInputFormat
2.16.4.N-Line Input Format
2.16.5.DB InputFormat: DBInputFormat
2.16.6.Binary Input Format

2.17. Output Formats in MapReduce.....106

2.17.1.Text Output Format

2.17.2.KeyValueTextOutputFormat

2.17.3.Database Output format

2.17.4.SequenceFileOutputFormat

2.17.5.SequenceFileAsBinaryOutputFormat

2.17.6..MultipleTextOutputFormat

2.18.MapReduce API (Application Programming Interface).....107

2.19.Difference between Hadoop OLD API and NEW API.....107

2.20.Combiner.....109

2.21.Better optimisation techniques in MapReduce.....110

2.22.Partitioner 110

2.23.How Partitioning works 112

2.24.Steps for developing Custom Partitioner.....112

2.25.Compression Techniques in MapReduce.....118

2.25.1.Types of compression techniques

2.25.1.1.GZip compression Technique

2.25.1.2LZO Codec

2.25.1.3.Snappy codec

2.26.How to enable compression.....118

2.27.Hadoop MapReduce Chaining (Advanced MapReduce).....124

2.28.Chaining MapReduce jobs in a sequence.....124

2.29.JOINS - In MapReduce.....131

2.29.1.Joining during the Map phase

2.29.2.Joining during the Reduce phase

2.30.Distributed Cache In Hadoop.....132

2.31.we will see Hadoop map side join with Distributed Cache below.....133

2.32.Data Localization Using MapReduce.....134

3.PIG.....135-185

3.1.pig.....136

3.2.Apache open source project.....136

3.2.1.Ease of programming

3.2.2.Optimization opportunities

3.2.3.Extensibility

3.3.Pig Latin.....137

3.4.Comments in Scripts.....137

3.5.Pig has two execution modes.....137

3.6.Running Pig Programs.....137

3.6.1.Grunt Shell
3.6.2.Script Mode
3.7.Data Types in Pig.....138
3.8.PIG Transformations Examples.....139
3.8.1.Grunt Shell (Local Mode)
3.8.2.Grunt Shell (HDFS Mode)
3.8.3.SCRIPT (Local Mode)
3.8.4.SCRIPT (HDFS Mode)
3.9.How To Work With Udf's In Pig.....141
3.10.How To Write Register Jar In Pig.....142
3.11.How To Write Custom Filter Function In Pig.....142
3.12.How To Write Custom Eval Function In Pig.....142
3.13.How To Write Custom Load Function In Pig.....143
3.14.How To Write Custom Store Function In Pig.....143
3.15.How To Write Macros In Pig.....144
3.16.Pig Latin – Efficiency and Performance improvement.....145
3.16.1.Using proper Data Types
3.16.2.Project Early
3.16.3.Filter Early
3.16.4.Avoid Splitting Of Operations In Code
3.16.5.Make Use Of Parallel
3.16.6.Limit The Output
3.16.7.Use Of Distinct Instead Of Group By Or Generate
3.17.Optimization Rules.....151
3.17.1.Use of Optimization rules
3.17.2.Proper Use Of Filter
3.17.3.Limit Optimizer
3.17.4.Foreach Optimization
3.17.5.Group All Optimization
3.18.Multi-Query Execution.....155
3.19.Other Techniques : Multi-Query Execution.....155
3.19.1.Explicit And Implicit Splits
3.19.2.Storing Intermediate Results
3.19.3.Store vs. Dump
3.19.4.Error Handling
3.19.5.Use of EXEC

3.20.User Defined Function (Udf) 157
3.21.Multiple Xml File Data Load To Pig Environment.....159
3.22.Pig Custom Data Types.....164
3.23.Examples on Pig 168

4.HIVE.....186-221
4.1.Hive.....187
4.2.Hive Architecture.....187
4.3.Functioning of Driver,Compiler and Executor.....188
4.4.Working of Hive.....189
4.5.Meta Store in Hive.....190
4.5.1.The metadata which metastore stores contains things like
4.5.2.There are three modes of metastore
4.5.2.1.Embedded Metastore
4.5.2.2.Local Metastore
4.5.2.3.Remote Metastore
4.6.Important metastore configuration properties.....194
4.7.Differences between RDBMS and Hive.....194
4.7.1.Schema on Write vs Schema on Read
4.8.Hive Tables.....195
4.8.1.Managed Table
4.8.2.External Table
4.9.Hive Services.....197
4.10.Hiveserver.....198
4.11.Data Types in hive.....199
4.12.Commands in hive.....200
4.13.Difference between Managed table and External Table.....205
4.14.Alter table in Hive.....205
4.15.Create table...as select.....208
4.16.Joins.....209
4.16.1.Inner joins
4.16.2.Left Outer joins
4.16.3.Right Outer Join
4.16.4.Full Outer Join
4.17.Partition.....211
4.17.1.Static Partitioning
4.17.2. Dynamic partitioning
4.18.Bucketing.....213
4.19.Order By And Sort By.....214
4.20.Hive : Sort By Vs Order By Vs Distribute By Vs Cluster By.....215
4.21.Collection Data Type In Hive
5.SQOOP.....222-245
5.1.Introduction.....223

5.2.History....223
5.3.Why Sqoop? 223
5.4.Where Sqoop is used?....224
5.5.Sqoop Architecture....224
5.6.What Sqoop Does?....225
5.7.Sqoop – Installation....225
5.8.Sqoop Tools....227
5.9.Using Generic and Specific Arguments....227
5.10.New Concepts in Sqoop Versions....228
5.11.sqoop-import....229
5.12.Specifying a Target Directory.... 230
5.13.Import all tables.... .230
5.14.Importing Only a Subset of Data.... 230
5.15.Using a File Format Other Than CSV.... 231
5.16.Compressing Imported Data.... 232
5.17.Speeding Up Transfers.... 233
5.18.Overriding Type Mapping.... 233
5.19.Controlling Parallelism.... 234
5.20.Encoding NULL Values.... 234
5.21.Importing all Tables except Some Tables....234
5.22.Importing particular columns....234
5.23.Importing data with diff. field separator (otherthan CSV)....234
5.24.Importing Only New Data....235
5.25.Incrementally Importing Mutable Data....235
5.26.Preserving the Last Imported Value(Sqoop Job)....235
5.27.Storing Passwords in the Metastore....236
5.28.Importing Data from Two Tables....237
5.29.Using Custom Boundary Queries....238
5.30.Renaming Sqoop Job Instances....238
5.31.Importing Queries with Duplicated Columns....238
5.31.Hive-Import....239
5.32.Using Partitioned Hive Tables....240
5.33.Replacing Special Delimiters During Hive Import....24
5.34.Using the Correct NULL String in Hive....241
5.35.Importing Data into Hbase....241
5.36.Importing All Rows into Hbase....242
5.37.Export....242
5.37.1.Transferring Data from Hadoop

5.37.2.Inserting Data in Batches
5.37.3.Exporting with All-or-Nothing Semantics
5.37.4.Updating an Existing Data Set
5.37.5.Updating or Inserting at the Same Time
5.38.Sqoop Codegen.....245
6.HBASE.....246-310
6.1.HBASE-Introduction.....247
6.2.Comparison of HBASE and RDBMS.....247
6.3.Comparison of HBASE and HDFS.....248
6.4.Comparison of HBASE and NoSQL.....249
6.5.When should we prefer HBASE over other database.....250
6.6.When not to use HBASE.....251
6.7.Hbase Architecture.....251
6.7.1.HMaster
6.7.2.RegionServer
6.7.3.Region
6.7.4.ZooKeeper
6.8.Data Modelling.....254
6.8.1.HTable
6.8.2.ColumnFamily
6.8.3.ColumnQualifierName (CQN)
6.8.4.RowKey
6.8.5.TimeStamp
6.8.6.cell
6.9.Sample table structure in HBASE.....255
6.10.Steps To Execute The Ddl And Dml Statements In Hbase Shell...257
6.10.1.Ddl In Hbase
6.10.2.Dml In Hbase
6.11.Java Api For Hbase.....268
6.11.1.Create table in HBASE using java code
6.11.2.Scan Table for first 3 rows using Java Code
6.11.3.Using Java Code scan and retrieve whole table content
6.11.4.Using Java Code scan and retrieve Rowkey based table content
6.11.5.Using Java code to retrieve table content for a specific ColumnFamily:ColumnQualifierName
6.11.6.Using Java code insert data into RowKey "0004" of the table "CUST_DET"
6.12.HBASE_MR Integration.....284
6.13.Execute the java code.....296

6.13.1.Method 1 :When using Eclipse in Ubuntu Terminal
6.13.2.Method 2:If the java code is placed on Windows based Eclipse IDE
6.14.Client Side Buffering In Hbase.....308
6.14.1. Implicit Flush
6.14.2. Explicit Flush
7.FLUME.....311-334
7.1.What is Flume? 312
7.2.Applications of Flume.....312
7.3.Advantages of Flume.....312
7.4.Advantages over ad-hoc solutions.....313
7.5.Features of Flume.....313
7.6.Core Concepts.....313
7.6.1.Event
7.6.2.Client
7.6.3.Agent
7.7.Streaming / Log Data.....315
7.7.1.Log file
7.8.Data Transfer To HDFS 315
7.8.1.HDFS put Command
7.8.1.1.Problem with put Command
7.8.2.Problem with HDFS
7.8.3.Available Solutions
7.8.3.1.Facebook's Scribe
7.8.3.2.Apache Kafka
7.8.3.3.Apache Flume
7.9.Flume Architecture.....317
7.9.1.Source
7.9.1.1.Different Source Types
7.9.2.Channel
7.9.3.Sink
7.9.3.1.Different types of Sinks
7.10.Additional Components of Flume Agent 318
7.10.1.Interceptors
7.10.2.Channel Selectors
7.10.3.Sink Processors
7.11.Flume Dataflow.....319
7.11.1.Multi-hop Flow
7.11.2.Fan-out Flow

7.11.3.Fan-in Flow
7.12.Failure Handling....321
7.13.Agent Data Ingest....321
7.14.Agent Data Drain....322
7.15.Agent Pipeline Flow....322
7.16.Delivery Guarantee....323
7.17.Aggregation Use-Case....323
7.18.Using Single Flume Agent....324
7.19.Using Multiple Flume Agents....324
7.20.Consolidation....325
7.21.Multiplexing the flow....326
7.22.Flume Installation....327
7.22.1.Configuring Flume
7.22.2.conf Folder
7.22.3.flume-env.sh
7.23.Verifying the Installation....331
7.24.Flume - Configuration....332
7.24.1.Naming the Components
7.24.2.Describing the Source
7.24.3.Describing the Sink
7.24.4.Describing the Channel
7.24.5.Binding the Source and the Sink to the Channel
7.25.Starting a Flume Agent....334
8.OOZIE....335-359
8.1.What is Apache Oozie?....336
8.2.Oozie Editors....337
8.3.Oozie Eclipse Plugin (OEP)....338
8.4.Oozie Workflow....338
8.5.Property File....346
8.6.Coordinators....349
8.7.Control Information....350
8.8.Coordinator Job Status....351
8.9.Bundle....353
8.10.Command Line Tools....356
8.11.Action Extensions....358
9.MONGODB....360-
9.1.MongoDB Introduction and Use....361
9.1.1.General database concept

9.2.What is MongoDB.....361
9.3.Why MongoDB.....361
9.4.How to use MongoDB.....362
9.5.Mongo java api.....365
10.MAHOUT.....369-381
10.1.What is Machine Learning?.....370
10.2.Types of machine learning.....370
10.3.What is Mahout?.....370
10.4.Mahout algorithms are roughly divided into 4 major sections.....372
10.4.1. Collaborative filtering
10.4.2. Clustering
10.4.3.Categorization
10.4.4.Mahout utilities
10.5.Mahout Classifier Demo Using Naive Bayes.....376
11.SCALA.....382-421
11.1.Scala Closures.....383
11.2.Functions in Scala.....385
11.2.1.Anonymus Functions in Scala
11.2.2.Higher Order Functions In Scala
1.Foreach()
2.Map()
3.Reduce()
11.3.Scala Loops.....388
11.3.1.While loop
11.3.2.Do...While loop
11.3.3.for loop
11.4.Scala Arrays.....393
11.5.Scala Iterators.....394
11.6.Scala Collections.....395
11.6.1.List
11.6.2.Set Operations
11.6.3.Scala Maps
11.7.Scala Tuples.....407
11.8.Flatten.....409
11.9.Pattern Matching in Scala.....411
11.10.Some Other.....415
11.11.Currying Function In Scala.....417
11.12.Creating Mutable Collections from Immutable Ones.....418

11.13.Examples on Lists & Sets.....419
11.14.Scala Classes & Objects.....420
11.15.Scala Traits.....421
11.16.When to Choose "Object Oriented" Vs "Functional Programming" languages ?
12.SPARK.....422-495
12.1.Spark.....423
12.2.Key Features of Spark.....423
12.2.1.Easy to Use
12.2.2.Fast
12.2.3.General Purpose
12,2.4.Scalable
12.2.5.Fault Tolerant
12.3.Basic Difference between Map Reduce and Spark.....425
12.4.Spark Architecture.....426
12.4.1.Spark Context
12.4.2.Cluster Manager
12.4.3.Executors
12.5.Different Cluster Managers In - Spark Architecture.....427
12.5.1.Apache Mesos
12.5.2.Hadoop YARN
12.6.Spark Architecture.....428
12.7.Resilient Distributed Datasets (RDD).....428
12.8.Key characteristics of an RDD.....428
12.8.1.Immutable
12.9.Rdd Operations.....429
12.9.1.Transformations
1)Map
2)filter
3)flatMap
4)mapPartitions
5)Union
6)Intersection
7)Subtract
8)Parallelize
9)Distinct
10)Cartesian
11)ZiP
12)ZipWithIndex

13)GroupBy
14)Sort By
15)Coalesce
16)Repartition
17)Sample
18)Keys
19)Values
20)MapValues
21)Join
22)LeftOuterJoin
23)RightOuterJoin
24)FullOuterJoin
25)GroupByKey
26)ReduceByKey
12.9.2.Actions
1)Collect
2)Count
3)Count By Value
4)first
5)max
6)min
7)Top
8)reduce
12.9.3.Actions on RDD of key-value Pairs
1)CountByKey
2)lookup
12.10.How To Login To Scala Shell In Spark.....451
12.11.Why Spark is "Lazy Evaluated" System?.....454
12.12.How RDDs are Fault Tolerant?.....454
12.13.How to Initialize Spark Context.....455
12.14.Parallelized collections.....456
12.15.pair RDD.....458
12.16.How To Access Spark Gui.....459
12.17.Word Count Use Case Using Spark Context – In Scala.....460
12.18. Spark Sql.....461
12.19.Diff Ways Of "Loading" & "Saving" Data Using Spark Sql.....476
12.20.Parquet.....476
12.21.Spark Sql – 2 X Examples With "Jdbc" Intraction.....478

12.22.Spark Streaming.....482

12.22.1.Process Flow in Spark Streaming

12.22.2.High-Level Architecture

12.22.3.StreamingContext

12.22.4.Starting Stream Computation

12.23.5.DStreams or discretized streams

12.23.6.Spark Streaming Use Case

13.STORM.....496-505

13.1.Introduction 497

13.2.Advantages of Storm.....497

13.3.Need of Storm.....497

13.4.Architecture of Storm.....497

13.5.Cluster Architectural flow.....499

13.6.Explanation of the Components.....499

13.7.Difference between Hadoop cluster and Storm cluster.....500

13.8.Internal Process of the data in the Apache Storm.....501

13.9.Components of Storm.....501

13.9.1. Topology

13.9.2. Stream

13.9.3.Spout

13.9.4.Bolt

13.10.Differences between Apache Spark, Apache Storm and Apache Kafka.505

14.KAFKA.....506-511

14.1.Distributed Messaging Architecture: Apache Kafka.....507

14.2.The following is how Kafka solved these problems.....509

14.2.1.Messaging System: The Kafka Way

HADOOP & HDFS

1.1.What is Big Data?

Big data is a term for data sets that are so large or complex that traditional data processing applications are inadequate to deal with them. Challenges include analysis, capture, data curation, search, sharing, storage, transfer, visualization, querying, updating and information privacy. The term "big data" often refers simply to the use of predictive analytics, user behavior analytics, or certain other advanced data analytics methods that extract value from data, and seldom to a particular size of data set.

Big data usually includes data sets with sizes beyond the ability of commonly used software tools to capture, acurate, manage, and process data within a tolerable elapsed time.Big data "size" is a constantly moving target, ranging from a few dozen terabytes to many petabytes and even exabytes of data. Big data requires a set of techniques and

technologies with new forms of integration to reveal insights from datasets that are diverse, complex, and of a massive scale

1.1.1 Some interesting facts of Big Data

- The data volumes are exploding, more data has been created in the past two years than in the entire previous history of the human race.
- Data is growing faster than ever before and by the year 2020, about 1.7 megabytes of new information will be created every second for every human being on the planet.
- By then, our accumulated digital universe of data will grow from 4.4 zettabytes today to around 44 zettabytes, or 44 trillion gigabytes.
- Every second we create new data. For example, we perform 40,000 search queries every second (on Google alone), which makes it 3.5 billion searches per day and 1.2 trillion searches per year.
- In Aug 2015, over 1 billion people used Facebook on a single day.
- Facebook users send on average 31.25 million messages and view 2.77 million videos every minute.
- We are seeing a massive growth in video and photo data, where every minute up to 300 hours of video are uploaded to YouTube alone.
- In 2015, a staggering 1 trillion photos will be taken and billions of them will be shared online. By 2017, nearly 80% of photos will be taken on smart phones
- This year, over 1. billion smart phones will be shipped – all packed with sensors capable of collecting all kinds of data, not to mention the data the users create themselves.
- By 2020, we will have over 6.1 billion smartphone users globally (overtaking basic fixed phone subscriptions).
- Within five years there will be over 50 billion smart connected devices in the world. all developed to collect, analyze and share data.
- By 2020, at least a third of all data will pass through the cloud (a network of servers connected over the Internet).
- Distributed computing (performing computing tasks using a network of computers in the cloud) is very real. Google uses it every day to involve about 1,000 computers in answering a single search query. which takes no more than 0.2 seconds to complete.
- The Hadoop (open source software for distributed computing) market is forecast to grow at a compound annual growth rate 58% surpassing $1 billion by 2020.
- And one of most important facts: At the moment less than 0.5% of all data is ever analyzed and used, just imagine the potential here.
- A 2015 report by Cap Gemini found that 56% of companies expect to increase their spending on big data in the next three years.
- There will be a shortage of talent necessary for organizations to take advantage of Big Data. By 2018, the United States alone could face a shortage of 140,000 to 190,000 skilled workers with

deep analytical skills as well as 1.5 million managers and analysts with the know-how to use Big Data analytics to make effective decisions.

1.2.Uses of Big Data

Organizations are increasingly turning to big data to discover new ways to improve decision-making, opportunities, and overall performance. For example, big data can be harnessed to address the challenges that arise when information that is dispersed across several different systems that are not interconnected by a central system. By aggregating data across systems, big data can help improve decision-making capability. It also can augment data warehouse solutions by serving as a buffer to process new data for inclusion in the data warehouse or to remove infrequently accessed or aged data.

Big data can lead to improvements in overall operations by giving organizations greater visibility into operational issues. Operational insights might depend on machine data, which can include anything from computers to sensors or meters to GPS devices. Big data provides unprecedented insight on customers' decision-making processes by allowing companies to track and analyze shopping patterns, recommendations, purchasing behavior and other drivers that are known to influence sales.

Cyber security and fraud detection is another use of big data. With access to real-time data, businesses can enhance security and intelligence analysis platforms. They can also process, store and analyze a wider variety of data types to improve intelligence, security and law enforcement insight.

1.3.What is Hadoop

Apache Hadoop is an open source software framework for storage and large scale processing of data-sets on clusters of commodity hardware. Hadoop is an Apache top-level project being built and used by a global community of contributors and users.

All the modules in Hadoop are designed with a fundamental assumption that hardware failures are a common occurrence and should be automatically handled by the framework.

Hadoop was created by Doug Cutting and Mike Cafarella in 2005. It was originally developed to support distribution for the Nutch search engine project. Doug named Hadoop project after his son's yellow coloured toy elephant.

1.4.Relation between Hadoop and Bigdata

If you research the term Big Data, you will invariably encounter Hadoop. The fundamental difference between big data and Hadoop is that the former is an asset, often a complex and

ambiguous one, while the latter is a program that accomplishes a set of goals and objectives for dealing with that asset.

Hadoop is one of the tools designed to handle big data. Hadoop works to interpret or parse the results of big data searches through specific proprietary algorithms and methods. Hadoop is an open-source program under the Apache license that is maintained by a global community of users. It includes various main components, including a MapReduce set of functions and a Hadoop distributed file system (HDFS)

Traditional data warehouses and relational databases process structured data and can store massive amounts of data, but the requirement for structure restricts the type of data that can be processed. Hadoop is designed to process large amounts of data, regardless of its structure.

1.5 Why to go ahead with Hadoop ?

1.5 1.Some of the key features of Hadoop are

1.5 1.1 Flexible

As it is a known fact that only 20% of data in organizations is structured, and the rest is all unstructured, it is very crucial to manage unstructured data which goes unattended. Hadoop manages different types of Big Data, whether structured or unstructured, encoded or formatted, or any other type of data and makes it useful for decision making process. Moreover, Hadoop is simple, relevant and schema-less! Though Hadoop generally supports Java Programming, any programming language can be used in Hadoop with the help of the MapReduce technique. Though Hadoop works best on Windows and Linux, it can also work on other operating systems like BSD and OS X.

1.5.1.2. Scalable

Hadoop is a scalable platform, in the sense that new nodes can be easily added in the system as and when required without altering the data formats, how data is loaded, how programs are written or even without modifying the existing applications. Hadoop is an open source platform and runs on industry-standard hardware. Moreover, Hadoop is also fault tolerant – this means, even if a node gets lost or goes out of service, the system automatically reallocates work to another location of the data and continues processing as if nothing had happened!

1.5.1.3. Building more efficient data economy

Hadoop has revolutionized the processing and analysis of big data world across. Till now, organizations were worrying about how to manage the non-stop data overflowing in their systems. Hadoop is more like a "Dam", which is harnessing the flow of unlimited amount of

data and generating a lot of power in the form of relevant information. Hadoop has changed the economics of storing and evaluating data entirely!

1.5.1.4.Robust Ecosystem

Hadoop has a very robust and a rich ecosystem that is well suited to meet the analytical needs of developers, web start-ups and other organizations. Hadoop Ecosystem consists of various related projects such as MapReduce, Hive, HBase, Zookeeper, HCatalog, Apache Pig, which make Hadoop very competent to deliver a broad spectrum of services.

1.5.1.5. Hadoop is getting more "Real-Time"!

Did you ever wonder how to stream information into a cluster and analyze it in real time? Hadoop has the answer for it. Yes, Hadoop's competencies are getting more and more real-time. Hadoop also provides a standard approach to a wide set of APIs for big data analytics comprising MapReduce, query languages and database access, and so on.

1.5.1.6. Cost Effective

Loaded with such great features, the icing on the cake is that Hadoop generates cost benefits by bringing massively parallel computing to commodity servers, resulting in a substantial reduction in the cost per terabyte of storage, which in turn makes it reasonable to model all your data. The basic idea behind Hadoop is to perform cost-effective data analysis present across world wide web!

1.5.1.7. Upcoming Technologies using Hadoop

With reinforcing its capabilities. Hadoop is leading to phenomenal technical advancements. For instance, HBase will soon become a vital Platform for Blob Stores (Binary Large Objects) and for Lightweight OLTP (Online Transaction Processing). Hadoop has also begun serving as a strong foundation for new-school graph and NoSQL databases, and better versions of relational databases.

1.5.1.8. Hadoop is getting cloudy!

Hadoop is getting cloudier! In fact, cloud computing and Hadoop are synchronizing in several organizations to manage Big Data. Hadoop will become one of the most required apps for cloud computing. This is evident from the number of Hadoop clusters offered by cloud vendors in various businesses. Thus, Hadoop will reside in the cloud soon!

The importance of Hadoop is evident from the fact that there are many global MNCs that are using Hadoop and consider it as an integral part of their functioning. It is a misconception that social media companies alone uses Hadoop. In fact, many other industries now use Hadoop to manage BIG DATA!

1.6.Some of the use cases of Hadoop

It was Yahoo!Inc. that developed the World's biggest application of Hadoop on February 19, 2008. In fact, if you've heard of 'The Yahoo! Search Webmap', it is a Hadoop application that runs on over 10,000 core Linux cluster and generates data that is now extensively used in each query of Yahoo! Web search.

Facebook, which has over 1.3 billion active users and it is Hadoop that brings respite to Facebook in storing and managing data of such magnitude. Hadoop helps Facebook in keeping track of all the profiles stored in it, along with the related data such as posts, comments, images, videos, and so on.

Linkedin manages over 1 billion personalized recommendations every week. All thanks to Hadoop and its MapReduce and HDFS features!

Today, more than half of the Fortune 50 companies run open source Apache Hadoop based on Cloudera

1.7.Scenarios to opt Hadoop technology in REAL TIME projects

1.7.1. Migration

Being open source Apache Hadoop has been the preferred choice of a number of organizations to replace the old, legacy software tools which demanded a heavy license fee to procure and a considerable fraction of it for maintenance. Unlike years ago, open source platforms have a large talent pool available for managers to choose from who can help design better, more accurate and faster solutions. Hadoop ecosystem has a very desirable ability to blend with popular programming and scripting platforms such as SQL, Java, Python and the like which makes migration projects easier to execute.

1.7.2. Integration

Businesses seldom start big. Most of them start as isolated, individual entities and grow over a period of time. Digital explosion of the present century has seen businesses undergo exponential growth curves. Given the operation and maintenance costs of centralized data centres, they often choose to expand in a decentralized, dispersed manner. Given the constraints imposed by time, technology, resources and talent pool, they end up choosing different technologies for different geographies and when it comes to integration. they find going tough.

That is where Apache Hadoop comes in. Given its ability to transfer, process and store data from heterogeneous sources in a fast, reliable and cost effective manner, they have been the preferred choice for integrating systems across organizations.

1.7.3. Scalability

As mentioned earlier, scalability is a huge plus with Apache Hadoop. Its ability to expand systems and build scalable solutions in a fast. efficient and cost effective manner outsmart a number of other alternatives. Apache Spark has been built in a way that it runs on top of Hadoop framework (for parallel processing of MapReduce jobs). As the data volumes grow, processing times noticeably go on increasing which adversely affects performance. Hadoop can be used to carry out data processing using either the traditional (map/reduce) or Spark based (providing interactive platform to process queries in real time) approach.

1.7.4. Cloud Hosting

Apache Hadoop is equally adept at hosting data at on-site, customer owned servers or in the Cloud. Cloud deployment saves a lot of time. cost and resources. Organizations are no longer required to spend over the top for procurement of servers and associated hardware infrastructure and then hire staff to maintain it. Instead, cloud service providers such as Google, Amazon and Microsoft provide hosting and maintenance services at a fraction of the cost. Cloud hosting also allows organizations to pay for actual space utilized whereas in procuring physical storage, companies have to keep in mind the growth rate and procure more space than required.

1.7.5. Specialized Data Analytics

Organizations often choose to store data in separate locations in a distributed manner rather than at one central location. Besides risk mitigation (which is the primary objective on most occasions) there can be other factors behind it such as audit, regulatory, advantages of localization, etc.

It is only logical to extract only the relevant data from warehouses to reduce the time and resources required for transmission and hosting. For example, in financial services there are a number of categories that require fast data processing (time series analysis, risk analysis, liquidity risk calculation, Monte Carlo simulations, etc.

Hadoop facilitates faster data extraction and processing to give actionable insights to users. Separate systems are built to carry out problem specific analysis and are programmed to use resources judiciously.

1.8.Challenges of Big Data

1.8.1.The Four -V's of Big Data

1.8.1.1.Volume

Big data is always large in volume. It actually doesn't have to be a certain number of petabytes to qualify. If your store of old data and new incoming data has gotten so large that you are having difficulty handling it, that's big data. Remember that it's going to keep getting bigger.Many small datasets that are considered big data do not consume much physical space but are particularly complex in nature. At the same time, large datasets that require significant physical space may not be complex enough to be considered big data.

The Cisco VNI Forecast—Historical Internet Context

Year	Global Internet Traffic
1992	100 GB per day
1997	100 GB per hour
2002	100 GBps

2007	2,000 GBps
2015	20,235 GBps
2020	61,386 GBps

Source: Cisco VNI, 2016

1.8.1.2.Variety

Variety points to the number of sources or incoming vectors leading to your databases.Variety references the different types of structured,semi-structuredand unstructured data that organizations can collect, such as transaction-level data, video, and audio, or text and log files.

1.8.1.3.Velocity

Velocity or speed refers to how fast the data is coming in, but also to how fast you need to be able to analyze and utilize it. If you have one or more business processes that require real-time data analysis, you have a velocity challenge.

1.8.1.4.Veracity

Veracity is an indication of data integrity and the ability for an organization to trust the data and be able to confidently use it to make crucial decisions.

1.9.How does Hadoop address the big Data changes

1.9.1.Hadoop is built to run on a cluster of machines

Lets start with an example. Let's say that we need to store lots of photos. We will start with a single disk. When we exceed a single disk, we may use a few disks stacked on a machine. When we max out all the disks on a single machine, we need to get a bunch of machines, each with a bunch of disks.This is exactly how Hadoop is built. Hadoop is designed to run on a cluster of machines from the get go.

1.9.2.Hadoop clusters scale horizontally

More storage and compute power can be achieved by adding more nodes to a Hadoop cluster. This eliminates the need to buy more and more powerful and expensive hardware.

1.9.3.Hadoop can handle unstructured/semi-structured data

Hadoop doesn't enforce a schema on the data it stores. It can handle arbitrary text and binary data. So Hadoop can digest any unstructured data easily.

1.9.4.Hadoop clusters provides storage and computing

We saw how having separate storage and processing clusters is not the best fit for big data. Hadoop clusters, however, provide storage and distributed computing all in one.

1.9.5.Hadoop provides storage for big data at reasonable cost

Storing big data using traditional storage can be expensive. Hadoop is built around commodity hardware, so it can provide fairly large storage for a reasonable cost. Hadoop has been used in the field at petabyte scale.

1.9.6.Hadoop allows for the capture of new or more data

Sometimes organizations don't capture a type of data because it was too cost prohibitive to store it. Since Hadoop provides storage at reasonable cost, this type of data can be captured and stored.

One example would be website click logs. Because the volume of these logs can be very high, not many organizations captured these. Now with Hadoop it is possible to capture and store the logs.

1.9.7.With Hadoop, you can store data longer

To manage the volume of data stored, companies periodically purge older data. For example, only logs for the last three months could be stored, while older logs were deleted. With Hadoop it is possible to store the historical data longer. This allows new analytics to be done on older historical data.

For example, take click logs from a website. A few years ago, these logs were stored for a brief period of time to calculate statistics like popular pages. Now with Hadoop, it is viable to store these click logs for longer period of time.

1.9.8.Hadoop provides scalable analytics

There is no point in storing all this data if we can't analyze them. Hadoop not only provides distributed storage, but also distributed processing as well, which means we can crunch a large volume of data in parallel. The compute framework of Hadoop is called MapReduce. MapReduce has been proven to the scale of petabytes.

1.9.9.Hadoop provides rich analytics

Native MapReduce supports Java as a primary programming language. Other languages like Ruby, Python and R can be used as well.

Of course, writing custom MapReduce code is not the only way to analyze data in Hadoop. Higher-level Map Reduce is available. For example, a tool named Pig takes English like data flow language and translates them into MapReduce. Another tool, Hive, takes SQL queries and runs them using MapReduce.

1.10.Comparison with other technologies

1.10.1.Extreme low cost per byte

HDFS uses commodity direct attached storage and shares the cost of the network & computers it runs on with the MapReduce / compute layers of the Hadoop stack. HDFS is open source software, so that if an organization chooses, it can be used with zero licensing and support costs. This cost advantage lets organizations store and process orders of magnitude more data per dollar than tradition SAN or NAS systems, which is the price point of many of these other systems. In big data deployments, the cost of storage often determines the viability of the system.

1.10.2.Very high bandwidth to support MapReduce workload

HDFS can deliver data into the compute infrastructure at a huge data rate, which is often a requirement of big data workloads. HDFS can easily exceed 2 gigabits per second per computer into the map-reduce layer, on a very low cost shared network. Hadoop can go

much faster on higher speed networks, but 10gigE, IB, SAN and other high-end technologies double the cost of a deployed cluster. These technologies are optional for HDFS. 2+ gigabits per second per computer may not sound like a lot, but this means that today's large Hadoop clusters can easily read/write more than a terabyte of data per second continuously to the MapReduce layer.

1.10.3.Rock solid data reliability

When deploying large distributed systems like Hadoop, the laws of probability are not on your side. Things will break every day, often in new and creative ways. Devices will fail and data will be lost or subtly mutated. The design of HDFS is focused on taming this beast. It was designed from the ground up to correctly store and deliver data while under constant assault from the gremlins that huge scale out unleashes in your data center. And it does this in software, again at low cost. Smart design is the easy part; the difficult part is hardening a system in real use cases. The only way you can prove a system is reliable is to run it for years against a variety of production applications at full scale. Hadoop has been proven in thousands of different use cases and cluster sizes, from startups to Internet giants and governments.

1.11.How does the HDFS competition stack up?

1.11.1.System not designed for Hadoop's scale

Many systems simply don't work at Hadoop scale. They haven't been designed or proven to work with very large data or many commodity nodes. They often will not scale up to petabytes of data or thousands of nodes. If you have a small use-case and value other attributes, such as integration with existing apps in your enterprise, maybe this is a good trade-off, but something that works well in a 10 node test system may fail utterly as your system scales up. Other systems don't scale operationally or rely on non-scalable hardware. Traditional NAS storage is a simple example of this problem. A NAS can replace Hadoop in a small cluster. But as the cluster scales up, cost and bandwidth issues come to the fore.

1.11.2.System that don't use commodity hardware or open source software

Many proprietary software / non-commodity hardware solutions are well tested and great at what they were designed to do. But, these solutions cost more than free software on commodity hardware. For small projects, this may be ok, but most activities have a finite

budget and a system that allows much more data to be stored and used at the same cost often becomes the obvious choice. The disruptive cost advantage of Hadoop & HDFS is fundamental to the current success and growing popularity of the platform. Many Hadoop competitors simply don't offer the same cost advantage.

1.11.3.Not designed for MapReduce's I/O patterns

Many of these systems are not designed from the ground up for Hadoop's big sequential scans & writes. Sometimes the limitation is in hardware. Sometimes it is in software. Systems that don't organize their data for large reads cannot keep up with MapReduce's data rates. Many databases and NoSql stores are simply not optimized for pumping data into MapReduce.

1.11.4.Unproven technology

Hadoop is interesting because it is used in production at extreme scale in the most demanding big data use cases in the world. As a result thousands of issues have been identified and fixed. This represents several hundred person-centuries of software development investment. It is easy to design a novel alternative system. A paper, a prototype or even a history of success in a related domain or a small set of use cases does not prove that a system is ready to take on Hadoop

1.12.Hadoop vs RDBMS

Disk latency has not improved proportionally to disk bandwidth i.e. seek time has not improved proportionally to transfer time. RDBMS uses B-tree for data access which is limited by disk latency, therefore it would take large time to access majority of data. Hadoop uses MapReduce model for data access which is limited by disk bandwidth. Hence for queries involving majority of database B-tree is less efficient than MapReduce.

RDBMS is more efficient for point queries where data is indexed to improve disk latency. Whereas Hadoop's MapReduce is more efficient for queries involving complete data.

Moreover MapReduce suits applications in which data is written once and read many times, whereas in RDBMS dataset is continuously updated.

	N0	MapReduce	RDBMS
1.	Size of data	Petabytes	Gigabytes
2.	Integrity of data	Low	High
3.	Data schema	Dynamic	Static
4.	Access method	Interactive and Batch	Batch
5.	Scaling	Linear	Nonlinear
6.	Data structure	Unstructured	Structured
7.	Normalization of data	Not Required	Required

1.13.Hadoop vs Data Warehouse

Data warehouse data models are static in the database, with both data integration and BI tools mapping directly to the models. They can be changed, although this may require days or even weeks because of the tightly integrated nature of the models and tools, versus hours with Hadoop.

Some data warehouses have not been able to keep up with the demand for new data sources, new types of data and different types of analysis. RDBMS databases are evolving by adding JSON and XML, which provide some, but not unlimited, schema flexibility.

1.14.Core Hadoop Components

The Hadoop Ecosystem comprises of 4 core components –

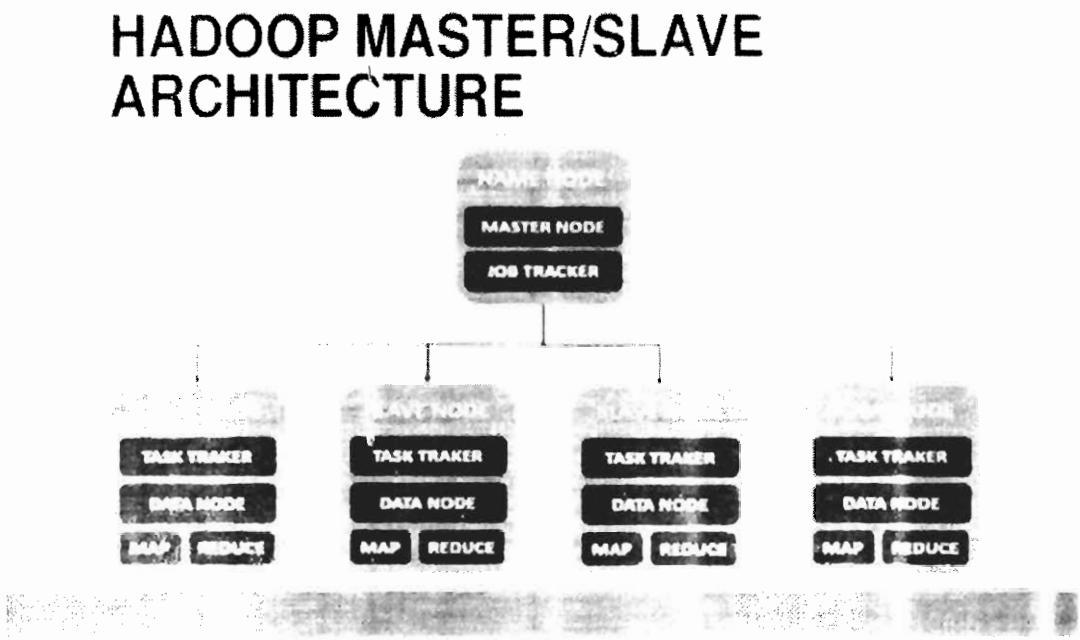
1.14.1. Hadoop Common

Apache Foundation has pre-defined set of utilities and libraries that can be used by other modules within the Hadoop ecosystem. For example, if HBase and Hive want to access HDFS they need to make of Java archives (JAR files) that are stored in Hadoop Common.

1.14.2. Hadoop Distributed File System (HDFS)

The default big data storage layer for Apache Hadoop is HDFS. HDFS is the "Secret Sauce" of Apache Hadoop components as users can dump huge datasets into HDFS and the data will sit there nicely until the user wants to leverage it for analysis. HDFS component creates several replicas of the data block to be distributed across different clusters for reliable and quick data

access. HDFS comprises of 3 important components-NameNode, DataNode and Secondary NameNode. HDFS operates on a Master-Slave architecture model where the NameNode acts as the master node for keeping a track of the storage cluster and the DataNode acts as a slave node summing up to the various systems within a Hadoop cluster.



HDFS Use Case-

Nokia deals with more than 500 terabytes of unstructured data and close to 100 terabytes of structured data. Nokia uses HDFS for storing all the structured and unstructured data sets as it allows processing of the stored data at a petabyte scale.

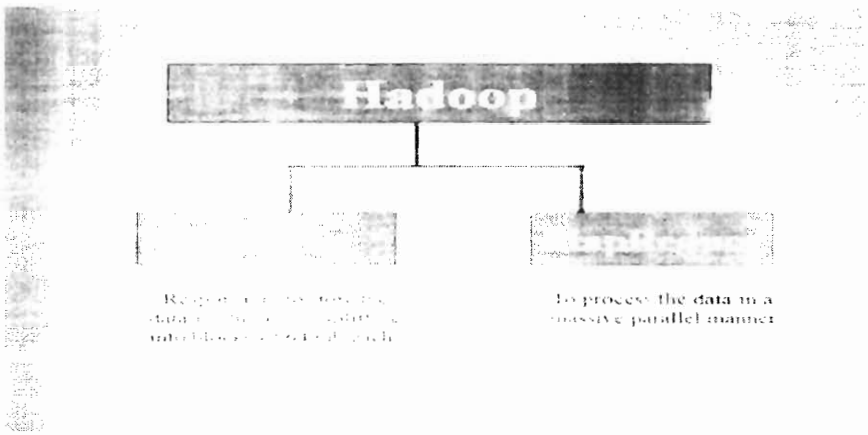
1.14.3.MapReduce- Distributed Data Processing Framework of Apache Hadoop

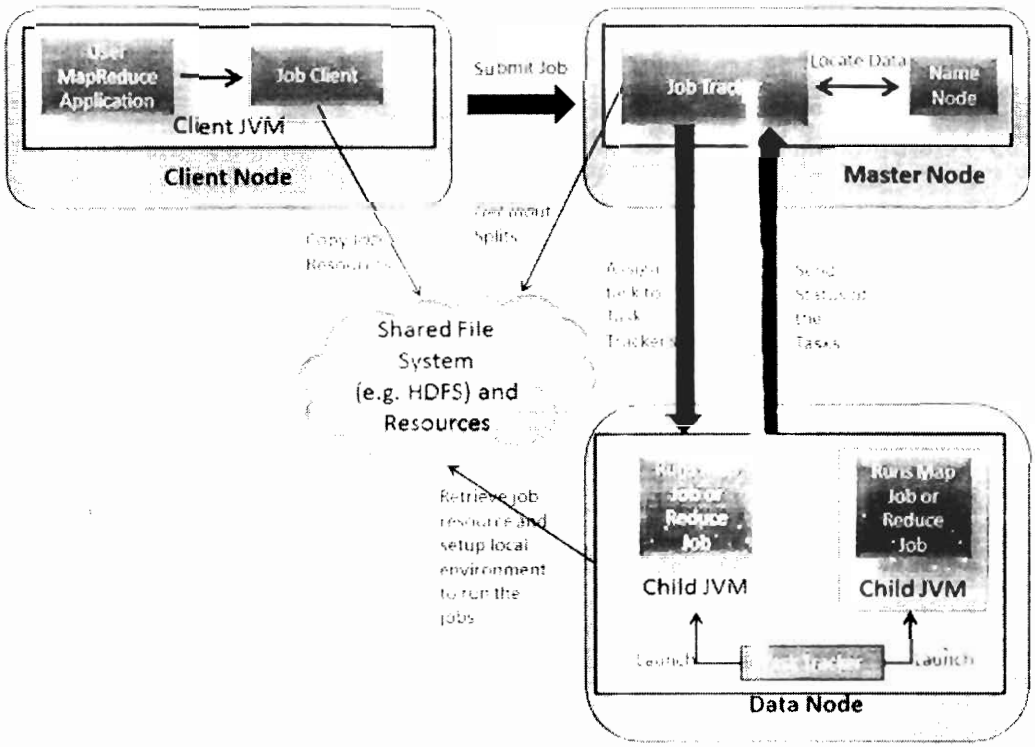
MapReduce is a Java-based system created by Google where the actual data from the HDFS store gets processed efficiently. MapReduce breaks down a big data processing job into smaller tasks. MapReduce is responsible for the analyzing large datasets in parallel before reducing it to find the results. In the Hadoop ecosystem, Hadoop MapReduce is a framework based on YARN architecture. YARN based Hadoop architecture, supports parallel processing of huge data sets and MapReduce provides the framework for easily writing applications on thousands of nodes, considering fault and failure management.

The basic principle of operation behind MapReduce is that the "Map" job sends a query for processing to various nodes in a Hadoop cluster and the "Reduce" job collects all the results to output into a single value. Map Task in the Hadoop ecosystem takes input data and splits into independent chunks and output of this task will be the input for Reduce Task. In The same Hadoop ecosystem Reduce task combines Mapped data tuples into smaller set of tuples. Meanwhile, both input and output of tasks are stored in a file system. MapReduce takes care of scheduling jobs, monitoring jobs and re-executes the failed task.

MapReduce framework forms the compute node while the HDFS file system forms the data node. Typically in the Hadoop ecosystem architecture both data node and compute node are considered to be the same.

The delegation tasks of the MapReduce component are tackled by two daemons- Job Tracker and Task Tracker as shown in the image below –





MapReduce Use Case:

Skybox has developed an economical image satellite system for capturing videos and images from any location on earth. Skybox uses Hadoop to analyse the large volumes of image data downloaded from the satellites. The image processing algorithms of Skybox are written in C++. Busboy, a proprietary framework of Skybox makes use of built-in code from java based MapReduce framework.

1.14.4.YARN

YARN forms an integral part of Hadoop 2.0.YARN is great enabler for dynamic resource utilization on Hadoop framework as users can run various Hadoop applications without having to bother about increasing workloads.

Hadoop 2 - YARN Architecture

ResourceManager (RM)

Central agent - Manages and allocates cluster resources

NodeManager (NM)

Per-Node agent - Manages and enforces node resource allocations

ApplicationMaster (AM)

Per-Application – Manages application lifecycle and task scheduling

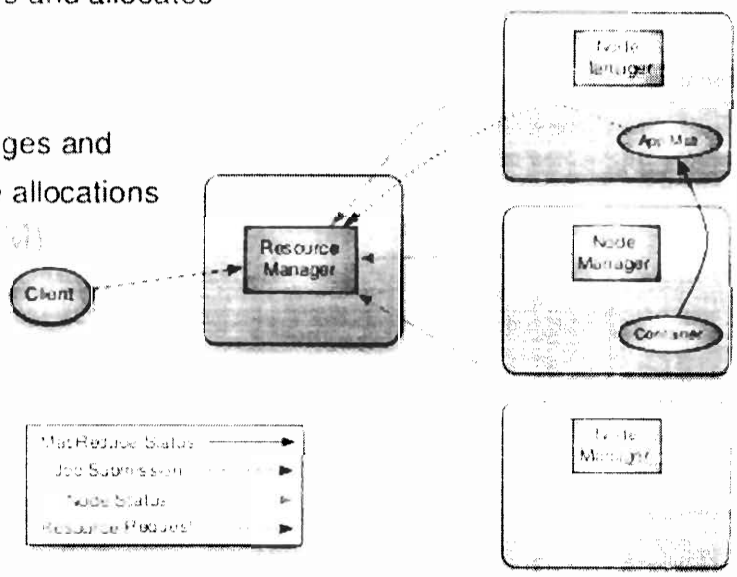


Image Credit: slidehshare.net

Key Benefits of Hadoop 2.0 YARN Component-

- It offers improved cluster utilization
- Highly scalable
- Beyond Java
- Novel programming models and services
- Agility

YARN Use Case:

Yahoo has close to 40,000 nodes running Apache Hadoop with 500,000 MapReduce jobs per day taking 230 compute years extra for processing every day. YARN at Yahoo helped them increase the load on the most heavily used Hadoop cluster to 125,000 jobs a day when compared to 80,000 jobs a day which is close to 50% increase.

The above listed core components of Apache Hadoop form the basic distributed Hadoop framework. There are several other Hadoop components that form an integral part of the Hadoop ecosystem with the intent of enhancing the power of Apache Hadoop in some way or the other like-providing better integration with databases, making Hadoop faster or developing novel features and functionalities

HDFS (Hadoop Distributed File System)

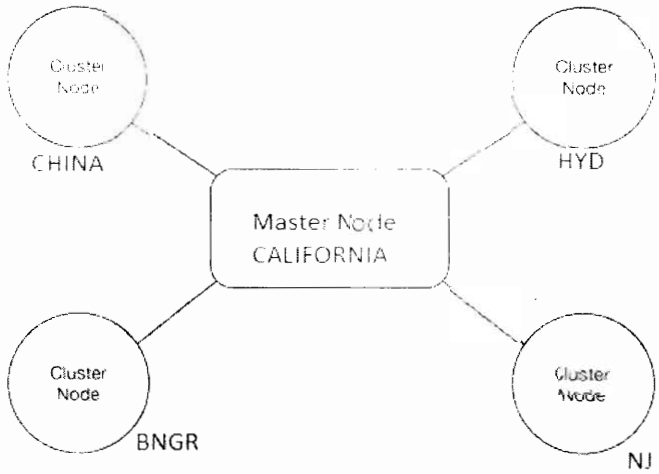
The Hadoop Distributed File System (HDFS) is designed to store very large data sets reliably, and to stream those data sets at high bandwidth to user applications.

1.15.Some of the basic terminology in Hadoop

1. Cluster
2. Hadoop Cluster Node
3. Hadoop Cluster

1.15.1.What is a cluster environment?

A cluster is a collection of all nodes being part of a common and dedicated network. It is also referred to as a Clustered Environment.



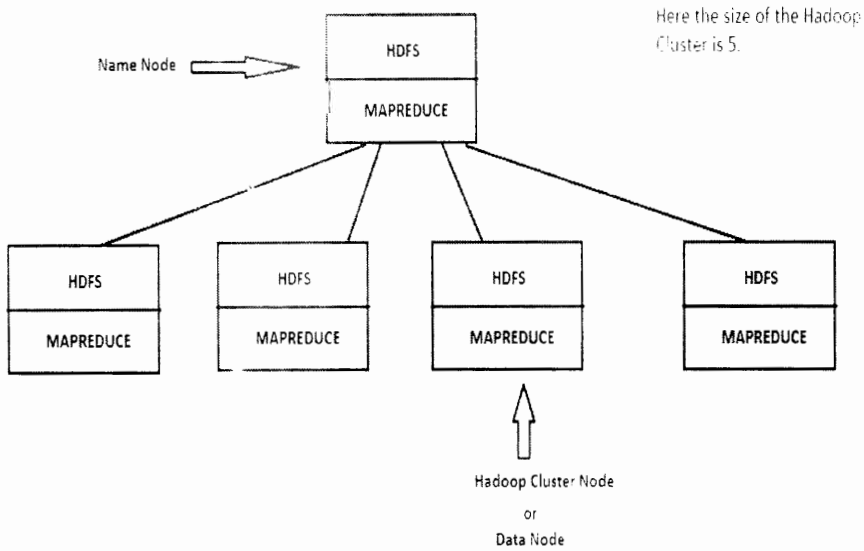
1.15.2.Hadoop Cluster Node

A Hadoop cluster node is a node which is part of a dedicated and common Hadoop network and has in its system both HDFS and MapReduce up and running.

1.15.3.Hadoop Cluster

A Hadoop Cluster is a collection of all the Hadoop cluster nodes being part of some common and dedicated network.

Hadoop clusters are often referred to as "shared nothing" systems because the only thing that is shared between nodes is the network that connects them.



Note : The pre-requisite of a Hadoop cluster is that all the systems or nodes in the cluster have to have HDFS and MapReduce installed, up and running.

In a typical RDBMS, the data is generally stored in tabular format. So the data should by in sync with the schema of the table. Where as in Hadoop (HDFS) the data is stored in a Schema-less File System (no-restrictions). The data is stored in "Blocks".

1.16.HDFS (Hadoop Distributed File System)

The Hadoop Distributed File System (HDFS) is the primary storage system used by Hadoop applications.

HDFS is a distributed file system that provides high-performance access to data across Hadoop clusters. When HDFS takes in data, it breaks the information down into smaller pieces (called blocks) and distributed throughout the cluster, allowing for parallel processing. The file system also copies each piece of data multiple times and distributes the copies to individual nodes, placing at least one copy on a different server rack than the others. As a result, the data on nodes that crash can be found elsewhere within a cluster, which allows processing to continue while the failure is resolved.

HDFS is built to support applications with large data sets, including individual files that reach into the terabytes. It uses a master/slave architecture, with each cluster consisting of

a single NameNode that manages file system operations and supporting DataNodes that manage data storage on individual compute nodes.Like other Hadoop-related technologies, HDFS has become a key tool for managing pools of big data and supporting big data analytics applications.

1.17.Features of HDFS

1.17.1.Fault Tolerance

Fault tolerance in HDFS refers to the working strength of a system in unfavourable conditions and how that system can handle such situations. HDFS is highly fault tolerant, in HDFS data is divided into blocks and multiple copies of blocks are created on different machines in the cluster (this replica creation is configurable). So whenever if any machine in the cluster goes down, then client can easily access their data from the other machine which contains the same copy of data blocks. HDFS also maintains the replication factor by creating replica of blocks of data on other rack. Hence if suddenly a machine fails, then user can access data from other slaves present in other rack. To learn more about Fault Tolerance follow this Guide.

1.17.2.High Availability

HDFS is a highly available file system, data gets replicated among the nodes in the HDFS cluster by creating replica of the blocks on the other slaves present in HDFS cluster. Hence whenever a user wants to access his data, they can access their data from the slaves which contains its blocks and which is available on the nearest node in the cluster. And during un-favorable situations like failure of a node, user can easily access their data from the other nodes. Because duplicate copies of blocks which contain user data are created on the other nodes present in the HDFS cluster. To learn more about high availability follow this Guide.

1.17.3.Reliability

HDFS is a distributed file system which provides reliable data storage. HDFS can store data in the range of 100s of petabytes. It stores data reliably on cluster of nodes. HDFS divides the data into blocks and these blocks are stored on nodes present in HDFS cluster.

It stores data reliably by creating replica of each and every block present on the nodes present in the cluster and hence provides fault tolerance facility. If node containing data goes down, then user can easily access that data from the other nodes which contains copy of same data in the HDFS cluster. HDFS by default creates 3 copies of blocks containing data present in the nodes in HDFS cluster. Hence data is quickly available to the users and hence user does not face the problem of data loss. Hence HDFS is highly reliable.

1.17.4.Replication

Data Replication is most important and unique feature of HDFS. In HDFS replication of data is done to solve the problem of data loss in unfavorable conditions like crashing of a node, hardware failure, and so on. As data is replicated across a number of machines in the cluster by creating blocks. The process of replication is maintained at regular intervals of time by HDFS and HDFS keeps creating replicas of user data on different machines present in the cluster. So whenever any machine in the cluster gets crashed, user can access their data from other machines which contains the blocks of that data. Hence there is no possibility of losing of user data. Follow this guide to learn more about the data read operation.

1. 7.5.Scalability

As HDFS stores data on multiple nodes in the cluster, when requirements increase we can scale the cluster. There are two scalability mechanism available: Vertical scalability – add more resources (CPU, Memory, Disk) on the existing nodes of the cluster. Another way is horizontal scalability – Add more machines in the cluster. Horizontal way is preferred as we can scale the cluster from 10s of nodes to 100s of nodes on the fly without any down-time.

1.17.6.Distributed Storage

In HDFS all the features are achieved via distributed storage and replication. In HDFS data is stored in distributed manner across the nodes in HDFS cluster. In HDFS data is divided into blocks and is stored on the nodes present in HDFS cluster. And then replicas of each and every block are created and stored on other nodes present in the cluster. So if a single

machine in the cluster gets crashed we can easily access our data from the other nodes which contains its replica.

1.18.Storage Aspects Of Hdfs

Filesystem Blocks: A block is the smallest unit of data that can be stored or retrieved from the disk. Filesystems deal with the data stored in blocks. Filesystem blocks are normally in few kilobytes of size. Blocks are transparent to the user who is performing filesystem operations like read and write.

1.18.1.HDFS Block

Hadoop distributed file system stores the data in terms of blocks. : A block is the smallest unit of data that can be stored or retrieved from the disk. HDFS deals with the data stored in blocks However the block size in HDFS is very large. The default size of HDFS block is 64MB. The files are split into 64MB blocks and then stored into the hadoop filesystem. The hadoop application is responsible for distributing the data blocks across multiple nodes

Default block size in Hadoop = 64 mb

Block size can be configured to 128 MB, 256MB, 512MB or 1024MB (1GB)

1.18.2.How to configure the block size

hdfs-site.xml is used to configure HDFS.Changing the **dfs.block.size** property in **hdfs-site.xml** will change the default block size for all the files placed into HDFS. Changing this setting will not affect the block size of any files currently in HDFS. It will only affect the block size of files placed into HDFS after this setting has taken effect.

Open the **hdfs-site.xml** file. This file is usually found in the **conf/** folder of the Hadoop installation directory. Change or add the following property to **hdfs-site.xml**:

i.e., /home/Hadoop/conf/hdfs-site.xml

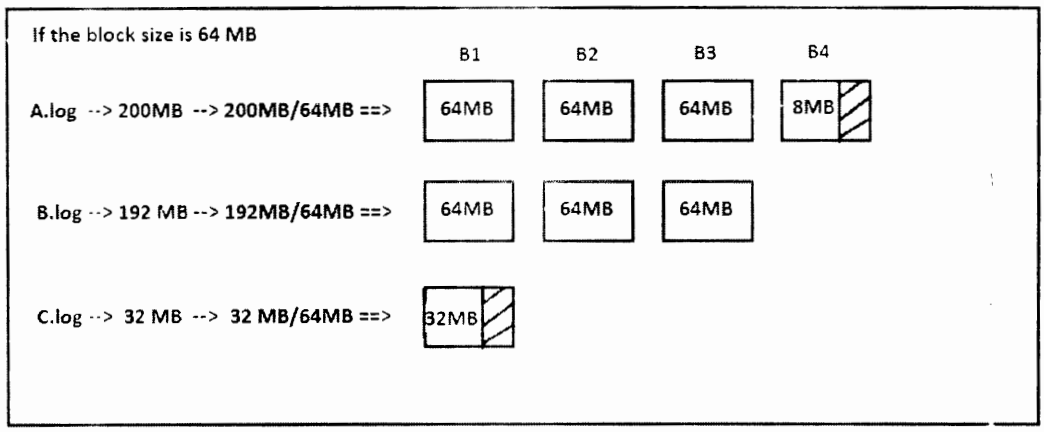
```
<property>
    <name>dfs.block.size</name>
    <value>134217728</value>
    <description>Block size</description>
</property>
```

NOTE

In this case, we set the **dfs.block.size** to 128 MB.

Value is in bytes

When a Master Node receives a File , the HDFS divides the data into the block sizes according to the values specified in the hdfs-site.xml. Example of how it is accomplished is illustrated in the below diagram.



Note: Except the last block, the rest of the blocks will hold equal amounts of data

Note: Minimum block size is 64 MB and Maximum block size is 128 MB

1.19.Why is a Block in HDFS So Large?

HDFS blocks are large compared to disk blocks, and the reason is to minimize the cost of seeks. By making a block large enough, the time to transfer the data from the disk can be made to be significantly larger than the time to seek to the start of the block. Thus the time to transfer a large file made of multiple blocks operates at the disk transfer rate. A quick calculation shows that if the seek time is around 10 ms, and the transfer rate is 100 MB/s, then to make the seek time 1% of the

transfer time, we need to make the block size around 100 MB. The default is actually 64 MB, although many HDFS installations use 128 MB blocks.

Second, making the unit of abstraction a block rather than a file simplifies the storage subsystem (since blocks are a fixed size, it is easy to calculate how many can be stored on a given disk). Having a block abstraction for a distributed filesystem brings several benefits. The first benefit is the most obvious: a file can be larger than any single disk in the network. There's nothing that requires the blocks from a file to be stored on the same disk, so they can take advantage of any of the disks in the cluster.

1.20.HDFS ARCHITECTURE

HDFS has a master and slaves architecture in which the master is called the *name node* and slaves are called *data nodes* . An HDFS cluster consists of a single name node that manages the file system namespace (or metadata) and controls access to the files by the client applications, and multiple data nodes (in hundreds or thousands) where each data node manages file storage and storage device attached to it.

Hadoop architecture can be generally reflected in two components

1. Storage Architecture
2. Processing Architecture

The Storage Architecture in Hadoop Further Classified into 3 Daemons

- Name Node
- Data Node
- Secondary Name Node

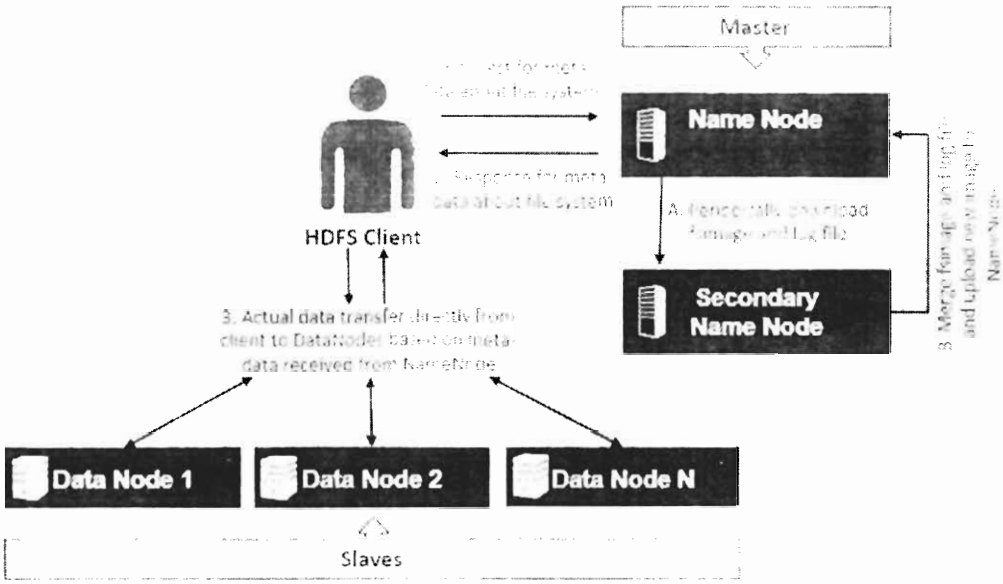


Illustration of Storage Architecture in Hadoop

1.20.1.Name Node

The NameNode in Hadoop is the node where Hadoop stores all the location information of the files in HDFS. In other words, it holds the metadata for HDFS. Whenever a file is placed in the cluster a corresponding entry of it location is maintained by the NameNode.

The name node is also responsible for managing file system namespace operations, including opening, closing, and renaming files and directories. The name node records any changes to the file system namespace or its properties. The name node contains information related to the replication factor of a file, along with the map of the blocks of each individual file to data nodes where those blocks exist.

- The name node keeps metadata related to the file system namespace in memory, for quicker response time. Hence, more memory is needed.
- The name node coordinates with hundreds or thousands of data nodes and serves the requests coming from client applications.

All this information, along with information related to data nodes and blocks stored in each of the data nodes, is recorded in the file system name space, called fsimage and stored as a file on the local host OS file system at the name node

daemon. This fsimage file is not updated with every addition or removal of a block in thefile system. Instead, the name node logs and maintains these add/remove operations in a separate edit log file, which exists as another file on the local host OS file system. Appending updates to a separate edit log achieves faster I/O.

1.20.2.Data Node

Datanodes are the workhorses of the filesystem. They store and retrieve blocks when they are told to (by clients or the namenode), and they report back to the namenode periodically with lists of blocks that they are storing.

Data nodes are responsible for serving read and write requests from the HDFS clients and perform operations such as block creation, deletion, and replication when the name node tells them to. Data nodes store and retrieve blocks when they are told to (by the client applications or by the name node), and they report back to the name node periodically with lists of blocks that they are storing, to keep the name node up to date on the current status.

A client application talks to the name node to get metadata information about the file system. It connects data nodes directly so that they can transfer data back and forth between the client and the data nodes.

The client communicates with the name node to get only metadata; the actual data transfer happens between the client and the data nodes. The name node is not involved in the actual data transfer.

1.20.3.Secondary name node

A secondary name node is another daemon. Contrary to its name, the secondary name node is not a standby name node, so it is not meant as a backup in case of name node failure. The primary purpose of the secondary name node is to periodically download the name node fsimage and edit the log file from the name node, create a new fsimage by merging the older fsimage and edit the log file. and upload the new fsimage back to the name node. By periodically merging the namespace fsimage with the edit log, the secondary name node prevents the edit log from becoming too large.

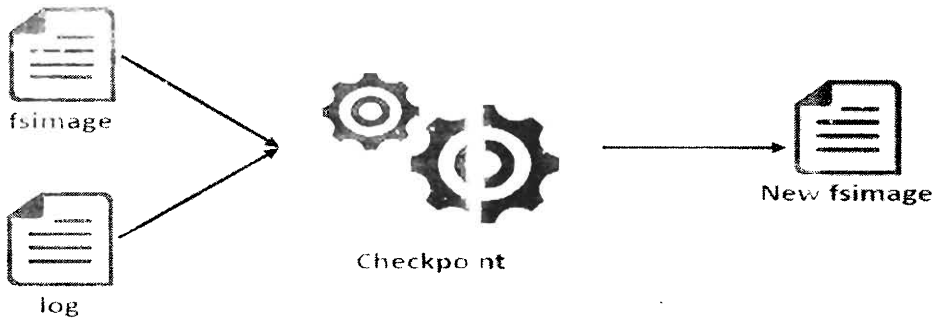
The name node machine is a single point of failure, so manual intervention is needed if it fails. When ever the name node is down , the secondary name node must be manually

triggered.. RPC (Remote Procedure Calls) will help continuos polling between name node and secondary name node

NOTE: Name Node: fsimage and Edit Log File

The fsimage and the edit log file are central data structures that contain HDFS file system metadata and namespaces. Any corruption of these files can cause the HDFS cluster instance to become nonfunctional. For this reason, the name node can be configured to support maintaining multiple copies of the fsimageand edit log to another machine. This is where the secondary name node comes into play.

he process of generating a new fsimage from a merge operation is called the *Checkpoint process* . Usually the secondary name node runs on a separate physical machine than the name node; it also requires plenty of CPU and as much as memory as the name node to perform the Checkpoint operation.



Checkpoint process

1.20.4.JobTracker

JobTracker is a deamon which is the component of processing architecture. JobTracker daemon runs on either name node or in large clusters , a designated job tracker node and has the responsibility of accepting map-reduce jobs from user and split it in sub-tasks. It then delegated each of the subtasks to different TaskTracker daemons.

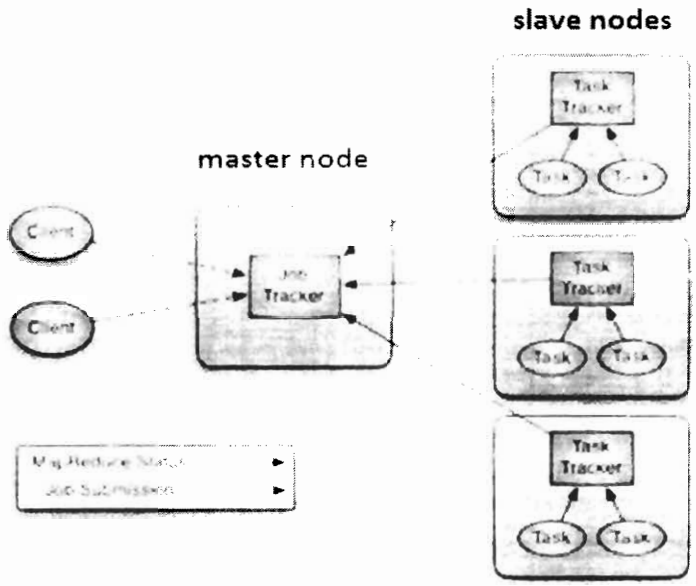
Its main responsibilities are:

1. Assigning the tasks to Task Tracker
2. Scheduling of the tasks
3. If any task fails, then how to re-schedule the tasks

1.20.5.TaskTracker

TaskTracker is a daemon which is the component of processing architecture. TaskTracker daemon runs in each of the slave nodes and has the responsibility of managing (running and status reporting) the tasks assigned to it by Job Tracker.A TaskTracker accepts tasks – like Map, Reduce and Shuffle operations - from a JobTracker. The TaskTrackers also send out heartbeat messages to the JobTracker, usually every few minutes, to reassure the JobTracker that it is still alive.

Note: On each and every data node, one Task tracker runs by default.



1.21.Replication in Hadoop

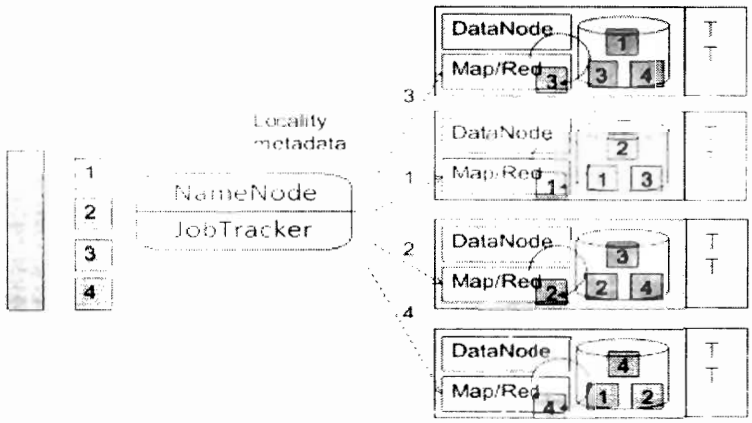
When a file is created in HDFS , the application can specify the number of replicas of each block of the file that HDFS must maintain. Multiple copies or replicas of each block makes it fault tolerant: If one copy is not accessible or gets corrupted, the data can be read from the other copy. The number of copies of each block is called the *replication factor* for a file, and it applies to all blocks of a file

The NameNode makes all decisions regarding replication of blocks.

The default replication value is 3, i.e. there will be 3 copies of the same block in the cluster.

1.21.1.Data Storage in Data Nodes

While storing a file, HDFS internally splits it into one or more blocks (chunks of 64MB, by default, which is configurable and can be changed at cluster level or when each file is created). These blocks are stored in a set of slaves, called *data nodes,* to ensure that parallel writes or reads can be done even on a single file. Multiple copies of each block are stored per replication factor (which is configurable and can be changed at the cluster level, or at file creation, or even at a later stage for a stored file) for making the platform devoid of faults.



TT= TaskTracker

1.21.2.Replilcation Configuration

hdfs-site.xml is used to configure HDFS. Changing the dfs.replication property in hdfs-site.xml will change the default replication for all files placed in HDFS.

Open the **hdfs-site.xml** file. This file is usually found in the **conf/** folder of the Hadoop installation directory. Change or add the following property to **hdfs-site.xml**:

i.e., /home/Hadoop/conf/hdfs-site.xml

To change the replication factor , change the value in the '<value> -</value>'

Stats for Replication:

The minimum Replication in Hadoop is 1.

Default Replication in Hadoop is 3

Maximum Replication in Hadoop is 512.

1.22.Core Design Principles of Hadoop

1.Replication is only applicable to Hadoop file system data but not the Meta Dat (data about data)

2.Keeps only one replica per slave node / data node per design

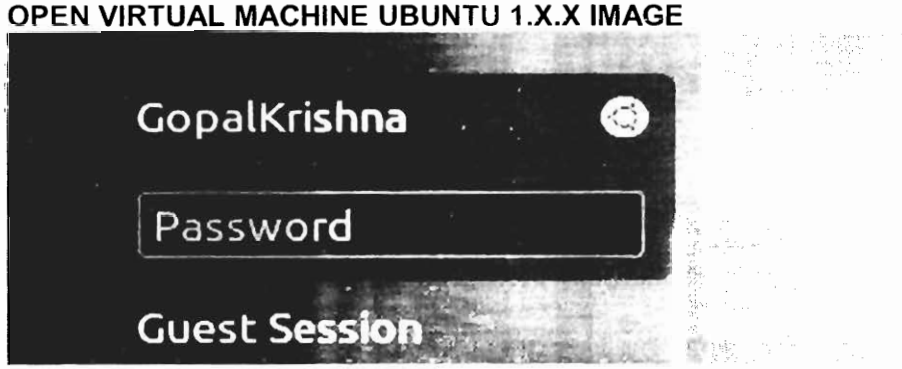
3.Replication only happens on Hadoop salve nodes but not on master nodes; As master node is only meant for meta-data management.

- Hadoop master data will only care about the block-size and replication when writing data only (not while reading /extracting data)
- There is not impact on already stored data on old block size if the block size or replication factor is changed.
- After the block size /replication factor is changed, the whole Hadoop cluster needs to be restarted.

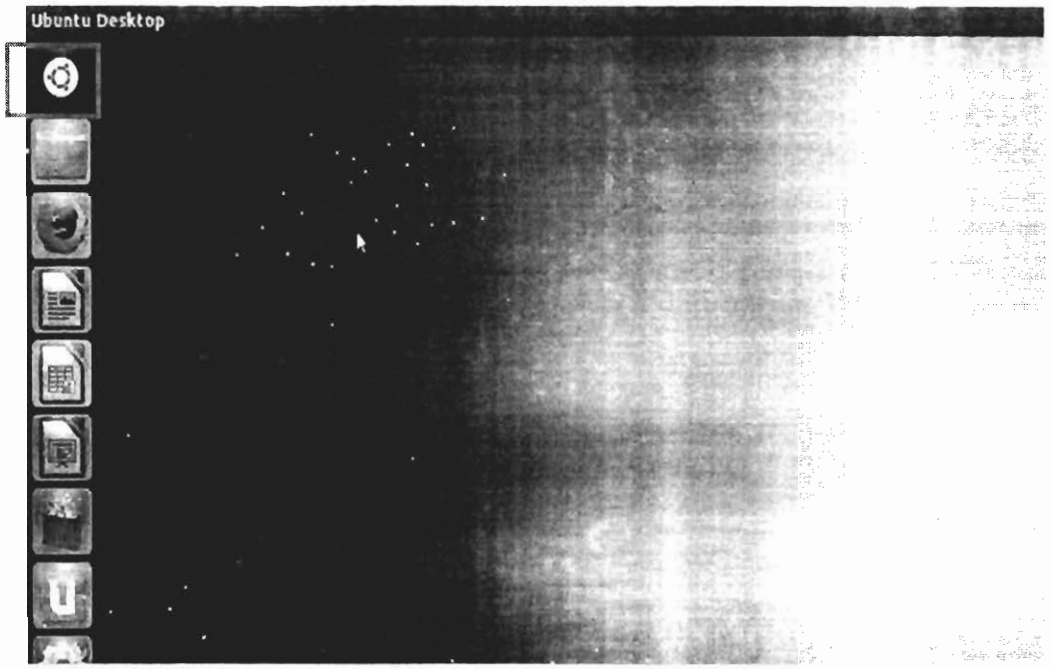
1.23.Commads Guide

1 . Basic Commands

OPEN VMWARE
OPEN VIRTUAL MACHINE UBUNTU 1.X.X IMAGE

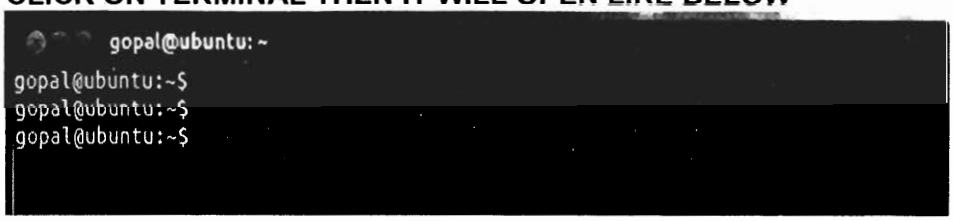


ENTER PASSWORD :gopal@123
AFTER ENTER PASSWORD IT WILL OPEN UBUNTU 1.X.XIMAGE



CLICK ON DASH BOARD AND TYPE TERMINAL OR (CTRL+ALT+T)

CLICK ON TERMINAL THEN IT WILL OPEN LIKE BELOW



LINUX COMMANDS

FOR DATE AND CALENDER

```
gopal@ubuntu: ~
gopal@ubuntu:~$
gopal@ubuntu:~$
gopal@ubuntu:~$ date
Sat Mar 18 05:52:42 PDT 2017
gopal@ubuntu:~$ cal
     March 2017
Su Mo Tu We Th Fr Sa
             1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
```

- mkdir: **MAKING A DIRECTORY IN LFS**

```
gopal@ubuntu:~$
gopal@ubuntu:~$ mkdir GOPAL
gopal@ubuntu:~$
```

- cd : **CHANGE TO GOPAL DIRECTORY**

```
gopal@ubuntu:~$ cd GOPAL
gopal@ubuntu:~/GOPAL$
```

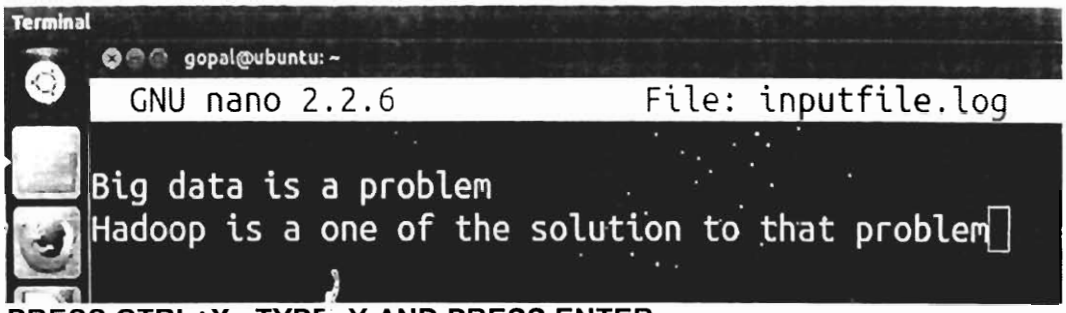
- pwd : **TO KNOW THE PRESENT WORKING DIRECTORY**

```
gopal@ubuntu: ~
gopal@ubuntu:~$
gopal@ubuntu:~$ pwd
/home/gopal
```

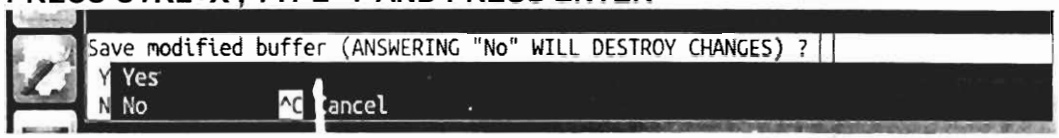
- nano filename: **TO CREATE NEW FILE IN LFS USING NANO EDITOR**

```
gopal@ubuntu:~/GOPAL$
gopal@ubuntu:~/GOPAL$ nano inputfile.log
```

INSERT DATA INTO inputfile.log

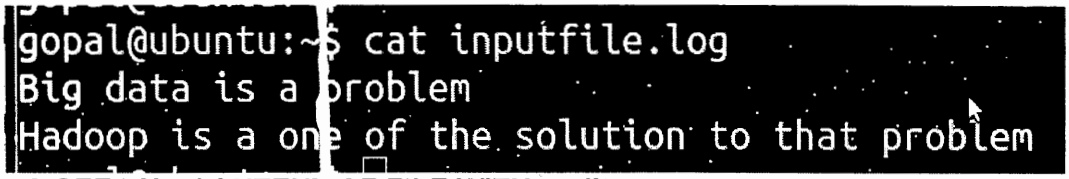


PRESS CTRL+X , TYPE Y AND PRESS ENTER

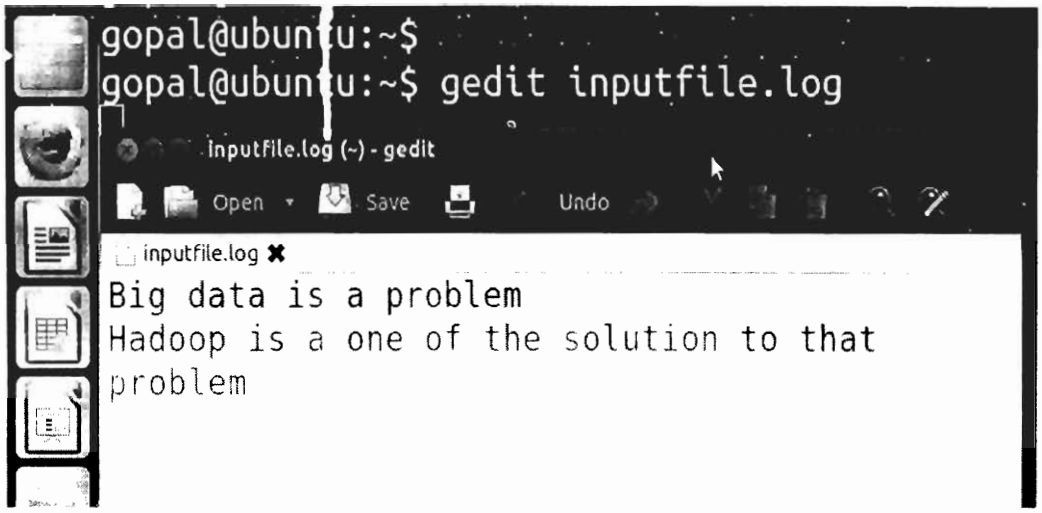


TO SEE THE CONTENT OF THAT A FILE

- **cat filename**



TO SEE THE CONTENT OF FILE WITH gedit



BY USING gedit YOU WILL GET A NOTEPAD LIKE EDITOR IF YOU WANT TO SAVE CONTENT OF gedit CLICK ON SAVEBUTTON& CLOSE THE FILE
TO CLEAR A TERMINAL TYPE CLEAR Or (CTRL+L)FROM ANY PATH

- **touch : TO CREATE EMPTYFILE USE TOUCH COMMAND**

```
gopal@ubuntu:~$
gopal@ubuntu:~$ touch dummyfile.log
gopal@ubuntu:~$
gopal@ubuntu:~$ ll dummyfile.log
-rw-r--r-- 1 gopal hadoop 0 Mar 18 06:15 dummyfile.log
gopal@ubuntu:~$
```

- **ll : TO SEE THE LONGLIST OF FILES**

```
gopal@ubuntu:~$
gopal@ubuntu:~$ ll
total 276
drwxr-xr-x 31 gopal hadoop  4096 Mar 18 06:15 ./
drwxr-xr-x  3 root  root    4096 Jul 13  2015 ../
-rw-r--r--  1 gopal hadoop    28 Feb  1 21:28 a1.log
-rw-------  1 gopal gopal   9888 Mar  8 04:52 .bash_history
-rw-r--r--  1 gopal hadoop   220 Jul 13  2015 .bash_logout
-rw-r--r--  1 gopal hadoop  4295 Feb 11 01:38 .bashrc
drwx------ 19 gopal hadoop  4096 Feb  1 22:12 .cache/
drwx------ 13 gopal hadoop  4096 Mar  8 23:02 .config/
-rw-r--r--  1 gopal hadoop  1306 Feb  4 21:48 .dbshell
```

- **ls**: TO SEE THE LIST OF COMMANDS

```
gopal@ubuntu:~$
gopal@ubuntu:~$ ls
a1.log      dummyfile         MangoDBJDBC.java  Pictures                    Templates
derby.log   examples.desktop  metastore_db      pig_1485846343060.log       Videos
Desktop     GOPAL             mongo             pig_1486810223994.log       workspace
Documents   inputfile.log     MongoDBJDBC.java  pig_1489058168244.log
Downloads   kalyani           Music             Public
```

- **cp file1 file2**
- **COPY CONTENTS ONE PLACE TO ANTHOR PATH**

```
gopal@ubuntu:~
gopal@ubuntu:~$
gopal@ubuntu:~$
gopal@ubuntu:~$ cp inputfile.log  input.log
gopal@ubuntu:~$
gopal@ubuntu:~$
gopal@ubuntu:~$ cat input.log
Big data is a problem
Hadoop is a one of the solution to that problem
gopal@ubuntu:~$
```

- **mv file1path file2path**

MOVING OR RENAME FILE NAME

```
gopal@ubuntu:~$
gopal@ubuntu:~$ mv input.log inputNEW.log
gopal@ubuntu:~$ cat inputNEW.log
Big data is a problem
Hadoop is a one of the solution to that problem
```

- **m filename : TO REMOVE FILE**

```
gopal@ubuntu:~$
gopal@ubuntu:~$ rm inputfile.log
gopal@ubuntu:~$
```

- **gzip filename : COMPRESS THE FILE IN LFS**

```
gopal@ubuntu:~$ gzip inputNEW.log
gopal@ubuntu:~$ ll
```

- **gunzip filename: TO UNZIP THE FILE**

```
gopal@ubuntu:~$
gopal@ubuntu:~$ ll inputfile.log.gz
-rw-r--r-- 1 gopal hadoop 91 Mar 18 04:03 inputfile.log.gz
gopal@ubuntu:~$
gopal@ubuntu:~$ gunzip inputfile.log.gz
gopal@ubuntu:~$
gopal@ubuntu:~$ ll inputfile.log
-rw-r--r-- 1 gopal hadoop 70 Mar 18 04:03 inputfile.log
gopal@ubuntu:~$
```

User Group 777 → others

Write 42 1 → Execute, Read

Chmod 477

Problem (How to reset permission again?)

- **chmod : TO CHANGE FILE ACCESS PERMISSIONS**

```
gopal@ubuntu:~$ ll a1.log
-rw-r--r-- 1 gopal hadoop 28 Feb  1 21:28 a1.log
gopal@ubuntu:~$ chmod 777 a1.log
gopal@ubuntu:~$ ll a1.log
-rwxrwxrwx 1 gopal hadoop 28 Feb  1 21:28 a1.log*
```

- **du: SHOWS AMOUNT OF SPACE IN BYTES**
- **df :SHOWS THE CAPACITY USED AND FREE SPACE**

```
gopal@ubuntu:~
gopal@ubuntu:~$
gopal@ubuntu:~$ du inputfile.log
4       inputfile.log
gopal@ubuntu:~$
gopal@ubuntu:~$ df inputfile.log
Filesystem     1K-blocks    Used Available Use% Mounted on
/dev/sda1       28768380 6652640  20631352  25% /
gopal@ubuntu:~$
```

TO KNOW HOST NAME

- **hostname**

```
gopal@ubuntu:~$
gopal@ubuntu:~$ hostname
ubuntu
gopal@ubuntu:~$
```

TO KNOW THE IFCONFIGURATIONS

- **ifconfig**

```
gopal@ubuntu:~$
gopal@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:5e:8e:51
          inet addr:192.168.20.131  Bcast:192.168.20.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe5e:8e51/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:232 errors:0 dropped:0 overruns:0 frame:0
          TX packets:145 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:34250 (34.2 KB)  TX bytes:17985 (17.9 KB)
          Interrupt:19 Base address:0x2000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:34 errors:0 dropped:0 overruns:0 frame:0
          TX packets:34 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:2754 (2.7 KB)  TX bytes:2754 (2.7 KB)
```

HDFS COMMANDS

HELP FOR ALL COMMANDS

```
gopal@ubuntu: ~

gopal@ubuntu:~$ hadoop
Usage: hadoop [--config confdir] COMMAND
where COMMAND is one of:
  namenode -format     format the DFS filesystem
  secondarynamenode    run the DFS secondary namenode
  namenode             run the DFS namenode
  datanode             run a DFS datanode
  dfsadmin             run a DFS admin client
  mradmin              run a Map-Reduce admin client
  fsck                 run a DFS filesystem checking utility
  fs                   run a generic filesystem user client
  balancer             run a cluster balancing utility
  oiv                  apply the offline fsimage viewer to an fsimage
  fetchdt              fetch a delegation token from the NameNode
  jobtracker           run the MapReduce job Tracker node
  pipes                run a Pipes job
  tasktracker          run a MapReduce task Tracker node
  historyserver        run job history servers as a standalone daemon
  job                  manipulate MapReduce jobs
  queue                get information regarding JobQueues
  version              print the version
  jar <jar>            run a jar file
  distcp <srcurl> <desturl> copy file or directories recursively
```

BY TYPING JPS WE CAN SEE DAEMONS

- **jps**

```
gopal@ubuntu: ~
gopal@ubuntu:~$
gopal@ubuntu:~$
gopal@ubuntu:~$ jps
5665 Jps
gopal@ubuntu:~$
```

- **start-all.sh** : TO START ALL DAEMONS

```
gopal@ubuntu:~$
gopal@ubuntu:~$ start-all.sh
starting namenode, logging to /usr/local/hadoop-1.2.1/libexec/../logs/hadoop-gopal-namenode-ubuntu.out
localhost: starting datanode, logging to /usr/local/hadoop-1.2.1/libexec/../logs/hadoop-gopal-datanode-ubun
tu.out
localhost: starting secondarynamenode, logging to /usr/local/hadoop-1.2.1/libexec/../logs/hadoop-gopal-seco
ndarynamenode-ubuntu.out
starting jobtracker, logging to /usr/local/hadoop-1.2.1/libexec/../logs/hadoop-gopal-jobtracker-ubuntu.out
localhost: starting tasktracker, logging to /usr/local/hadoop-1.2.1/libexec/../logs/hadoop-gopal-tasktracke
r-ubuntu.out
gopal@ubuntu:~$ jps
6636 Jps
6558 TaskTracker
6227 SecondaryNameNode
6322 JobTracker
5984 DataNode
5749 NameNode
gopal@ubuntu:~$
```

TO KNOW ABOUT **SAFEMODE RELATED STATUS**

```
gopal@ubuntu: ~
gopal@ubuntu:~$
gopal@ubuntu:~$
gopal@ubuntu:~$
gopal@ubuntu:~$ hadoop dfsadmin -safemode get
Safe mode is OFF
gopal@ubuntu:~$
gopal@ubuntu:~$ hadoop dfsadmin -safemode enter
Safe mode is ON
gopal@ubuntu:~$
gopal@ubuntu:~$
gopal@ubuntu:~$ hadoop dfsadmin -safemode leave
Safe mode is OFF
```

HADOOP MAKE A DIRECTORY

- **hadoop fs –mkdir /directoryname**

```
gopal@ubuntu:~$
gopal@ubuntu:~$ hadoop fs -mkdir /GOPAL_HDFS
gopal@ubuntu:~$
```

- **hadoop fs –ls /**
- **LIST THE DIRECTORIES IN HDFS**

```
gopal@ubuntu:~$
gopal@ubuntu:~$ hadoop fs -ls /
Found 7 items
drwxr-xr-x   - gopal supergroup          0 2017-03-18 03:59 /GOPAL_HDFS
drwxr-xr-x   - gopal supergroup          0 2017-03-09 03:37 /PIG
drwxr-xr-x   - gopal supergroup          0 2017-03-08 23:57 /app
drwxr-xr-x   - gopal supergroup          0 2017-03-08 04:48 /hbase
drwxr-xr-x   - gopal supergroup          0 2017-03-09 00:37 /helthcareoutput
drwxr-xr-x   - gopal supergroup          0 2017-03-08 23:19 /pdf
drwxr-xr-x   - gopal supergroup          0 2017-03-09 03:37 /tmp
gopal@ubuntu:~$
```

- **hadoop fs –put LFSFILE /HDFS_DIR**
- **LOAD DATA FROM LOCAL SYSTEM TO HDFS**

```
gopal@ubuntu:~$
gopal@ubuntu:~$ hadoop fs -put input.log  /GOPAL_HDFS
gopal@ubuntu:~$
gopal@ubuntu:~$
gopal@ubuntu:~$ hadoop fs -ls /GOPAL_HDFS
Found 2 items
-rw-r--r--   1 gopal supergroup         70 2017-03-18 04:03 /GOPAL_HDFS/input.log
-rw-r--r--   1 gopal supergroup         70 2017-03-18 04:00 /GOPAL_HDFS/inputNEW.log
```

- **hadoop fs –ls /GOPAL_HDFS**
- **TO SEE THE LIST OF FILES IN PARTICULAR DIRECTORY IN HDFS**

```
gopal@ubuntu:~$
gopal@ubuntu:~$
gopal@ubuntu:~$ hadoop fs -ls /GOPAL_HDFS
Found 2 items
-rw-r--r--   1 gopal supergroup          0 2017-03-18 04:21 /GOPAL_HDFS/EmptyFile.txt
-rw-r--r--   1 gopal supergroup         70 2017-03-18 04:15 /GOPAL_HDFS/input.log
gopal@ubuntu:~$
```

- **hadoop fs –cat /GOPAL_HDFS/INPUTFILENAME**
- **TO SEE THE CONTENT OF A FILE WHICH IS LOCATED UNDER A PARTICULAR HDFS DIRECTORY**

```
gopal@ubuntu:~$
gopal@ubuntu:~$ hadoop fs -cat /GOPAL_HDFS/input.log
Big data is a problem
Hadoop is a one of the solution to that problem
gopal@ubuntu:~$
```

- **moveFromLocal:**
- **TO MOVE FILE LFS TO HDFS**

```
gopal@ubuntu:~$
gopal@ubuntu:~$
gopal@ubuntu:~$ hadoop fs -moveFromLocal abcd.log /GOPAL_HDFS
gopal@ubuntu:~$
gopal@ubuntu:~$
gopal@ubuntu:~$ hadoop fs -ls /GOPAL_HDFS
Found 3 items
-rw-r--r--   1 gopal supergroup          0 2017-03-18 04:21 /GOPAL_HDFS/EmptyFile.txt
-rw-r--r--   1 gopal supergroup         76 2017-03-18 20:57 /GOPAL_HDFS/abcd.log
-rw-r--r--   1 gopal supergroup         70 2017-03-18 04:15 /GOPAL_HDFS/input.log
gopal@ubuntu:~$
```

- **AFTER TRIGGER moveFromLocal COMMAND WE CANNOT FIND FILE IN LFS**
- **copyFromLocal: TO COPY DATA FROM LFS**

```
Terminal
gopal@ubuntu: ~
gopal@ubuntu:~$
gopal@ubuntu:~$
gopal@ubuntu:~$ hadoop fs -copyFromLocal sampleData.log /GOPAL_HDFS
gopal@ubuntu:~$ hadoop fs -ls /GOPAL_HDFS
Found 2 items
-rw-r--r--   1 gopal supergroup         70 2017-03-18 04:00 /GOPAL_HDFS/inputNEW
.log
-rw-r--r--   1 gopal supergroup         69 2017-03-18 04:08 /GOPAL_HDFS/sampleDa
ta.log
gopal@ubuntu:~$
```

- **AFTER TRIGGER copyFromLocal COMMAND WE CAN FIND FILE IN LFS AND HDFS**
- **-mv : TO MOVE HDFS DIR1 TO HDFS DIRECTORY2**

```
gopal@ubuntu:~$
gopal@ubuntu:~$ hadoop fs -mkdir /GOPAL_HDFS1
gopal@ubuntu:~$
gopal@ubuntu:~$ hadoop fs -mv /GOPAL_HDFS/sampleData.log /GOPAL_HDFS1
gopal@ubuntu:~$ hadoop fs -ls /GOPAL_HDFS1
Found 1 items
-rw-r--r--   1 gopal supergroup         69 2017-03-18 04:08 /GOPAL_HDFS1/sampleD
ata.log
```

- hadoop dfsadmin –report : TO KNOW THE CLUSTER HELTH

```
gopal@ubuntu:~$ hadoop dfsadmin -report
Configured Capacity: 29458821120 (27.44 GB)
Present Capacity: 21178306560 (19.72 GB)
DFS Remaining: 21129400320 (19.68 GB)
DFS Used: 48906240 (46.64 MB)
DFS Used%: 0.23%
Under replicated blocks: 13
Blocks with corrupt replicas: 0
Missing blocks: 0

-------------------------------------------------
Datanodes available: 1 (1 total, 0 dead)

Name: 127.0.0.1:50010
Decommission Status : Normal
Configured Capacity: 29458821120 (27.44 GB)
DFS Used: 48906240 (46.64 MB)
Non DFS Used: 8280551424 (7.71 GB)
DFS Remaining: 21129363456(19.68 GB)
DFS Used%: 0.17%
DFS Remaining%: 71.73%
Last contact: Sat Mar 18 04:19:31 PDT 2017
```

- touchz : TO CREATE EMPTY FILE IN HDFS

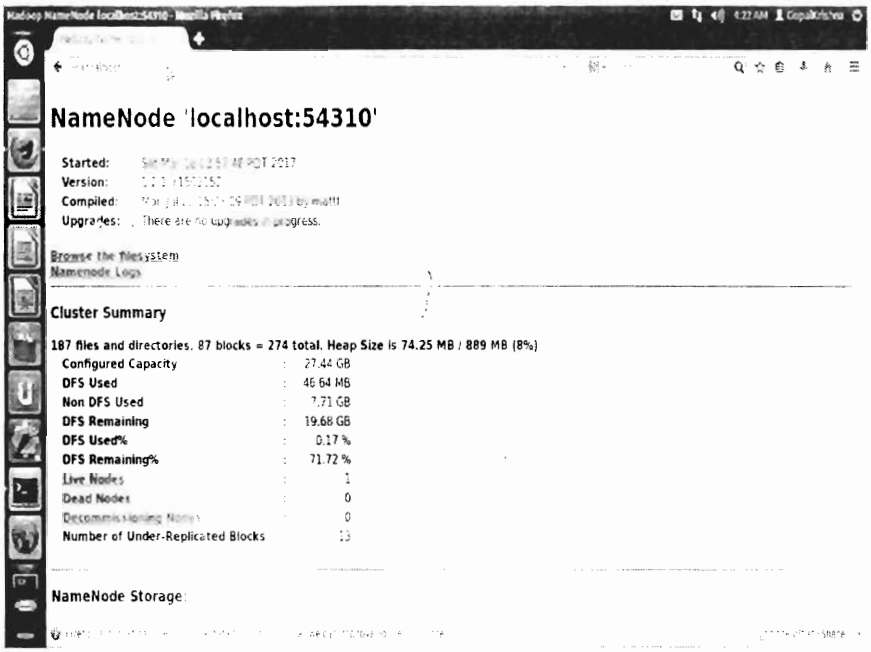
```
gopal@ubuntu:~$ hadoop fs -touchz /GOPAL_HDFS/EmptyFile.txt
gopal@ubuntu:~$ hadoop fs -ls /GOPAL_HDFS
Found 2 items
-rw-r--r--   1 gopal supergroup          0 2017-03-18 04:21 /GOPAL_HDFS/EmptyFil
e.txt
-rw-r--r--   1 gopal supergroup         70 2017-03-18 04:15 /GOPAL_HDFS/input.lo
g
```

TO GET DATA HDFS TO LFS USE BELOW COMMANDS

- -get
- -copyToLocal

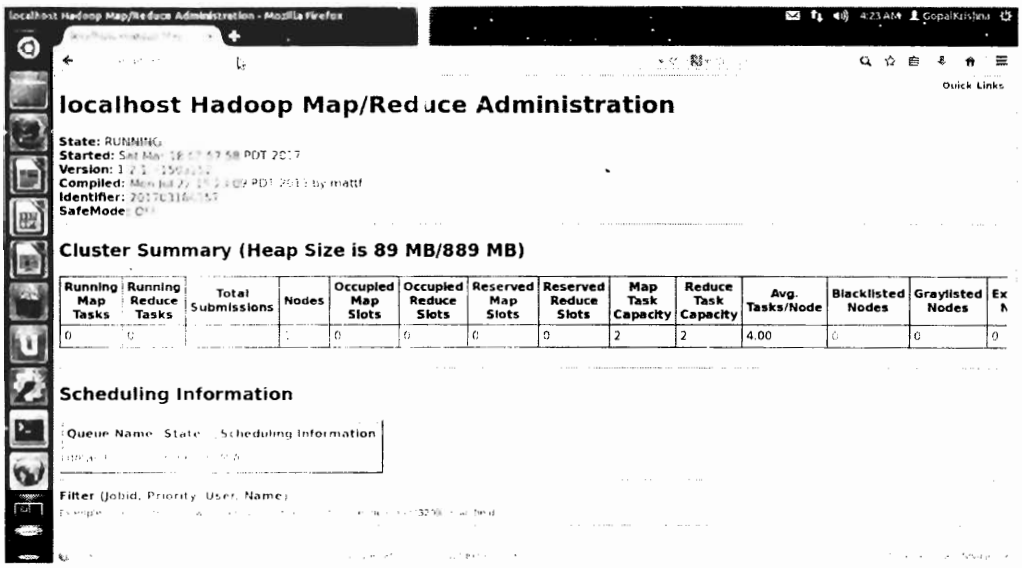
```
gopal@ubuntu:~$
gopal@ubuntu:~$
gopal@ubuntu:~$
gopal@ubuntu:~$ hadoop fs -get /GOPAL_HDFS/sampleData.log /home/gopal/
gopal@ubuntu:~$

gopal@ubuntu:~$
gopal@ubuntu:~$ hadoop fs -ls /GOPAL_HDFS
Found 3 items
-rw-r--r--   1 gopal supergroup          0 2017-03-18 04:21 /GOPAL_HDFS/EmptyFile.txt
-rw-r--r--   1 gopal supergroup         76 2017-03-18 20:57 /GOPAL_HDFS/abcd.log
-rw-r--r--   1 gopal supergroup         70 2017-03-18 04:15 /GOPAL_HDFS/input.log
gopal@ubuntu:~$
gopal@ubuntu:~$ hadoop fs -copyToLocal /GOPAL_HDFS/input.log /home/gopal/
gopal@ubuntu:~$
gopal@ubuntu:~$ ll
total 288
```

TO ACCESS NAMENODE IN GUI
TYPE localhost:50070in BROWSER



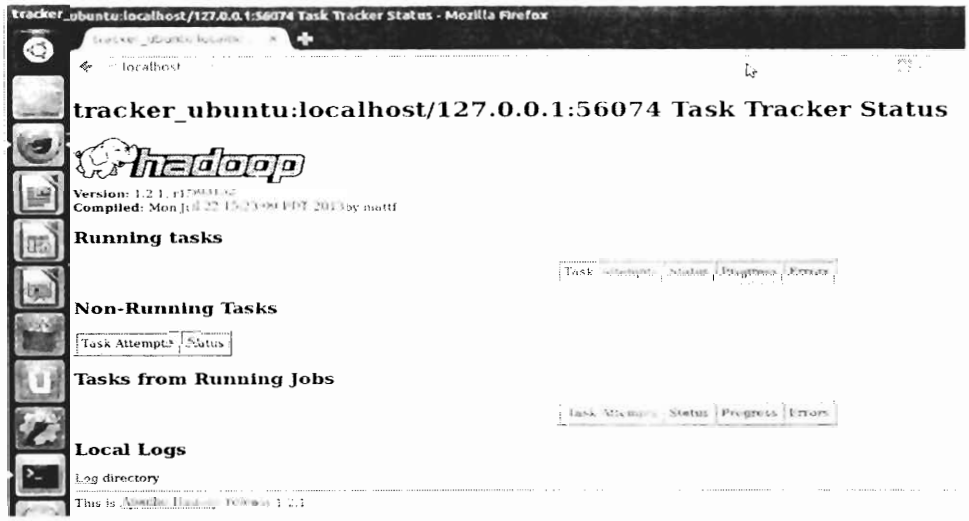
TO ACCESS JOBTRACKER IN GUI

- **TYPE localhost:50030 in BROWSER**



TO ACCESS TASKTRACKER IN GUI

- **TYPE localhost:50060 in BROWSER**



2. User Commands

Commands useful for users of a hadoop cluster.

2.1. archive

Creates a hadoop archive. More information can be found at Hadoop Archives.

```
Usage: hadoop archive -archiveName NAME <src>* <dest>
```

-archiveName NAME	Name of the archive to be created.
Src	Filesystem pathnames which work as usual with regular expressions.
Dest	Destination directory which would contain the archive.

2.2. distcp

Copy file or directories recursively. More information can be found at Hadoop DistCp Guide.

```
Usage: hadoopdistcp<srcurl><desturl>
```

Srcurl	Source Uri
Desturl	Destination Url

2.3. fs

```
Usage: hadoop fs [GENERIC_OPTIONS] [COMMAND OPTIONS]
```

Runs a generic filesystem user client.

The various COMMAND_OPTIONS can be found at File System Shell Guide.

2.4. fsck

Runs a HDFS filesystem checking utility. See Fsckfor more info.

```
Usage: hadoopfsck [GENERIC_OPTIONS] <path> [-move | -delete |
-openforwrite] [-files [-blocks [-locations | -racks]]]
```

<path>	Start checking from this path.
-move	Move corrupted files to /lost+found
-delete	Delete corrupted files.
-openforwrite	Print out files opened for write.
-files	Print out files being checked.
-blocks	Print out block report.
-locations	Print out locations for every block.
-racks	Print out network topology for data-node locations.

2.5. fetchdt

Gets Delegation Token from a NameNode. See fetchdtfor more info.

```
Usage: hadoopfetchdt [GENERIC_OPTIONS] [--
webservice<namenode_http_addr>] <path>
```

<fileName>	File name to store the token into.
--webservice<namenode_http_addr>	use http protocol instead of RPC

2.6. jar

Runs a jar file. Users can bundle their Map Reduce code in a jar file and execute it using this command.

```
Usage: hadoop jar <jar> [mainClass] args. .
```

The streaming jobs are run via this command. Examples can be referred from Streaming examples

Word count example is also run using jar command. It can be referred from Wordcount example

2.7.job

Command to interact with Map Reduce Jobs.

```
Usage: hadoop job [GENERIC_OPTIONS] [-submit <job-file>] | [-status <job-id>] | [-counter <job-id><group-name><counter-name>] | [-kill <job-id>] | [-events <job-id><from-event-#><#-of-events>] | [-history [all] <jobOutputDir>] | [-list [all]] | [-kill-task <task-id>] | [-fail-task <task-id>] | [-set-priority <job-id><priority>]
```

-submit <job-file>	Submits the job.
-status <job-id>	Prints the map and reduce completion percentage and all job counters.
-counter <job-id><group-name> <counter-name>	Prints the counter value.
-kill <job-id>	Kills the job.
-events <job-id><from-event-#> <#-of-events>	Prints the events' details received by jobtracker for the given range.
-history [all] <jobOutputDir>	-history <jobOutputDir> prints job details, failed and killed tip details. More details about the job such as successful tasks and task attempts made for each task can be viewed by specifying the [all] option.
-list [all]	-list all displays all jobs. -list displays only jobs which are yet to complete.
-kill-task <task-id>	Kills the task. Killed tasks are NOT counted against failed attempts.
-fail-task <task-id>	Fails the task. Failed tasks are counted against failed attempts.
-set-priority <job-id><priority>	Changes the priority of the job. Allowed priority values are VERY_HIGH, HIGH, NORMAL, LOW, VERY_LOW

2.8. pipes

Runs a pipes job.

```
Usage: hadoop pipes [-conf<path>] [-jobconf<key=value>,
<key=value>, ...] [-input <path>] [-output <path>] [-jar
<jar file>] [-inputformat<class>] [-map <class>] [-
partitioner<class>] [-reduce <class>] [-writer <class>] [-
program <executable>] [-reduces <num>]
```

-conf<path>	Configuration for job
-jobconf<key=value>, <key=value>, ...	Add/override configuration for job
-input <path>	Input directory
-output <path>	Output directory
-jar <jar file>	Jar filename
-inputformat<class>	InputFormat class
-map <class>	Java Map class
-partitioner<class>	Java Partitioner
-reduce <class>	Java Reduce class
-writer <class>	Java RecordWriter
-program <executable>	Executable URI
-reduces <num>	Number of reduces

2.9. queue

command to interact and view Job Queue information

```
Usage : hadoop queue [-list] | [-info <job-queue-name>
[-showJobs]] | [-showacls]
```

-list	Gets list of Job Queues configured in the system. Along with scheduling information associated with the job queues.
-info <job-queue-name> [-showJobs]	Displays the job queue information and associated scheduling information of particular job queue. If -showJobs options is present a list of jobs submitted to the particular job queue is displayed.

`-showacls`	Displays the queue name and associated queue operations allowed for the current user. The list consists of only those queues to which the user has access.

2.10. version

Prints the version.

```
Usage: hadoop version
```

2.11. CLASSNAME

hadoop script can be used to invoke any class.

```
Usage: hadoop CLASSNAME
```

Runs the class named CLASSNAME.

2.12. classpath

Prints the class path needed to get the Hadoop jar and the required libraries.

```
Usage: hadoopclasspath
```

3. Administration Commands

Commands useful for administrators of a hadoop cluster.

3.1. balancer

Runs a cluster balancing utility. An administrator can simply press Ctrl-C to stop the rebalancing process. See Rebalancerfor more details.

```
Usage: hadoop balancer [-threshold <threshold>]
```

`-threshold <threshold>`	Percentage of disk capacity. This overwrites the default threshold.

3.2. daemonlog

Get/Set the log level for each daemon.

```
Usage: hadoopdaemonlog -getlevel<host:port><name>
Usage: hadoopdaemonlog -setlevel<host:port><name><level>
```

-getlevel<host:port><name>	Prints the log level of the daemon running at <host:port>. This command internally connects to http://<host:port>/logLevel?log=<name>
-setlevel<host:port><name><level>	Sets the log level of the daemon running at <host:port>. This command internally connects to http://<host:port>/logLevel?log=<name>

3.3. datanode

Runs a HDFS datanode.

```
Usage: hadoopdatanode [-rollback]
```

-rollback	Rollsback the datanode to the previous version. This should be used after stopping the datanode and distributing the old hadoop version.

3.4. dfsadmin

Runs a HDFS dfsadmin client.

```
Usage: hadoopdfsadmin [GENERIC_OPTIONS] [-report] [-safemode
enter | leave | get | wait] [-refreshNodes] [-finalizeUpgrade]
[-upgradeProgress status | details | force] [-metasave filename]
[-setQuota<quota><dirname>...<dirname>] [-
clrQuota<dirname>...<dirname>] [-help [cmd]]
```

-report	Reports basic filesystem information and statistics.
-safemode enter | leave | get | wait	Safe mode maintenance command. Safe mode is a Namenode state in which it 1. does not accept changes to the name space (read-only) 2. does not replicate or delete blocks. Safe mode is entered automatically at Namenode startup, and leaves safe mode

	automatically when the configured minimum percentage of blocks satisfies the minimum replication condition. Safe mode can also be entered manually, but then it can only be turned off manually as well.
-refreshNodes	Re-read the hosts and exclude files to update the set of Datanodes that are allowed to connect to the Namenode and those that should be decommissioned or recommissioned.
-finalizeUpgrade	Finalize upgrade of HDFS. Datanodes delete their previous version working directories, followed by Namenode doing the same. This completes the upgrade process.
-upgradeProgress status | details | Force	Request current distributed upgrade status, a detailed status or force the upgrade to proceed.
-metasave filename	Save Namenode's primary data structures to <filename> in the directory specified by hadoop.log.dir property. <filename> will contain one line for each of the following 1. Datanodes heart beating with Namenode 2. Blocks waiting to be replicated 3. Blocks currrently being replicated 4. Blocks waiting to be deleted
-setQuota quota <dirname>...<dirname>	Set the quota <quota> for each directory <dirname>. The directory quota is a long integer that puts a hard limit on the number of names in the directory tree. Best effort for the directory, with faults reported if 1. N is not a positive integer, or 2. user is not an administrator, or 3. the directory does not exist or is a file, or 4. the directory would immediately exceed the new quota.
-clrQuota dirname...<dirname>	Clear the quota for each directory <dirname>. Best effort for the directory, with fault reported if 1. the directory does not exist or is a file, or 2. user is not an administrator. It does not fault if the directory has no quota.
-help [cmd]	Displays help for the given command or all commands if none is specified

3.5. mradmin

Runs MR admin client

```
Usage: hadoopmradmin [ GENERIC_OPTIONS] [-refreshQueueAcls]
```

`-refreshQueueAcls`	Refresh the queue acls used by hadoop, to check access during submissions and administration of the job by the user. The properties present in `mapred-queue-acls.xml` is reloaded by the queue manager.

3.6. jobtracker

Runs the MapReduce job Tracker node.

```
Usage: hadoopjobtracker [-dumpConfiguration]
```

`-dumpConfiguration`	Dumps the configuration used by the JobTracker alongwith queue configuration in JSON format into Standard output used by the jobtracker and exits.

3.7. namenode

Runs the namenode. More info about the upgrade, rollback and finalize is at Upgrade Rollback

```
Usage: hadoopnamenode [-format] | [-upgrade] | [-rollback] |
[-finalize] | [-importCheckpoint]
```

`-format`	Formats the namenode. It starts the namenode, formats it and then shut it down.
`-upgrade`	Namenode should be started with upgrade option after the distribution of new hadoop version.
`-rollback`	Rollsback the namenode to the previous version. This should be used after stopping the

	cluster and distributing the old hadoop version.
`-finalize`	Finalize will remove the previous state of the files system. Recent upgrade will become permanent. Rollback option will not be available anymore. After finalization it shuts the namenode down.
`-importCheckpoint`	Loads image from a checkpoint directory and save it into the current one. Checkpoint dir is read from property fs.checkpoint.dir

3.8. secondarynamenode

Runs the HDFS secondary namenode. See Secondary Namenodefor more info.

```
Usage: hadoopsecondarynamenode [-checkpoint [force]] | [-
geteditsize]
```

`-checkpoint [force]`	Checkpoints the Secondary namenode if EditLog size >= fs.checkpoint.size. If -force is used, checkpoint irrespective of EditLog size.
`-geteditsize`	Prints the EditLog size.

MAP REDUCE

2.1.Map Reduce

A core processing component of hadoop which is meant for processing of huge data in a parallel fashion on commodity hardware meachines.

The MapReduce algorithm contains two important tasks, namely Map and Reduce. 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 a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

2.2.Daemons of Hadoop

Name Node | Secondary Name Node | Data Node

HDFS Demons

Job Tracker | Task Tracker

MapReduce Demons

Basically MapReduce uses two daemons name Job Tracker and Task Tracker.

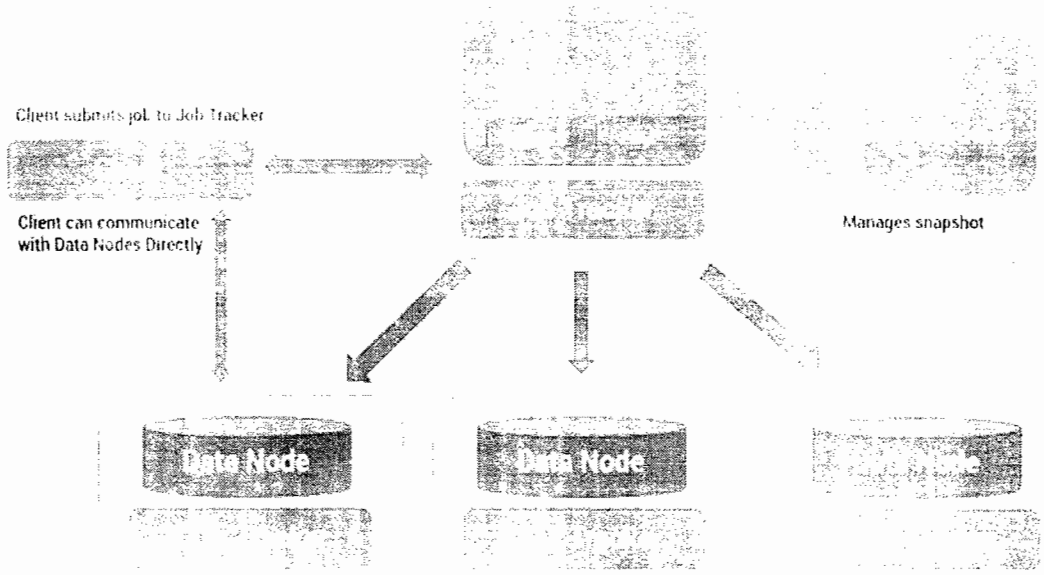
Job Tracker

- Only one instance of this process runs on a master node same as Name Node
- Any MapReduce job is submitted to Job Tracker first
- Job Tracker checks for the location various file blocks used in MapReduce processing
- Than it initiates the separate tasks on various Data Nodes(where blocks are present) by communicating with Task Tracker Daemons

Task Tracker

- This process has multiple instances running on the slave nodes(typically runs on the slave nodes where Data Node process is running)
- It receives the job information from Job Tracker daemon and initiates a task on that slave node
- In most of the cases, Task Tracker initiates the task on the same node where there physical data block is present
- Same as Data Node daemon, this process also periodically sends heart bits to Job Tracker to make Job Tracker aware that slave process is running.

2.3.Hadoop 1.x Architecture



2.4.Limitations of Hadoop 1.x Architecture

- As we also have seen in above description, major drawback of Hadoop 1.x Architecture is Single Point of Failure as there is no backup Name Node.

- Job scheduling, resource management and job monitoring are being done by Job Tracker which is tightly coupled with Hadoop. So Job Tracker is not able to manage resources outside Hadoop.

- Job Tracker has to coordinate with all task tracker so in a very big cluster it will be difficult to manage huge number of task trackers altogether.

Job Tracker may fail because of the overloaded with responsibilities. sometimes : this problem is solved in latest versions of MapReduce by spilting of the two major functionalities of JobTracker (resource management and job scheduling/monitoring) into two separate daemons: a global Resource Manager and per-application ApplicationMaster.

2.5.MapReduce Phases

MapReduce program executes in three phase, namely mapper phase, Sort&shuffle phase, and reducer stage.

- **Mapper Phase** : The map or mapper's job is to process the input data Generally the input data is in the form of file or directory and is stored in the Hadoop file system (HDFS). The input file is passed to the mapper function line by line. The mapper processes the data and creates several small chunks of data.The input for mapper phase is (key,value) pair and o/p is (key,value) pair.

- **Sort&Shuffle Phase**:It is framework driven phase. The input for Sort & shuffle phase is (key, value) pair and o/p is (key,list of values) .

- **Reduce Phase** :The Reducer's job is to process the data that comes from the Sort &Shuffle phase, after processing, it produces a new set of output, which will be stored in the HDFS.

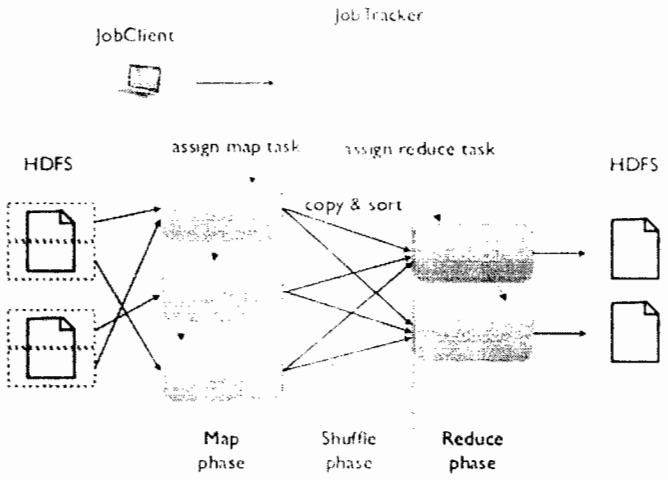
- During a MapReduce job, Hadoop sends the Map and Reduce tasks to the appropriate servers in the cluster.

- The framework manages all the details of data-passing such as issuing tasks, verifying task completion and copying data around the cluster between the nodes.

- Most of the computing takes place on nodes with data on local disks that reduces the network traffic.

- After completion of the given tasks, the cluster collects and reduces the data to form an appropriate result, and sends it back to the Hadoop server.

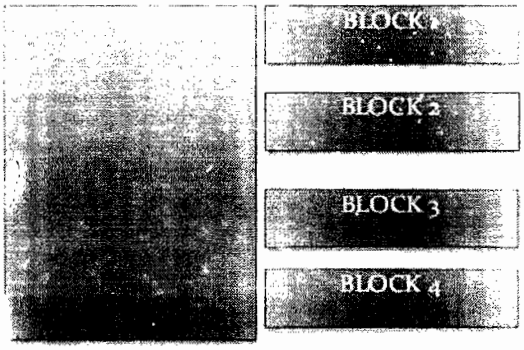
Hadoop MapReduce



Input Files

- *Input files* are where the data for a MapReduce task is initially stored
- The input files typically reside in a distributed file system (e.g. HDFS)
- The format of input files is arbitrary
  - Line-based log files
  - Binary files
  - Multi-line input records
  - Or something else entirely

HDFS Blocks



- Big File is divided into multiple blocks and stored in hdfs.
- This is a physical division of data
- dfs.block.size (64MB default size)

InputFormat

- An InputFormat is responsible for creating the input splits and dividing them into records.
- How we get the data to mapper

```
public abstract class InputFormat<K, V> {

    public abstract List<InputSplit> getSplits(JobContext context)
    throws IOException, InterruptedException;

    public abstract RecordReader<K, V>
    createRecordReader(InputSplit split, TaskAttemptContext
    context) throws IOException, InterruptedException;
}
```

InputSplits and Records

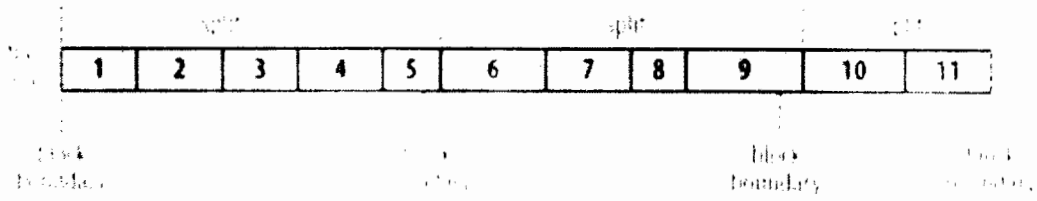
Input split LOGICAL DIVISION

- A chunk of data processed by a mapper
- Further divided into records
- Map process these records
  Record = key + value
- How to correlate to a DB table
  Group of rows → split
  Row → record

Key	Value
R1	R1
R2	R2
R3	R3

InputSplit and HDFS blocks

- So even in data locality some remote reading is done.. a slight overhead.
- Split gives logical record boundaries
- Blocks – physical boundaries (size)

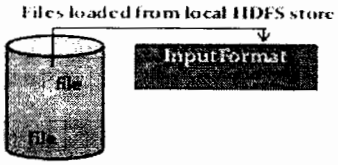


InputFormat

- How the input files are split up and read is defined by the *InputFormat*

- InputFormat is a class that does the following:

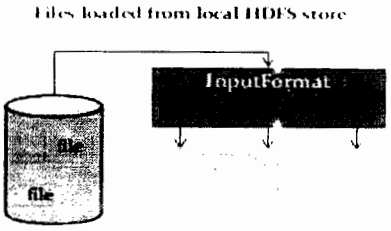
1. Select the files that should be used for input
2. Defines the *InputSplits* that break a file
3. Provides a factory for *RecordReader* objects that read the file



InputSplit Continued...

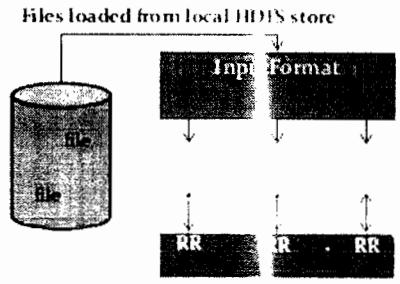
- An *input split* describes a unit of work that comprises a single map task in a MapReduce program
- By default, the InputFormat breaks a file up into 64MB splits
- By dividing the file into splits, we allow several map tasks to operate on a single file in parallel
- If the file is very large, this can improve performance significantly through parallelism

- Each map task corresponds to a *single* input split.



RecordReader

- The input split defines a slice of work but does not describe how to access it
- The *RecordReader* class actually loads data from its source and converts it into (K, V) pairs suitable for reading by Mappers
- The RecordReader is invoked repeatedly on the input until the entire split is consumed
- Each invocation of the RecordReader leads to another call of the map function defined by the programmer

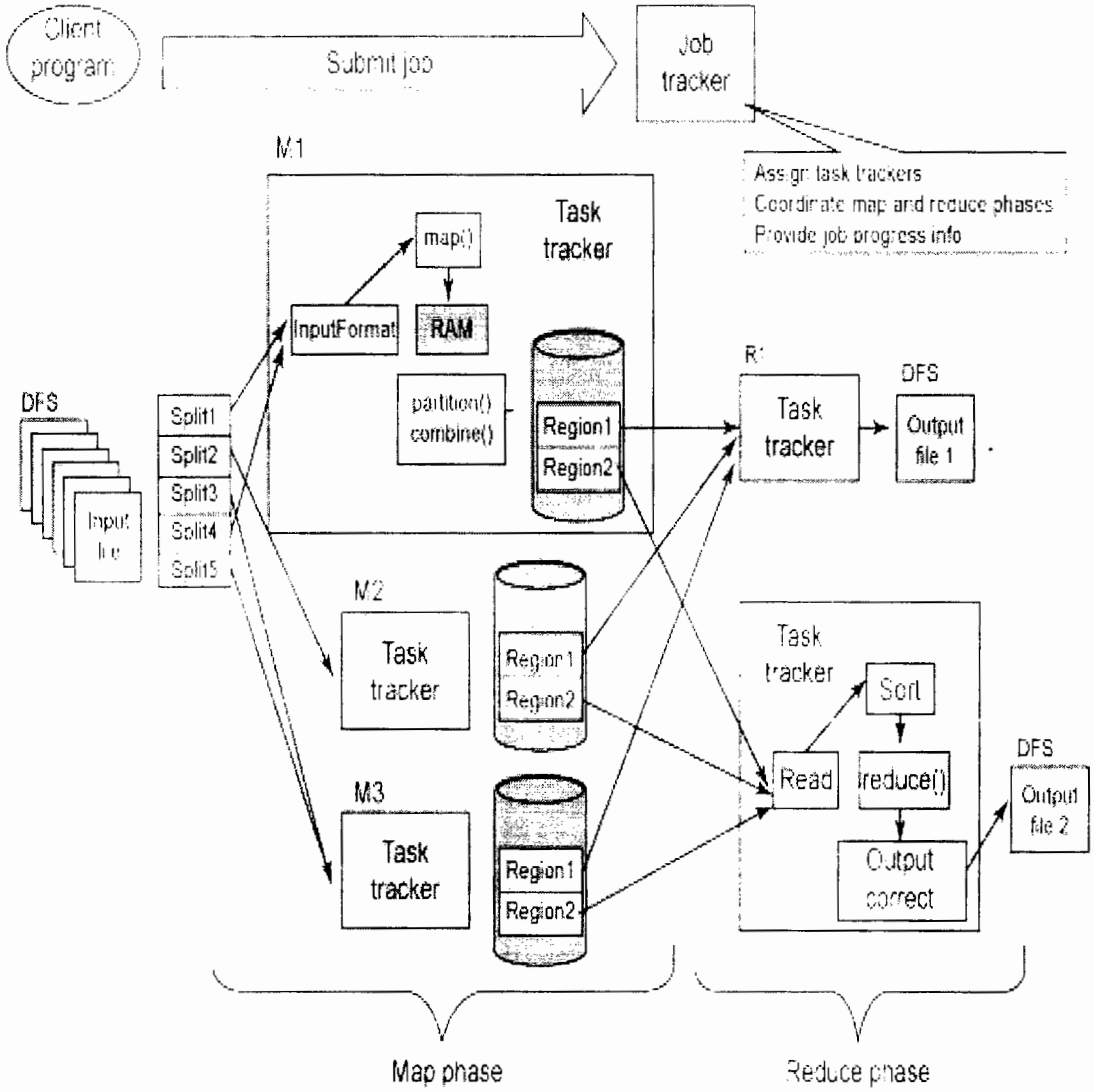


2.6.Map Reduce Life Cycle

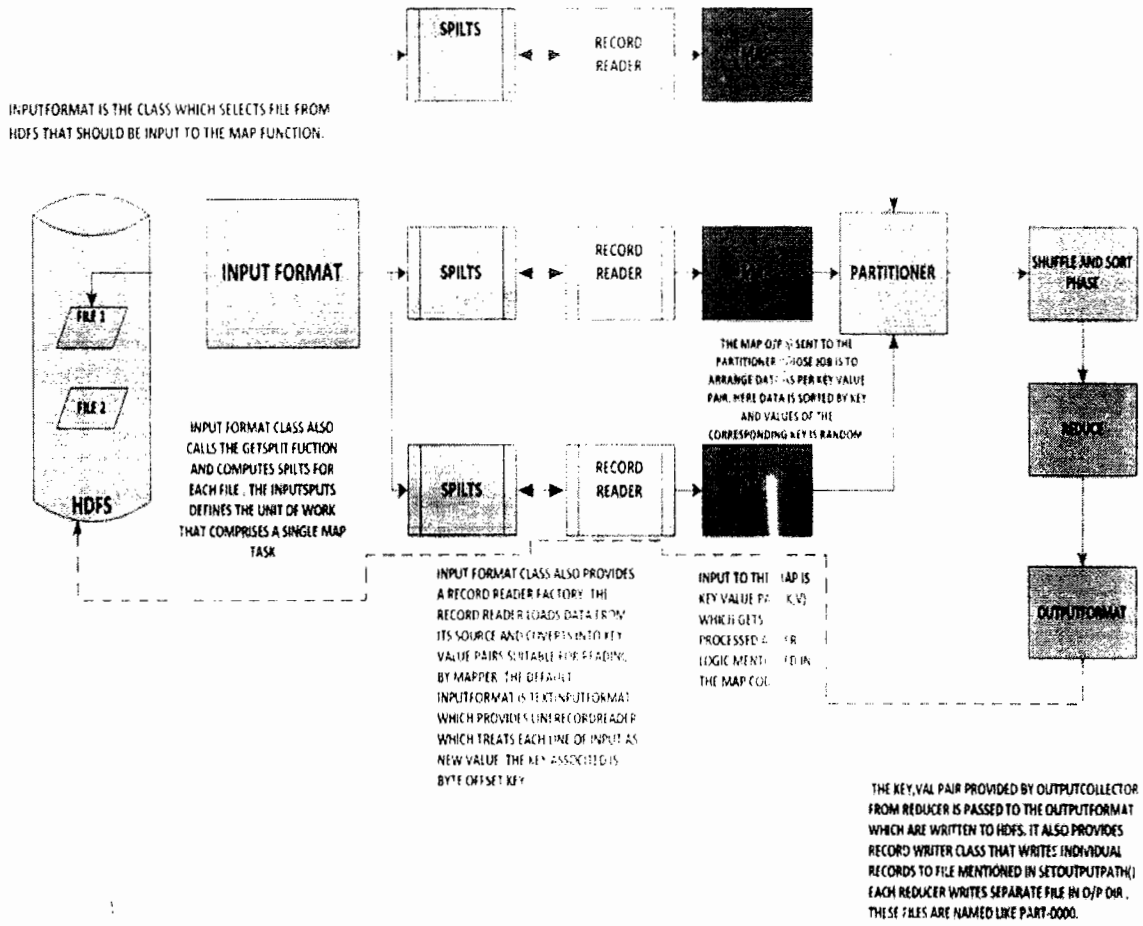
A Map-Reduce job usually splits the input data-set into independent chunks, which are processed by the map tasks in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the reduce tasks. Typically both the input and the output of the job are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks.

Map-Reduce applications specify the input/output locations and supply map and reduce functions via implementations of appropriate Hadoop interfaces, such as Mapper and Reducer. These, and other job parameters, comprise the job configuration. The Hadoop job client then submits the job (jar/executable, etc.) and configuration to the JobTracker, which then assumes the responsibility of distributing the software/configuration to the slaves, scheduling tasks and monitoring them, providing status and diagnostic information to the job-client.

The Map/Reduce framework operates exclusively on <key, value> pairs — that is, the framework views the input to the job as a set of <key, value> pairs and produces a set of <key, value> pairs as the output of the job, conceivably of different types.



Here is the typical data-flow in a Map-Reduce application:



Now before processing it needs to know on which data to process, this is achieved with the **InputFormat** class. InputFormat is the class which selects file from HDFS that should be input to the map function. An InputFormat is also responsible for creating the **input splits** and dividing them into records. The data is divided into number of splits(typically 64/128mb) in HDFS. An input split is a chunk of the input that is processed by a single map.

InputFormat class calls the getSplits() function and computes splits for each file and then sends them to the jobtracker, which uses their storage locations to schedule map tasks to process them on the tasktrackers. On a tasktracker, the map task passes the split to the createRecordReader() method on InputFormat to obtain a **RecordReader** for that split. The RecordReader loads data from its source and which treats each value of input a new value and the associated key is converts into key-value pairs suitable for reading by mapper. The default InputFormat is TextInputFormat byte offset.

A RecordReader is little more than an iterator over records, and the map task uses one to generate record key-value pairs, which it passes to the map function. We can see this by looking at the Mapper's run() method:

```
public void run(Context context) throws IOException, InterruptedException {
setup(context);
while (context.nextKeyValue()) {
map(context.getCurrentKey(), context.getCurrentValue(), context);
}
cleanup(context);
}
```

After running **setup(),** the nextKeyValue() is called repeatedly on the Context. (which delegates to the identically-named method on the the RecordReader) to populate the key and value objects for the mapper. The key and value are retrieved from the Record Reader by way of the Context, and passed to the map() method for it to do its work. Input to the map function which is the key-value pair (K, V) gets processed as per the logic mentioned in the map code.

When the reader gets to the end of the stream, the nextKeyValue() method returns false, and the map task runs its **cleanup()** method.

The output of the mapper is sent to the partitioner. **Partitioner** controls the partitioning of the keys of the intermediate map-outputs. The key (or a subset of the key) is used to derive the partition, typically by a hash function. The total number of partitions is the same as the number of reduce tasks for the job. Hence this controls which of the m reduce tasks the intermediate key (and hence the record) is sent for reduction. The use of partitioners is optional.**All values corresponding to the same key will go the same reducer.**

Output of the mapper is first written on the local disk for **sorting and shuffling** process. It is also in the form of key-value pair. And then it is merged and finally given to reducer.

MapReduce makes the guarantee that the input to every reducer is sorted by key. The process by which the system performs the **sort**—and transfers the map outputs to the reducers as inputs—is known as the **shuffle**. It is said that the shuffle is the heart of MapReduce and is where the "magic" happens.

Output of all mapper goes to all reducer. During the reduce phase, the reduce function is invoked for each key in the sorted output. The output of this phase is written to the output filesystem, typically HDFS. The key-value pair provided as output by reducer is passed to

the **OutputFormat** which are then written to HDFS. It also provides RecordWriter class that writes individual records to the file mentioned in setOutputPath(). Each reducer writes separate file in the output directory and these files are named as **part-00000.**

One can also use the **combiner** for the optimization purpose. Combiner is conceptually placed after the map block and it reduces the output particular to that block of map. It is generally termed as mini-reducer. It also reduces the network lag.

2.7.The Parts of a Hadoop MapReduce Job

The user configures and submits a MapReduce job (or just *job* for short) to the framework, which will decompose the job into a set of map tasks, shuffles, a sort, and a set of reduce tasks.

The framework will then manage the distribution and execution of the tasks, collect the output, and report the status to the user.

Parts of a MapReduce Job	**Part Handled By**
Configuration of the job	User
Input splitting and distribution	Hadoop framework
Start of the individual map tasks with their input split	Hadoop framework
Map function, called once for each input key/value pair	User
Shuffle, which partitions and sorts the per-map output	Hadoop framework
Sort, which merge sorts the shuffle output for each partition of all map outputs	Hadoop framework

2.8.MapReduce Programming Model

A typical MapReduce job is performed according to the following steps:

1. Split the data into independent chunks based on key-value pair. This is done by Map task in a parallel manner.
2. The output of the Map job is sorted based on the key values
3. The sorted output is the input to the Reduce job. And then it produces the final output to the processing and returns the result to the client.

2.9.MapReduce Framework

The Apache Hadoop MapReduce framework is written in Java. The framework consists of master-slave configuration. The master is known as JobTracker and the slaves are known as TaskTrackers. The master controls the task processed on the slaves (which are nothing but the nodes in a cluster). The computation is done on the slaves. So the compute and storages nodes are the same in a clustered environment. The concept is ' move the computation to the nodes where the data is stored', and it makes the processing faster.

2.10.MapReduce Processing

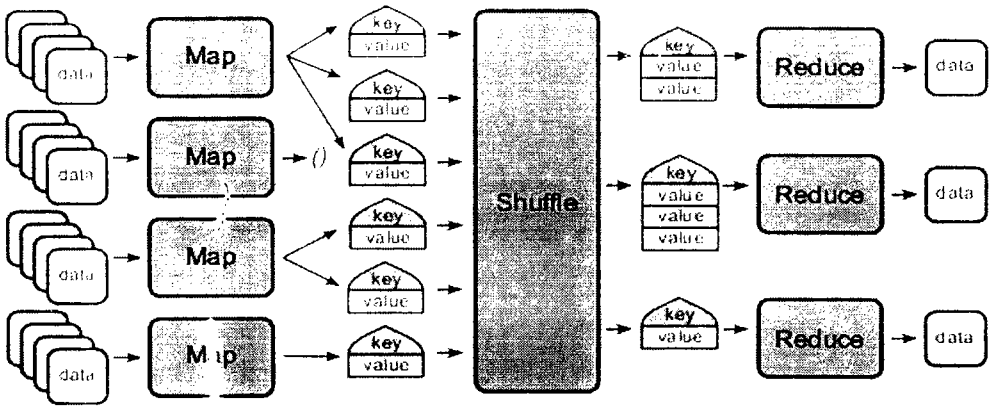
The MapReduce framework model is very lightweight. So the cost of hardware is low compared with other frameworks. But at the same time, we should understand that the model works efficiently only in a distributed environment as the processing is done on nodes where the data resides. The other features like scalability, reliability and fault tolerance also works well on distributed environment.

2.11.MapReduce Implementation

Now it is time to discuss the implementation of the MapReduce model using the Java programming platform. The following are the different components of the entire end-to-end implementation.

- The *client program*that is the driver class and initiates the process
- The *Map function*that performs the split using the key-value pair.
- The *Reduce function*that aggregate the processed data and send the output back to the client.

A graphical view of the MapReduce programming model



Map: mostly read batches of data in parallel and do some initial filtering

Shuffle is a global comm. like group-by or MPIAlltoall

Reduce is often where the computation happens

David Gleich · Purdue

Hadoop DataTypes

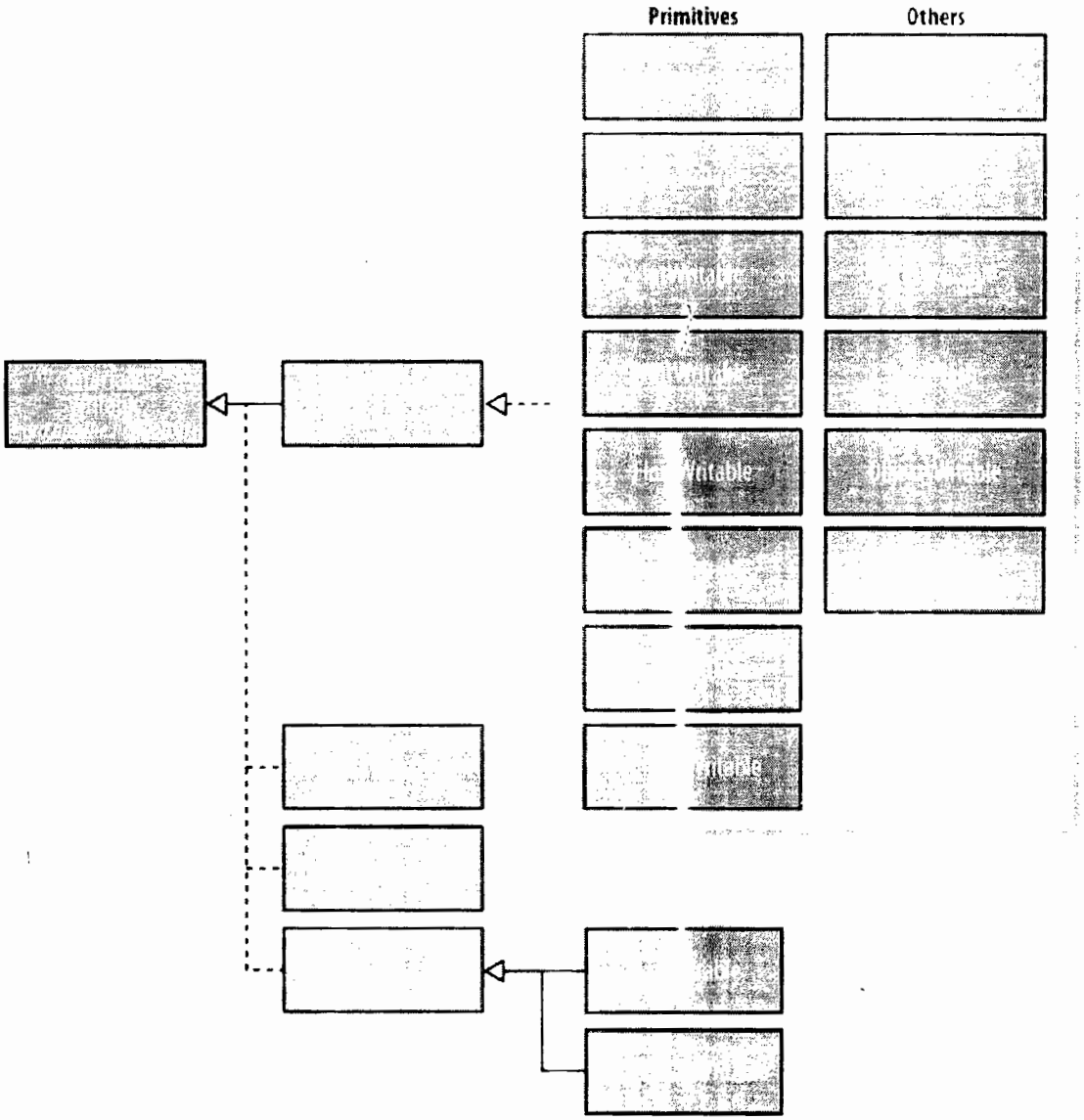
MR has a defined way of keys and values types → for it to move across cluster

Values → Writable

Keys → WritableComparable<T>

WritableComparable = Writable+Comparable<T>

Hadoop type	Wrapper for Java type
BooleanWritable	Boolean
ByteWritable	Byte
DoubleWritable	Double
IntWritable	Integer
LongWritable	Long
Text	String
NullWritable	Placeholder when key/value not needed


Primitives
Others

2.12.Basic MapReduce Program

A MapReduce program consists of the following 3 parts :

1. Driver
2. Mapper
3. Reducer

2.12.1.Driver

The Driver code runs on the client machine and is responsible for building the configuration of the job and submitting it to the Hadoop Cluster. The Driver code will contain the main() method that accepts arguments from the command line.

Some of the common libraries that are included for the Driver class :

```
1 import org.apache.hadoop.fs.Path;
2 import org.apache.hadoop.io.*;
3 import org.apache.hadoop.mapred.*;
```

Driver code is the entry point for any MapReduce program and hence it should be part of Main() method only.

Below are the five subsections of any driver class

1. Configuration details w.r.t job, jar etc.
2. Mapper, Combiner and Reducer class name details.
3. Final output key,value datatype details.
4. HDFS Input path,Outpath details.
5. System exit process.

Sample Driver code:

```
public static void main(String[] args) throws Exception {
    System.out.println("*****In the Driver Code*****");
    //Configuration Details w.r.to JOB,JAR etc(Section 1 Details)
  Configuration conf = new Configuration();
  Job job = new Job(conf, "WORD COUNT JOB");
```

```
    job.setJarByClass(WordCountNew.class);
    // Mapper , Combiner & Reducer Class Name details (Section 2 Details)
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    // Final Output KEY , VALUE Datatype details(Section 3 Details)
    job.setOutputKeyClass(Text.class); //hadoop
    job.setOutputValueClass(IntWritable.class); //4
    // HDFS Input Path , HDFS Output Path Details    (Section 4 Details)
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    //System Exit process (Section 5 Details)
    System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

2.12.2.Mapper

The Mapper code reads the input files as <Key,Value> pairs and emits key value pairs. The Mapper class extends MapReduceBase and implements the Mapper interface. The Mapper interface expects four generics, which define the types of the input and output key/value pairs. The first two parameters define the input key and value types, the second two define the output key and value types.

Some of the common libraries that are included for the Mapper class

```
import java.io.IOException;
import java.util.*;
import org.apache.hadoop.io.*
import org.apache.hadoop.mapred.*;
```

The map() function accepts the key, value, OutputCollector and an Reporter object. The OutputCollector is resposible for writing the intermediate data generated by the Mapper.

Sample Mapper code:

```
public class WordCountNew
{
 public static class TokenizerMapper
    extends Mapper<Object, Text, Text, IntWritable>
     {
     // 1111L , hadoop is one of the bigdata tool---> hadoop , 1
  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();
  // To Write the Mapper Business Logic  , we need to override below
  // map method
  public void map(Object key, Text value, Context context
              ) throws IOException, InterruptedException
         {
   StringTokenizer itr = new StringTokenizer(value.toString());
   while (itr.hasMoreTokens())
            {
    word.set(itr.nextToken());
    System.out.println("Word Is:" + word);
    context.write(word, one); // hadoop , 1  is , 1----tool,1
     }
  }
 }// End Of Map Method
```

2.12.3.Reducer

The Reducer code reads the outputs generated by the different mappers as <Key,Value> pairs and emits key value pairs. The Reducer class extends MapReduceBase and implements the Reducer interface. The Reducer interface expects four generics, which define the types of the input and output key/value pairs. The first two parameters define the intermediate key and value types, the second two define the final output key and value types. The keys are WritableComparables. the values are Writables.

Some of the common libraries that are included for the Reducer class :

```
import java.io.IOException;
import java.util.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
```

The reduce() function accepts the key, an iterator , OutputCollector and an Reporter object.

The OutputCollector is resposible for writing the final output result.

Sample Reducer code:

```
public static class IntSumReducer
    extends Reducer<Text,IntWritable,Text,IntWritable> {
      // hadoop , 1,1,1,1---> hadoop    4
  private IntWritable result = new IntWritable();
  // To Write the Reducer Business Logic  , we need to override below
  // reduce method
  public void reduce(Text key, Iterable<IntWritable> values,
                Context context
                 throws IOException, InterruptedException {
    // hadoop , 1,1,1,1
   int sum = 0;  //4
       for (IntWritable val : values) { // hadoop,,,,
    sum += val.get(); // sum = sum+val.get();
                 //  3 = 3 + 1;

   }
   System.out.println("Sum is--->" + sum);
    result.set(sum);
    context.write(key, result);
  }
}
```

2.13.Sample MapReduce Program for dynamic wordcount:

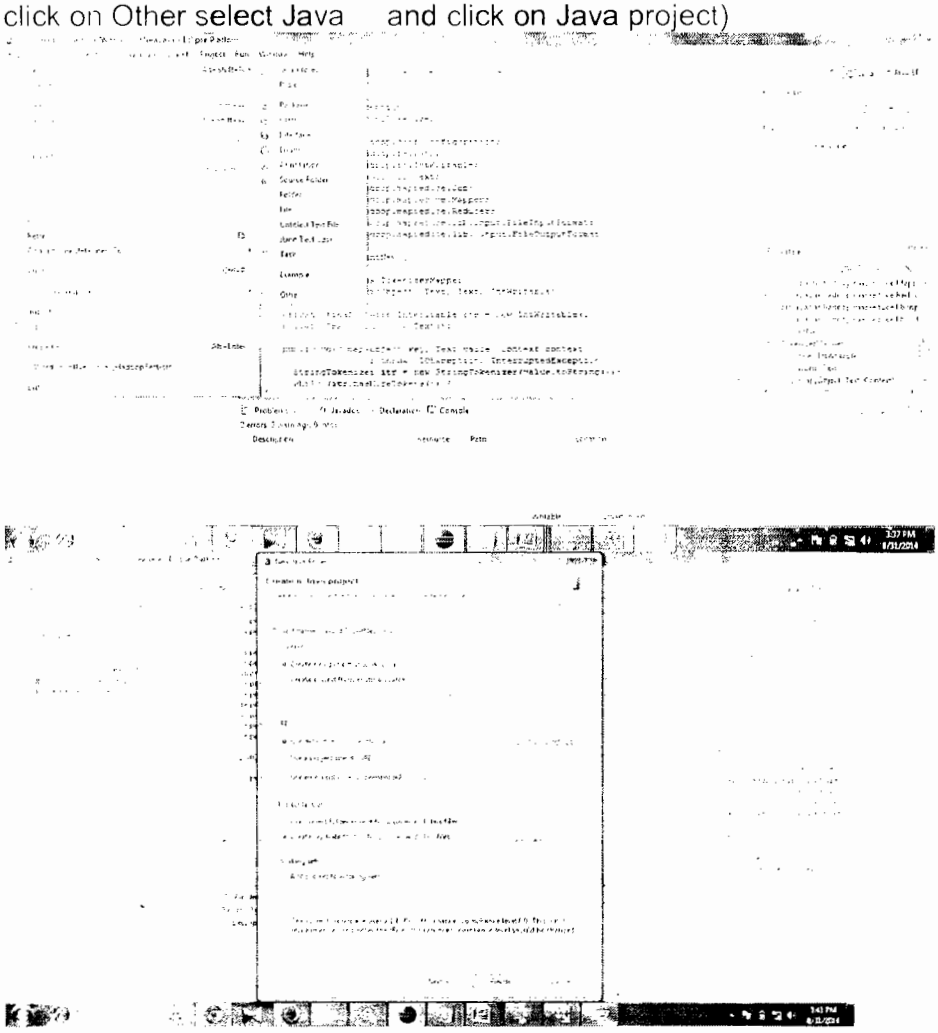
```
package com.dynamic.wordcount;
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class DynamicWordCount {

  public static class Map extends Mapper<LongWritable, Text, Text, IntWritable>
  {
      private final static IntWritable one = new IntWritable(1);
      private Text word = new Text();
```

```
        public void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException
        {
                String line = value.toString();
                String[] words=line.split(" ");
                Configuration conf=context.getConfiguration();
                String newWord=conf.get("RunTimeArg");
                for(String wrd:words)
                {
                        if(wrd.equals(newWord))
                        {
                                word.set(wrd);
                                context.write(word, one);
                        }
                }
        }
      }

  public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable>
  {
        public void reduce(Text key, Iterable<IntWritable> values, Context context) throws
IOException, InterruptedException
        {
                int sum = 0;
                for (IntWritable val : values)
                {
                        sum += val.get();
                }
                context.write(key, new IntWritable(sum));
        }
  }

  public static void main(String[] args) throws Exception
  {
        Configuration conf = new Configuration();
        conf.set("RunTimeArg",args[2]);
        Job job = new Job(conf, "DynamicWordCount");
         job.setJarByClass(DynamicWordCount.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        job.setMapperClass(Map.class);
        job.setReducerClass(Reduce.class)
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        job.waitForCompletion(true);
  }
}
```

2.14.Creation of jar file and Execution Process in Clustered Environment

2.14.1.Process of Creation of Jar file

Step1: Open Eclipse, click on file, click on New then click on Java Project(if not available click on Other select Java and click on Java project)

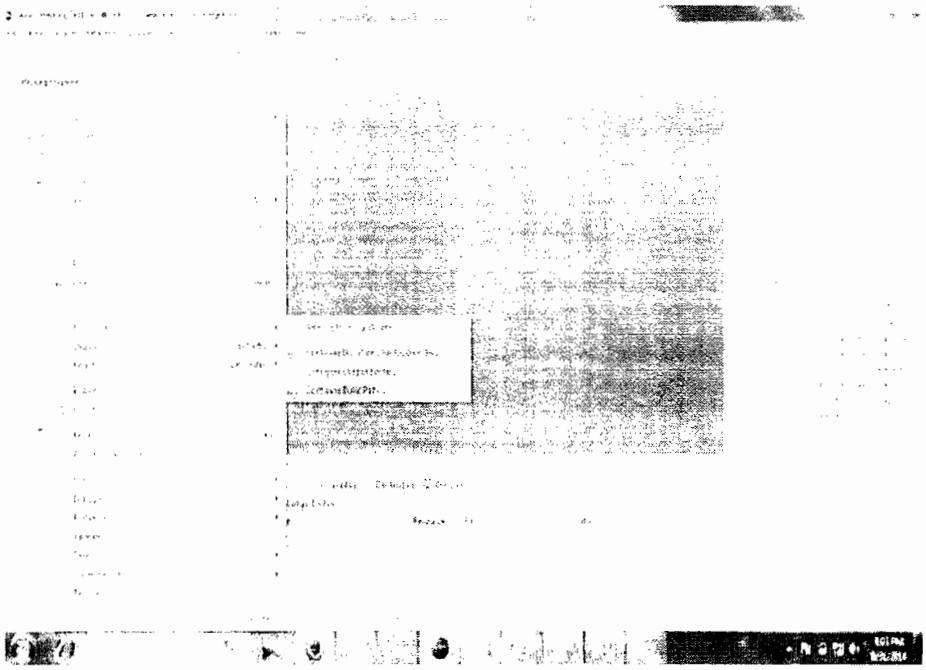


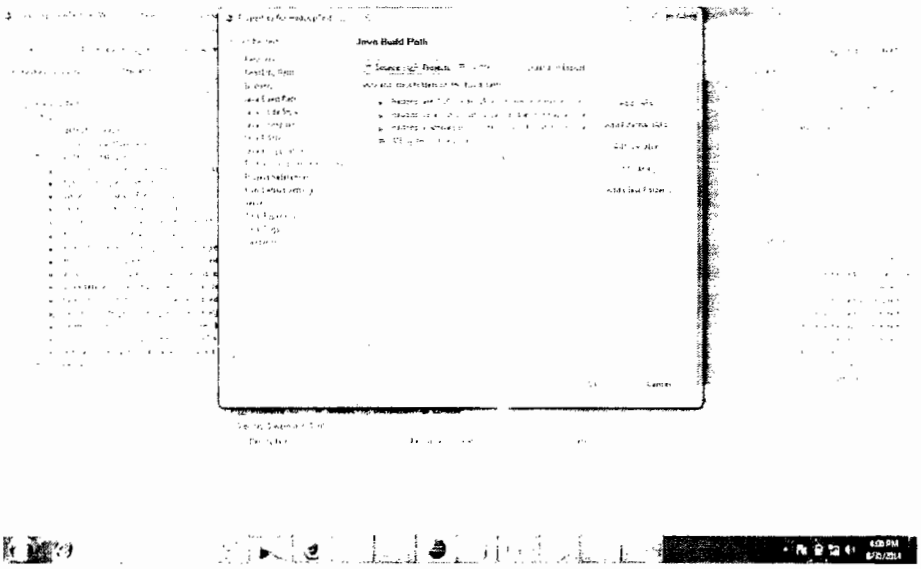
Step2:

Copy the provided code of Dynamicwordcount usecase

Step3:

Right Click on "src" which is in Package Explorer tab -> Click On "Build Path"->Configure Build Path->Click on Libraries tab->Add External JARs..->Press "Ok" Button.





Step4: Step-3: Right Click on "src" ->New->Class->give the name of Class File same as in code->Press "Finish" Button->Paste Code/Write the Code

Step5: Again Right Click on "src"->Export->Select JAR File under the Java->give the path as well as Name of JAR File->Press Finish Button

2.14.2.Execution Process

Step1: copy the jar file and input text file(provided) into download folder(provided server shared path).

Step2: copy the text file into local folder and copy the jar file into local folder.

Step3: Create the new directory in HDFS path (hadoop fs –mkdir MrInput)

Step4: copy the text file from LFS to HDFS by using –put or –copyFromLocal.

Step5:

root@ubuntu:**/home/gopal# hadoop jar BATCH38-DYNAMICWORDCOUNT.jar WordCountNew /root/user/MrInput/Input-Big.txt /root/user/MROutput**

2.15.IDENTITY MAPPER and IDENTITY REDUCER IN MAPREDUCE

- org.apache.hadoop.mapred.lib.IdentityMapper Implements the identity function, mapping inputs directly to outputs. If MapReduce programmer do not set the Mapper Class using JobConf.setMapperClass then IdentityMapper.class is used as a default value.
- org.apache.hadoop.mapred.lib.IdentityReducer Performs no reduction, writing all input values directly to the output. If MapReduce programmer do not set the Reducer Class using JobConf.setReducerClass then IdentityReducer.class is used as a default value.

identity mappers and reducers doesn\'t have a body, it only generates key-value pairs.

```
jobConf.setMapperClass(IdentityMapper.class);
jobConf.setReducerClass(IdentityReducer.class);
```

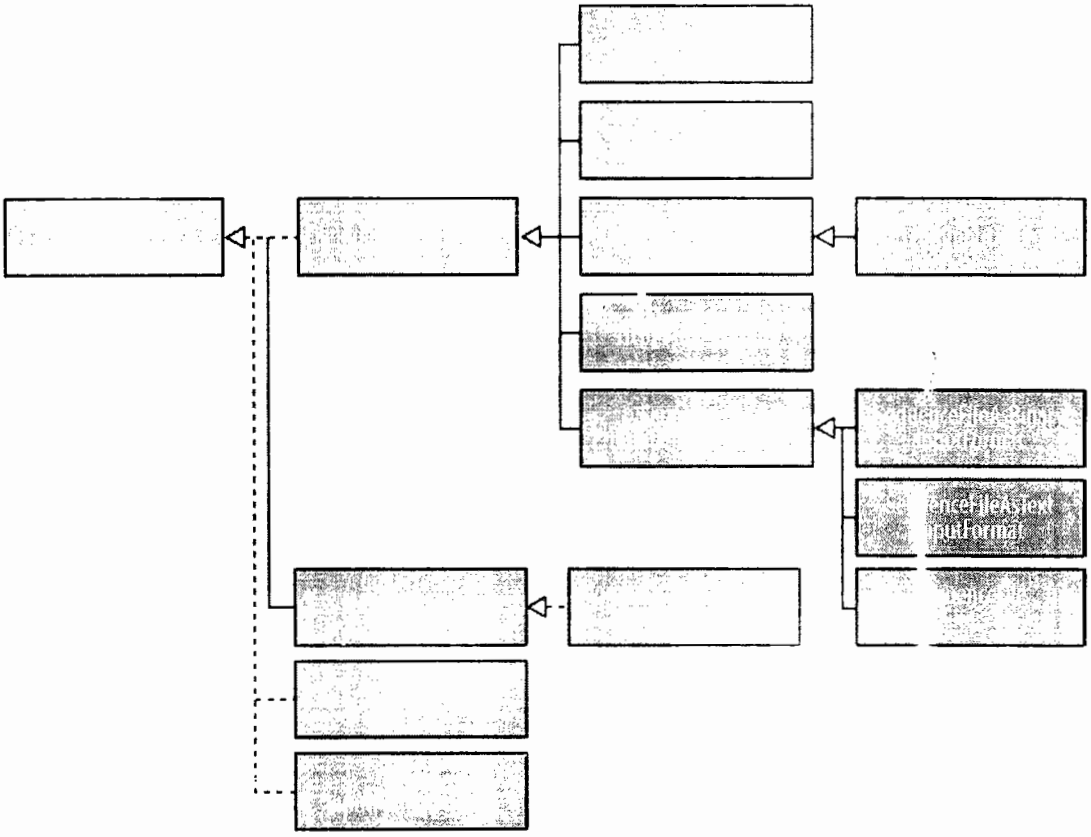
2.16.Input Format's in MapReduce

Hadoop can process many different types of data formats, from flat text files to databases. In this section, we explore the different formats available.

The InputFormat of a Hadoop MapReduce computation generates the key-value pair inputs for the mappers by parsing the input data. InputFormat also performs the splitting of the input data into logical partitions, essentially determining the number of Map tasks of a MapReduce computation and indirectly deciding the execution location of the Map tasks.

2.16.1.File Input Format

It is the base implementation class for all kinds of Input Formats that use filesas their data source.It provides two things: a place to define which files are included as the input to a job, and an implementation for generating splits for the input files.



2.16.2.Text Input Format

This is the default input format in MapReduce.Each record is a line of input. The key, a LongWritable, is the byte offset within the file of the beginning of the line. The value is the contents of the line.

job.setInputFormatclass(TextInputFormat.class)

The Test Input File Format generally,

Key ------> byte offset value

value-----> Line of text

Note: Framework will generates the byteoffset value

Ex: key	value
10101011L	hadoop is a big data tool
10101012L	HDFS is for storing huge amount of data
10101013L	MapReduce is for processing huge amount of data.

2.16.3.KeyValueTextInputFormat

An InputFormat for plain text files. Each line is divided into key and value parts by a separator byte. If no such a byte exists, the key will be the entire line and value will be empty. The separator byte can be specified in config file

configuration conf = new configuartion();

conf.set("mapreduce.input.keyvaluelinerecordreader.key.value.separator","\t")

. By default KeyValueTextInputFormat is tab space (\t) delimited.As we have specific key in this file format , no byte offset value gets generated.

we have to define the class in the code as,

job.setInputFormatClass(KeyValueTextInputFormat.class)

EX: key		value
hadoop	\t	bigdata
hdfs	\t	storage
MapReduce	\t	processing

2.16.4.N-Line Input Format

With TextInputFormat and KeyValueTextInputFormat, each mapper receives a variable number of lines of input. The number depends on the size of the split and the length of the lines.

Whenever we rae providing the data multiple splits into mapper functions, if we have variable number of records in each and every split then we will not have proper idea on when exactly a particular split wiil be completed.

with the same reason If you want your mappers to receive a fixed number of lines of input, then NLineInputFormat is the InputFormat to use.So inorder to control the processing time and to acheive performance gain, we use N-LineInputFormat.

Like TextInputFormat, the keys are the byte offsets within the file and the values are the lines themselves.*N* refers to the number of lines of input that each mapper receives. With *N* set to one (the default), each mapper receives exactly one line of input. we can also set as conf.setInt(NLineInputFormat.LINES_PER_MAP, 1000); // sets N value to 1000

job.setInputFormatClass(NLineInputFormat.class)

Note:N value shpuld be defined in the driver code in such a way not degrading ideal performance by selecting too less or too high 'N' value.

2.16.5.DB InputFormat:

DBInputFormat is an input format for reading data from a relational database. using JDBC.To import data from RDBMS to HDFS or to load data from HDFS **to** RDBMS, the alternative approach is SQOOP.

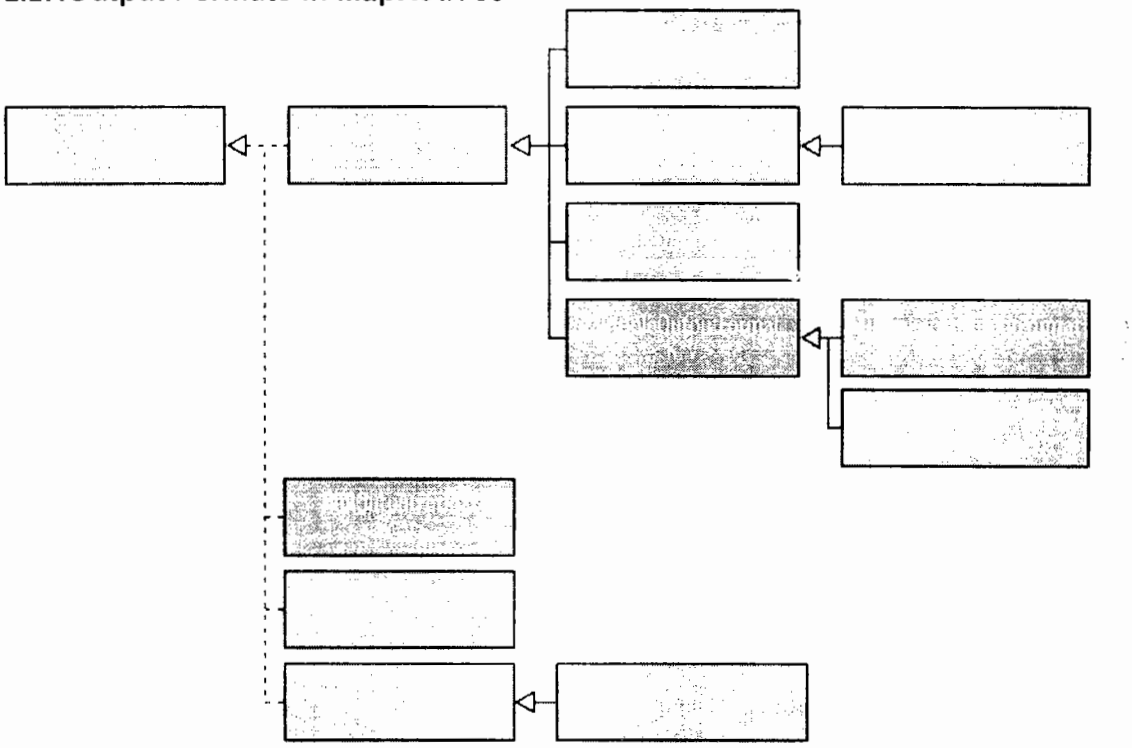
2.16.6.Binary Input Format

Hadoop MapReduce is not just restricted to processing textual data—it has support for binary formats, too.

SequenceFileInputFormat: An InputFormat for SequenceFiles. We will give text input in binary form in this format.

MultiFileInputFormat: This is an abstract class that provides a way for a single task to receive multiple input files as the task's input split. This is commonly done for performance tuning.

2.17.Output Formats in MapReduce



2.17.1.Text Output Format

The default output format class is TextOutputFormat, writes records as lines of text.

job.setOutputFormatclass(TextOutputFormat.class);

set output path , TextOutputformat.setoutputpath(Job , newpath(mypath));

2.17.2.KeyValueTextOutputFormat

It is the TextOutputFormat.

2.17.3.Database Output format

The output formats for writing to relational databases is **DB output Format.**

DBOutputFormat, which is useful for dumping job outputs (of modest size) into a database.

2.17.4.SequenceFileOutputFormat

As the name indicates. SequenceFileOutputFormat writes sequence files for its output.

2.17.5.SequenceFileAsBinaryOutputFormat

SequenceFileAsBinaryOutputFormat is the counterpart to SequenceFileAsBinaryInput Format, and it writes keys and values in raw binary format into a SequenceFile container.

2.17.6..MultipleTextOutputFormat

This format allows you to write output records to different files based on the key and value. The test case

org.apache.hadoop.mapred.TestMultipleTextOutputFormat

provides a sample implementation.

2.18.MapReduce API (Application Programming Interface)

Two types of API,
1.NEW API
2.DEPRICATRD API
On and above Hadoop Version ---->0.20.X----------> New MapReduce API
Befor HadoopVersion ------->0.20.X----------> Depricated API

2.19.Difference between Hadoop OLD API and NEW API

Below are Difference between Hadoop OLD API (0.20) and New API (1.X or 2.X)

Diffrence	New API	OLD API
Mapper & Reducer	New API useing Mapper and Reducer as***Class*** So can add a method (with a default implementation) to an abstract class without breaking old implementations of the class	IN OLD API used Mapper & Reduceer as***Interface*** (still exist in New API as well)
Package	new API is in the***org.apache.hadoop.mapreduce*** package	old API can still be found in***org.apache.hadoop.mapred.***
User Code to commnicate with MapReduce Syaterm	use "***context***" object to communicate with mapReduce system	***JobConf***, the ***OutputCollector***, and the***Reporter*** object use for communicate with Map reduce System
Control Mapper and Reducer execution	new API allows both mappers and reducers to control the execution flow by ***overriding the run()*** method.	Controlling mappers by writing a***MapRunnable***, but no equivalent exists for reducers.

JOB control	Job control is done through the ***JOB*** classin New API	Job Control was done through ***JobClient*** (not exists in the new API)
Job Configuration	Job Configuration done through***Configuration*** class via some of the helper methods on Job.	**jobconf** objet was use for Job configuration.which is extension of Configuration class. java.lang.Object extended by org.apache.hadoop.conf.Configuration extended by org.apache.hadoop.mapred.JobConf
OutPut file Name	In the new API map outputs are named***part-m-nnnnn***, and reduce outputs are named ***part-r-nnnnn*** (where nnnnn is an integer designating the part number, starting from zero).	in the old API both map and reduce outputs are named ***part-nnnnn***
reduce() method passes values	In the new API, the reduce() method passes values as a ***java.lang.Iterable***	In the Old API, the reduce() method passes values as a ***java.lang.Iterator***

Mapper code ---------> In NEW API

```
public static class ABC mapper extends mapper<IK,IV,OK,OV> {
public void map(IK,IV,Context con) {
con.write(OK ,OV);
}
}
```

Mapper code ---------> In Depricated API

```
public static class ABC mapper extends MapReduceBase Implements mapper
<IK,IV,OK,OV>
{
public void map(IK,IV,outputcollector <ok,ov> output,Reporter rep) {
output.collect (OK ,OV);
}
}
```

2.20.COMBINER

To acheive better network optimisation , we use combiner class.The purpose of a combiner is to reduce the volume of data that must be passed to the reducer from a map task by summarizing output records that share the same key.

To optimize the n/w and bandwidth limitations combiner concept will be used in MapReduce.Inorder to reduce n/w bandwidth, we can reduce/mitigate the data at the mapper output itself.we will apply combiner logic at the mapper output phase to acheive this.Combiner will act as local reducer (or minireducer ----> whatever the data consumed by reducer phase , the same copy will reside in the combiner.The other two names for

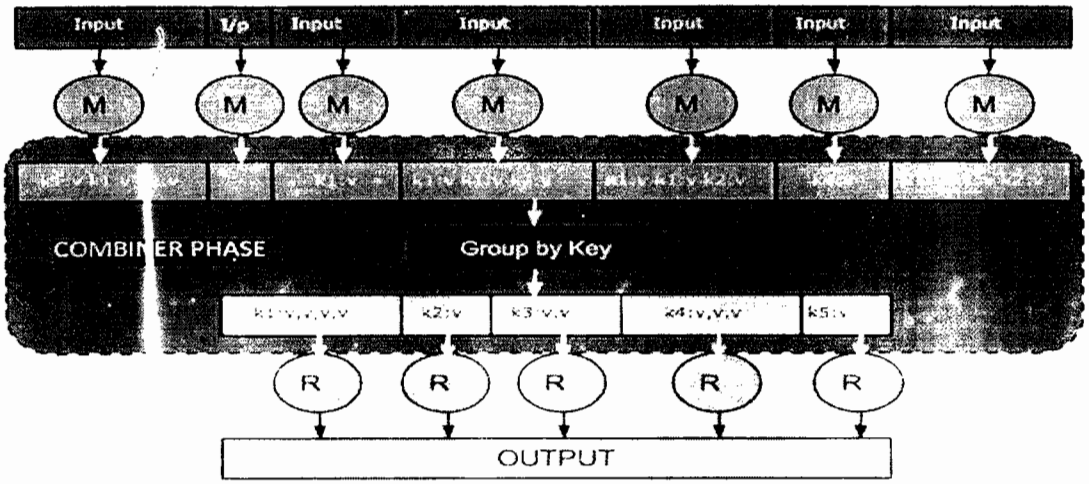
combiner are LocalReducer (or) MiniReducer.

public static combiner class < > extends Reducer < >

Our Combiber will always extends Reducer class only.

Note: Combiner is not mandate in each and every MapReduce Usecase. Its usage purely depends upon Business Logic.

Note:Combiner function does not replace the Reduce function.



When the map task has completed and the partitions are sorted, the combiner may run over the partitions and aggregate values, reducing the total number of key/value pairs that must go over the network to the reduce task.

For example, suppose the map partition dataset originally contained the following:

Key Value

A 1

A 1

The 1

The 1

The 1

Xylophone 1

After the combiner has completed, the map partition dataset would contain these keys and values:

Key Value

A 2

The 3

Xylophone 1

2.21.Better optimisation techniques in MapReduce

These three are the better optimisation techniques in MR,

1.By increasing number of reduce tasks.

2.By using Compression Techniques.

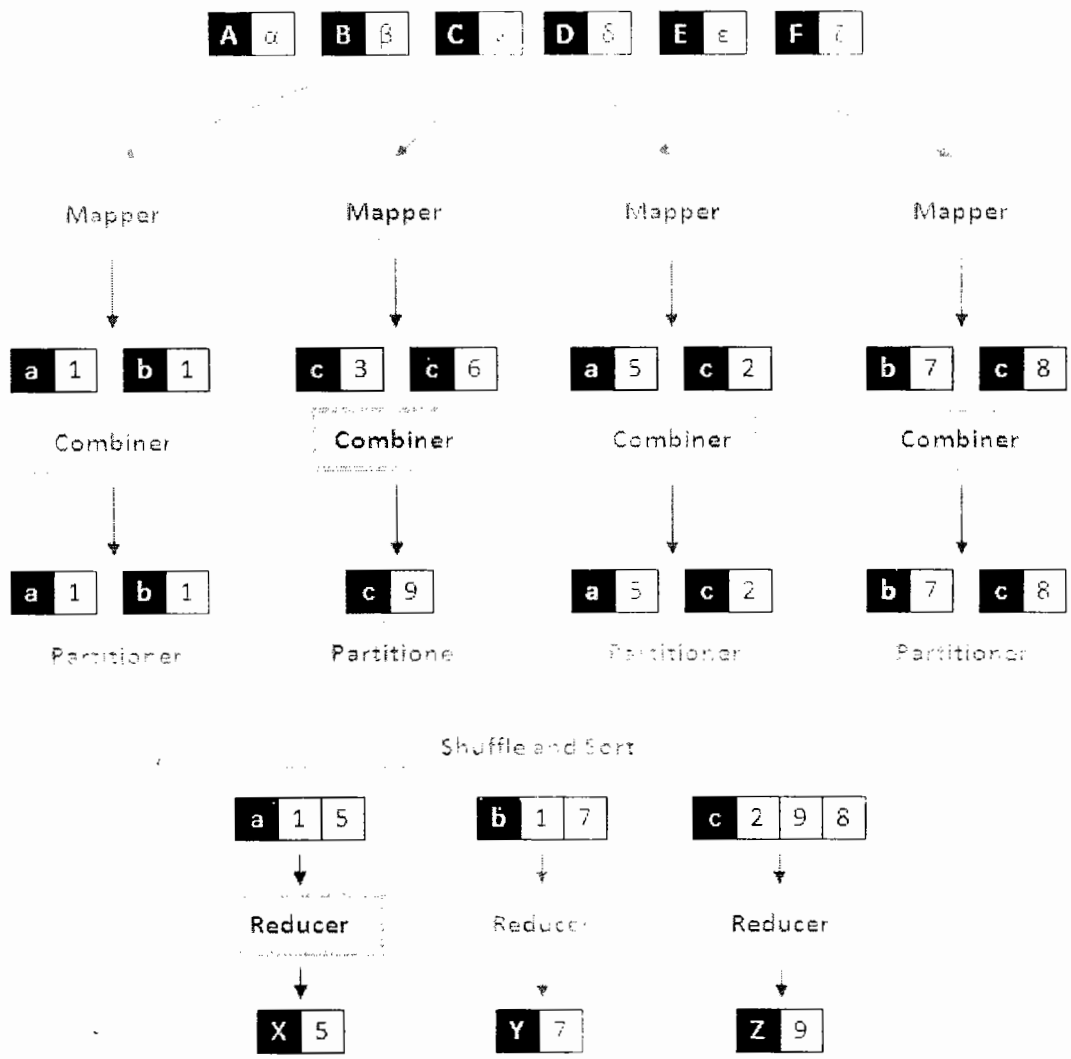
3.By using Combiner Class.

2.22.PARTITIONER

A Partitioner partitions the data using a user-defined condition, which works like a hash function.

The mechanism of dividing the similar mapper outputs into the corresponding reducres based on some business logic , is known as **Custom Partitioner.** The total number of partitions is same as the number of Reducer tasks for the job.

The partition phase takes place after the Map phase and before the Reduce phase.The number of partitioners is equal to the number of reducers. That means a partitioner will divide the data according to the number of reducers. Therefore, the data passed from a single partitioner is processed by a single Reducer.



There are four types of partitioners in Hadoop namely,

1.**Binary Partitioner**
2.**Hash Partitioner , which is default.**
3.**TotalOrderPartitioner, which provides a way to partition by range**
4.**KeyFieldBasedPartitioner, which provides a way to partition by parts of the key**

2.23.How Partitioning works

When the framework is performing the shuffle, each key output by the mapper is examined. and the following operation is performed:

int partition = partitioner.getPartition(key, value, partitions);

The value partitions is the number of reduce tasks to perform. The key, if actually output by the

reducer, will end up in the output file part partition, with an appropriate number of leading zeros so that the file names are all the same length.

2.24.Steps for developing Custom Partitioner

1. In the driver code, we have to add the below method,

jobobj.setPartitionerclass(samplePartitioner.class)

2.We have to develop the same partitioner class by extending the partitioner class,

public static class Agepartitioner extends Partitioner <text,text>

3.We have to override the below method to achieve the custom partitioner,

public int getPartition(Text key,Text value,int numReduceTasks)

4.In the driver code we have to set the corresponding number of tasks using below method

job.setNumReduceTasks(..;

(bcoz, the relation b/w no. of partitions and no. of reducers is always one-to-one)

Example :

Let us see one example showing in detail how partitioning works,

Input data: Input_Partitioner.log

Id	Name	Age	Gende	Salary
1201	gopal	45	Male	0,000
1202	manisha	40	emale	50,000
1203	khalil	34	Male	0,000
1204	prasanth	30	Male	30,000
1205	kiran	20	Male	0,000
1206	laxmi	25	Female	35,000
1207	bhavya	20	emale	15,000
1208	reshma	19	emale	15,000
1209	kranthi	22	Male	22,000
1210	Satish	24	Male	5,000
1211	Krishna	25	Male	25,000
1212	Arshad	28	Male	20,000
1213	lavanya	18	Female	8,000

Partition Logic: (Requirement)

From the input given above,

who are under 20 yrs of age ----------> Partition 0

who are between 20 and 50 yrs of age ----------> Partition 1

who are above 50 yrs of age ---------->partition 2

In this way, we have to partition the data as per age (above)

Partitioner Task

The partitioner task accepts the key-value pairs from the map task as its input. Partition implies dividing the data into segments. According to the given conditional criteria of partitions, the input key-value paired data can be divided into three parts based on the age criteria

Input – The whole data in a collection of key-value pairs.

key = Gender field value in the record.

value = Whole record data value of that gender.

Check the age value with the following conditions.

----->Age less than or equal to 20

----->Age Greater than 20 and Less than or equal to 50.

------>Age Greater than 50.

Read the age field value from the input key-value pair.

Output – The whole data of key-value pairs are segmented into three collections of key-value pairs. The Reducer works individually on each collection.

Reduce Tasks

The number of partitioner tasks is equal to the number of reducer tasks. Here we have three partitioner tasks and hence we have three Reducer tasks to be executed.

Input – The Reducer will execute three times with different collection of key-value pairs.

key = gender field value in the record.

value = the whole record data of that gender.

Output – Finally, you will get a set of key-value pair data in three collections of different age groups.

After executing the Map, the Partitioner, and the Reduce tasks, the three collections of key-value pair data are stored in three different files as the output.

SamplePartitioner.java:

```
package com.mapred.partiexam;

import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Partitioner;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
public class SamplePartitioner
{
    public static class PartitionMapper extends      Mapper<Object, Text, Text, Text>
  {
    public void map(Object key, Text value, Context context)throws IOException,
InterruptedException
      {
                String[] tokens = value.toString() split("\t");
                String gender = tokens[2].toString();
                String nameAgeScore = tokens[0]+"\t"+tokens[1]+"\t"+tokens[3];
                context.write(new Text(gender), new Text(nameAgeScore));
            }
    }
```

```
//AgePartitioner is a custom Partitioner to partition the data according to age.
//The age is a part of the value from the input file.
//The data is partitioned based on the range of the age.
//In this example, there are 3 partitions, the first partition contains the information
where the age is less than 20
//The second partition contains data with age ranging between 20 and 50 and the third
partition contains data where the age is >50.
public static class AgePartitioner extends Partitioner<Text, Text>
  {
    public int getPartition(Text key, Text value, int numReduceTasks)
  {
    String [] nameAgeScore = value.toString().split("\t");
        String age = nameAgeScore[1];
        int ageInt = Integer.parseInt(age);
        //if the age is <20, assign partition 0
        if(ageInt <=20){
            return 0;   // part-r-00000
        }
        //else if the age is between 20 and 50, assign partition 1
        if(ageInt >20 && ageInt <=50)
          {
            return 1 % numReduceTasks;   //part -r -00001
        }
        //otherwise assign partition 2
        else
            return 2 % numReduceTasks;  //part-r-00002

    }
}
//The data belonging to the same partition go to the same reducer. In a particular
partition, all the values with the same key are iterated and the person with the maximum
score is found.
//Therefore the output of the reducer will contain the male and female maximum

scorers in each of the 3 age categories.

// the type parameters are the input keys type, the input values type, the
// output keys type, the output values type
static class ParitionReducer extends Reducer<Text, Text, Text, Text>
{
    @Override
    public void reduce(Text key, Iterable<Text> values, Context context)
            throws IOException, InterruptedException
            {
        int maxScore = Integer.MIN_VALUE;
        String name = " ";
        String age = " ";
        String gender = " ";
        int score = 0;
        //iterating through the values corresponding to a particular key
        for(Text val: values)
    {
            String [] valTokens = val.toString().split("\t");
```

```
                    score = Integer.parseInt(valTokens[2]);
        //if the new score is greater than the current maximum score, update the fields as
they will be the output of the reducer after all the values are processed for a particular key

                    if(score > maxScore){
                        name = valTokens[0];
                        age = valTokens[1];
                        gender = key.toString();
                            maxScore = score;
                    }
                }
                context.write(new Text(name), new Text("age-
"+age+"\t"+gender+"\tscore-"+maxScore));
            }
    }
    public static void main(String[] args) throws Exception
      {
        Configuration conf = new Configuration();
      Job job = new Job(conf, "word count");
      job.setJarByClass(SamplePartitioner.class);
      job.setNumReduceTasks(3);
      job.setMapperClass(PartitionMapper.class);
      job.setPartitionerClass(AgePartitioner.class);
      job.setReducerClass(ParitionReducer.  class);
      job.setOutputKeyClass(Text.class);
      job.setOutputValueClass(Text.class);
      FileInputFormat.addInputPath(job, new Path(args[0]));
      FileOutputFormat.setOutputPath(job, new Path(args[1]));
      System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

2.25.COMPRESSION TECHNIQUES IN MAPREDUCE

The Hadoop framework supports several types of compression and several compression formats.Both Compression and decompression are going to be applicable through the codec.

2.25.1Types of compression techniques

2.25.1.1.GZip compression Technique

This is default Compression technique.

Reasonal Compression----------Reasonal Decompression

2.25.1.2.LZO Codec

To achieve reasonal compression and faster decompression.For LZO Codec use

BZip2codec.class

Reasonal Compression----------Faster Decompression

2.25.1.3.Snappy codec

To get neither faster compression nor faster decompression

Neither faster compression-------Nor faster decompression.

High speed transmission of the data is the beauty of SnappyCodec.

2.26.How to enable compression

1.By default compression technique will be disabled(False). We have to enable it to true to use compression technique.

In mapred-site.xml:

<property>

<name>mapred.output.compress</name>

<value>false</value> ----------> // enable it to true

</property>

2. Define the type of compression(GZip or snappy or LZO)

mapred.outpit.compressioncodec ---------->value: GZip or snappy

Above two are the mportant properties as compression is concerned.

Configurations w.r.t Compression Techniques:

In mapred-site.xml:

```
<property>
<name>mapred.output.compression.codec</name>
<value>org.apache.hadoop.io.compress.Defaultcodec</value>
</property>
```

To specify the compression techniques based on phases:

Mapper :

```
<property>
<name>mapred.map.output.compression.codec</name>
<value>org.apache.hadoop.io.compress.Defaultcodec</value>
</property>
```

Reducer :

```
<property>
<name>mapred.Reduce.output.compression.codec</name>
<value>org.apache.hadoop.io.compress.Defaultcodec</value>
</property>
```

CONCLUSION:

Algorithm	Compression Ratio	IO performance increase
Snappy	40%	25%
LZF	40%	21%
LZO	41%	5%
ZLIB	48%	-16%

EXAMPLE :

Input:

Sachin 20 against Australia on 22/02/2011
Dravid 45 against SouthAfrica on 18/03/2011
Ponting 65 against India on 21/03/2011
Sachin 148 against Australia on 05/02/2011
Laxman 281 against Australia on 07/03/2011
Lara 400 against England on 17/03/2011
Hayden 123 against Pakistan on 17/03/2011
Dravid 105 against SriLanka on 15/02/2011
Sachin 13 against Australia on 05/03/2011
Laxman 31 against Pakistan on 06/02/2011
Sachin 25 against England on 25/02/2011
Dravid 11 against WestIndies on 22/02/2011
Inzamam 55 against India on 01/03/2011
Kallis 69 against SriLanka on 15/03/2011
Kallis 75 against Australia on 06/03/2011
Hayden 1 against Bangladesh on 08/03/2011

Code:

Write three phases of logic,
1.Driver code
2.Mapper code
3.Reducer code

1.Driver code:

```
package com.gopal.compressTest;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.compress.CompressionCodec;
import org.apache.hadoop.io.compress.GzipCodec;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;

public class Driver
{	public static void main(String[] args) throws Exception
	{
		if (args.length != 2)
		{
			System.err.println("Usage: FileDemo1: <input path> <output path>");
			System.exit(-1);
		}
		JobConf conf = new JobConf(Driver.class);
		conf.setJobName( "demo3" );
		FileInputFormat.addInputPath( conf, new Path( args[0] ) );
		FileOutputFormat.setOutputPath( conf, new Path( args[1] ) );
		conf.setMapperClass( MyMapper.class );
		conf.setReducerClass( MyReducer.class );
		conf.setBoolean( "mapred.output.compress" , true );
		conf.setClass( "mapred.output.compression.codec" , GzipCodec.class ,
CompressionCodec.class );
		conf.setOutputKeyClass( Text.class );
```

```
            conf.setOutputValueClass( IntWritable.class );
            JobClient.runJob( conf );

        }
}
```

2.Mapper code:

```
package com.gopal.compressTest;
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;
public class MyMapper extends MapReduceBase implements Mapper<LongWritable, Text,
Text, IntWritable>
{
    public void map(LongWritable k1, Text k2,
                OutputCollector<Text, IntWritable> k3, Reporter k4)
                throws IOException
    {
    String line = k2.toString();
    StringTokenizer itr = new StringTokenizer(line);//Iterating for every separate words in
the line.

        String player=itr.nextToken();  //it gives the name of the batsman.
        String playerRun=itr.nextToken(); //it gives the runs scored in a particular match by
that batsman.
        IntWritable run=new IntWritable( Integer.parseInt(playerRun) );//typecasted as
required by the framework.
        k3.collect( new Text(player) , run);//Mapper output<-->Reducer input.
    }
}
```

3.Reducer code:

```
package com.gopal.compressTest;
import java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;

public class MyReducer extends MapReduceBase implements Reducer<Text, IntWritable,
Text, IntWritable>
{
    public void reduce(Text k1, Iterator<IntWritable> k2,
                OutputCollector<Text, IntWritable> k3, Reporter k4)
```

```
                    throws IOException
    {
            int sum = 0;
        while (k2.hasNext()) {
            sum += k2.next().get();
        }

        k3.collect( k1 , new IntWritable(sum) );
    }

}
```

output:
We will get the compressed output.
Note: the compressed output will not be in readable form.
command to get the compressed data in to readable form:
hadoop fs -text /compressout64 /part-00000.gz (this is depricated API, so o/p part-00000)

2.27.Hadoop MapReduce Chaining (Advanced MapReduce)

Small data processing tasks can be accomplished by a single MapReduce job but complex tasks need to be broken down into simpler subtasks, and each should be accomplished by an individual MapReduce job .

There are cases where we need to write more than one MapReduce Job.

Map1--Reduce1--Map2--Reduce2

2.28.Chaining MapReduce jobs in a sequence

Two jobs can be executed manually one after the other, it s more convenient to automate the execution sequence. You can chain MR jobs to run sequentially, with the output of one MapReduce job being the input to the next.

The output of one Mapreduce job will be piped as a input for next MapReduce job and output of next job will be piped as input to next job and so on.

Eventhough we have multiple jobs , there will be only one driver class with in the MapReduce usecase.

Chaining MapReduce jobs involves calling the driver of one MapReduce job after another. The driver at each job will have to create a new JobConf object and set its input path to be the output path of the previous job.

Note: In Multi-Node cluster only, job chaining happens in parallel fashion,where as in SingleNode Cluster, job chaining happens sequentially.

Example:
Input:
Name sub1 sub2 sub3

aa	45	50	55
bb	58	52	69
cc	85	45	86
dd	45	78	89
ee	36	95	98

Logic:

Base on the input ,

Job1--------> to find the avg marks of student

job2--------->based on avg, grade the student as brilliant, good,poor.

Code to run above usecase:

Driver:

```
package com.gopal.demo;
import java.io.IOException;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.fs.s3.FileSystemStore;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.join.Parser.StrToken;
import org.apache.hadoop.metrics.file.FileContext;
public class Driver
{
	public static void main(String[] args) throws IOException
	{
		if (args.length != 2)
		{
			System.err.println("Usage: FileDemo1: <input path> <output path>");
			System.exit(-1);
		}
		//For the First Job...
		JobConf job1 = new JobConf(Driver.class);
		job1.setJobName( "Job 1" );
		//job1.set( "fs.default.name" , "hdfs://10.87.18.243:9000" );
		job1.set( "fs.default.name" , "hdfs://localhost:8020" );
		String temporaryPath = "/user/job1out";
		FileInputFormat.addInputPath( job1, new Path( args[0] ) );
		FileOutputFormat.setOutputPath( job1, new Path( temporaryPath ) );
		job1.setMapperClass( MyMapper class );
		job1.setReducerClass( MyReducer.class );
		job1.setMapOutputKeyClass( Text.class );
		job1.setMapOutputValueClass( IntWritable.class );
		job1.setOutputKeyClass( Text.class );
		job1.setOutputValueClass( Text.class );
		JobClient.runJob( job1 );
		// For the second Job...
		JobConf job2 = new JobConf(Driver.class);
		job2.setJobName( "Job 2" );
		//job2.set( "fs.default.name" , "hdfs://10.87.18.243:9000" );
```

```
            job2.set( "fs.default.name" , "hdfs://localhost:8020" );

            FileInputFormat.addInputPath( job2 , new Path( temporaryPath ) );
            FileOutputFormat.setOutputPath( job2, new Path( args[1] ) );
            job2.setMapperClass( MyMapper1.class );
            job2.setReducerClass( MyReducer1.class );
            job2.setMapOutputKeyClass( Text.class );
            job2.setMapOutputValueClass( Text.class );
            job2.setOutputKeyClass( Text.class );
            job2.setOutputValueClass( Text.class );
            JobClient.runJob( job2 );
        }
}
```

Mapper1:

```
package com.gopal.demo;
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;
public class MyMapper extends MapReduceBase implements Mapper<LongWritable, Text,
Text, IntWritable>
{
   private Text word = new Text();
      @Override
      public void map( LongWritable k1, Text k2, OutputCollector<Text, IntWritable> k3,
                     Reporter k4 ) throws IOException {

            String line = k2.toString();
            StringTokenizer itr = new StringTokenizer(line);
            String student=itr.nextToken();
         String marks=itr.nextToken();
         IntWritable marksNew=new IntWritable( Integer.parseInt(marks) );
         k3.collect( new Text(student) , marksNew);
      }
}
```

Mapper2:

```
package com.gopal.demo;
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
```

```
import org.apache.hadoop.mapred.Reporter;

public class MyMapper1 extends MapReduceBase implements Mapper<LongWritable,
Text, Text, Text>
{
    @Override
    public void map(LongWritable k1, Text k2, OutputCollector<Text, Text> k3,
                    Reporter k4) throws IOException {

        String line = k2.toString();
        StringTokenizer itr = new StringTokenizer(line);
        String student = itr.nextToken();
        String marks = itr.nextToken();
        //IntWritable marksNew=new IntWritable( Integer.parseInt(marks) );
        k3.collect( new Text(student) , new Text(marks) );
    }
}
```

Reducer1:

```
package com.gopal.demo;
import java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;

public class MyReducer extends MapReduceBase implements Reducer<Text, IntWritable,
Text, Text>
{
    @Override
    public void reduce(Text k1, Iterator<IntWritable> k2,
                    OutputCollector<Text, Text> k3, Reporter k4) throws IOException {
            float sum = 0, count=0 ,average=0;
        while (k2.hasNext()) {
          sum += k2.next().get();
          count++;
        }

        average = (sum/count);
        String averageString = ""+average;
        //sum: Total runs scored by that particular batsman in the tournament.
        k3.collect( k1, new Text( averageString ));
        //Final output from the Reducer, that is the total runs scored by each batsman in
the tournament.
                }
    }
```

Reducer2:

```
package com.gopal.demo;
import java.io.IOException;
import java.util.iterator;
```

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;

public class MyReducer1 extends MapReduceBase implements Reducer<Text, Text, Text,
Text>
{
@Override
    public void reduce(Text k1, Iterator<Text> k2,
                    OutputCollector<Text, Text> k3, Reporter k4) throws IOException {
            String comment="Fair";
            String result="";
        while (k2.hasNext()) {
          result = result+ k2.next().toString();
        }

        if( Float.parseFloat(result)>=80 && Float.parseFloat(result)<=100 )
        {
            comment="Very Good";
        }
        else if ( Float.parseFloat(result)>=70 && Float.parseFloat(result)<80 )
        {
                comment="Good";
            }
        else if ( Float.parseFloat(result)>=60 && Float.parseFloat(result)<70 )
        {
            comment="Fair";
            }
        else if ( Float.parseFloat(result)>=50 && Float.parseFloat(result)<60 )
        {
            comment="Satisfactory";
            }
        else
        {
                comment="Fail";
            }

        k3.collect( k1, new Text( comment ) );
    }
}
```

This is how we will achieve Job Chaining in MapReduce.

2.29.JOINS - In MapReduce

Joining two datasets in HADOOP can be implemented using two techniques:
1.Joining during the Map phase.----------> Map side joins
2.Joining during the Reduce phase.---------->Reduce side joins

2.29.1.Joining during the Map phase

This is recommended when the data to join with is small. Joining during the map phase allows to join quickly when one set is small, and also allows to join on separate keys with multiple sets, or with single set repeatedly (for example in warehouse to warehouse transfer of products data, the "from warehouse" and "to warehouse" can be mapped), this is something which cannot be done in a single job if Reduce-phase joins are used.

Map side join is done in the map phase and in the memory.
Map side joins are faster because join operation is done in the memory.
Mapper does Mapper-side join
Replicate a relatively smaller input source to the cluster.Put the replicated data set into a local hash table.
Joins relatively larger input source with each local Hashtable.
The most common problem with map-side joins are out of memory exceptions on slave nodes.
The drawback of mapside joins is , all the data should be in sorted order only.

2.29.2.Joining during the Reduce phase

Reduce side join is a technique for merging data from different sources based on a apecific key. There are no memory restrictions.No need of sorting.This technique is recommended when both datasets are large.

2.30.DISTRIBUTED CACHE IN HADOOP

DistributedCache is a facility provided by the Map-Reduce framework to cache files (text, archives, jars etc.) needed by applications.

If you are writing Map Reduce Applications, where you want some files to be shared across all nodes in Hadoop Cluster. *Hadoop Map Reduce Project provides us this facility with something called as DistributedCache*.

This Distributed Cache is configured with Job Configuration, What it does is, it provides read only data to all machine on the cluster.

DistributedCache can be used to distribute simple, read-only data/text files and/or more complex types such as archives, jars etc. Archives (zip, tar and tgz/tar.gz files) are un-archived at the slave nodes. Jars may be optionally added to the classpath of the tasks, a rudimentary software distribution mechanism. Files have execution permissions. In older version of Hadoop Map/Reduce users could optionally ask for symlinks to be created in the working directory of the child task. In the current version symlinks are always created. If the URL does not have a fragment the name of the file or directory will be used. If multiple files or directories map to the same link name, the last one added, will be used. All others will not even be downloaded.

Step 1 : Put file to HDFS

```
# hdfs dfs -put /tmp/file1 /cachefile1
```

Step 2: Add cachefile in Job Configuration

```
Configuration conf = new Configuration();
Job job = new Job(conf, "wordcount");
```

```
DistributedCache.addCacheFile(new URI("/cachefile1"),job.getConfiguration());
```

Step 3: Access Cached file

```
Path[] cacheFiles = context.getLocalCacheFiles();
FileInputStream fileStream = new FileInputStream(cacheFiles[0].toString());
```

2.31.We will see Hadoop map side join with Distributed Cache below

Note: we need to remember one thing is that we always put the small file in the distributed cache and send large file to map phase.

Table 1 (Always Small Table):

Department table with deptno and deptname(tab seperated)

```
001	hadoop
002	hive
003	pig
004	sqoop
005	oozie
006	mapreduce
```

Table 2(Always LargeTable):

Employee table with Employee info(tab separated)

```
empid	empname	salary	deptid
1	name	2000	001
2	name2	4000	002
3	name3	5000	005
4	name4	7000	003
5	name5	8000	004
```

Output:

```
1	name	2000	001	hadoop
2	name2	4000	002	hive
3	name3	5000	005	oozie
4	name4	7000	003	pig
5	name5	8000	004	sqoop
```

To get the above output we need to join employee table with dept table so that we will get the dept name from the dept table where we join deptID in employee table with dept table.

2.32.DATA LOCALIZATION USING MAPREDUCE

To avoid network related problems, there is a provision in MapReduce called **DataLocalisation**.

The mechanism of storing the mapper outputs in the LFS temporary path (/tmp) is called DataLocalisation.

Every mapper output will be stored in Localfile.Once the mapper phase has been completed by the Taktracker,The mapper output will be sent to local file system temporary path (\tmp).The TaskTracker will the sends data from the LFS \tmp path to the HDFS.The reason is , all the mapper output if we put on the network. there may be some data loss due to network failures.In that case we need to redo the entire mapper process which is a time consuming process and performance degrades.In that case inorder not to lose mapper o/p we use **Data Localisation** which is the process of storing every mapper output in to the LFS temporary path before loading to HDFS.

Note:Data Localisation is only for the mapper , but not for reducer, the reason might be the data whatever we get from reducer is the final output.

PIG

3.1.pig

Pig was initially developed at Yahoo! to allow people using Hadoop to focus more on analyzing large data sets and spend less time having to write mapper and reducer programs. Like actual pigs, who eat almost anything, the Pig programming language is designed to handle any kind of data—hence the name!

Pig is made up of two components: the first is the language itself, which is called PigLatin, and the second is a runtime environment where PigLatin programs are executed.

Pig is a platform for analyzing large data sets that consists of a high-level language for expressing data analysis programs, coupled with infrastructure for evaluating these programs. The salient property of Pig programs is that their structure is amenable to substantial parallelization, which in turns enables them to handle very large data sets.

3.2.Apache open source project

Can handle structured. semi-structured and un-structured data ,Can work in any framework including Hadoop .All data processing features like Joins, group, filter. order by, etc. are provided by Pig.

At the present time, Pig's infrastructure layer consists of a compiler that produces sequences of Map-Reduce programs, for which large-scale parallel implementations already exist (e.g., the Hadoop subproject). Pig's language layer currently consists of a textual language called Pig Latin, which has the following key properties:

3.2.1.Ease of programming

It is trivial to achieve parallel execution of simple, "embarrassingly parallel" data analysis tasks. Complex tasks comprised of multiple interrelated data transformations are explicitly encoded as data flow sequences, making them easy to write, understand and maintain.

3.2.2.Optimization opportunities

The way in which tasks are encoded permits the system to optimize their execution automatically, allowing the user to focus on semantics rather than efficiency.

3.2.3.Extensibility

Users can create their own functions to do special-purpose processing. Pig Setup Requirements Mandatory.

3.3.Pig Latin

A high level programming language provided by Pig.Can be used in any framework including Hadoop.Knowledge on Java is not required (as in MapReduce).Contains all Data processing features like- joins, group by, filter, order by, etc.

supports a variety of data types,use of UDFs (User Defined Functions) to write custom operations in Java, Python and JavaScript.

3.4.Comments in Scripts

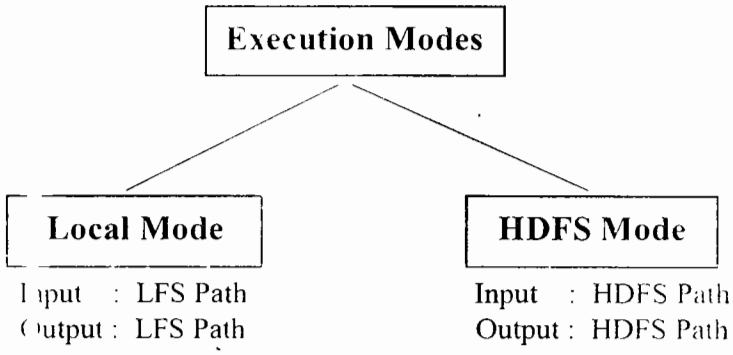
You can include comments in Pig scripts:

For multi-line comments use /* .... */

For single-line comments use –

3.5.Pig has two execution modes

1. Local mode
2. HDFS mode (or) MapReduce mode

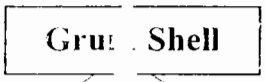


3.6.Running Pig Programs

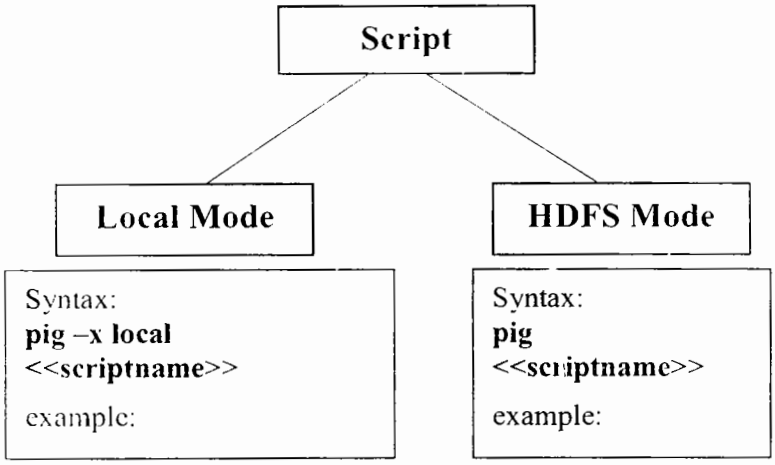
There are three ways of running a pig program.

- ***Grunt***
- ***Script***
- ***Embedded***

3.6.1.Grunt Shell



3.6.2.Script Mode



3.7.Data Types in Pig

SIMPLE TYPES

int, long, float, double, boolean, chararray. bytearray, ...

COMPLEX TYPES

bag. tuple, field, map

3.8.PIG Transformations Examples

3.8.1.Grunt Shell (Local Mode)

INPUT (Local Path)

```
root@ubuntu:/home/BATCH44/PIG# cat student1.txt
1001    Student1    CSE
1002    Student2    ECE
1003    Student3    EEE
1004    Student4    CSE
```

1005 Student5 IT

CODE

```
root@ubuntu:/home/BATCH44/PIG# pig -x local
grunt> A = LOAD 'student1.txt' using PigStorage('\t');
grunt> B = FILTER A by $2 == 'CSE';
grunt> STORE B INTO 'FILTER-OUT';
```

OUTPUT (Local Path)

```
root@ubuntu:/home/BATCH44/PIG# cat FILTER-OUT/part-m-00000
1001    Student1    CSE
1004    Student4    CSE
```

3.8.2.Grunt Shell (HDFS Mode)

INPUT (HDFS Location)

```
root@ubuntu:/home/BATCH44/PIG# hadoop fs -cat /PIG/student1.txt
1001    Student1    CSE
1002    Student2    ECE
1003    Student3    EEE
1004    Student4    CSE
1005    Student5    IT
```

CODE

```
root@ubuntu:/home/BATCH44/PIG# pig
grunt> A = LOAD '/PIG/student1.txt' using PigStorage('\t') AS
        ` (id:int, name:chararray, branch:chararray);
grunt> B = GROUP A BY branch;
grunt> STORE B INTO '/PIG/GROUP-BY-OUT';
```

OUTPUT (HDFS Location)

```
root@ubuntu:/home/BATCH44/PIG# hadoop fs -cat /PIG/GROUP-BY-OUT/part-r-00000
IT      {(1005,Student5,IT)}
CSE     {(1001,Student1,CSE),(1004,Student4,CSE)}
ECE     {(1002,Student2,ECE)}
EEE     {(1003,Student3)}
```

3.8.3.SCRIPT (Local Mode)

INPUT (Local Path)

```
root@ubuntu:/home/BATCH44/PIG# cat student1.txt
1001    Student1    CSE
1002    Student2    ECE
1003    Student3    EEE
```

```
1004    Student4    CSE
1005    Student5    IT
root@ubuntu:/home/BATCH44/PIG# cat student2.txt
1001    Maths  78
1002    Physics    65
1005    English    91
1006    Maths  95
1007    Physics    7
```

CODE

```
root@ubuntu:/home/BATCH44/PIG# nano UnionScript.pig
root@ubuntu:/home/BATCH44/PIG# cat UnionScript.pig
firstData = LOAD 'student1.txt' using PigStorage('\t');
secondData = LOAD 'student2.txt' using PigStorage('\t');
uniData = UNION firstData, secondData;
DUMP uniData;
root@ubuntu:/home/BATCH44/PIG# pig -x local UnionScript.pig
```

OUTPUT

```
(1001,Student1,CSE)
(1002,Student2,ECE)
(1003,Student3,EEE)
(1004,Student4,CSE)
(1005,Student5,IT)
(1001,Maths,78)
(1002,Physics,65)
(1005,English,91)
(1006,Maths,95)
(1007,Physics,72)
```

3.8.4.SCRIPT (HDFS Mode)

INPUT (HDFS Location)

```
root@ubuntu:/home/BATCH44/PIG# hadoop fs -cat /PIG/student1.txt
1001    Student1    CSE
1002    Student2    ECE
1003    Student3    EEE
1004    Student4    CSE
1005    Student5    IT
```

CODE

```
root@ubuntu:/home/BATCH44/PIG# cat SplitScript.pig
```

A = LOAD '/PIG/student1.txt' using PigStorage('\t') AS (id:int, name:chararray, branch:chararray);

SPLIT A into B if branch=='CSE',C if id>1002;

STORE B into '/PIG/SPLIT-OUT1'; STORE C into /PIG/'SPLIT-OUT2';

root@ubuntu:/home/BATCH44/PIG# **pig SplitScript.pig**

OUTPUT

root@ubuntu:/home/BATCH44/PIG# Hadoop fs -cat /PIG/SPLIT-OUT1/part-m-00000

1001 Student1 CSE

1004 Student4 CSE

root@ubuntu:/home/BATCH44/PIG#hadoop fs -cat SPLIT-OUT2/part-m-00000

1003 Student3 EEE

1004 Student4 CSE

1005 Student5 IT

3.9.How To Work With Udf's In Pig

records = LOAD '/home/gopal/pig_inputs/max_temparature' AS (year:int, temperature:int, quality:int);

filtered_records = FILTER records BY temperature != 9999 AND (quality == 0 OR quality == 1 OR quality == 4 OR quality == 5 OR quality == 9);

3.10.How To Write Register Jar In Pig

REGISTER /home/gopal/pig_inputs/pig-examples.jar;

3.11.How To Write Custom Filter Function In Pig

filtered_records = FILTER records BY temperature != 9999 AND com.gopal.kelly.hadoop.training.pig.udfs.IsGoodQuality(quality);

or

DEFINE isGood com.gopal.kelly.hadoop.training.pig.udfs.IsGoodQuality();

filtered_records = FILTER records BY temperature != 9999 AND isGood(quality);

DUMP filtered_records;

(1950,0,1)

(1950,22,1)

(1950,-11,1)

(1949,111,1)

(1949,78,1)

3.12.How To Write Custom Eval Function In Pig

A = LOAD '/home/gopal/pig_inputs/udfs_A' AS (fruit:chararray);

```
DUMP A;
( pomegranate)
(banana  )
(apple)
( mango )
DESCRIBE A;
A: {fruit: chararray}
B = FOREACH A GENERATE com.gopal.kelly.hadoop.training.pig.udfs.Trim(fruit);
DUMP B;
(pomegranate)
(banana)
(apple)
(mango)
DEFINE trim com.gopal.kelly.hadoop.training.pig.udfs.Trim();
B = FOREACH A GENERATE trim(fruit);
DESCRIBE B;
B: {chararray}
DEFINE trim InvokeForString('org.apache.commons.lang.StringUtils.trim', 'String');
B = FOREACH A GENERATE trim(fruit);
DUMP B;
(pomegranate)
(banana)
(apple)
(mango)
DESCRIBE B;
B: {chararray}
```

3.13.How To Write Custom Load Function In Pig

```
records = LOAD '/home/gopal/pig_inputs/udfs_sample.txt' USING
com.gopal.kelly.hadoop.training.pig.udfs.CutLoadFunc('16-19,88-92,93-93') AS (year:int,
temperature:int, quality:int);
DUMP records;
(1950,0,1)
(1950,22,1)
(1950,-11,1)
```

(1949,111,1)

(1949,78,1)

3.14.How To Write Custom Store Function In Pig

```
student = load '/home/gopal/pig_inputs/student.txt' AS (name:chararray, id:int, course:chararray, year:int);
DEFINE store1 com.gopal.kelly.hadoop.pig.udfs.MyStoreFunc();
DEFINE store2 com.gopal.kelly.hadoop.pig.udfs.MyStoreFunc(':');
DEFINE store3 com.gopal.kelly.hadoop.pig.udfs.MyStoreFunc(',', '{"number.multiply":3,"text.multiply":2}');
STORE student INTO '/home/gopal/pig_inputs/student_1' USING store1();
STORE student INTO '/home/gopal/pig_inputs/student_2' USING store2();
STORE student INTO '/home/gopal/pig_inputs/student_3' USING store3();
STORE student INTO '/home/gopal/pig_inputs/student_4' USING com.gopal.kelly.hadoop.training.pig.udfs.MyStoreFunc();
STORE student INTO '/home/gopal/pig_inputs/student_5' USING com.gopal.kelly.hadoop.training.pig.udfs.MyStoreFunc(':');
STORE student INTO '/home/gopal/pig_inputs/student_6' USING com.gopal.kelly.hadoop.training.pig.udfs.MyStoreFunc(',', '{"number.multiply":3,"text.multiply":2}');
```

3.15.How To Write Macros In Pig

```
DEFINE max_by_group(X, group_key, max_field) RETURNS Y
{
 A = GROUP $X by $group_key;
 $Y = FOREACH A GENERATE group, MAX($X.$max_field);
}
records = LOAD '/home/gopal/pig_inputs/max_temparature' AS (year:chararray, temperature:int, quality:int);
filtered_records = FILTER records BY temperature != 9999 AND (quality == 0 OR quality == 1 OR quality == 4 OR quality == 5 OR quality == 9);
max_temp = max_by_group(filtered_records, year, temperature);
DUMP max_temp
(1949,111)
(1950,22)
```

```
records = LOAD '/home/gopal/pig_inputs/sample.txt' AS (year:chararray, temperature:int, quality:int);
filtered_records = FILTER records BY temperature != 9999 AND (quality == 0 OR quality == 1 OR quality == 4 OR quality == 5 OR quality == 9);
macro_max_by_group_A_0 = GROUP filtered_records by (year);
max_temp = FOREACH macro_max_by_group_A_0 GENERATE group, MAX(filtered_records.(temperature));
DUMP max_temp
(1949,111)
(1950,22)
```

To foster reuse, macros can be defined in separate files to Pig scripts, in which case they need to be imported into any script that uses them. An import statement looks like this:

```
IMPORT '/home/gopal/pig_inputs/max_temp.macro';
tuple_records = LOAD '/home/gopal/pig_inputs/tuple.txt' AS (t:(year:chararray, temperature:int, quality:int));
dump tuple_records;
((1950,0,1))
((1950,22,1))
((1950,-11,1))
((1949,111,1))
((1949,78,1))
bag_records = LOAD '/home/gopal/pig_inputs/bag.txt' AS (b:{t:(year:chararray, temperature:int, quality:int)});
dump bag_records;
({(1950,0,1)})
({(1950,22,1)})
({(1950,-11,1)})
({(1949,111,1)})
({(1949,78,1)})
multi_records = LOAD '/home/gopal/pig_inputs/multi.txt' AS (b:{t:(year:chararray, temperature:int, quality:int)},t:(year:chararray, temperature:int));
dump multi_records;
({(1950,0,1)},(1950,0))
({(1950,22,1)},(1950,22))
```

({(1950,-11,1)},(1950,-11))

({(1949,111,1)},(1949,111))

({(1949,78,1)},(1949,78))

3.16.Pig Latin – Efficiency and Performance improvement

Some techniques to improve the performance and efficiency of Pig Latin programming are:

By using performance enhancers like

1.Using proper Data Types

2.Project and Filter early

3.Avoid splitting of operations in code

4.Drop Null values Before a Join

5.Make use of PARALLEL

6.LIMIT the output

7.Use of DISTINCT instead of GROUP BY or GENERATE

3.16.1.Using proper Data Types

Use of proper Data types while loading data will have advantages like Saving space early detection of error in code increases the speed of data loading and Arithmetic computation

Example

Query

StudentData = load 'Student_Data' as(RegNO. Name. Age. Maths, Physics, Chemistry).

Student_Marks = foreach StudentData generate RegNO, Name,

Age,Maths+Physics+Chemistry as Total;

Pig will give a warning for implicit data type casting as " Encountered Warning IMPLICIT_CAST_TO_DOUBLE <n>time(s)".

Since Pig Latin will assign 'Double' to all numeric data if the data type is not specified which affects the processing speed.

Optimized query

StudentData = load 'Student_Data' as(RegNO:int, Name:chararray, Age:int, Maths:int.

Physics:int, Chemistry:int);

Student_Marks = foreach StudentData generate RegNO, Name, Age, Maths+Physics+Chemistry as Total;

This query will be executed successfully without any warnings and will run more efficiently than the 1st query

3.16.2.Project early

Drop fields which are not used any more. As this will help in reducing the processing time and will also reduce the amount of data carried to MapReduce phase by pig script.

3.16.3.FILTER early

Apply FILETR before JOIN where ever possible, as this will reduce the amount of data being carried to JOIN and will increase the processing speed and reduce the time of execution. Thus increases the efficiency of pig query.

FILTER early will not be feasible in only one scenario where the FILTER is applied on large amount of data and only few data is filtered out.

Project early

Example:

Consider the scenario where each Student's RegisterNO, Name and Total fees has to be extracted from the data files.

In the optimized query, only the necessary columns alone extracted using FOREACH. Thus the unnecessary loading of data on join is avoided to improve the efficiency.

Query:	Optimized Query:

Student = load 'StudentData' as (RegNO,Name,Age);	Student = load 'StudentData' as (RegNO,Name,Age); Student1 = foreach Student generate RegNO,Name;
Fees = load 'FeesData' as (RegNO,TotalFees,Sports,Tution);	Fees = load 'FeesData' as (RegNO,TotalFees,Sports,Tution) ; Fees1 = foreach Fees generate RegNO,TotalFees
StudFees = join Student by RegNO, Fees by RegNO;	StudFees = join Student1 by RegNO, Fees1 by RegNO.
Final = foreach StudFees generate	

FILTER early

Example: Consider the scenario where Students with Grade=1 are joined with I-class. In the optimized query, Students with Grade=1 are extracted from other students for processing thus reducing the amount of data carried to JOIN in order to improve efficiency

Query:	**Query:**
StudentData = load 'Student_Data.txt' using PigStorage(',') as (RegNO,Name,Grade);	StudentData = load 'Student_Data.txt' using PigStorage(',') as(RegNO,Name,Grade);
StudentSubject = load 'Student_Subjects.txt using PigStorage(',') as(Class,Subject1,Subject2,Subject3);	StudentSubject = load 'Student_Subjects.txt' using PigStorage(',') as (Class,Subject1,Subject2,Subject3);

StudentClass = join StudentData by Grade StudentSubject by Class; FinalList = group StudentClass by Class; StudentCount = foreach FinalList generate group COUNT($1); SelectedStudents = filter StudentCount by StudentClass.Grade==1;	SelectedStudents = filter StudentData by Grade==1; StudentClass = join SelectedStudents by Grade StudentSubject by Class; FinalList = group StudentClass by Class; StudentCount = foreach FinalList generate group, COUNT($1); StudentCount = foreach FinalList generate group, COUNT($1);

3.16.4.Avoid splitting of operations in code

Code can be split to smaller units for clarity.

But combining one or more small steps together will improve the performance.

Example: Consider the scenario where FirstName and LastName of Students have to be concatenated.

Query:	**Optimized query:**
StudentData = load 'StudentFile' as (RegNO, FirstName, LastName, mark1, mark2, mark3); StudentName = foreach StudentData generate FirstName, LastName;	StudentData = load 'StudentFile' as (RegNO, FirstName, LastName, mark1, mark2, mark3);

Final = foreach StudentName generate CONCAT(FirstName, LastName);	Final = foreach StudentData generate CONCAT(FirstName, LastName);

3.16.5.Make use of PARALLEL

Setting the number of Reducer task in pig script will improve the performance and efficiency of Pig query.

The Number of Reducers can be set in 2 ways

At the script level

```
A = LOAD 'ProductFile' AS (ProdID,Name,Price);
B = GROUP A BY ProdID PARALLEL 11;
```

At the Operator level

```
SET default_parallel 21;
A = LOAD 'ProductFile' AS (ProdID,Name,Price);
B = GROUP A BY ProdID;
```

3.16.6.LIMIT the output

Use LIMIT operator in case of sample output or to take the top level records.This will improve the performance and will reduce the high data flowing through the pipeline.

Example:

To get sample records of Product data

```
ProductData = load 'ProductFile' as (ProdID,Name,Price);
sampleData = limit ProductData 10;
```

To get top results like - Products with Higher prices

```
ProductData  = load 'ProductFile' as (ProdID,Name,Price);
PriceData  = order ProductData by price;
TopData = limit PriceData 10;
```

3.16.8.Use of DISTINCT instead of GROUP BY or GENERATE

The DISTINCT operator is used to get all the unique values from a column. The same can also be done by using GROUP BY – GENERATE operators. But when comparing the two, DISTINCT is much more faster and very efficient and also it

decreases the number of coding steps or pig data processing which in turn will increase the performance.

Example : Consider the scenario where the various sports being played by students have to be extracted. (To extract the unique sports name played by students)

Query: using GROUP BY – GENERATE

Step 1: loading of Student data file to an alias

```
StudentData = load 'Student_Data' as
(RegNO,Name,Age,Sports);
Student_Sports = foreach StudentData generate Sports;
```

Step 2: extraction of unique sports names

```
Student_Sports_Temp = group Student_Sports by Sports;
Sports_Data= foreach Student_Sports_Temp generate group as
uniquekey;
```

Optimized query - using DISTINCT

Step 1: loading of Student data file to an alias

```
StudentData = load 'Student_Data' as
(RegNO,Name,Age,Sports);
Student_Sports = foreach StudentData generate Sports;
```

Step 2: extraction of unique sports names

```
Sports_Data = distinct Student_Sports ;
```

3.17.Optimization Rules

By default all the optimization rules are turned on. The command to turn off is:

pig -optimizer_off [opt_rule | all]

Example:

To turn off all the pig optimization rules

To turn off a specific optimization rule:

pig –optimizer_off <optimisation rule name>

pig –optimizer_off StreamOptimizer

Some Optimization rules are mandatory and cannot be turned off

1. Use of Optimization rules
2. Proper use of FILTER
3. Push Down FOREACH FLATTEN
4. Limit Optimizer
5. FOREACH optimization
6. GROUP BY optimization

3.17.1.Use of Optimization rules

By following the below techniques while using FILTER in script, the execution will be more efficient and will increase the performance.

Use Pre-Calculation while using constants in FILTER

Elimination of naive TRUE expression

Elimination of complementary expressions in OR

Splitting of FILTER statement

Elimination of logically implied expressions in AND, OR, NOT

Use Pre-Calculation while using constants in FILTER

Example:

Query: Mark = FILTER Student BY MathsMark > 50+40;

Optimized Query: Mark = FILTER Student BY MathsMark > 90;

Elimination of naive TRUE expression

Example:

Query: Product = FILTER Price BY 1==1;

Optimized Query: no FILTER required

Elimination of complementary expressions in OR

Example:

Query: Mark = FILTER Student BY (MathsMark > 50 OR MathsMark <= 50)

Optimized Query: no FILTER required

Splitting of FILTER statement

Student_Maths = load 'StudentFile1' as (RegNO,Name,MathsMark)

Student_Science = load 'StudentFile2' as (RegNO,Name,ScienceMark);

StudentMarks = join Student_Maths by RegNO, Student_Science by RegNO;

SelectedStudents = FILTER StudentMarks by MathsMark > 90 and ScienceMark > 90;

Splitting filter as below:

SelectedStudents1 = FILTER StudentMarks by MathsMark > 90;

SelectedStudents2 = FILTER StudentMarks by ScienceMark > 90;

3.17.2.Proper use of FILTER

Elimination of logically implied expressions in AND

Example:

Query: Mark = FILTER Student BY (MathsMark > 50 **AND** MathsMark > 70);

Optimized Query: Mark = FILTER Student BY MathsMark > 70;

Elimination of logically implied expressions in OR

Example

Query: Mark = FILTER Student BY ((MathsMark > 50) **OR** (MathsMark > 70 AND ScienceMark > 50)); **Optimized Query**: Mark = FILTER Student BY MathsMark > 50

Elimination of logically implied expressions in NOT

Example:

Query: Mark = FILTER Student BY **NOT** (**NOT**(MathsMark > 50) OR ScienceMark > 50)

Optimized Query: Mark = FILTER Student BY MathsMark > 50 AND ScienceMark > 50;

3.17.3.Limit Optimizer

The LIMIT command can be used at its best when used along with a ORDER BY command. If the given data is arranged in some order and if the LIMIT command is used, then the resultant output will be much more useful than LIMIT used alone. By pushing LIMIT into the ORDER BY command, some top level data can be obtained as result

Example:

ProductData = load 'ProductFile' as (ProdID,Name,Price);

PriceData = order ProductData by price;

TopData = limit PriceData 10;

3.17.4.FOREACH optimization

Add FOREACH

trim columns which are not used in any of the forthcoming lines in the script, to improve efficiency.

Example: Consider the scenario to extract data where salary is greater than Rs.5000.

Query:

```
EmpData = load 'EmployeeFile' as (EmpID,Name,Age,Salary,Addres);
EmpDataOrder = order EmpData by Age;
EmpData_Sal = filter EmpDataOrder by Salary>5000;
```

trim columns which are not used in any of the forthcoming lines in the script, to improve efficiency.

Example: Consider the scenario to extract data where salary is greater than Rs.5000. the columns EmpID, Name, Age and Address which are not used in the forthcoming columns are trimmed using FOREACH.

Optimized query:

```
EmpData = load 'EmployeeFile as (EmpID,Name,Age,Salary,Addres);
EmpDataOrder = order EmpData by Age;
EmpDataOrder_Tmp = foreach EmpDataOrder generate Salary;
EmpData_Sal = filter EmpDataOrder_Tmp by Salary>5000;
```

3.17.5.GROUP ALL optimization

The PARALLEL command is used for setting the number of reducers in the script. But in case of 'GROUP ALL', even if the PARALLEL is set to 'n' number of reducers, only one reducer will be giving the output and all other reducers will give an empty result. Hence it is better to set PARALLEL to '1' in case of GROUP BY, to avoid unnecessary empty reducer results.

Example:

A = LOAD 'ProductFile' AS (ProdID,Name,Price);

B = GROUP A all PARALLEL 1;

3.18.Multi-Query Execution

Multi-Query Execution – processing of an entire Pig Latin script or a batch of statements at once.

In Multi-Query Execution for batch mode execution the below steps are done by Pig:

The entire script is first parsed

Parsing is done to find if the intermediate tasks can be combined together Combining of intermediate tasks will reduce the amount of work to be done, thus increasing performance and efficiency.

Execution of script will start only after parsing of script

By default Multi-Query Execution is turned on

The command to turn off and work with dump/store (to execute without optimization) is:

pig -M myscript.pig

pig -no_multiquery myscript.pig

3.19.Other techniques : Multi-Query Execution

1. Explicit and Implicit Splits
2. Storing Intermediate Results
3. Store vs. Dump
4. Error Handling
5. Use of EXEC

3.19.1.Explicit and Implicit Splits

Pig Multi-Query Execution handles the processing of same data multiple times using Explicit and Implicit splitting.

Example: Consider the scenario where students have to separated according to their preferred subjects.

Step 1: loading of Student data:

StudentData = load 'Student_Data' as (RegNO,Name,Age,PreferredSubject);

Step 2: separating:

Students: Students_ScienceGrp = FILTER StudentData BY PreferredSubject == 'Science'; Students_MathsGrp = FILTER

StudentData BY PreferredSubject == 'Math';

Here the same Student data is processed 2 times which affects performance. In case if the Multi-Query Execution is not turned off, then the script is parsed and an Implicit split is done and gives multiple outputs in single process. In simple, Multi-Query Execution converts the step2 as below:

Optimized Step 2: separating students

SPLIT StudentData INTO

Students_ScienceGrp IF PreferredSubject == 'Science', Students_MathsGrp IF PreferredSubject == 'Math;

3.19.2.Storing Intermediate Results

Whenever data is loaded by Pig, the multi-query execution will process the alias and will store the data even if the STORE command is not given in the script. But it is always better to store the result using the STORE command in the script to avoid duplication of data while re-loading of alias.

3.19.3.Store vs. Dump

Using the interactive command DUMP will disable the multi-query execution and will also slow down the execution. Hence it is best to use STORE instead of DUMP.

Example:

Step 1: loading of Student data:

StudentData = load 'Student_Data' as (RegNO,Name,Age,PreferredSubject);

Step 2: Storing of alias data:

Store StudentData into 'Student_Data_File' using PigStorage(',');

3.19.4.Error Handling

Even if some intermediate jobs fail Pig tries to execute all the jobs since, multi-query execution will process the entire script at once. Hence it is necessary to check the return code jobs in Pig and make the script to continue or stop the process depending on the value of the return code.

<u>Various Return Codes in Pig:</u>

Job Status	Return Code
All jobs succeeded	**0**
Used for retrievable errors	**1**
All jobs have failed	**2**
Some jobs have failed	**3**

To stop the script execution once a job failure is detected, use one of the below commands:

pig -F myscript.pig	Or	pig -stop_on_failure myscript.pig

3.19.5.Use of EXEC

Example:

To make the script work EXEC command is used as below.

The EXEC command will trigger the execution of statements that produce the 'EmployeeData_File'.

3.20.User Defined Function (UDF)

A User Defined Function or UDF is a way to specify custom processing. For logic that cannot be done using the system-defined functions in Pig, developers can write their own functions. UDFs can be used to do column transformation, filtering, ordering, custom aggregation and much more.

Business Scenario:

Problem Statement:

Lets say we have a log file, which has time in epoch format

12946472

12942346

12946412

12923462

12245734

134998034

190874678

23123455

5674466

34556

123445556

We need to convert this epoch time (number of seconds that have elapsed since January 1, 1970) to a "yyyy-MM-dd:HH" date format and store the output in a new file. We don't have a built in function in Pig to convert epoch time.

Approach to solution:

We need to come up with a UDF for converting epoch time. Let's now look into the implementation of the epoch converter UDF

epochScript

```
REGISTER EPOCH-UC.jar;
iniData = LOAD '/home/PIG/InputDates.txt'  as (timeline:c  ararray);
iniData = LOAD 'epochTimeData.log'  as (timeline:chararr  y);
forTimeData = FOREACH iniData GENERATE timeline ,
com.epoch.pigusecase.EPOCH(timeline);
STORE forTimeData INTO 'EPOCH-OUTPUT1' using Pig Storage('|');
package com.epoch.pigusecase;
```

JAVA CODE FOR EPOCH:

```
import java.io.IOException;
import java.text.*;
import java.util.*;
import java.lang.*;
import org.apache.pig.EvalFunc;
import org.apache.pig.data.Tuple;
import org.apache.pig.impl.util.WrappedIOException;
public class EPOCH extends EvalFunc<String>
{
    public String exec(Tuple input) throws IOException {
        if (input == null || input.size() == 0)
            return null;
        try{
            String str = (String)input.get(0);
            String date = new java.text.SimpleDateFormat("yyyy-MM-
dd:HH").format(new java.util.Date (integer.parseInt(str)*1000));
```

```
                return date;
            }catch(Exception e){
                throw WrappedIOException.wrap("Caught exception processing input row ", e);
            }
        }
    }
```

3.21. Multiple XML file data load to Pig Environment

Source Data information :

It comprises the HRA data from Cisco and allows to process all the HR data in Hadoop Eco-System. The data is in XML format and contains various categories defining it NAME ,JOB,COUNTRY,CURRENCY,SALARY.

Multiple source xml files received from client:

xml file name : names.xml **xml file name : names1.xml**

Problem Statement : Analyze the data in Hadoop Eco-system to :

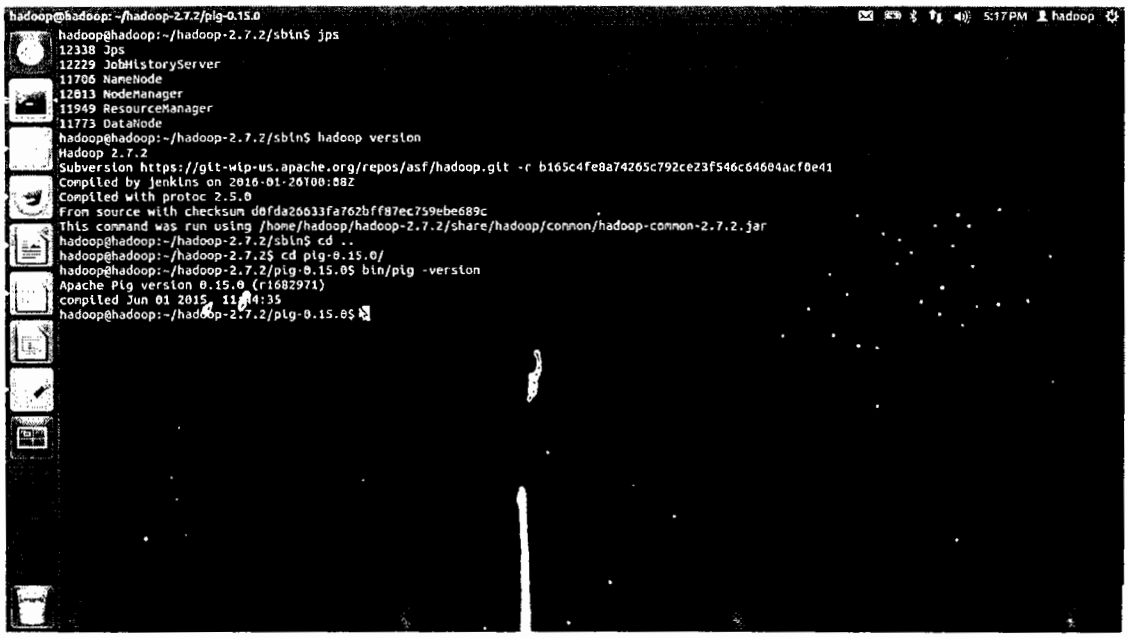
1.Fetch the XML data's(semi-structured) into Hadoop Distributed File System and store the data by using PIG scripts for further level of analysis.

2.Keep all the data in proper and readable format for easy level of analysis.

Objective : To drive this use case we can apply :

- **All xml files should be available in HDFS .**
- **Use Pig software to drive the case.**
- **Use piggybank default utility jar from pig.**
- **Use Regex utlity to understand all data inside various categories.**

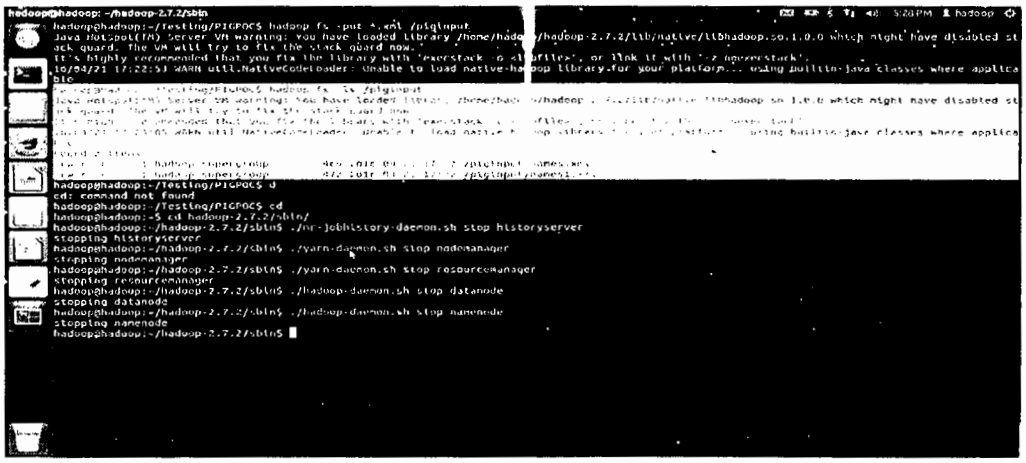
Environment details :



Step 1 :

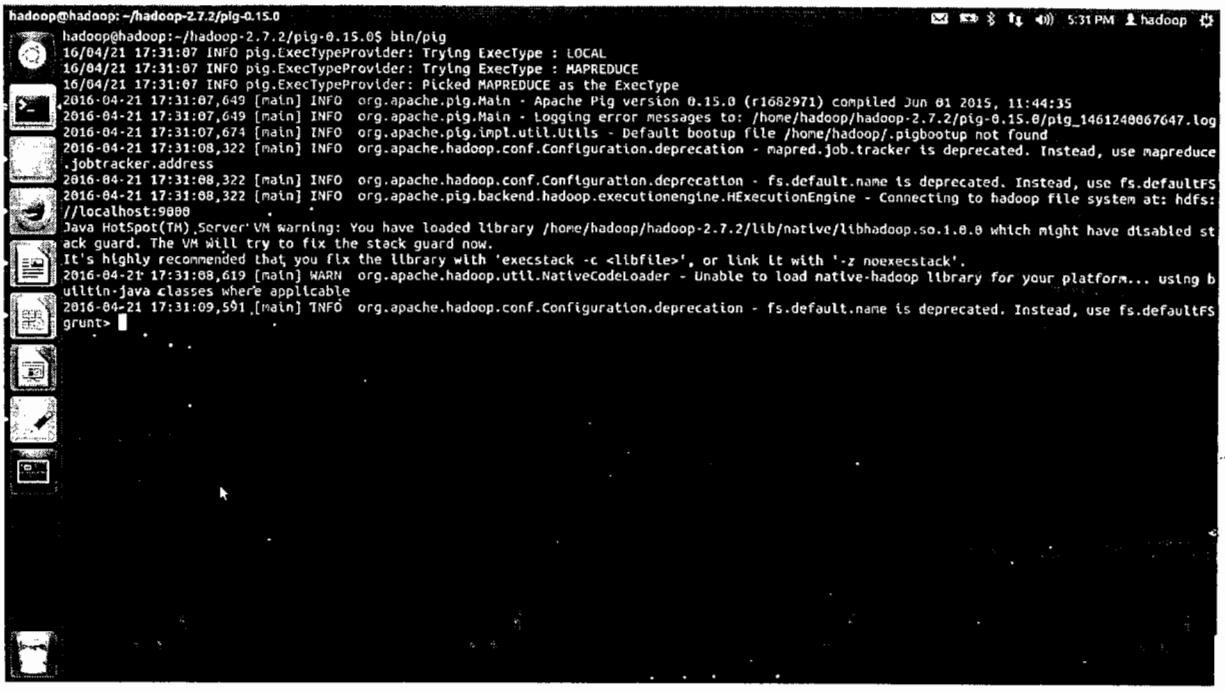
gopal@ubuntu:~/Testing/PIGPOC$ hadoop fs -put *.xml /piginput

gopal@ubuntu:~/Testing/PIGPOC$ hadoop fs -ls /piginput



Step 2:

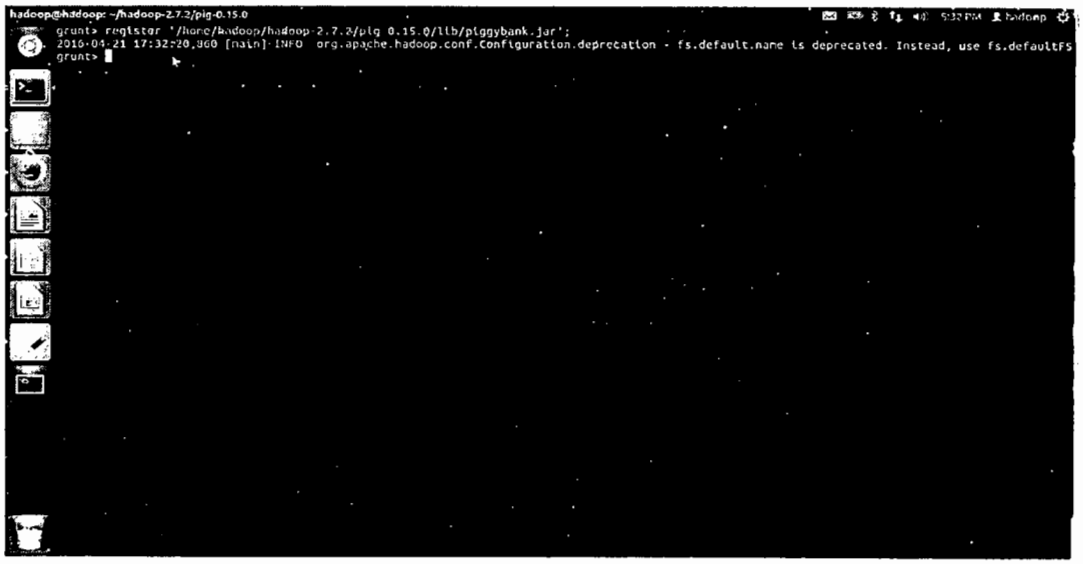
Open PIG grunt HDFS terminal



Step 3:

Register piggybank default jar from lib folder :

grunt> register '/home/hadoop/hadoop-2.7.2/pig-0.15.0/lib/piggybank.jar';



Step 4:

Load multiple XML file datas to Pig ennvironment

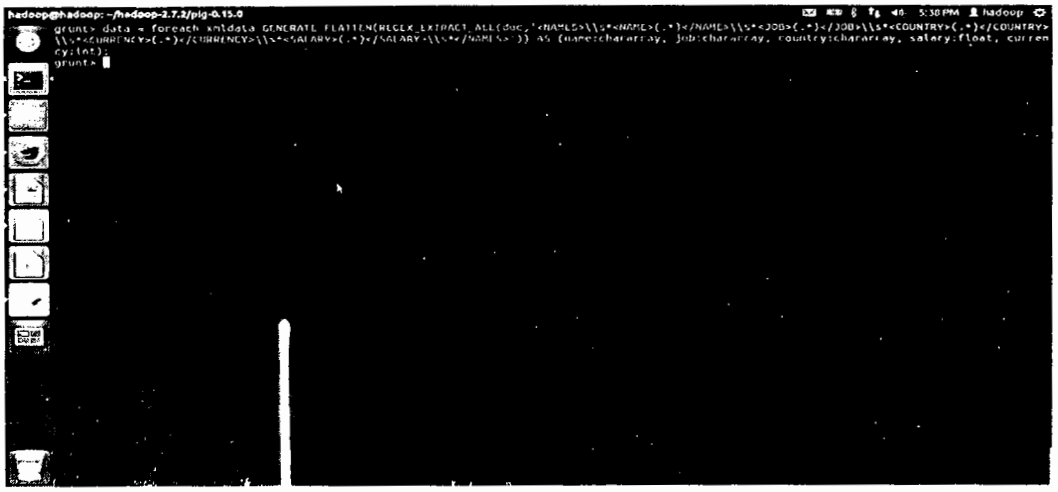
grunt> xmldata = load '/piginput/*.xml' USING

org.apache.pig.piggybank.storage.XMLLoader('NAMES') as(doc:chararray);

Step 5 :

Use Regex utility

grunt> data = foreach xmldata GENERATE FLATTEN(REGEX_EXTRACT_ALL(doc,'<NAMES>\\s*<NAME>(.*)</NAME>\\s*<JOB>(.*)</JOB>\\s*<COUNTRY>(.*)</COUNTRY>\\s*<CURRENCY>(.*)</CURRENCY>\\s*<SALARY>(.*)</SALARY>\\s*</NAMES>')) AS (name:chararray, job:chararray, country:chararray, salary:float, currency:int);



Sep 6:

To view the desired output

grunt> dump data;

```
hadoop@hadoop: ~/hadoop-2.7.2/pig-0.15.0

Job DAG:
job_1461240036408_0002

2016-04-21 17:38:52,155 [main] INFO  org.apache.hadoop.yarn.client.RMProxy - Connecting to ResourceManager at /0.0.0.0:8032
2016-04-21 17:38:52,161 [main] INFO  org.apache.hadoop.mapred.ClientServiceDelegate - Application state is completed. FinalApplicationStatus=SUC
CEEDED. Redirecting to job history server
2016-04-21 17:38:52,216 [main] INFO  org.apache.hadoop.yarn.client.RMProxy - Connecting to ResourceManager at /0.0.0.0:8032
2016-04-21 17:38:52,225 [main] INFO  org.apache.hadoop.mapred.ClientServiceDelegate - Application state is completed. FinalApplicationStatus=SUC
CEEDED. Redirecting to job history server
2016-04-21 17:38:52,273 [main] INFO  org.apache.hadoop.yarn.client.RMProxy - Connecting to ResourceManager at /0.0.0.0:8032
2016-04-21 17:38:52,281 [main] INFO  org.apache.hadoop.mapred.ClientServiceDelegate - Application state is completed. FinalApplicationStatus=SUC
CEEDED. Redirecting to job history server
2016-04-21 17:38:52,339 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2016-04-21 17:38:52,341 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2016-04-21 17:38:52,342 [main] INFO  org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2016-04-21 17:38:52,352 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2016-04-21 17:38:52,352 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(ABHI,BIGDATA DEVELOPER,INDIA-BANGALORE,RS,60000)
(RAVI,BIGDATA ARCHITECT,INDIA-HYDERABAD,RS,80000)
(YOGESH,BIGDATA ETL DEVELOPER,INDIA-CHENNAI,RS,70000)
(YASH,BIGDATA ARCHITECT,USA,DOLAR,8000.80)
(KARTIK,BIGDATA SOLUTION ARCHITECT,RIYADH,RIAL,30000.50)
(PRAKASH,BIGDATA ETL DEVELOPER,USA,DOLLAR,5000)
grunt>
grunt>
grunt>
grunt>
grunt>
grunt>
grunt>
grunt>
grunt>
grunt>
grunt>
grunt>
grunt>
grunt>
grunt>
grunt>
```

Step 7 :

verified the above output with below source data

```
hadoop@hadoop: ~/Testing/PIGPOC
<CATALOG>

<NAMES>
        <NAME>YASH</NAME>
        <JOB>BIGDATA ARCHITECT</JOB>
        <COUNTRY>USA</COUNTRY>
        <CURRENCY>DOLAR</CURRENCY>
        <SALARY>8000.80</SALARY>

</NAMES>
<NAMES>
        <NAME>KARTIK</NAME>
        <JOB>BIGDATA SOLUTION ARCHITECT</JOB>
        <COUNTRY>RIYADH</COUNTRY>
        <CURRENCY>RIAL</CURRENCY>
        <SALARY>30000.50</SALARY>

</NAMES>
<NAMES>
        <NAME>PRAKASH</NAME>
        <JOB>BIGDATA ETL DEVELOPER</JOB>
        <COUNTRY>USA</COUNTRY>
        <CURRENCY>DOLLAR</CURRENCY>
        <SALARY>5000</SALARY>

</NAMES>

"names.xml" 26 lines, 409 characters
```

name1.xml original data

3 22.PIG Custom Data Types

T ıple Usage

c ıt data;

(3 ,8,9) (4,5,6)

(1 ,4,7) (3,7,5)

(2 ,5,8) (9,5,8)

A = LOAD 'data' AS (t1:tuple(t1a:int, t1b:int,t1c:int),t2:tuple(t2a:int,t2b:int,t2c:int));

DUMP A;

((3,8,9),(4,5,6))

((1,4,7),(3,7,5))

((2,5,8),(9,5,8))

X = FOREACH A GENERATE t1.t1a,t2.$0;

DUMP X;

(3 4)

(1 3)

(2 9)

Map

map is a set of key value pairs.

Syntax (<> denotes optional)

[key#value <, key#value ...>]

Terms

[ ]	Maps are enclosed in straight brackets [ ].
#	Key value pairs are separated by the pound sign #.
key	Must be chararray data type. Must be a unique value.
value	Any data type (the defaults to bytearray).

Example

In this example the map includes two key value pairs.

```
[name#John,phone#5551212]
cat data;
[open#apache]
[apache#hadoop]
A = LOAD 'data' AS (M:map [])
A = LOAD 'data' AS (M:[]);
DESCRIBE A;
a: {M: map[ ]}
DUMP A;
([open#apache])
([apache#hadoop])
```

This example shows the use of a typed maps.

```
/* Map types are declared*/
a = load '1.txt' as(map[int]); --Map value is int
b = foreach a generate (map[(i:int)])a0; -- Map value is tuple
b = stream a through `cat` as (m:map[{(i:int,j:chararray)}]); -- Map value is bag

/* The MapLookup of a typed map will result in a datatype of the map value */
a = load '1.txt' as(map[int]);
b = foreach a generate $0#'key';
```

/* Schema for b */

b: {int}

In this example the schema defines a tuple, bag, and map.

A = LOAD 'mydata' AS (T1:tuple(f1:int, f2:int), B:bag{T2:tuple(t1:float,t2:float)}, M:map[]);

A = LOAD 'mydata' AS (T1:(f1:int, f2:int), B:{T2:(t1:float,t2:float)}, M:[]);

Examples

Suppose we have relation A.

A = LOAD 'data' AS (f1:int, f2:int, B:bag{T:tuple(t1:int,t2:int)});

DUMP A;

(10,1,{(2,3),(4,6)})

(10,3,{(2,3),(4,6)})

(10,6,{(2,3),(4,6),(5,7)})

In this example the modulo operator is used with fields f1 and f2.

X = FOREACH A GENERATE f1, f2, f1%f2;

DUMP X;

(10,1,0)

(10,3,1)

(10,6,4)

STREAM

Sends data to an external script or program.

Syntax

alias = STREAM alias [, alias ...] THROUGH {`command` | cmd_alias } [AS schema]

EExample:

A = LOAD 'data';

B = STREAM A THROUGH `stream.pl -n 5`;

ABOUT PIG SCHEMA (FULL SCHEMA, PARTIAL SCHEMA, WITH OUT SCHEMA)

```
records = load '/home/gopal/pig_inputs/sample.txt' AS (year, temperature, quality);
describe records;
records: {year: bytearray,temperature: bytearray,quality: bytearray}

records = load '/home/gopal/pig_inputs/sample.txt' AS (year:int, temperature:int, quality:int);
describe records;
records: {year: int,temperature: int,quality: int}

records = load '/home/gopal/pig_inputs/sample.txt' AS (year, temperature:int, quality:int);
describe records;
records: {year: bytearray,temperature: int,quality: int}
records = load '/home/gopal/pig_inputs/sample.txt';
describe records;
Schema for records unknown.
```

3.23.EXAMPLES ON PIG

```
// select year,temperature,quality from records;
projected_records = foreach records generate $0, $1, $2;
dump projected_records;
(1950,0,1)
(1950,22,1)
(1950,-11,1)
(1949,111,1)
(1949.78,1)
describe projected_records;
projected_records: {bytearray,bytearray,bytearray}
```

schema is used at run time, if the value of the field cannot be cast, then a null is replaced with it

```
A = load '/home/gopal/pig_inputs/schema_A' AS (id:int, item:int);
describe A;
A: {id: int,item: int}
dump A;
```

```
(2,)
(4,)
(3,)
(1,)
B = foreach A generate $0, $1;
describe B;
B: {id: int,item: int}
```

For Each:

```
A = load '/home/gopal/pig_inputs/foreach_A' AS (f0:chararray, f1:chararray, f2:int);
dump A;
(Joe,cherry,2)
(Ali,apple,3)
(Joe,banana,2)
(Eve,apple,7)
store A into 'out' using PigStorage(':');
cat out
Joe:cherry:2
Ali:apple:3
Joe:banana:2
Eve:apple:7
B = foreach A generate $0, $2+1, 'Constant';
dump B;
(Joe,3,Constant)
(Ali,4,Constant)
(Joe,3,Constant)
(Eve,8,Constant)
describe B;
B: {f0: chararray,int,chararray}
C = foreach A generate $0, (int) $2 +1 AS f , 'Constant' AS f2;
dump C;
(Joe,2,Constant)
(Ali,3,Constant)
(Joe,2,Constant)
```

(Eve,7,Constant)

describe C;

C: {f0: chararray,f1: int,f2: chararray}

Sort:

A = load '/home/gopal/pig_inputs/sort_A';

dump A;

(2,3)

(1,2)

(1,5)

(2,4)

B = **ORDER** A BY $0, $1 DESC;

dump B;

(1,5)

(1,2)

(2,4)

(2,3)

D = **LIMIT** B 2;

dump D;

(1,5)

(1,2)

Combine:

A = load '/home/gopal/pig_inputs/combine_A' AS (f0:int, f1:int);

B = load '/home/gopal/pig_inputs/combine_B' AS (f0:chararray, f1:chararray, f2:int);

dump A;

(2,3)

(1,2)

(2,4)

dump B;

(z,x,8)

(w,y,1)

C = **UNION** A. B;

describe A;

A: {f0: int,f1: int}

describe B;

B: {f0: chararray,f1: chararray,f2: int}

describe C;

Schema for C unknowr

store C into 'out' using PigStorage(':');

dump C;

(z,x,8)

(w,y,1)

(2,3)

(1,2)

(2,4)

D = foreach C generate $0,$1,$2;

dump D;

(2,3,)

(1,2,)

(2,4,)

(z,x,8)

(w,y,1)

Group:

A = load '/home/gopal/pig_inputs/group_A';

dump A;

(Joe,cherry)

(Ali,apple)

```
(Joe,banana)
(Eve,apple)

B = GROUP A BY $1;
dump B;
(apple,{(Ali,apple),(Eve,apple)})
(banana,{(Joe,banana)})
(cherry,{(Joe,cherry)})

B = GROUP A BY SIZE($1);
dump B;
(5,{(Ali,apple),(Eve,apple)})
(6,{(Joe,cherry),(Joe,banana)})

describe B;
B: {group: long,A: {()}}

C = GROUP A ALL;
dump C;
(all,{(Joe,cherry),(Ali,apple),(Joe,banana),(Eve,apple)})

describe C;
C: {group: chararray,A: {()}}
D = foreach C generate COUNT(A);
dump D;
(4)
```

Join:

```
A = load '/home/gopal/pig_inputs/join_A' AS (id:int, name:chararray);
B = load '/home/gopal/pig_inputs/join_B' AS (name:chararray, id:int);
dump A;
(2,Tie)
(4,Coat)
(3,Hat)
```

(1,Scarf)

dump B;

(Joe,2)

(Hank,4)

(Ali,0)

(Eve,3)

(Hank,2)

inner join:

C = JOIN A BY $0, B BY $1;

dump C;

(2,Tie,Joe,2)

(2,Tie,Hank,2)

(3,Hat,Eve,3)

(4,Coat,Hank,4)

left outer join:

C = JOIN A BY $0 LEFT OUTER, B BY $1;

dump C;

(1,Scarf,,)

(2,Tie,Joe,2)

(2,Tie,Hank,2)

(3,Hat,Eve,3)

(4,Coat,Hank,4)

right outer join:

C = JOIN A BY $0 RIGHT OUTER, B BY $1;

dump C;

(,,Ali,0)

(2,Tie,Joe,2)

(2,Tie,Hank,2)

(3,Hat,Eve,3)

(4,Coat,Hank,4)

full outer join:

C = JOIN A BY $0 FULL OUTER, B BY $1;

dump C;

(,,Ali,0)

(1,Scarf,,)

(2,Tie,Joe,2)

(2,Tie,Hank,2)

(3,Hat,Eve,3)

(4,Coat,Hank,4)

MapSide Join:

```
C = JOIN A BY $0, B BY $1 USING 'replicated';
C = JOIN A BY $0, B BY $1 USING 'skewed';
C = JOIN A BY $0, B BY $1 USING 'merge';

D = COGROUP A BY $0, B BY $1;
dump D;
(0,{},{(Ali,0)})
(1,{(1,Scarf)},{})
(2,{(2,Tie)},{(Joe,2),(Hank,2)})
(3,{(3,Hat)},{(Eve,3)})
(4,{(4,Coat)},{(Hank,4)})

E = COGROUP A BY $0 INNER, B BY $1;
dump E;
(1,{(1,Scarf)},{})
(2,{(2,Tie)},{(Joe,2),(Hank,2)})
(3,{(3,Hat)},{(Eve,3)})
(4,{(4,Coat)},{(Hank,4)})

G = COGROUP A BY $0, B BY $1 INNER;
dump G;
(0,{},{(Ali,0)})
(2,{(2,Tie)},{(Joe,2),(Hank,2)})
(3,{(3,Hat)},{(Eve,3)})
(4,{(4,Coat)},{(Hank,4)})

G = COGROUP A BY $0 INNER, B BY $1 INNER;
```

```
dump G;
(2,{(2,Tie)},{(Joe,2),(Hank,2)})
(3,{(3,Hat)},{(Eve,3)})
(4,{(4,Coat)},{(Hank,4)})

F = foreach E generate FLATTEN(A), B.$0;
dump F;
(1,Scarf,{})
(2,Tie,{(Joe),(Hank)})
(3,Hat,{(Eve)})
(4,Coat,{(Hank)})

H = foreach G generate $1;
dump H;
({(2,Tie)})
({(3,Hat)})
({(4,Coat)})

H = foreach G generate $2;
dump H;
({(Joe,2),(Hank,2)})
({(Eve,3)})
({(Hank,4)})
H = foreach G generate FLATTEN($1);
dump H;
(2,Tie)
(3,Hat)
(4,Coat)

H = foreach G generate FLATTEN($2);
dump H;
(Joe,2)
(Hank,2)
(Eve,3)
```

```
(Hank,4)

H = foreach G generate FLATTEN($1), $2;
dump H;
(2,Tie,{(Joe,2),(Hank,2)})
(3,Hat,{(Eve,3)})
(4,Coat,{(Hank,4)})

H = foreach G generate FLATTEN($1), FLATTEN($2);
dump H;
(2,Tie,Joe,2)
(2,Tie,Hank,2)
(3,Hat,Eve,3)
(4,Coat,Hank,4)

I = CROSS A, B;
dump I;
(2,Tie,Joe,2)
(2,Tie,Hank,4)
(2,Tie,Ali,0)
(2,Tie,Eve,3)
(2,Tie,Hank,2)
(4,Coat,Joe,2)
(4,Coat,Hank,4)
(4,Coat,Ali,0)
(4,Coat,Eve,3)
(4,Coat,Hank,2)
(3,Hat,Joe,2)
(3,Hat,Hank,4)
(3,Hat,Ali,0)
(3,Hat,Eve,3)
(3,Hat,Hank,2)
(1,Scarf,Joe,2)
(1,Scarf,Hank,4)
```

(1,Scarf,Ali,0)

(1,Scarf,Eve,3)

(1,Scarf,Hank,2)

Disambiguate:

```
A = LOAD '/home/gopal/pig_inputs/join_A' AS (id:int, name:chararray);
dump A;
(2,Tie)
(4,Coat)
(3,Hat)
(1,Scarf)

B = LOAD '/home/gopal/pig_inputs/join_B' AS (name:chararray, id:int);
dump B;
(Joe,2)
(Hank,4)
(Ali,0)
(Eve,3)
(Hank,2)

C = JOIN A by id, B by id;
dump C;
(2,Tie,Joe,2)
(2,Tie,Hank,2)
(3,Hat,Eve,3)
(4,Coat,Hank,4)

describe C;
C: {A::id: int,A::name: chararray,B::name: chararray,B::id: int}


D = FOREACH C GENERATE A::name.
D = FOREACH C GENERATE $1;
dump D;
```

(Tie)

(Tie)

(Hat)

(Coat)

Different effects of FLATTEN:

```
C = LOAD '/home/gopal/pig_inputs/types_C' AS (f0:chararray, f1:chararray);
dump C
(a,pomegranate)
(b,apple)
D = FOREACH C GENERATE TOTUPLE(TOTUPLE(f0 , TOTUPLE(f1));
DUMP D
(((a),(pomegranate)))
(((b),(apple)))
F = FOREACH D GENERATE FLATTEN($0);
DUMP F
((a),(pomegranate))
((b),(apple))

F1 = FOREACH F GENERATE FLATTEN($0);
dump F1
(a)
(b)

F1 = FOREACH F GENERATE FLATTEN($0), FLATTEN($1);
dump F1
(a,pomegranate)
(b,apple)
B = FOREACH C GENERATE TOBAG(f0, f1);
DUMP B
({(a),(pomegranate)})
({(b),(apple)})
```

```
F = FOREACH B GENERATE FLATTEN($0);
DUMP F
(a)
(pomegranate)
(b)
(apple)

B = FOREACH C GENERATE f0, TOBAG(f1, f1);
DUMP B
(a,{(pomegranate),(pomegranate)})
(b,{(apple),(apple)})

F = FOREACH B GENERATE $0, FLATTEN($1);
DUMP F
(a,pomegranate)
(a,pomegranate)
(b,apple)
(b,apple)

H = FOREACH C GENERATE TOMAP(f0, f1);
DUMP H;
([a#pomegranate])
([b#apple])
C = LOAD '/home/gopal/pig_inputs/types_D' AS (f0:chararray, f1:chararray,
f2:chararray,f3:chararray);
dump C;
(a,pomegranate,raj,venkat)
(b,apple,hello,ravi)
(c,orange,hi,appu)

B = FOREACH C GENERATE f0, TOBAG(f1, f1), TOBAG(f2, f3);
DUMP B
(a,{(pomegranate),(pomegranate)},{(raj),(venkat)})
(b,{(apple),(apple)},{(hello),(ravi)})
```

(c ,{(orange),(orange)},{(hi),(appu)})

```
F = FOREACH B GENERATE $0, FLATTEN($1), FLATTEN($2);
dump F;
(a,pomegranate,raj)
(a,pomegranate,venkat)
(a,pomegranate,raj)
(a,pomegranate,venkat)
(b,apple,hello)
(b,apple,ravi)
(b,apple,hello)
(b,apple,ravi)
(c ,orange,hi)
(c ,orange,appu)
(c ,orange,hi)
(c ,orange,appu)
```

Max Temparature:

```
records = LOAD '/home/gopal/pig_inputs/max_temparature_comma' using PigStorage(',') AS (year:chararray, temperature:int, quality:int);
DUMP records;
(1950,0,1)
(1950,22,1)
(1950,-11,1)
(1949,111,1)
(1949,78,1)
(1949,9999,-1)
(1950,1,-11)
records = LOAD '/home/gopal/pig_inputs/max_temparature' AS (year:chararray, temperature:int, quality:int);
DUMP records;
(1950,0,1)
(1950,22,1)
(1950,-11,1)
(1949,111,1)
```

```
(1949,78,1)
(1949,9999,-1)
(1950,1,-11)

DESCRIBE records;
records: {year: chararray,temperature: int,quality: int}

filtered_records = FILTER records BY temperature != 9999 AND (quality == 0
OR quality == 1 OR quality == 4 OR quality == 5 OR quality == 9);
DUMP filtered_records;
(1950,0,1)
(1950,22,1)
(1950,-11,1)
(1949,111,1)
(1949,78,1)

grouped_records = GROUP filtered_records BY year;
DUMP grouped_records;
(1949,{(1949,111,1),(1949,78,1)})
(1950,{(1950,0,1),(1950,22,1),(1950,-11,1)})

DESCRIBE grouped_records;
grouped_records: {group: chararray,filtered_records: {(year:
chararray,temperature: int,quality: int)}}

max_temp = FOREACH grouped_records GENERATE group,
MAX(filtered_records.temperature);
DUMP max_temp;
(1949,111)
(1950,22)
gopal@ubuntu$ hadoop fs -ls /
grunt> exec /home/gopal/pig_inputs/max_temp.pig;
gopal@ubuntu$ pig -x local /home/gopal/pig_inputs/max_temp.pig;
grunt> exec -param_file /home/gopal/pig_inputs/max_temp.properties
```

/home/gopal/pig_inputs/max_temp_properties.pig;

gopal@ubuntu$ pig -x local -param_file /home/gopal/pig_inputs/max_temp.properties /home/gopal/pig_inputs/max_temp_properties.pig;

Missing Fields:

```
A = LOAD '/home/gopal/pig_inputs/missing_fields';
DUMP A;
(2,Tie)
(4,Coat)
(3)
(1,Scarf)

B = FILTER A BY SIZE(TOTUPLE(*)) > 1;
DUMP B;
(2,Tie)
(4,Coat)
(1,Scarf)
```

Multiquery:

```
A = LOAD '/home/gopal/pig_inputs/multiquery_A';
B = FILTER A BY $1 == 'banana';
C = FILTER A BY $1 != 'banana';
STORE B INTO '/home/gopal/pig_inputs/output_b';
STORE C INTO '/home/gopal/pig_inputs/output_c';
```

make a store.pig file with above 5 lines

```
$ pig -x local '/home/gopal/pig_inputs/store.pig';
```

Null Values:

```
records = LOAD '/home/gopal/pig_inputs/sample_corrupt.txt'AS (year:chararray, temperature:int, quality:int);
DUMP records;
(1950,0,1)
(1950,22,1)
(1950,-1,1)
```

```
(1949,111,1)
(1949,78,1)

corrupt_records = FILTER records BY temperature is null;
DUMP corrupt_records;
(1950,,1)

grouped = GROUP corrupt_records ALL;
all_grouped = FOREACH grouped GENERATE group,
COUNT(corrupt_records);
DUMP all_grouped;
(all,1)

good_records = FILTER records BY temperature is not null;
bad_records = FILTER records BY temperature is null;
or
SPLIT records INTO good_records IF temperature is not null, bad_records IF
temperature is null;
DUMP good_records;
(1950,0,1)
(1950,22,1)
(1949,111,1)
(1949,78,1)
DUMP bad_records;
(1950,,1)
records = LOAD '/home/gopal/pig_inputs/sample_corrupt.txt' AS
(year:chararray, temperature, quality:int);
DUMP records;
(1950,0,1)
(1950,22,1)
(1950,e,1)
(1949,111,1)
(1949,78,1)
filtered_records = FILTER records BY temperature != 9999 AND (quality == 0
```

```
OR quality == 1 OR quality == 4 OR quality == 5 OR quality == 9);
grouped_records = GROUP filtered_records BY year;
max_temp = FOREACH grouped_records GENERATE group,
MAX(filtered_records.temperature);
DUMP max_temp;
(1949,111.0)
(1950,22.0)
```

Stream:

```
A = LOAD '/home/gopal/pig_inputs/foreach_A' AS (f0:chararray, f1:chararray,
f2:int);
dump A;
(Joe,cherry,2)
(Ali,apple,3)
(Joe,banana,2)
(Eve,apple,7)
C = STREAM A THROUGH `cut -f 2`;
DUMP C;
(cherry)
(apple)
(banana)
(apple)
C = STREAM A THROUGH `cut -f 2 | grep apple`;
DUMP C;
(apple)
(apple)
```

Tuples:

```
A = LOAD '/home/gopal/pig_inputs/tuples_A' AS (f0, t0:tuple(f1:int,
f2:chararray, t1:tuple(f3:int, f4:chararray)));
A = LOAD '/home/gopal/pig_inputs/tuples_A' AS (f0, t0:(f1:int, f2:chararray,
t1:(f3:int, f4:chararray)));
DUMP A;
(1,(1,'pomegranate',(2,'apple')))
(2,(3,'banana',(4,lychee)))
```

Types:

```
A = LOAD '/home/gopal/pig_inputs/types_A' AS (t0:tuple(f0:int, f2:chararray));
DUMP A;
((1,pomegranate))
DESCRIBE A;
A: {t0: (f0: int,f2: chararray)}
one = LOAD '/home/gopal/pig_inputs/types_one';
B = FOREACH one GENERATE (chararray) 'pomegranate' AS f0;
B = FOREACH one GENERATE (1,'pomegranate') AS t0:tuple(f0:int,
f2:chararray);
DUMP B;
((1,pomegranate))
DESCRIBE B;
B: {t0: (f0: int,f2: chararray)}
C = FOREACH one GENERATE ['a'#'pomegranate'] AS t0:map[];
DUMP C;
([a#pomegranate])
DESCRIBE C;
C: {t0: map[]}
C = LOAD '/home/gopal/pig_inputs/types_C' AS (f0:chararray, f1:chararray);
D = FOREACH C GENERATE TOTUPLE(f0, f1);
DUMP D;
((a,pomegranate))
((b,apple))

E = FOREACH C GENERATE TOBAG(f0, f1);
DUMP E;
({(a),(pomegranate)})
({(b),(apple)})

F = FOREACH C GENERATE TOMAP(f0, f1);
DUMP F;
([a#pomegranate])
([b#apple])
```

HIVE

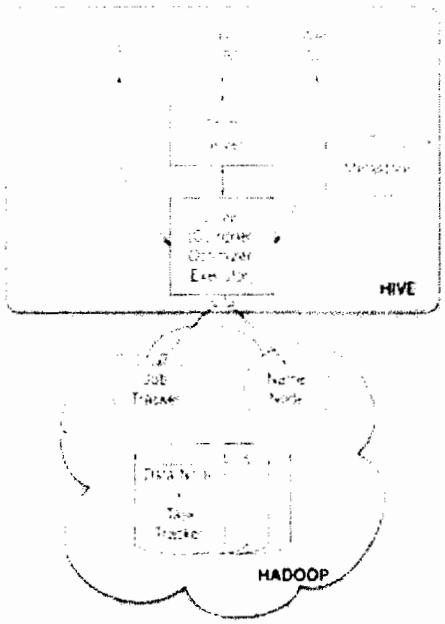
4.1.Hive

Hive is a data warehousing infrastructure based on Apache Hadoop. Hadoop provides massive scale out and fault tolerance capabilities for data storage and processing on commodity hardware.

Hive is designed to enable easy data summarization, ad-hoc querying and analysis of large volumes of data. It provides SQL which enables users to do ad-hoc querying. summarization and data analysis easily. At the same time, Hive's SQL gives users multiple places to integrate their own functionality to do custom analysis, such as User Defined Functions (UDFs).

Hive is not designed for online transaction processing. It is best used for traditional data warehousing tasks.

4.2.Hive Architecture



The diagram represents CLI (Command Line Interface), JDBC/ODBC and Web GUI (Web Graphical User Interface).This represents when user comes with CLI(Hive Terminal) it directly connected to Hive Drivers, When User comes with JDBC/ODBC(JDBC Program) at that time by using API(Thrift Server) it connected to Hive driver and when the user comes with Web GUI(Ambari server) it directly connected to Hive Driver.

The hive driver receives the tasks (Queries) from user and send to Hadoop architecture. The Hadoop architecture uses name node, data node, job tracker and task tracker for receiving and dividing the work what Hive sends to Hadoop.

4.3.Functioning of Driver,Compiler and Executor

Driver

The Driver is used for receives the quires from UI .This component implements the notion of session handles and provides execute and fetch APIs modeled on JDBC/ODBC interfaces. Driver reads the configuration file hive-site.xml.

Compiler

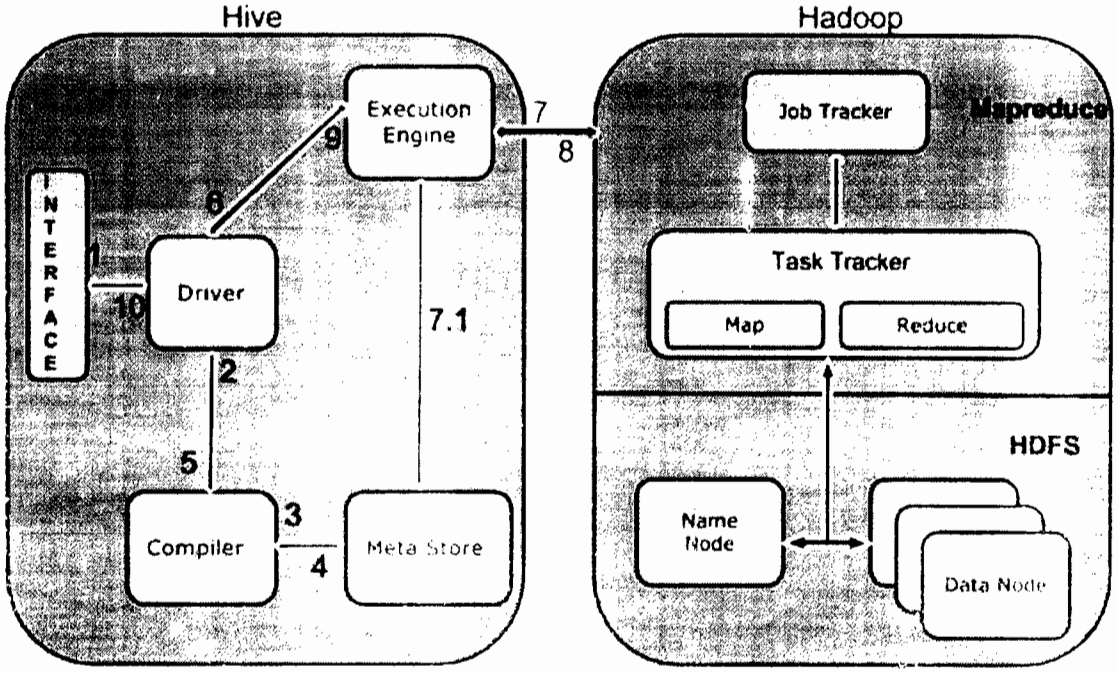
The component that parses the query, does semantic analysis on the different query blocks and query expressions and eventually generates an execution plan with the help of the table and partition metadata looked up from the metastore.

Executor

The component which executes the execution plan created by the compiler. The plan is a DAG of stages. The executor manages the dependencies between these different stages of the plan and executes these stages on the appropriate system components.

4.4.Working of Hive

The following diagram depicts the workflow between Hive and Hadoop.



Step No.	Operation
1	Execute Query The Hive interface such as Command Line or Web UI sends query to Driver (any database driver such as JDBC, ODBC, etc.) to execute.
2	Get Plan The driver takes the help of query compiler that parses the query to check the syntax and query plan or the requirement of query.
3	Get Metadata The compiler sends metadata request to Metastore (any database).
4	Send Metadata Metastore sends metadata as a response to the compiler.
5	Send Plan The compiler checks the requirement and resends the plan to the driver. Up to here, the parsing and compiling of a query is complete.
6	Execute Plan The driver sends the execute plan to the execution engine.
7	Execute Job Internally, the process of execution job is a MapReduce job. The execution engine sends the job to JobTracker, which is in Name node and it assigns this job to TaskTracker, which is in Data node. Here, the query executes MapReduce job.
7.1	Metadata Ops Meanwhile in execution, the execution engine can execute metadata operations with Metastore.
8	Fetch Result The execution engine receives the results from Data nodes.

9 Send Results
The execution engine sends those resultant values to the driver.

10 Send Results
The driver sends the results to Hive Interfaces.

4.5.Meta Store in Hive

The component that stores all the structure information of the various tables and partitions in the warehouse including column and column type information, the serializers and deserializers necessary to read and write data and the corresponding HDFS files where the data is stored. It is the central repository of Hive metadata.

Metadata is the internal database of hive, which is responsible for managing metadata information of the tables.

4.5.1.The metadata which metastore stores contains things like

a) Ids of database

b) Ids of Tables and index

c) Time of creation of index

d) Time of creation of tables

e) Ids of roles assigned to a particular user

f) Input format used for tables

g) Output format used for tables etc.

4.5.2.There are three modes of metastore

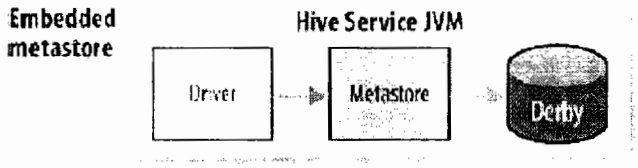
1. Embedded Metastore
2. Local Metastore
3. Remote Metastore

4.5.2.1.Embedded Metastore

The metastore is divided into two pieces: a service and the backing store for the data.

By default, the metastore service runs in the same JVM as the Hive service and contains an embedded Derby database instance backed by the local disk. This is called the embedded metastore.

Using an embedded metastore is a simple way to get started with Hive; however, only one embedded Derby database can access the database files on disk at any one time, which means you can only have one Hive session open at a time that shares the same metastore . Trying to start a second session gives the error "Failed to start database metastore_db".



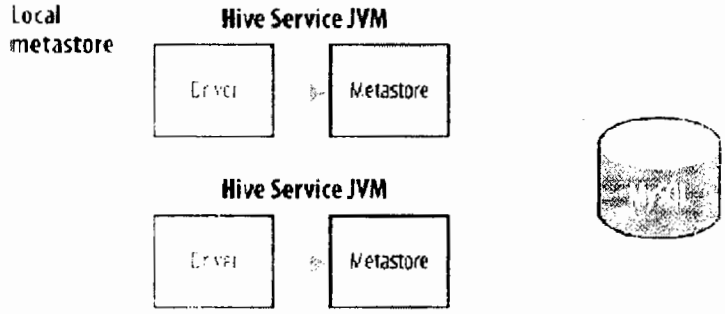
Default configuration for Embedded Metastore

Below are the metastore configuration detail present in "hive-site.xml".

```
<property>
  <name>javax.jdo.option.ConnectionURL</name>
  <value>jdbc:derby:;databaseName=/var/lib/hive/metastore/metastore_db;create=true</value>
  <description>JDBC connect string for a JDBC metastore</description>
</property>

<property>
  <name>javax.jdo.option.ConnectionDriverName</name>
  <value>org.apache.derby.jdbc.EmbeddedDriver</value>
  <description>Driver class name for a JDBC metastore</description>
</property>
```

4.5.2.2.Local Metastore

The solution to supporting multiple sessions (and therefore multiple users) is to use a standalone database. This configuration is referred to as a local metastore, since the metastore service still runs in the same process as the Hive service, but connects to a database running in a separate process, either on the same machine or on a remote machine.



Any JDBC-compliant database may be used by setting the javax.jdo.option.* configuration properties.

If we chose MySQL database as the standalone metastore,then we need to set below property in "hive-site.xml".

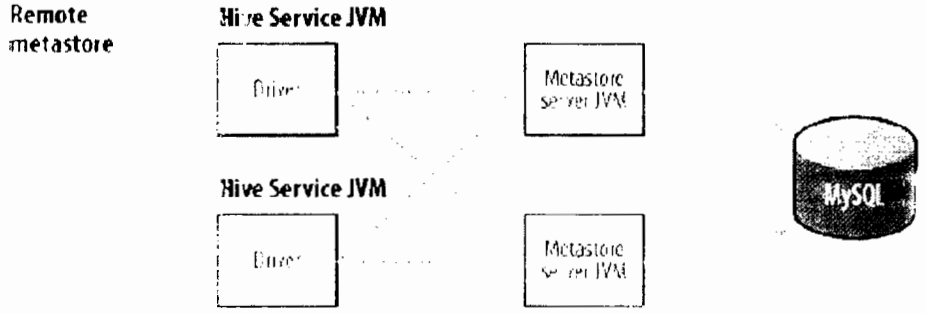
```
<property>
  <name>javax.jdo.option.ConnectionURL</name>
  <value>jdbc:mysql://host/dbname?createDatabaseIfNotExist=true</value>
  <description>JDBC connect string for a JDBC metastore</description>
</property>

<property>
  <name>javax.jdo.option.ConnectionDriverName</name>
  <value>com.mysql.jdbc.Driver</value>
  <description>Driver class name for a JDBC metastore</description>
</property>
```

Note- The JDBC driver JAR file for MySQL must be on Hive's class path , which is simply achieved by placing it in Hive's lib directory.

4.5.2.3.Remote Metastore

In a remote metastore, where one or more metastore servers run in separate processes to the Hive service.

This brings better manageability and security, since the database tier can be completely firewalled off, and the clients no longer need the database credentials.



4.6.Important metastore configuration properties

Property Name	Type	Default value	Description

hive.metastore.warehouse.dir	URI	/user/hive/ warehouse	The directory relative to fs.default.name where managed tables are stored.
hive.metastore. local	boolean	True	Whether to use an embedded metastore server (true), or connect to a remote instance (false). If false, then hive.metastore.uris must be set.
javax.jdo.option. ConnectionURL	URI	jdbc:derby:;database Name=metastore_db; create=true	The JDBC URL of the metastore database
javax.jdo.option. ConnectionDriverName	String	org.apache.derby. jdbc.EmbeddedDriver	. The JDBC driver classname.
javax.jdo.option. ConnectionUserName	String	APP	The JDBC user name.
javax.jdo.option. ConnectionPassword	String	Mine	The JDBC password.

4.7.Differences between RDBMS and Hive

RDBMS	HIVE
It is a fully database	It is a datawarehouse component of hadoop.
It enforces "Schema on write".	It enforces "Schema on read".
It is designed for "Read and Write many".	It is based on "write once and read many".
In RDBMS. record level updates,insertions,deletes,transactions and indexes are possible.	Whereas these are not allowed in Hive because hive is built to operate over HDFS data.
It support OLTP processing.	It doesnot support OLTP processing due to high latency.whereas it best suits for OLAP.
RDBMS is best suited for dynamic data analysis.	Hive is suited for datawarehouse applications, where relatively static data is analyzed.
It can handle data in GBs.	Whereas it can handle data in PBs &

	more

4.7.1.Schema on Write vs Schema on Read

Schema on Write	Schema on Read
In a traditional database, a table's schema is enforced at data load time. If the data being loaded doesn't conform to the schema, then it is rejected. This design is called schema on write, since the data is checked against the schema when it is written into the database.	Hive,doesn't verify the data when it is loaded, but rather when a query is issued. This is called schema on read.At the time of query execution to pull the data If loaded data doesnot conform the schema of table,then it is rejected. It is called Schema on Read,since the data is checked against the schema when it is being read from the hive table.
Schema on write makes query time performance faster, since the database can index columns and perform compression on the data.	Schema on read makes for a very fast initial load, since the data does not have to be read, parsed, and serialized to disk in the database's internal format. The load operation is just a file copy or move.

4.8.Hive Tables

A Hive table is logically made up of the data being stored and the associated metadata describing the layout of the data in the table. The data typically resides in HDFS.

There are two types of Hive table.

i> Managed Table
ii> External Table.

4.8.1.Managed Table

When you create a table in Hive, by default Hive will manage the data, which means that Hive moves the data into its warehouse directory.

These are the default tables of hive, and all the managed tabular data is stored in one of the default warehouse location of HDFS called "/user/hive/warehouse/<<TableName>>/<<TableData>>".

Syntax to create managed table in hive

```
CREATE TABLE <tableName> (schema defination)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY <delimiter>
LINES TERMINATED BY <delimiter>
STORED AS <file format>;
```

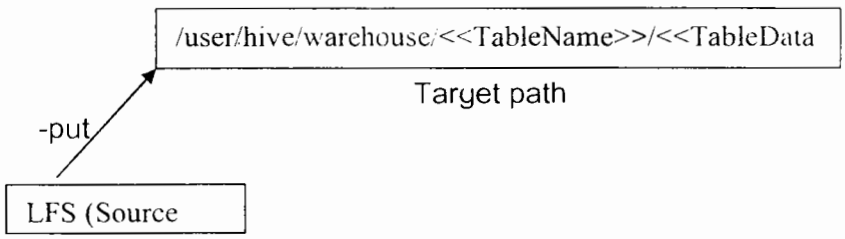
Loading of data in managed table

There are two ways to load the data in managed table:

i> Local mode
ii> HDFS mode

Local mode

In this mode, hive will accepts the input data from one of the local filesystem path and it loads the data in hive warehouse directory.

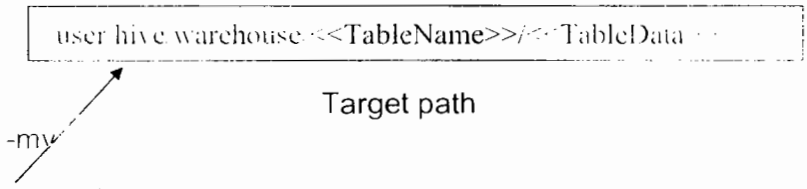


Syntax

```
LOAD DATA LOCAL INPATH '<LocalPath>' INTO TABLE <tablename>;
```

HDFS mode

In this mode, hive will accepts the input data from one of the HDFS path and once the data get loaded it will move the data to hive warehouse directory.



HDFS

Source path

Syntax

LOAD DATA INPATH '<HDFS Abs Path>' INTO TABLE <TableName>;

4.8.2.External Table

These tables are external to the hive warehouse path. i.e- external table data will be stored in an external location of HDFS which we specify at the table schema defination time using "location " keyword.

Syntax to create external table in hive

CREATE EXTERNAL TABLE <tableName> (schema defination)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY <delimiter>
LINES TERMINATED BY <delimiter>
STORED AS <file format> LOCATION 'HDFS Path' ;

Note: External table is loaded with data same as the managed table.

4.9.Hive Services

The Hive shell is only one of several services that we can run using the hive command. We can specify the service to run using the --service option. Type hive --service help to get a list of available service names.

```
root@ubuntu:/home/mrinmoy# hive --service help;
Usage ./hive <parameters> --service serviceName <service parameters>
Service List: cli help hiveserver jar lineage metastore rcfilecat
Parameters parsed:
  --auxpath : Auxillary jars
  --config : Hive configuration directory
  --service : Starts specific service/component. cli is default
Parameters used:
  HADOOP_HOME : Hadoop install directory
  HIVE_OPT : Hive options
For help on a particular service:
  ./hive --service serviceName --help
```

For help on a particular service, we need to use below command as:

hive --service <<serviceName>> --help;

```
root@ubuntu:/home/mrinmoy# hive --service jar --help;
Used for applications that require Hadoop and Hive classpath and environment.
./hive --service jar <yourjar> <yourclass> HIVE_OPTS <your_args>
root@ubuntu:/home/mrinmoy#
```

The most useful are described below

Cli
The command line interface to Hive (the shell). This is the default service

```
root@ubuntu:/home/mrinmoy# hive --service cli --help;
usage: hive
 -e <quoted-query-string>         SQL from command line
 -f <filename>                    SQL from files
 -H,--help                        Print help information
 -h <hostname>                    connecting to Hive Server on remote host
    --hiveconf <property=value>   Use value for given property
 -i <filename>                    Initialization SQL file
 -p <port>                        connecting to Hive Server on port number
 -S,--silent                      Silent mode in interactive shell
 -v,--verbose                     Verbose mode (echo executed SQL to the
                                  console)
```

4.10.Hiveserver

Runs Hive as a server exposing a Thrift service, enabling access from a range of clients written in different languages. Applications using the Thrift, JDBC, and ODBC connectors need to run a Hive server to communicate with Hive. Set the HIVE_PORT environment variable to specify the port the server will listen on (defaults to 10,000).

```
root@ubuntu:/home/mrinmoy# hive --service hiveserver --help;
usage HIVE_PORT=xxxx ./hive --service hiveserver
  HIVE_PORT : Specify the server port
root@ubuntu:/home/mrinmoy#
```

4.11.Data Types in hive

Below are the list of primitive data types and their descriptions in hive.

Type	Description	Literal examples
TINYINT	1-byte (8-bit) signed integer, from -128 to 127	1
SMALLINT	2-byte (16-bit) signed integer, from -32,768 to 32,767	1
INT	4-byte (32-bit) signed integer, from -2,147,483,648 to 2,147,483,647	1
BIGINT	8-byte (64-bit) signed integer, from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	1
FLOAT	4-byte (32-bit) single-precision floating-point number	1.0
DOUBLE	8-byte (64-bit) double-precision floating-point number	1.0
BOOLEAN	true/false value	TRUE
STRING	Character string	'a', "a"
BINARY	Byte array	Not supported
TIMESTAMP	Timestamp with nanosecond precision	1325502245000, '2012-01-02 03:04:05.123456789'

Below are the complex datatypes and their descriptions in hive:

Type	Description	Literal examples
ARRAY	An ordered collection of fields. The fields must all be of the same type.	array(1, 2)[a]
MAP	An unordered collection of key-value pairs. Keys must be primitives; values may be any type. For a particular map, the keys must be the same type, and the values must be the same type	map('a', 1, 'b', 2)
STRUCT	A collection of named fields. The fields may be of different types.	struct('a', 1, 1.0)[b]

To enter in hive shell

```
gopal@ubuntu:~$ hive

Logging initialized using configuration in jar:file:/usr/local/hadoop-1.2.1/apa
rties
hive>
```

To quit the hive shell

```
hive> quit;
gopal@ubuntu:~$
```

```
hive> exit;
gopal@ubuntu:~$
```

To Show Databases in hive

```
hive> show databases;
OK
default
hive73
Time taken: 0.908 seconds, Fetched: 2 row(s)
hive>
```

To create database

```
hive> create database batch73;
OK
Time taken: 0.263 seconds
hive>
```

To use a particular database

```
hive> use batch73;
OK
Time taken: 0.036 seconds
hive>
```

To create managed table

```
hive> create table mantab73 (cid int, cname string, cheadcnt int, cloc string)
    > row format delimited
    > fields terminated by '|'
    > lines terminated by '\n'
    > stored as textfile;
OK
Time taken: 0.672 seconds
```

```
hive> create table if not exists mantab73 (cid int, cname string, cheadcnt int, cloc int )
    > row format delimited
    > fields terminated by '|'
    > lines terminated by '\n'
    > stored as textfile;
OK
Time taken: 0.057 seconds
```

Note:

1. "If not exists" clause is used to create the table if the table donot exist.
2. "lines terminated by " clause is optional to tell how the row would be terminated
3. "row format delimited" tells that how fields of any row would be terminated.
4. "stored is textfile" tells that the table would be stored as text file. Below are the available file format which can be used in place of textfield:

"textfile","sequence file", "rcfile","orc file","parquet file".

To get the schema information of a table

```
hive> desc mantab73;
OK
cid                     int
cname                   string
cheadcnt                int
cloc                    string
Time taken: 0.254 seconds, Fetched: 4 row(s)
hive>
```

To get the detailed information about table

```
hive> desc extended mantab73;
OK
cid                     int
cname                   string
cheadcnt                int
cloc                    string

Detailed Table Information      Table(tableNa:mantab73, dbName:batch73, owner:gopal, createTime:1482259609, lastAccessTime:0, retention:0
:StorageDescriptor(cols:[FieldSchema(name:cid, type:int, comment:null), FieldSchema(name:cname, type:string, comment:null), FieldSchema(nam
eadcnt, type:int, comment:null), FieldSchema(name:cloc, type:string, comment:null)], location:hdfs://localhost:54310/user/hive/warehouse/ba
3.db/mantab73, inputFormat:org.apache.hadoop.mapred.TextInputFormat, outputFormat:org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputForma
ompressed:false, numBuckets:-1, serdeInfo:SerDeInfo(name:null, serializationLib:org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe, paramet
{serialization.format=|, line.delim=
, field.delim=|}), bucketCols:[], sortCols:[], parameters:{}, skewedInfo:SkewedInfo(skewedColNames:[], skewedColValues:[], skewedColValueLo
onMaps:{}), storedAsSubDirectories:false), partitionKeys:[], parameters:{transient_lastDdlTime=1482259609}, viewOriginalText:null, viewExpa
Text:null, tableType:MANAGED_TABLE)
Time taken: 0.15 seconds, Fetched: 7 row(s)
hive>
```

To load the data in the managed table (Local mode)

```
hive> load data local inpath '/home/gopal/emp' into table mantab73;
Copying data from file:/home/gopal/emp
Copying file: file:/home/gopal/emp
Loading data to table batch73.mantab73
Table batch73.mantab73 stats: [numFiles=1, numRows=0, totalSize=193, rawDataSize=0]
OK
Time taken: 2.979 seconds
```

To see the content of a table

```
hive> select * from mantab73;
OK
100     Infosys 32000   Hyderabad
101     TCS     32410   Bangalore
102     IBM     456210  Hyderabad
103     IBM     45645   Bangalore
104     IBM     456210  Pune
105     IBM     456210  Delhi
106     Attra   78542   Bangalore
107     HCL     75842   Hyderabad
NULL    NULL    NULL    NULL
Time taken: 2.251 seconds, Fetched: 9 row(s)
hive>
```

Note: High latency is the known problem in hive as it doesnot maintain indexes.

To load data in managed table (HDFS mode)

```
hive> load data inpath '/emp' into table mantab73;
Loading data to table batch73.mantab73
Table batch73.mantab73 stats: [numFiles=2, numRows=0, totalSize=386, rawDataSize=0]
OK
Time taken: 0.423 seconds
hive>
```

In HDFS mode, we load the file from HDFS location(here, /emp) into the table.

To drop managed table

```
hive> drop table mantab73;
OK
Time taken: 2.075 seconds
hive> select * from mantab73;
FAILED: SemanticException [Error 10001]: Line 1:14 Table not found 'mantab73'
hive>
```

When we drop managed table its data as well as as metadata gets deleted.
After dropping the table,if we fetch the data from the table then it
"Table not found" exception.

To create external table

```
hive> create external table if not exists extntab73 (id int, name string, sal int)
    > row format delimited
    > fields terminated by '\t'
    > lines terminated by '\n'
    > stored as textfile location '/extn';
OK
Time taken: 0.186 seconds
```

To load data in external table (Local mode)

```
hive> load data local inpath '/home/gopal/emp1y' into table extntab73;
Copying data from file:/home/gopal/emp1y
Copying file: file:/home/gopal/emp1y
Loading data to table batch73.extntab73
Table batch73.extntab73 stats: [numFiles=0, numRows=0, totalSize=0, rawDataSize=0]
OK
Time taken: 0.835 seconds
hive>
```

To drop external table

```
hive> drop table extntab73;
OK
Time taken: 0.86 seconds
hive> select * from extntab73;
FAILED: SemanticException [Error 10001]: Line 1:14 Table not found 'extntab73'
hive>
```

As we can see, the tabular data persist on the HDFS location after the deletion of external table.

```
hive> !hadoop fs -cat /extn/emp1y ;
100     Mudassar        100000
101     Ravi    40000
102     Junaid  45600
103     Dibya   145620
104     Ahmad   156420
105     Pandey  58463
hive>
```

4.13.Difference between Managed table and External Table

Managed Table	External Table
When a managed table get dropped off then tabular data as well as meta data (from the metastore) get deleted.	When texternal table get dropped off then metadata (from metastore) get deleted but actual data persist in the external HDFS location.
These are the default table of hive,which will be managed by hive warehouse.	These tables are external to the hive warehouse system.

Managed tabular data will be stored on default hive warehouse location.	External table data will be stored in an external location of HDFS other than warehouse generally.
No need to use any extra keywords for table creation.	We have to use "external" and "location" keyword in the table creation.

4.14.ALTER TABLE IN HIVE

Alter Table is used to alter the attributes of a table such as changing its table name, changing column names, adding columns, and deleting or replacing columns.

Syntax

The statement takes any of the following syntaxes based on what attributes we wish to modify in a table.

ALTER TABLE name RENAME TO new_name
ALTER TABLE name ADD COLUMNS (col_spec[, col_spec ...])
ALTER TABLE name DROP [COLUMN] column_name
ALTER TABLE name CHANGE column_name new_name new_type
ALTER TABLE name REPLACE COLUMNS (col_spec[, col_spec ...])

Rename To... Statement

The following query renames the table from dept to Department.

```
hive> alter table dept RENAME TO Department;
OK
Time taken: 0.276 seconds
hive>
```

Change Statement

Change statement is used to change the name or type (or both) of a particular column in a table.

Before using change statement

```
hive> desc Department;
OK
deptid                  int
deptname                string
dstrength               int
dloc                    string
Time taken: 0.167 seconds, Fetched: 4 row(s)
hive>
```

Uses of change statement

```
hive> alter table Department CHANGE dstrength dheadcnt int;
OK
Time taken: 0.193 seconds
hive>
```

After using change statement

```
hive> desc Department;
OK
deptid                  int
deptname                string
dheadcnt                int
dloc                    string
Time taken: 0.092 seconds, Fetched: 4 row(s)
hive>
```

Here.dstrength is changed to dheadcnt.

Add Columns Statement

The following query adds a column named dperformance to the Department table.

```
hive> alter table Department ADD COLUMNS (dperformance string);
OK
Time taken: 0.134 seconds
hive> desc Department;
OK
```

After adding column dperformance in Department

```
hive> desc Department;
OK
deptid                  int
deptname                string
dheadcnt                int
dloc                    string
dperformance            string
Time taken: 0.101 seconds, Fetched: 5 row(s)
hive>
```

Replace Statement

The following query deletes all the columns from the Department table and replaces it with empid and ename columns:

```
hive> alter table Department REPLACE COLUMNS (empid int, ename string);
OK
Time taken: 0.568 seconds
hive> desc Department;
OK
```

After Replacing the column

```
hive> desc Department;
OK
empid                   int
ename                   string
Time taken: 0.253 seconds, Fetched: 2 row(s)
hive>
```

4.15.CREATE TABLE...AS SELECT

It's often very convenient to store the output of a Hive query in a new table, perhaps because it is too large to be dumped to the console or because there are further processing steps to carry out on the result.

The new table's column definitions are derived from the columns retrieved by the SELECT clause. In the following query, the target table has two columns named col1 and col2 whose types are the same as the ones in the source table:

CREATE TABLE target AS SELECT col1, col2 FROM source;

A CTAS operation is atomic, so if the SELECT query fails for some reason, then the table is not created.

```
hive> create table Section
    > as
    > select empid,ename from Department;
```

Below screen shot shows that the table "Section" has been created.

```
hive> show tables;
OK
department
section
Time taken: 0.034 seconds, Fetched: 2 row(s)
hive>
```

Below are the description of "Section " table that are created through CTAS.

```
hive> desc section;
OK
empid                   int
ename                   string
Time taken: 0.239 seconds, Fetched: 2 row(s)
hive>
```

4.16.JOINS

To fetch the data from more than one table, we perform join operation on the tables.

4.16.1.Inner joins

The simplest kind of join is the inner join, where each match in the input tables results in a row in the output. Consider two small demonstration tables: sales, which lists thenames of people and the ID of the item they bought; and things, which lists the item ID and its name:

hive> SELECT * FROM sales;

```
Joe     2
Hank    4
Ali     0
Eve     3
Hank    2
```

hive> SELECT * FROM things;

```
2    Tie
4    Coat
3    Hat
1    Scarf
```

We can perform an inner join on the two tables as follows:

hive> SELECT sales.*, things.* FROM sales JOIN things ON (sales.id = things.id);

```
Joe   2  2  Tie
Hank  2  2  Tie
Eve   3  3  Hat
Hank  4  4  Coat
```

The table in the FROM clause (sales) is joined with the table in the JOIN clause (things), using the predicate in the ON clause. Hive only supports equijoins, which means that only equality can be used in the join predicate, which here matches on the id column

4.16.2.Left Outer joins

Outer joins allow us to find non-matches in the tables being joined. If we change the join type to LEFT OUTER JOIN, then the query will return a row for every row in the left table (sales), even if there is no corresponding row in the table it is being joined to (things):

SELECT sales.*, things.* FROM sales LEFT OUTER JOIN things ON (sales.id = things.id);

```
Ali     0   NULL   NULL
Joe     2   2      Tie
Hank    2   2      Tie
Eve     3   3      Hat
Hank    4   4      Coat
```

4.16.3.Right Outer Join

Hive supports right outer joins, which reverses the roles of the tables relative to the left join. In this case, all items from the things table are included, even those that weren't purchased by anyone (a scarf):

hive > SELECT sales.*, things.* FROM sales RIGHT OUTER JOIN things ON (sales.id = things.id);

```
NULL    NULL   1   Scarf
Joe     2      2   Tie
Hank    2      2   Tie
Eve     3      3   Hat
Hank    4      4   Coat
```

4.16.4.Full Outer Join

Finally, there is a full outer join, where the output has a row for each row from both tables in the join:

hive > SELECT sales.*, things.* FROM sales FULL OUTER JOIN things ON (sales.id = things.id);

```
Ali     0      NULL   NULL
NULL    NULL   1      Scarf
Joe     2      2      Tie
Hank    2      2      Tie
Eve     3      3      Hat
Hank    4      4      Coat
```

4.17.PARTITION

Hive organizes tables into partitions. It is a way of dividing a table into related parts based on the values of partitioned columns such as date, city, and department. Using partition, it is easy to query a portion of the data.

There are mainly two types of partitioning in hive.

1. Static Partitioning.
2. Dynamic partitioning.

4.17.1.Static Partitioning

Creation of partitioned table:

```
hive> create table sbatab73 (bankid string, bankrepname string,banktover string)
    > PARTITIONED BY (country string)
    > row format delimited
    > fields terminated by '|'
    > stored as textfile;
```

Sbatab73 is the partitioned table partitioned on 'country'.

LOADING OF DATA in PARTITIONED TABLE

```
hive> load data local inpath '/home/gopal/Desktop/SBA_NG_Data.log' into table sbatab73
    > PARTITION (country = 'NIGERIA');
```

```
hive> load data local inpath '/home/gopal/Desktop/SBA_NM_Data.log' into table sbatab73
    > PARTITION (country = 'NAMABIA');
```

```
hive> load data local inpath '/home/gopal/Desktop/SBA_UG_Data.log' into table sbatab73
    > PARTITION (country = 'UGANDA');
```

Here, we have loaded data in three partitions Nigeria,Namabia and Uganda.

The partitioned table "sbatab73" will be present under the below path as a directory (Folder).

/user/hive/warehouse/batch73.db/sbatab73

Hive will now create subdirectories reflecting the partitioning structure. For example

.../sbatab73/country=NIGERIA
.../sbatab73/country=NAMABIA
.../sbatab73/country=UGANDA

In the above scenario, we were knowing the value of partition column before loading the data in partitioned table, i.e- we were knowing (about the data) before loading the data, that which partition it belongs to.

The above one is the case of "static partitioning".

If we know the value of partition column for the data being loaded in the partition table at the time of loading,then this type of partitioning is known as "static partitioning".

4.17.2. Dynamic partitioning

Creation of partitioned table "orctab73".

```
hive> create table orctab73 (id int,name string,sal int)
    > PARTITIONED BY (city string)
    > row format delimited
    > fields terminated by '\t'
    > stored as textfile;
OK
Time taken: 0.266 seconds
hive>
```

Before inserting the data in the partition table (in case of dynamic partitioning) ,we need to set some parameter.
SET hive.exec.dynamic.partition = true // set to true to enable dynamic partitioning.

SET hive.exec.dynamic.partition.mode = nonstrict // set to nonstrict to enable all partitions to be determined dynamically */

statement.
) and then
n.

```
hive> insert overwrite table orctab73
    > PARTITION (city) select * from stagetab73;
```

Here, we did not give the value of partition column (i.e – value of city) at the time of inserting data.This type of partitioning is called "Dynamic partitioning".

Note:- If we use static and dynamic partitioning both,the ststic partitioning must come first in the insert command.

4.18.Bucketing

Bucketing is the another data sharing mechanism in hive to improve the query performance along with partitioning.

Bucketing helps organize data inside the partition into multiple files so that same set of data will always be written in the same bucket.

In order to divide the data into said no of buckets,we have to use "cluster by" class.

In the way of dividing the data into corresponding buckets, bucketing concept internally making use of hashing algorithm;

Where, key = bucket no.
Value = corresponding data in the bucket.

```
Mandate property to use bucketing concept:
SET  hive.enforce.bucketing = true;
```

Creation of bucketed table:

```
hive> create table buckettab73 (id int,name string,sal int)
    > CLUSTERED BY (id) INTO 4 BUCKETS
    > row format delimited
    > fields terminated by '\t'
    > stored as textfile;
OK
Time taken: 0.364 seconds
hive>
```

Insertion of data from one table to bucketed table.

```
hive> SET hive.enforce.bucketing = true;
hive> insert overwrite table buckettab73
    > select * from sampletab73;
Total jobs = 1
```

Here,we first set the bucketing parameter true and then we inserted the table data into bucketed table.

TABLESAMPLE

The TABLESAMPLE clause allows the users to write queries for samples of the data instead of the whole table. The TABLESAMPLE clause can be added to any table in the FROM clause. The buckets are numbered starting from 1. colname indicates the column on which to sample each row in the table. colname can be one of the non-partition columns in the table or rand() indicating sampling on the entire row instead of an individual column.

Sampling Bucketized Table
table_sample: TABLESAMPLE (BUCKET x OUT OF y [ON colname])
4.19.ORDER BY and SORT BY

The ORDER BY and SORT BY clause are used for sorting operations.

The ORDER BY clause is familiar from other SQL dialects. It performs a total ordering of the query result set. This means that all the data is passed through a single reducer, which may take an unacceptably long time to execute for larger data sets.

Hive adds an alternative, SORT BY, that orders the data only within each reducer, thereby performing a local ordering, where each reducer's output will be sorted. Better performance is traded for total ordering.

In both cases, the syntax differs only by the use of the ORDER or SORT keyword. You can specify any columns you wish and specify whether or not the columns are ascending using the ASC keyword (the default) or descending using the DESC keyword.

Here is an example using ORDER BY:

```
SELECT s.ymd, s.symbol, s.price_close FROM stocks s
ORDER BY s.ymd ASC, s.symbol DESC;
```

Here is the same example using SORT BY instead

```
SELECT s.ymd, s.symbol, s.price_close
FROM stocks s
SORT BY s.ymd ASC, s.symbol DESC;
```

The two queries look almost identical, but if more than one reducer is invoked, the output will be sorted differently. While each reducer's output files will be sorted, the data will probably overlap with the output of other reducers.
Because ORDER BY can result in excessively long run times, Hive will require a LIMIT clause with ORDER BY if the property hive.mapred.mode is set to strict. By default, it is set to nonstrict.

NOTE:

There is single reducer concept in order by clause irrespective of amount of data unlike sort by clause.

4.20.Hive : Sort By Vs Order By Vs Distribute By Vs Cluster By

Sort By

Hive uses the columns in SORT BY to sort the rows before feeding the rows to a reducer. The sort order will be dependent on the column types. If the column is of numeric type, then the sort order is also in numeric order. If the column is of string type, then the sort order will be lexicographical order.It orders data at each of 'N' reducers , but each reducer can have overlapping ranges of data.

Outcome : N or more sorted files with overlapping ranges.

Let's understand with an example :-

SELECT key, value FROM src SORT BY key ASC, value DESC

The query had 2 reducers, and the output of each is

Reducer 1 :

```
0 5
0 3
3 6
9 1
```

Reducer 2 :

```
0 4
0 3
1 1
2 5
```

As, we can see, each reducer output is ordered but total ordering is missing , since we end up with multiple outputs per reducer.

Order By

This is similar to ORDER BY in SQL Language.In Hive, ORDER BY guarantees total ordering of data, but for that it has to be passed on to a single reducer, which is normally

unacceptable and therefore in strict mode, hive makes it compulsory to use LIMIT with ORDER BY so that reducer doesn't get overburdened.

Ordering : Total Ordered data.

Outcome : Single output i.e. fully ordered

For example:

SELECT key, value FROM src ORDER BY key ASC, value DESC

Reducer :

```
0 5
0 4
0 3
0 3
1 1
2 5
3 6
9 1
```

Distribute By

Hive uses the columns in Distribute By to distribute the rows among reducers. All rows with the same Distribute By columns will go to the same reducer.

It ensures each of N reducers gets non-overlapping ranges of column, but doesn't sort the output of each reducer. You end up with N or more unsorted files with non-overlapping ranges.

Example:

We are Distributing By x on the following 5 rows to 2 reducer:

```
x1
x2
x4
x3
x1
```

Reducer 1 got

```
x1
x2
x1
```

Reducer 2 got

```
x4
x3
```

Note that all rows with the same key x1 is guaranteed to be distributed to the same reducer (reducer 1 in this case), but they are not guaranteed to be clustered in adjacent positions.

Cluster By

Cluster By is a short-cut for both Distribute By and Sort By.CLUSTER BY x ensures each of N reducers gets non-overlapping ranges, then sorts by those ranges at the reducers.

Ordering : Global ordering between multiple reducers.

Outcome : N or more sorted files with non-overlapping ranges.

For the same example as above , if we use Cluster By x, the two reducers will further sort rows on x:

Reducer 1 got

```
x1
x1
x2
```

Reducer 2 got

```
x3
x4
```

Instead of specifying Cluster By, the user can specify Distribute By and Sort By, so the partition columns and sort columns can be different.

4.21.Collection Data Type in Hive

1. Array – to store homogeneous data
2. Struct – to store hetrogeneous data
3. Map – to store data in key.value form.

Usage of Collection data type:

```
hive> create table Collectmantab (companyid bigint, companyName struct <actualName:string,aliasName:string>,
    > companyAdress struct <roadnum:int,street:string,city:string,state:string>,
    > Previousyrs_Revenues array<smallint>,growth_Prefereances map<string,boolean> )
    > row format delimited
    > fields terminated by '|'
    > collection items terminated by ','
    > map keys terminated by ':'
    > lines terminated by '\n'
    > stored as textfile;
```

After loading the data in collectmantab:

```
hive> select * from collectmantab;
OK
101010  {"actualname":"Infosys pvt ltd","aliasname":"Infy"}    {"roadnum":121,"street":"Isb main street","city":"Hyderabad","state":"Andhrapr
desh"}  [56,95,78,79,87]        {"AsiaPacific":true,"European":false,"China":true}
101020  {"actualname":"Tata consultancy ltd","aliasname":"TCS"} {"roadnum":156,"street":"electronic city","city":"Bangalore","state":"Karnatak
"}      [15,78,95,62,99]        {"Europe":true,"AsiaPacific":false,"China":false}
Time taken: 5.553 seconds, Fetched: 2 row(s).
hive>
```

HOW TO SELECT THE DATA FROM COLLECTION ITEMS

1. To see only alias name of company

```
hive> select companyname.aliasName from collectmantab;
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201701142055_0001, Tracking URL = http://localhost:50030/jobdetails.jsp?jobid=job_201701142055_0001
Kill Command = /usr/local/hadoop-1.2.1/libexec/../bin/hadoop job  -kill job_201701142055_0001
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2017-01-14 22:18:17,175 Stage-1 map = 0%,  reduce = 0%
2017-01-14 22:18:24,541 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.14 sec
2017-01-14 22:18:27,660 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 2.14 sec
MapReduce Total cumulative CPU time: 2 seconds 140 msec
Ended Job = job_201701142055_0001
MapReduce Jobs Launched:
Job 0: Map: 1  Cumulative CPU: 2.14 sec   HDFS Read: 484 HDFS Write: 9 SUCCESS
Total MapReduce CPU Time Spent: 2 seconds 140 msec
OK
Infy
TCS
Time taken: 61.627 seconds, Fetched: 2 row(s)
hive>
```

2.To see state and city info only

```
hive> select companyadress.city,companyadress.state from collectmantab;
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201701142055_0002, Tracking URL = http://localhost:50030/jobdetails.jsp?jobid=job_201701142055_0002
Kill Command = /usr/local/hadoop-1.2.1/libexec/../bin/hadoop job  -kill job_201701142055_0002
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2017-01-14 22:24:31,613 Stage-1 map = 0%,  reduce = 0%
2017-01-14 22:24:39,846 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.13 sec
2017-01-14 22:24:42,901 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 1.13 sec
MapReduce Total cumulative CPU time: 1 seconds 130 msec
Ended Job = job_201701142055_0002
MapReduce Jobs Launched:
Job 0: Map: 1  Cumulative CPU: 1.13 sec  HDFS Read: 484 HDFS Write: 44 SUCCESS
Total MapReduce CPU Time Spent: 1 seconds 130 msec
OK
Hyderabad       Andhrapradesh
Bangalore       Karnataka
Time taken: 33.78 seconds, Fetched: 2 row(s)
hive>
```

3.To see 3rd year revenue

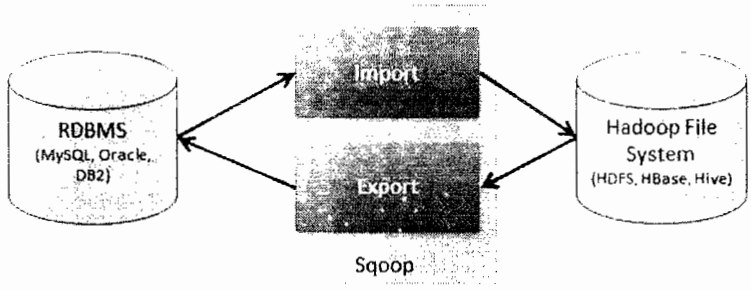
```
hive> select previousyrs_revenues [2] from collectmantab;
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201701142055_0003, Tracking URL = http://localhost:50030/jobdetails.jsp?jobid=job_201701142055_0003
Kill Command = /usr/local/hadoop-1.2.1/libexec/../bin/hadoop job  -kill job_201701142055_0003
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2017-01-14 22:28:30,453 Stage-1 map = 0%,  reduce = 0%
2017-01-14 22:28:47,514 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 6.34 sec
2017-01-14 22:28:49,548 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 6.34 sec
MapReduce Total cumulative CPU time: 6 seconds 340 msec
Ended Job = job_201701142055_0003
MapReduce Jobs Launched:
Job 0: Map: 1  Cumulative CPU: 6.34 sec  HDFS Read: 484 HDFS Write: 6 SUCCESS
Total MapReduce CPU Time Spent: 6 seconds 340 msec
OK
78
95
Time taken: 34.343 seconds, Fetched: 2 row(s)
hive>
```

SQOOP

5.1.Introduction

Sqoop is a tool designed to transfer data between Hadoop and relational databases. You can use Sqoop to import data from a relational database management system (RDBMS) such as MySQL or Oracle into the Hadoop Distributed File System (HDFS) and it can also export the data back into an RDBMS.

Sqoop works with relational databases such as Teradata, Netezza, Oracle, MySQL, Postgres, and HSQLDB.



5.2.History

Apache Sqoop is a tool designed for efficiently transferring bulk data between Apache Hadoop. Sqoop got the name from **sql+hadoop**. Sqoop became a top-level Apache project in March 2012.

Version

Latest stable release is 1.4.6

5.3.Why Sqoop?

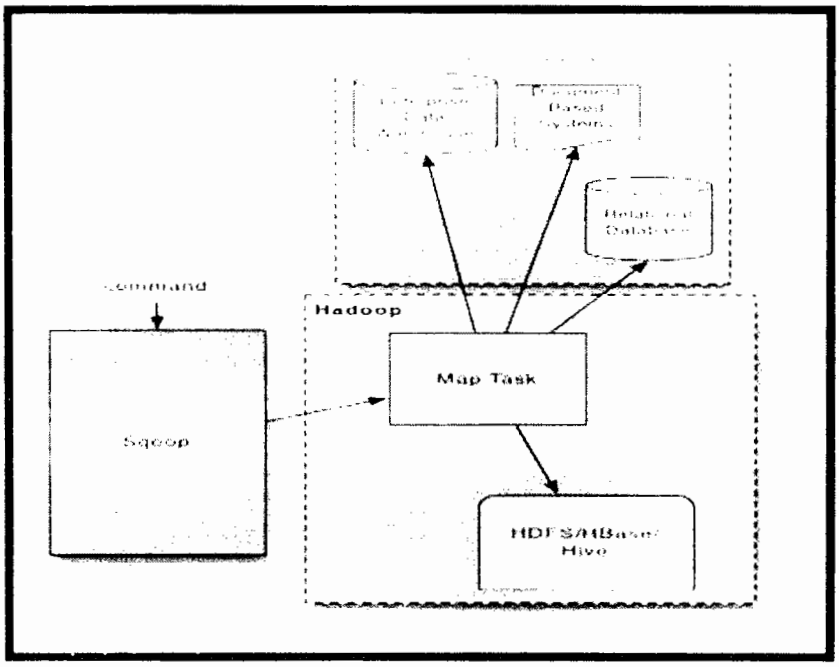
In real time Developers feel the transferring of data between relational database systems and HDFS is not interesting, the interesting work starts after data is loaded into HDFS. Here they always write custom scripts to transfer data in and out of Hadoop.If several MapReduce programs needed to do similar joins, the database server would experience very high load, in addition to a large number of concurrent connections, while MapReduce programs were running, possibly causing performance issues.

Sqoop makes this possible with a single command-line.
Sqoop uses MapReduce to import and export the data, which provides parallel operation as well as fault tolerance.

5.4.Where Sqoop is used?

As we are dealing with Big Data, Hadoop stores and processes the Big Data using differen processing frameworks like MapReduce, Hive, HBase, Cassandra, Pig etc and storage frameworks like HDFS to achieve benefit of distributed computing and distributed storage. In order to store and analyze the Big Data from relational databases. Data need to be transferred between database systems and Hadoop Distributed File System (HDFS). Here. Sqoop comes into picture. You can import data and export data between relational database systems and Hadoop and its eco-systems directly using sqoop.

5.5.Sqoop Architecture



Sqoop command submitted by the end user is parsed by Sqoop and launches Hadoop Map only job to import or export data because Reduce phase is required only when aggregations are needed. Sqoop just imports and exports the data; it does not do any aggregations. Map job launch multiple mappers depends on the number defined by user in the command line. Sqoop distributes the input data among the mappers equally to get high performance. Then each mapper creates connection with the database using JDBC and fetches the part of data assigned by Sqoop and writes it into HDFS or Hive or HBase based on the option provided in the command line.

5.6.What Sqoop Does?

Import sequential datasets from mainframe	Satisfies the growing need to move data from mainframe to HDFS
Data imports	Moves certain data from external stores and EDWs into Hadoop to optimize cost-effectiveness of combined data storage and processing
Parallel data transfer	For faster performance and optimal system utilization
Fast data copies	From external systems into Hadoop
Efficient data analysis	Improves efficiency of data analysis by combining structured data with unstructured data in a schema-on-read data lake
Load balancing	Mitigates excessive storage and processing loads to other systems

5.6.Sqoop ("SQL-to-Hadoop") capabilities

- Imports individual tables or entire databases to files in HDFS
- Generates Java classes to allow you to interact with your imported data
- Provides the ability to import from SQL databases straight into your Hive data warehouse

5.7.Sqoop – Installation

Follow the steps given below to install Sqoop on your system.

Step 1: Verifying JAVA Installation

You need to have Java installed on your system before installing Sqoop. Let us verify Java installation using the following command:

$ java -version

If Java is already installed on your system, you get to see the following response:

```
java version "1.7.0_71"
Java(TM) SE Runtime Environment (build 1.7.0_71-b13)
Java HotSpot(TM) Client VM (build 25.0-b02, mixed mode)
```

Step 2: Verifying Hadoop Installation

Hadoop must be installed on your system before installing Sqoop. Let us verify the Hadoop installation using the following command:

```
$ hadoop version
```

If Hadoop is already installed on your system, then you will get the following response:

```
Hadoop 2.4.1
--
Subversion https://svn.apache.org/repos/asf/hadoop/common -r 1529768
Compiled by hortonmu on 2013-10-07T06:28Z
Compiled with protoc 2.5.0
From source with checksum 79f8fb2479e4b1628b240f09af31e1e6
```

Step 3: Installing Sqoop

The following commands are used to extract the Sqoop tar ball and move it to "/usr/lib/sqoop" directory.

```
$tar -xvf sqoop-1.4.4.bin__hadoop-2.0.4-alpha.tar.gz
# mv sqoop-1.4.4.bin__hadoop-2.0.4-alpha /usr/lib/sqoop
```

Step 5: Configuring bashrc

You have to set up the Sqoop environment by appending the following lines to ~/**.bashrc** file:

```
export SQOOP_HOME=      lib sqoop export PATH $PATH:$SQOOP_HOME/bin
```

The following command is used to execute ~/**.bashrc** file.

```
$ source ~/.bashrc
```

Step 6: Download and Configure mysql-connector-java

We can download **mysql-connector-java-5.1.30.tar.gz** file from the following link.

The following commands are used to extract mysql-connector-java tarball and move **mysql-connector-java-5.1.30-bin.jar**to /usr/lib/sqoop/lib directory.

```
$ tar -zxf mysql-connector-java-5.1.30.tar.gz

# cd mysql-connector-java-5.1.30
# mv mysql-connector-java-5.1.30-bin.jar /usr/lib/sqoop/lib
```

Step 7: Verifying Sqoop

The following command is used to verify the Sqoop version.

```
$ sqoop-version
```

Expected output:

```
Sqoop 1.4.5 git commit id 5b34accaca7de251fc91161733f906af2cdbe83

Compiled by abe on Fri Aug 1 11:19:26 PDT 2014
```

5.8.Sqoop Tools

Sqoop is a collection of related tools. To use Sqoop, you specify the tool you want to use and the arguments that control the tool.

```
$ sqoop tool-name [tool-arguments]
```

Sqoop ships with a help tool. To display a list of all available tools. You can display help for a specific tool by entering: sqoop help (tool-name);

```
$ sqoop help
usage: sqoop COMMAND [ARGS]

Available commands:
  codegen            Generate code to interact with database records
  create-hive-table  Import a table definition into Hive
  eval               Evaluate a SQL statement and display the results
  export             Export an HDFS directory to a database table
  help               List available commands
  import             Import a table from a database to HDFS
  import-all-tables  Import tables from a database to HDFS
  list-databases     List available databases on a server
  list-tables        List available tables in a database
  version            Display version information
```

5.9.Using Generic and Specific Arguments

To control the operation of each Sqoop tool, you use generic and specific arguments. For example:

```
$ sqoop help import
usage: sqoop import [GENERIC-ARGS] [TOOL-ARGS]

Common arguments:
   --connect <jdbc-uri>     Specify JDBC connect string
   --connection-manager <class-name>    Specify connection manager class to use
   --driver <class-name>    Manually specify JDBC driver class to use
   --hadoop-mapred-home <dir>+      Override $HADOOP_MAPRED_HOME
   --help                   Print usage instructions
-P                          Read password from console
   --password <password>    Set authentication password
   --username <username>    Set authentication username
   --verbose                Print more information while working
   --hadoop-home <dir>+     Deprecated. Override $HADOOP_HOME

[...]

Generic Hadoop command-line arguments:
(must preceed any tool-specific arguments)
Generic options supported are
-conf <configuration file>     specify an application configuration file
-D <property=value>            use value for given property
-fs <local|namenode:port>      specify a namenode
-jt <local|jobtracker:port>    specify a job tracker
-files <comma separated list of files>    specify comma separated files to be copied to the map reduce cluster
-libjars <comma separated list of jars>    specify comma separated jar files to include in the classpath.
-archives <comma separated list of archives>    specify comma separated archives to be unarchived on the compute machines.
```

You must supply the generic arguments **-conf, -D**, and so on after the tool name but **before** any tool-specific arguments (such as **--connect**). Note that generic Hadoop arguments are preceeded by a single dash character (**-**), whereas tool-specific arguments start with two dashes (**--**), unless they are single character arguments such as **-P**.

For example, the **-D mapred.job.name=<job_name>** can be used to set the name of the MR job that Sqoop launches, if not specified, the name defaults to the jar name for the job - which is derived from the used table name.

5.10.New Concepts in Sqoop Versions

Recent Sqoop releases extended data integration between Apache Hadoop and relational data stores.

1.4.6 – July 2015	• **Support importing mainframe sequential datasets** • **Export data from HDFS back to an RDMS** • **Import data from database to Hive as Parquet files** • **Import data to HDFS as a set of Parquet files** • **Upsert export for SQL Server**
1.4.5 – August 2014	• Added high-performance Oracle connector • Enhancements to HCatalog integration • Support for Apache Accumulo
1.4.4 – July 2013	• HCatalog integration • Support for Oracle Wallet • Bulk load to PostgreSQL • Support for composite HBase keys

1.4.3 – March 2013

- Custom schemas for SQL Server and PostgreSQL
- Bulk import from PostgreSQL
- Support for using stored procedures during exports

5.11.sqoop-import

Sqoop is designed to import tables from a database into HDFS. To do so, you must specify a *connect string* that describes how to connect to the database. The *connect string* is similar to a URL, and is communicated to Sqoop with the **--connect** argument.

```
gopal@ubuntu:~$ sqoop import --connect jdbc:mysql://localhost/mydb --table emp;
```

This string will connect to a MySQL database named **mydb** on the host **emp**.You might need to authenticate against the database before you can access it. You can use the **--username** and **--password** or **-P** parameters to supply a username and a password to the database. For example:

```
gopal@ubuntu:~$ sqoop import --connect jdbc:mysql://localhost/mydb --username root --password root --table emp;
```

Or

```
gopal@ubuntu:~$ sqoop import --connect jdbc:mysql://localhost/mydb --username root --P --table emp;
```

The **-P** argument will read a password from a console prompt, and is the preferred method of entering credentials.

Sqoop automatically supports several databases, including MySQL.You can use Sqoop with any other JDBC-compliant database. First, download the appropriate JDBC driver for the type of database you want to import, and install the .jar file in the **$SQOOP_HOME/lib** directory on your client machine. For example, MySQL's Connector/Jlibrary has a driver class of **com.mysql.jdbc.Driver**. This class must be provided as an argument to Sqoop with **--driver**.

You can inspect the created files' contents by using the following command:

hadoop fs -cat /user/gopal/part-m-*

5.12.Specifying a Target Directory

If you plan to incorporate Sqoop into your Hadoop workflows. In order to do so, you want to specify the directory. Sqoop offers two parameters for specifying custom output directories: **--target-dir** and **--warehouse-dir**.

Use the **--target-dir** parameter to specify the directory on HDFS where Sqoop should import your data.

```
$ sqoop import --connect jdbc:mysql://localhost/mydb --table emp --target-dir /varma -m 1;
```

5.13.Import all tables

Use the **--warehouse-dir** parameter to specify the parent directory on HDFS where Sqoop import your data.

```
$ sqoop import all-tables --connect jdbc:mysql://localhost/mydb --warehouse-dir /gopal/varma --m 1;
```

You can inspect the created file contents by using the following command.

```
gopal@ubuntu:~$ hadoop fs -ls /gopal/varma/
Found 2 items
drwxr-xr-x   - gopal supergroup          0 2016-05-11 04:52 /gopal/varma/dept
drwxr-xr-x   - gopal supergroup          0 2016-05-11 04:54 /gopal/varma/emp
```

5.14.Importing Only a Subset of Data

Instead of importing an entire table, you need to transfer only a subset of the rows based on various conditions that you can express in the form of a SQL statement with a **--WHERE** clause.

```
$ sqoop import --connect jdbc:mysql://localhost/mydb --table emp --where "ename='scott'" --target-dir /common/put;
```

Protecting Your Password

Typing your password into the command-line interface is insecure. It can be easily retrieved from listing the operating systems running processes.You have two options besides specifying the password on the command line with the**--password** parameter.

The first option is to use the parameter **-P** that will instructSqoop to read the password from standard input.

```
$ sqoop import --connect jdbc:mysql://localhost/mydb -username root -P --table emp;
```

The first method, using the parameter **-P**, will instruct Sqoop to prompt the user for the password before any other Sqoop action is taken.

```
16/05/11 05:19:20 INFO sqoop.Sqoop: Running Sqoop version: 1.4.5
Enter password:
```

Alternatively, you can save your password in a file and specify the path to this file with the parameter **--password-file**.

```
$ sqoop import --connect jdbc:mysql://localhost/mydb -username root -password-file my-password --table emp;
```

The second solution, using the parameter **--password-file**, will load the password from any specified file on your HDFS cluster

```
echo"my-secret-password" >my-password
```

```
hadoop dfs -put my-password  /user/$USER/my-password
hadoop dfs -chmod 400 /user/$USER/my-password
rm my-password
sqoop import --password-file /user/$USER/my-password ...
```

5.15.Using a File Format Other Than CSV

The tab-separated CSV file that Sqoop uses by default does not suit your use case. Youprefer a binary format over plain text.

Sqoop supports three different file formats; one of these is text, and the other two are binary. The binary formats are Avro and Hadoop'sSequenceFile.

```
$ sqoop import --connect jdbc:mysql://localhost/mydb --table emp --username root -P --as-sequencefile
```

Avro File Format

```
$ sqoop import --connect jdbc:mysql://localhost/mydb --table emp --username root -P --as-avrodatafile
```

Binary formats have a few benefits over that of text files. First, binary formats are a natural fit for storing binary values like images or PDF documents. They are also more suited for storing text data. In order to access the binary data, you need to implement extra functionality or load special libraries in your application.

The SequenceFile is a special Hadoop file format that is used for storing objects and implements the Writable interface. This format was customized for MapReduce, and thus it expects that each record will consist of two parts: key and value.

Apache Avro is a generic data serialization system. Specifying the **--as-avrodatafile** parameter instructs Sqoop to use its compact and fast binary encoding format. Avro is a very generic system that can store any arbitrary data structures. The schema is usually encoded as a JSON string so that it's decipherable by the human eye. You don't need to import any special class, such as in the SequenceFile case, as all required metadata is embedded in the imported files themselves.

5.16.Compressing Imported Data

You want to decrease the overall size occupied on HDFS by using compression for generated files.

```
$ sqoop import --connect jdbc:mysql://localhost/mydb --table emp --compress
```

By default, when using the **--compress** parameter, output files will be compressed using the **GZipcodec**, and all files will end up with a .gz extension.
You can choose any other codec using the **--compression-codec** parameter.

```
gopal@ubuntu:~$ sqoop import --connect jdbc:mysql://localhost/mydb --table emp --compress --compression-codec org.apache.hadoop.io.com
press.BZip2Codec
```

Note: For example, if in the mapred-site.xml file, the property **mapred.output.compress** is set to false with the final flag, then Sqoop won't be able to compress the output files even when you call it with the **--compress** parameter.

Some codecs do not support seeking to the middle of the compressed file without reading all previous content, effectively preventing Hadoop from processing the input files in a parallel manner.

Table 2-2. Compression codecs

Splittable	Not Splittable
BZip2, LZO	Gzip, Snappy

5.17.Speeding Up Transfers

Sqoop is a great tool, and it's processing bulk transfers very well. For some databases you can take advantage of the direct mode by using the **–direct** Parameter

```
$ sqoop import --connect jdbc:mysql://localhost/mydb --table emp --direct
```

Rather than using the JDBC interface for transferring data, the direct mode delegates the job of transferring data to the native utilities provided by the database vendor. In the case of MySQL, the **mysqldump** and **mysqlimport** will be used for retrieving datafrom the database server or moving data back. In the case of PostgreSQL, Sqoop willtake advantage of the **pg_dump** utility to import data.

Sqoop has direct support only for **MySQL** and **PostgreSQL**.

Limitations

For example, in the case of MySQL, each node hosting a TaskTrackerservice needs to have both mysqldump and mysqlimport utilities installed. As the native utilities usually produce text output, binary formats like SequenceFileor Avro won't work.

5.18.Overriding Type Mapping

The default type mapping that Sqoop provides between relational databases and Hadoop usually works well. You have use cases requiring you to override the mapping.

Use Sqoop's ability to override default type mapping using the parameter **–map-column-java**. For example, to override the type of column id to Java type Long:

```
$ sqoop import --connect jdbc:mysql://localhost/mydb --table emp --map-column-java empno=long
```

5.19.Controlling Parallelism

Sqoop by default uses four concurrent map tasks to transfer data to Hadoop. Transferring bigger tables with more concurrent tasks should decrease the time required to transfer all data. You want the flexibility to change the number of map tasks used on a per-job basis.

Use the parameter **--num-mappers** or **–m** if you want Sqoop to use a different number of mappers.

```
$ sqoop import --connect jdbc:mysql://localhost/mydb --table emp --num-mappers 5
```

5.20.Encoding NULL Values

Sqoop encodes database NULL values using the nullstring constant.Your downstream processing (Hive queries, custom MapReduce job, or Pig script) uses a different constant for encoding missing values. You would like to override the default one.

You can override the NULL substitution string with the **--null-string** and **--null-non-string**parameters to any arbitrary value. For example, use the following command to override it to **\N**

```
$ sqoop import --connect jdbc:mysql://localhost/mydb --table emp --null-string '\\N' --null-non-string '\N'
```

5.21.Importing all Tables except Some Tables

If you need to import all but a few tables, you can use the parameter **--exclude-tables**that accept a comma-separated list of table names that should be excluded from the bulk

import.

```
$ sqoop import-all-tables --connect jdbc:mysql://localhost/mydb --exclude-tables emp
```

5.22.Importing particular columns

If you need to import particular columns in a table, then you can use the parameter **--columns**

```
$ sqoop import --connect jdbc:mysql://localhost/mydb --table emp --columns 'empno, ename' --target-dir /varma
```

5.23.Importing data with diff. field separator (otherthan CSV)

If you want to import data with a different field separator instead of default separator ',' then use **--fields-terminated-by**

```
$ sqoop import --connect jdbc:mysql://localhost/mydb --table dept --fields-terminated-by '|'
```

5.24.Importing Only New Data

You have a database table with an INTEGER primary key. You are only appending new rows, and you need to periodically sync the table's state to Hadoop for further processing.Activate Sqoop's incremental feature by specifying the **--incremental** parameter.

Incremental import also requires two additional parameters: **--check-column**indicates a column name that should be checked for newly appended data, and **--last-value**contains the last value that successfully imported into Hadoop.

```
gopal@ubuntu:~$ sqoop import --connect jdbc:mysql://localhost/mydb --table dept --incremental append --check-column deptno --last-valu
e 0
```

5.25.Incrementally Importing Mutable Data

While you would like to use the incremental import feature, the data in your table is also being updated, ruling out use of the **append** mode

Use the **lastmodified** mode instead of the **append** mode. For example, use the following command to transfer rows whose value in column **updated_date**is greater than 2016-05-12 06:49:01:

```
gopal@ubuntu:~$ sqoop import --connect jdbc:mysql://localhost/mydb --table ntab --incremental lastmodified --check-column updated_date
 --last-value 2016-05-12 06:53:01
```

The incremental mode **lastmodified** requires a column holding a date value (suitabletypes are **date, time, datetime**, and **timestamp**) containing information as to when eachrow was last updated. Sqoop will import only those rows that were updated after thelast import. This column should be populated to the current time on every new row insertion or on a change to an existing row. This ensures that Sqoop can pick up changed rows accurately.

5.26.Preserving the Last Imported Value(Sqoop Job)

Incremental import is a great feature that you're using a lot. Shouldering the responsibility for remembering the last imported value is getting to be a hassle.You can take advantage of the built-in Sqoop metastore that allows you to save all parameters for later reuse. You can create a simple incremental import job with the following command:

```
gopal@ubuntu:~$ sqoop job --create incrappnd -- import --connect jdbc:mysql://localhost/mydb --table dept --incremental append --check
-column deptno --last-value 0
```

And start it with the **--exec** parameter:

```
gopal@ubuntu:~$ sqoop job --exec incrappnd
```

The Sqoop metastore is a powerful part of Sqoop that allows you to retain your jobdefinitions and to easily run them anytime. Each saved job has a logical name that is used for referencing. You can list all retained jobs using the **--list** parameter:

```
gopal@ubuntu:~$ sqoop job --list
```

You can remove the old job definitions that are no longer needed with the **--deleteparameter**, for example:

```
gopal@ubuntu:~$ sqoop job --delete incrappnd
```

And finally. you can also view content of the saved job definitions using the **--showparameter**, for example:

```
gopal@ubuntu:~$ sqoop job --show incrappnd
```

Output of the **--show** command will be in the form of properties. Unfortunately, Sqoopcurrently can't rebuild the command line that you used to create the saved job.

5.27.Storing Passwords in the Metastore

You like the built-in Sqoop metastore for its ability to store jobs and the option to elegantly run them at your convenience. As a next step, you would like to automate the process and start the jobs automatically. Unfortunately, each execution requires you to enter a password, which is not easily automated by a script.

Sqoop offers two ways to run jobs from within the metastore without requiring any user input. The first and more secure method is by using the parameter **--password-file**to pass in the file containing the password. The second, less secure method is to set the property **sqoop.metastore.client.record.password** in the **sqoop-site.xml** to true:

```
<property>
  <name>sqoop.metastore.client.autoconnect.url</name>
  <value>jdbc:hsqldb:hsql://your-metastore:16000/sqoop</value>
</property>
```

Running the metastore as a service will start the embedded HSQLDB database that will be exposed to the rest of your cluster.The default port is 16000, and you can configure it in the sqoop-site.xml file with the **sqoop.metastore.server.port** configuration property.

5.28.Importing Data from Two Tables

You need to import one main table. The important values are stored in the referenced dictionary tables, and the main table contains only numeric foreign keys pointing to the values in the dictionaries rather than to natural keys as in the original cities table.

Instead of using table import, use free-form query import. In this mode, Sqoop willallow you to specify any query for importing data. Instead of the parameter **--table**, use the parameter **--query** with the entire query for obtaining the data you would like to transfer.

```
gopal@ubuntu:~$ sqoop import --connect jdbc:mysql://localhost/mydb --query "select empno, ename, sal, dname from emp join dept using (
deptno) where $CONDITIONS" --split-by empno --target-dir /gopal/varma
```

There is a lot to be aware of when using free-form query imports. By using query imports, Sqoop can't use the database catalog to fetch the metadata. This is one of the reasons why using table import might be faster than the equivalent free-form query import.

In addition to the **--query** parameter, you need to specify the **--split-by** parameter with the column that should be used for slicing your data into multiple parallel tasks.

The third required parameter is **--target-dir**, which specifies the directory on HDFS where your data should be stored.

Sqoop performs highly efficient data transfers by inheriting Hadoop's parallelism. To help Sqoop split your query into multiple chunks that can be transferred in parallel, you need to include the **$CONDITIONS** placeholder in the where clause of your query. While you could skip **$CONDITIONS** by forcing Sqoop to run only one job using the **--num-mappers** 1 parameter, such a limitation would have a severe performance impact.

5.29.Using Custom Boundary Queries

Prior to starting any data transfer in MapReduce, Sqoop takes a long time to retrieve the minimum and maximum values of the column specified in the **--split-by**parameter that are needed for breaking the data into multiple independent tasks.

```
gopal@ubuntu:~$ sqoop import --connect jdbc:mysql://localhost/mydb --query "select empno, ename, sal, dname from emp join dept using (
deptno) where \$CONDITIONS" --split-by empno --target-dir /gopal/varma --boundary-query "select min(empno), max(empno) from emp"
```

You can specify any valid query to fetch minimum and maximum values of the **--split-by** column using the **--boundary-query** parameter:

5.30.Renaming Sqoop Job Instances

You run several concurrent free-form query imports from various databases at the sametime on your Hadoop cluster. All MapReduce jobs are named **QueryResult.jar**, so it's very hard to see which MapReduce job belongs to which imported query.

You can use the command-line parameter **--mapreduce-job-name**to specify the name of the generated MapReduce job. This name will then show up in the JobTracker web UI.

```
gopal@ubuntu:~$ sqoop import --connect jdbc:mysql://localhost/mydb --query "select empno, ename, sal, dname from emp join dept using (
deptno) where \$CONDITIONS" --split-by empno --target-dir /gopal/varma --mapreduce-job-name empdata
```

WebUI Output:

[illegible]	Wed May 11 21:49:33 PDT 2016	NORMAL	gopal	empdata	100.00%	4	4	100.00%	0	0	NA	NA

5.31.Importing Queries with Duplicated Columns

You have more than one table that you're joining in your free-form query. Your Sqoopimport is failing with an error message about duplicate columns

Imported Failed: Duplicate Column identifier specified: 'deptno'

You can do that using the **AS**syntax. For example, to import city names from the tables **emp** and **dept**, you can use the following query:

Before submitting the MapReduce job, Sqoop performs several actions. One such action is to fetch metadata about the transferred columns and their associated types. During this step, Sqoop will generate a Java class that contains one attribute for each column that will be named as the column itself. Java attributes must be unique; therefore, all columns in your query must have unique names.

```
gopal@localhost:~$ sqoop import --connect jdbc:mysql://localhost/mydb --query "select empno, ename, sal, dname, emp.empno AS emp_deptno f
rom emp join dept using (deptno) where \$CONDITIONS" --split-by empno --target-dir /gopal/varma
```

During a join operation two columns from different tables will have the same name. This is especially problematic if your query selects all columns from all join tables using fragments like select **table1.*, table2.***. In this case, you must break the general statement down, name each column separately, and use the **AS** clause to rename the duplicate columns so that the query will not have duplicate names.

5.31.Hive-Import

Sqoop supports importing into Hive. Add the parameter --hive-import to your command to enable it

```
$ sqoop import --connect jdbc:mysql://localhost/mydb --table emp --hive-import;
```

If you're creating a new Hive table Sqoop will convert the data types of each column from your source table to a type compatible with Hive. Usually this conversion is straightforward: for example, JDBC types **VARCHAR, CHAR**, and other string-based types are all mapped to Hive **STRING**.

Sometimes the default mapping doesn't work correctly for your needs; in those cases, you can use the parameter **--map-column-hive** to override it. This parameter expects a

comma-separated list of **key-value pairs** separated by the equal sign (=) in order to specify which column should be matched to which type in Hive.

```
gopal@ubuntu:~$ sqoop import --connect jdbc:mysql://localhost/mydb --table emp --hive-import --hive-table emp,ept --map-column-hive e
mpno=int,mgr=float,deptno=int;
```

If your table already exists and contains data, Sqoop will append to the newly importeddata. You can change this behavior by using the parameter **--hive-overwrite**, which will instruct Sqoop to truncate an existing Hive table and load only the newly imported one.

5.32.Using Partitioned Hive Tables

You want to import data into Hive on a regular basis (for example, daily), and for that purpose your Hive table is partitioned. You would like Sqoop to automatically import data into the partition rather than only to the table.

Sqoop supports Hive partitioning out of the box. In order to take advantage of this functionality, you need to specify two additional parameters: **--hive-partition-key**, which contains the name of the partition column, and **--hive-partition-value**.

```
gopal@ubuntu:~$ sqoop import --connect jdbc:mysql://localhost/mydb --table dept --hive-import --hive-table dept --hive-partition-key c
ountry --hive-partition-value "US";
```

Sqoop mandates that the partition column be of type **STRING**. The current implementation is limited to a single partition level.

5.33.Replacing Special Delimiters During Hive Import

You've imported the data directly into Hive using Sqoop's**--hive-import**feature. When you call **SELECT count(*) FROM your_table** query to see how many rows are in the imported table, you get a larger number than is stored in the source table on the relational database side.

This issue is quite often seen when the data contains characters that are used as Hive's delimiters. You can instruct Sqoop to automatically clean your data using **--hive-drop-import-delims**, which will remove all \n, \t, and \01 characters from all string-based columns.

```
gopal@ubuntu:~$ sqoop import --connect jdbc:mysql://localhost/mydb --table emp --hive-import --hive-table emp --hive-drop-import-delim
s;
```

If removing the special characters is not an option in your use case, you can take advantage of the parameter **--hive-delims-replacement**, which will accept a replacement string. Instead of removing separators completely, they will be replaced with a specified string.

```
gopal@ubuntu:~$ sqoop import --connect jdbc:mysql://localhost/mydb --table emp --hive-import --hive-table emp --hive-delims-replacemen
t "NN";
```

Sqoop will, by default, import data into comma-separated text files where each linerepresents one row. However, if your data contains the new-line character (**\n**), such a row will create two separate lines that will consequently be processed as two separate rows by Hive.

Even though both parameters contain hive in their names, they are not restricted toworking in tandem with the **--hive-import** parameter.

5.34.Using the Correct NULL String in Hive

You've imported your data into Hive using Sqoop and now you're trying to query it. However, you can see that some columns contain the correct **NULL** value but some containthe string literal null and are not selected using the expression column **IS NULL**.

Due to differences in the default **NULL** substitution string between Sqoop and Hive, youhave to override the Sqoop default substitution strings to be compatible with Hive.

```
gopal@ubuntu:~$ sqoop import --connect jdbc:mysql://localhost/mydb --table emp --hive-import --null-string '\\N' --null-non-string '\\
N';
```

Similarly, when you export data from Hive into your relational database, you should use the parameters **--input-null-string** and **--input-null-non-string**and set both to value **\N**.

5.35.Importing Data into Hbase

Instead of importing data into an HDFS file or a Hive table, you would like to transfer it into Hbase.

Sqoop has out-of-the-box support for HBase. To enable import into HBase, you need to supply two additional parameters: **--hbase-table** and **--column-family**.

```
$ sqoop import --connect jdbc:mysql://localhost/mydb --table emp --hbase-table emp --column-family emp_details;
```

To insert data into HBase there are three mandatory parameters: the table name, a column family name within the table, and the id of the row. Sqoop uses one table and one column family per import job, so you have to specify them using the **--hbase-table** and **--column-family** parameters on the command line.

To identify each individual row in HBase, Sqoop defaults to the column name specified in the **--split-by** parameter or usually the primary key of the table. You can override this behavior using the **--hbase-row-key** parameter.

Both the HBase table and the column family must exist prior to runningthe Sqoop import command. If you want Sqoop to create the tableautomatically, you'll need to specify the parameter **--create-hbase-table**.

5.36.Importing All Rows into Hbase

You've imported your table directly into HBase, but it seems that there are fewer rowsthan in your source table.

You might need to enable inserting the row key into the value using the property**sqoop.hbase.add.row.key**.

```
gopal@ubuntu:~$ sqoop import -D sqoop.hbase.add.row.key=true --connect jdbc:mysql://localhost/mydb --table dept --hbase-table dept --c
olumn-family emp_details;
```

HBase does not allow the insertion of empty values: each cell needs to have at least onebyte. Sqoop serialization, however, skips all columns that contain a NULL value, resulting in skipping rows containing NULL value in all columns.

This explains why Sqoopimports fewer rows than are available in your source table.

5.38.Export

5.38.1.Transferring Data from Hadoop

You have a workflow of various Hive and MapReduce jobs that are generating data on a Hadoop cluster. You need to transfer this data to your relational database for easy querying.

You can use Sqoop's export feature that allows you to transfer data from the Hadoop ecosystem to relational databases. For example, to export data from the **export-dir**directory.Export works similarly to import, except export transfers data in the other direction.

```
$ sqoop export --connect jdbc:mysql://localhost/exp --table emp --export-dir /emp
```

Instead of transferring data from the relational database using **SELECT** queries, Sqoopwill transfer the data to the relational database using **INSERT**statements.

After you execute the Sqoop command, Sqoop will connect to your database to fetch various metadata about your table, including the list of all columns with their appropriate types. Using this metadata, Sqoop will generate and compile the Java class.

5.38.2.Inserting Data in Batches

While Sqoop's export feature fits your needs, it's too slow. It seems that each row isinserted in a separate insert statement.

Sqoop offers multiple options for inserting more than one row at a time.First, you can enable JDBC batching using the **--batch** parameter:

```
$ sqoop export --connect jdbc:mysql://localhost/exp --table emp  --export-dir /emp --batch
```

The second option is to use the property **sqoop.export.records.per.statement** tospecify the number of records that will be used in each insert statement.

```
gopal@ubuntu:~$ sqoop export -D sqoop.export.records.per.statement=10 [illegible] --connect jdbc:mysql://localhost/exp --table emp --export-dir /
mp
```

Finally, you can set how many rows will be inserted per transaction with the **sqoop.export.statements.per.transaction** property.

```
gopal@ubuntu:~$ sqoop export -D sqoop.export.statements.[illegible] --connect jdbc:mysql://localhost/exp --table emp --export-
/emp
```

5.38.3.Exporting with All-or-Nothing Semantics

You need to ensure that Sqoop will either export all data from Hadoop to your databaseor export no data (i.e., the target table will remain empty).

You can use a staging table to first load data to a temporary table before making changesto the real table. The staging table name is specified via the **--staging-table** parameter.

```
$ sqoop export --connect jdbc:mysql://localhost/exp --table emp --export-dir /emp --staging-table stag_tab

INFO mapreduce.ExportJobBase: Data will be staged in the table: stag_tab
INFO mapreduce.ExportJobBase: Beginning export of emp

INFO manager.SqlManager: Migrated 14 records from `stag_tab` to `emp`
```

When using a staging table, Sqoop will first export all da a into this staging table instead of the main table that is present in the parameter **--table**. Sqoop opens a new transaction to move data from the staging table to the final destination, if and only if all parallel tasks successfully transfer data.

Sqoop requires that the structure of the staging table be the same as that of the targettable. The number of columns and their types must be the same; otherwise, the exportoperation will fail.

Note: The staging table is not automatically created by Sqoop and must exist prior to starting the export process.

You can specify the parameter **--clear-staging-table** to instruct Sqoop to automatically clean the staging table for you. If supported by the database, Sqoop will use a **TRUNCATE** operation to clean up the staging table as quickly as possible.

5.38.4.Updating an Existing Data Set

You previously exported data from Hadoop, after which you ran additional processingthat changed it. Instead of wiping out the existing data from the database, you prefer tojust update any changed rows.

```
$ sqoop export --connect jdbc:mysql://localhost/exp --table emp --export-dir /emp --update-key empno;
```

You can take advantage of the update feature that will issue **UPDATE** instead of **INSERT**statements. The update mode is activated by using the parameter **--update-key** thatcontains the name of a column that can identify a changed row—usually the primarykey of a table.

The parameter **--update-key** is used to instruct Sqoop to update existing rows rather than insert new ones.

5.38.5.Updating or Inserting at the Same Time

You have data in your database from a previous export, but now you need to propagate updates from Hadoop.

```
gopal@ubuntu:~$ sqoop export --connect jdbc:mysql://localhost/exp --table emp --export-dir /emp --update-key empno --update-mode allow
insert;
```

If you need both updates and inserts in the same job, you can activate the so-called**upsert** mode with the **--update-modeallowinsert** parameter.

The ability to conditionally insert a new row or update an existing one is an advanced database feature known as **upsert**. This feature is not available on all database systems nor supported by all Sqoop connectors.

5.39.Sqoop Codegen

From the viewpoint of object-oriented application. every database table has one DAO class that contains 'getter' and 'setter' methods to initialize objects. This tool (**-codegen** generates the DAO class automatically.

It generates DAO class in Java, based on the Table Schema structure. The Java definition is instantiated as a part of the import process. The main usage of this tool is to check if Java lost the Java code. If so, it will create a new version of Java with the default delimiter between fields.

Let us take an example that generates Java code for the emp table in the mydb database.

```
$ sqoop codegen --connect jdbc:mysql://localhost/mydb --table emp;

INFO manager.CompilationManager: Writing jar file: /tmp/sqoop-gopal/compile/4a01fedfc5522f8d50a385bd5e17e424/emp.jar

gopal@ubuntu:~$ ls /tmp/sqoop-gopal/compile/4a01fedfc5522f8d50a385bd5e17e424/
emp.class              emp.java
```

HBASE

6.1.HBASE-Introduction

HBASE- Hadoop Data**BASE** that runs on top of Hadoop.It is a Database which is an open source ,non-relational ,distributed .NoSQL(Not Only SQL) database modeled after Google's BigTable and written in Java.It is developed as part of Apache Software Foundation's (AFS) Apache Hadoop project and runs on HDFS(Hadoop Distributed FileSystem)

Hadoop can perform only batch processing, and data will be accessed only in a sequential manner leading to low latency.HBASE internally uses Hash tables and provides random access, and it stores the data in HDFS files that are indexed (sorted) by their key for faster lookups thus providing high latency compared to HDFS.

6.2.Comparison of HBASE and RDBMS

HBASE	RDBMS
Column-oriented,Open Sourced, Distributed and versioned database storage system	Row Oriented,Tabular Structured Data
Does not have a query language like SQL and uses API's and Map Reduce for accessing data	Uses structured query language like SQL to access the data
Can handle Structured and semi-structured data	Can handle Structured data
Designed to store De-Normalized data and is a Non Relation database	Designed to store Normalized data and is a Relational database
Stores data in key,value pairs	
Multidimentional database (RowKey,ColumnFamily)	Two dimensional database
Flexible Schema- as Columns can be easily added for specific ColumnFamilies	Fixed Schema
HBase internally uses Hash tables and provides random access, and it stores the data in HDFS files that are indexed (sorted) by their key for faster lookups.	Randomness cause of indexing
Cannot be used for Transaction Processing	Can be used for Transaction processing
Easily Scalable- Scaling out by adding new servers to the cluster by Horizontal scalability (both storage and processing) using commodity hardware as it is based on distributed computing	Difficult to scale.-Scaling up by upgrading the servers by Horizontal Scalability

HBASE	RDBMS
architecture.	
Doesn't directly support JOINS or Aggregations	Supports JOINS,Aggregations using SQL
Used for Online Analytical Processing	Used for Online Transaction Process OLTP
Runs on top of HDFS	
Can handle huge volume of data	Can handle Limited amount of data at a time

6.3.Comparison of HBASE and HDFS

HBASE	HDFS
HBase is a distributed, column oriented database and stores data in key/value pairs	HDFS is a distributed file system and stores data in form of flat files
Random reads and writes	Sequential file access .Random writes are not possible as it allows to Write once read many times.
Suitable for Low Latency operations by providing access to the specific row data from the big volume	Suitable for High Latency operations through batch processing
Random access of rows through their row keys	Designed for batch processing and hence doesn't have a concept of random reads/writes
Offline access i.e; batch access through map-reduce queries	Designed for batch processing
Addresses Hadoop's Architecture failure i.e; Single Point of Failure (SPOF) with ZooKeeper	
Runs on top of HDFS and uses HDFS for storage	

HBASE	HDFS
W.r.t. Performance HBASE is slow than HDFS as HBASE has to read the KeyValues, returning the most current row or specified timestamps. etc.	

6.4.Comparison of HBASE and NoSQL

HBASE	NoSQL
HBASE is a NoSql database	
Datastores in Key,Value pair	By Default Value is stored in Key,Value pair
Horizontal Scalability	Horizontal Scalability
Non Relational DB,No ACID properties	Non Relational DB,No ACID properties
Can handle FileSystem Format only	Can handle Structured, SemiStructured, Unstructured data
HBASE uses MapReduce for processing the data	Can perform basic CRUD operations.Complex aggregations are tough to handle so we need to integrate with solutions like Hadoop for complex processing .
Used for online Analytical Processing	Used for online Analytical Processing OLTP
HBase permits two types of access: random access of rows through their row keys and offline or batch access through map-reduce queries	Random access of data is possible
Integrated with Big Data technology stack only .	Can be integrated with Big Data technology stack
Master slave model to address parallel processing	

6.5.When should we prefer HBASE over other database

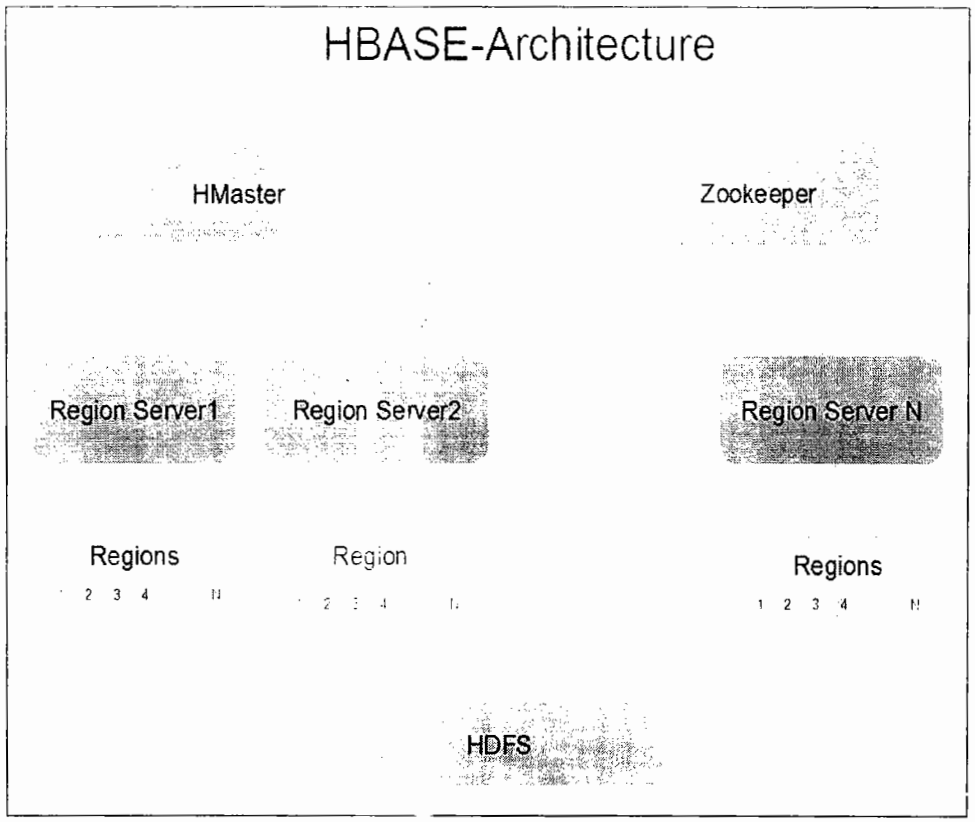
- When each row is slightly different in the schema.
- When each row can have different number of column and most of the columns may have NULL values.
- When we can't add columns fast enough and most of them are NULL in each row
- If we need key based access to data when storing or retrieving
- When we have to perform high write throughput , low latency reads and need to perform range based large data scans.
- Where applications has **CP** requirement of the **CAP** requirement
- **Consistency** (all nodes see same data at same time) ,Availability (guarantee that every request receives response whether its success or failure) and are Partition tolerant (system continues to operate despite of arbitary data loss or failure) i.e; HBASE distributed system addresses **Consistency** and **Partition** Tolerant requirement of the Applications
- Applications with fast read and writes and scalability

6 **6.When not to use HBASE**

- When we need to handle transactions and relational analytics.
- When Applications need data to be rolled up,aggregated or analyzed across row.

6.7.HBASE ARCHITECTURE

HBASE Architecture consists of HMaster,Region Server,Regions,Zookeeper



6.7.1.HMaster

Below are the key responsibilities of H Master

- To store the Meta data information of the HBASE tables using -**ROOT**- and .**META**. Files
- To bootstrap RegionServer ie;starting Region Server
- RegionServer failover/fault tolerance and backup mechanism is handled by **HMASTER**
- HMASTER should be part of a Hadoop Node
- Interface for creating. deleting. updating tables

6.7.2.RegionServer

- Regionservers do not run any ob like TaskTrackers . They just store data and are responsible for serving and managing egions.
- Each Region Server is responsible to serve a set of Regions(one or more).Howeve each Region can be associated to only on Region Server
- Region Server should be part of Hadoop Node
- Region Server should be only on Data Node

6.7.3.Region

- Regions are a subset of the table's data, and they are essentially a contiguous, sorted range of rows that are stored together.
- Each Region is assigned to a Region Server on startup and the master can decide to move a Region from one Region Server to another as the result of a load balance operation.
- Mapping of a Region to Region Server is handled by HMaster
- Intially there will be one region for one table. When region grows too large its gets splits into two and this information will be reported to HMASTER.
- Each Region is 1GB by default

6.7.4.ZooKeeper

- Hadoop Architecture is Single Point of Failure since it has only one Name Node and no backup or failure mechanism to address it.In HBASE ZooKeeper addresses the SPOF.
- Zookeeper cannot be compared to Secondary Name Node
- ZooKeeper is like HIVE,SQOOP,PIG component.
- Zookeeper is used to achieve coordination of distributed nodes between components of Hadoop.Plays important role in distributed clustered environment.
- As part of HBASE we may or may not have ZooKeeper
- HBase by default manages a ZooKeeper cluster. It gets started and stopped as part of the HBase start/stop process. But, you can also manage the ZooKeeper ensemble independent of HBase and just point HBase at the cluster it should use.
- Zookeeper is used in other stacks for distributed coordination.It is not restricted to HBASE.
- Each Region Server is connected to Zookeeper.HMaster keeps track of these connections.Zookeeper maintains heartbeats with timeout.On timeout Master declares RegionServer as dead and starts recovery process by reassigning regions to different active RegionServer
- ZooKeeper stores the **.META.** File

-ROOT- Table

-ROOT- Table keeps track of .META. Table

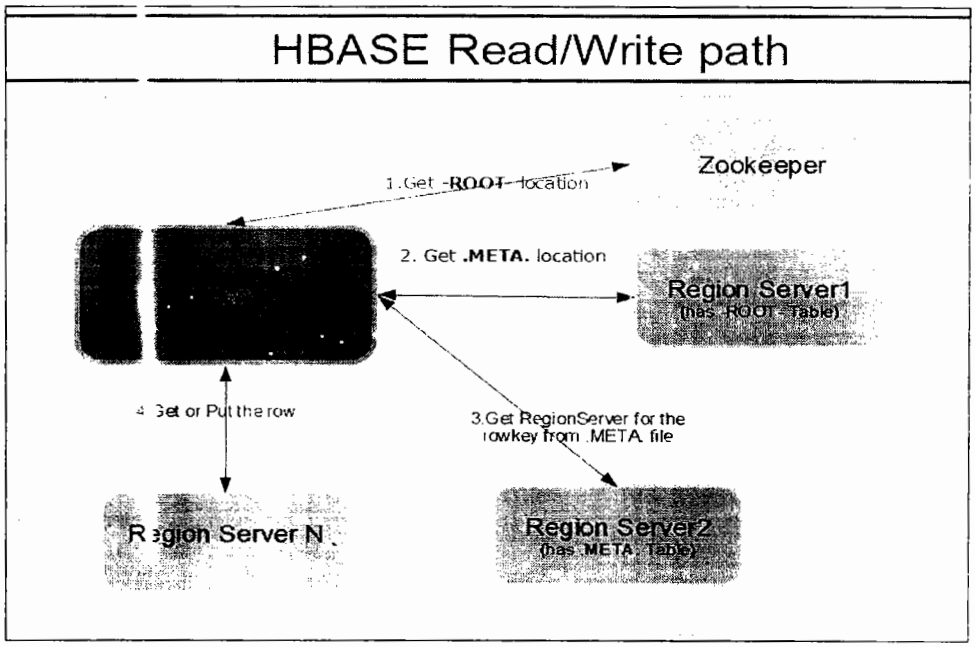
.META. Table

- META table. holds the location of the all the regions in the cluster/system.
- The .META. table structure is as follows: - Key: region start key,region id- Values:

RegionServer

- ZooKeeper stores the location of the META table.
- The client gets the Region server that hosts the META table from ZooKeeper.
- The client will query the Region server that hosts .META. table to get the region server corresponding to the row key it wants to access.
- The client caches region server(thats has the row key) information along with the META table location.
- It will get the Row from the corresponding Region Server

Reference :-https://www.mapr.com/blog/in-depth-look-hbase-architecture



6.8.Data Modelling

6.8.1.HTable

HBase tables are more like logical collection of rows stored in separate partitions called Regions

6.8.2.ColumnFamily

Data within a row is grouped by Column Family.

- As Column families also impact the physical arrangement of data stored in Hbase ,we need to define the ColumnFamily up front and they are not easily modified.
- Each Column Family has one or more Columns and these Columns in a family are stored together in a low level storage file known as Hfile
- Every row in a table has the same column families, although a row need not store data in all its families

6.8.3.ColumnQualifierName (CQN)

Data within a Column Family is tagged under ColumnQualifier/Column.

- No need to specify Column qualifiers in advance.Column qualifiers need not be consistent between rows.
- Column qualifiers do not have a data type and are always treated as a byte[ ](bytearray).

6.8.4.RowKey

A row is one instance of data in a table and is identified by a rowkey.

- Rowkeys are unique in a Table and are always treated as a byte[](bytearray).
- Row keys do not have a data type.

6.8.5.TimeStamp

Values within a cell are versioned. Versions are identified by their version number, which by default is the timestamp of when the cell was written.

- If a timestamp is not specified during a write, the current timestamp is used.
- If the timestamp is not specified for a read, the latest one is returned.
- The number of cell value versions retained by HBase is configured for each column family.
- The default number of cell versions is three.

6.8.6.cell

A combination of row key, column family, and column qualifier uniquely identifies a cell.

- In HBASE table ,the entity that contains the data is called a cell
- The data stored in a cell is referred to as that cell's value. Values also do not have a data type and are always treated as a byte[ ](bytearray).

6.9.Sample table structure in HBASE

Table1:-SW_TECH

A table **SW_TECH** in HBase consisting of two column families, **Database** and **Programming Language**, each having two columns.**Database** ColumnFamily comprises of **SQL** column and **NoSQL** column. Programming Language Column Family comprises of Web Programming and Software Programming Columns.

Software Technologies Table content :-**SW_TECH**

	Column (Database)	**Family**-DB	**Column Family**-PRG_LNG (Programming Language)	
RowKe	**CQN**-SQL	**CQN**-NoSQL	**CQN**-	**CQN**-SW_PRG

y			Web_PRG	
'0001	MySQL	Cassandra	HTML	C
'0002	Oracle	MongoDB	JavaScript	C++
'0003	Sybase	HBASE	VBScript	Java

Table2:-CUST_DET

A **CUST_DET** table in HBase consisting of two column families CustomerName(with 3 columns i.e; First Name,MiddleName,LastName),Shipping Address(with 2 columns i.e;Address1,Address2) and CustomerPersonalInformation (with 2 columns i.e;Gender,Age). The entity that contains the data is called a cell. The rows are sorted based on the row keys.

Customer Details Table content **:-CUST_DET**

	Column Family- CUST_NAME			**Column Family-** SHIP_ADDRESS		**Column Family-** CUST_PERSONAL_INFO	
Row Key	**CQN-** FIRST_NAME	**CQN-** MIDDLE_NAME	**CQN-** LAST_NAME	**CQN-** ADD1	**CQN-** ADD2	**CQN-** GENDER	**CQN-** AGE
'0001	Krishna	Kanth	Ch	123-A	KPHB colony,Hyderabad	M	25
'0002	Raju		M	SV Apartment,BFM layout		M	27
'0003	Sandhya	Rani	A	H.No-155,First Floor	Street-10,DharamKarm Road,Ameerpet	F	29

6.10.Steps to execute the DDL and DML statements in HBASE shell

1. Access VMWare WorkStation >> PowerOn this virtual machine
2. login into the ubuntu by providing the username and password
3. Open Ubuntu Terminal Applications >> Accessories >> Terminal
4. On ubuntu terminal login with username and password
5. mrinmoyroot@ubuntu: su root
6. password:pwd
7. Ensure HMASTER is running by executing the jps command

root@ubuntu:/home/mrinmoy# jps

```
943 DataNode
1461 NameNode
1352 JobTracker
1683 TaskTracker
1572 SecondaryNameNode
2276 Jps
1840 QuorumPeerMain
```

8. We need to kill the ZooKeeper and also start the HMASTER on this cluster

   root@ubuntu:/home/mrinmoy# kill 1840

9. root@ubuntu:/usr/lib/hbase/bin# ./start-hbase.sh

```
starting master, logging to /usr/lib/hbase/bin/../logs/hbase-root-master-ubuntu.out

root@ubuntu:/usr/lib/hbase/bin# jps
943 DataNode
2765 Hmaster
1461 NameNode
 1352 JobTracker
1683 TaskTracker
2907 Jps
1572 SecondaryNameNode
```

6.10.1.DDL in HBASE

Let us consider to create and insert data into the table Software Technologies , Table : "**SW_TECH**" mentioned above.

create - Creates a table.

If we provide just table name we will get error as below.

We need to provide at least one column family while creating the table in HBASE.

```
hbase(main):002:0> create 'SW_TECH','CF_DB'
    ERROR: Table must have at least one column family
```

Syntax:- create '<<tablename>>','<<co umnfamily>>'

```
hbase(main):002:0> create 'SW_TECH','CF_DB'
0 row(s) in 1.2160 seconds
```

list - Lists all the tables in HBASE.

Syntax:- list

```
TABLE
Chaitanya
DEPT_HBASE
DHW
DUMMY1
DUMMY2
EMP_HBASE
Employee
Gopal
```

```
NewTable
RK
SW_TECH
TEMP_HBASE
company_det
htab48
order
x
16 row(s) in 0.0910 seconds
```

Syntax:- list '<<tablename>>'

```
hbase(main):004:0> list 'SW_TECH'
TABLE
SW_TECH
1 row(s) in 0.0770 seconds
```

disable - Disables a table from making any changes to the data.

Before making any alternation to the table. We need to disable the table.

Syntax:- disable '<<tablename>>'

```
hbase(main):026:0> disable 'x'
0 row(s) in 2.0720 seconds
```

is_disabled - Verifies whether a table is disabled. Will return true if the table is disabled from performing any DML operations on the table.

Syntax:- is_disable '<<tablename>>'

```
hbase(main):009:0> is_disabled 'SW_TECH'
false
0 row(s) in 0.0130 seconds
```

enable - Enables a table. If table is in enabled mode then we cannot alter the table. However we can perform DML operations on the table.

Syntax:- enable '<<tablename>>'

```
hbase(main):004:0> enable 'SW_TECH'
1 row(s) in 0.0770 seconds
```

is_enabled - Verifies whether a table is enabled. Will return true. if the table is in enabled mode.

Syntax:- is_enable '<<tablename>>'

```
hbase(main):007:0> is_enabled 'SW_TECH'
true
0 row(s) in 0.0090 seconds
```

describe - Provides the description of a table.

Syntax:- describe '<<tablename>>'

```
hbase(main):006:0> describe 'SW_TECH'
  DESCRIPTION
ENABLED
 {NAME => 'SW_TECH', FAMILIES => [{NAME => 'CF_DB', BLOOMFILTER
=> 'NONE', REPLICATION_SCOPE => '0',COMPRES true
 SION => 'NONE', VERSIONS => '3', TTL => '2147483647', BLOCKSIZE =>
'65536', IN_MEMORY => 'false', BLOCKCACH
 E => 'true'}]}
  1 row(s) in 0.0570 seconds
```

alter - Alters a table. (Add new Column Family,Version etc)

If the table is not disabled. We will get an error.

Lets try to add new Column Family CF_PRG_LNG to the table 'SW_ ECH'

Syntax:- alter '<<tablename>>', {NAME=>>'<<columnfamily>>'}

```
hbase(main):010:0> alter 'SW_TECH', {NAME=>>'CF_PRG_LNG'}

ERROR: Table SW_TECH is enabled. Disable it first before altering.

Here is some help for this command:
Alter column family schema;  pass table name and a dictionary
specifying new column family schema. Dictionaries are described
on the main help command output. Dictionary must include name
of column family to alter. For example,

To change or add the 'f1' column family in table 't1' from defaults
to instead keep a maximum of 5 cell VERSIONS, do:

  hbase> alter 't1', NAME => 'f1', VERSIONS => 5

To delete the 'f1' column family in table 't1', do:

  hbase> alter 't1', NAME => 'f1', METHOD => 'delete'

or a shorter version:

  hbase> alter 't1', 'delete' => 'f1'

You can also change table-scope attributes like MAX_FILESIZE
MEMSTORE_FLUSHSIZE, READONLY, and DEFERRED_LOG_FLUSH.

For example, to change the max size of a family to 128MB, do:

  hbase> alter 't1', METHOD => 'table_att', MAX_FILESIZE => '134217728'

You can also remove a table-scope attribute:
```

```
hbase> alter 't1', METHOD => 'table_att_unset', NAME =>
'MAX_FILESIZE'

There could be more than one alteration in one command:
```

Steps to alter table i.e; Add new columnfamily 'CF_PRG_LNG' to the

table 'SW_TECH'

```
hbase(main):012:0> disable 'SW_TECH'
0 row(s) in 2.0870 seconds

hbase(main):013:0> is_enabled 'SW_TECH'
false
0 row(s) in 0.0230 seconds

hbase(main):014:0>                    is_disabled                    'SW_TECH'
        true
0 row(s) in 0.0150 seconds
   hbase(main):015:0> alter 'SW_TECH',{NAME=>>'CF_PRG_LNG'}
```

exists - Verifies whether a table exists.

Syntax:- alter '<<tablename>>', {NAME=>>'<<columnfamily>>'}

```
hbase(main):005:0> exists 'SW_TECH'
Table SW_TECH does exist
0 row(s) in 0.0700 seconds
```

drop - Drops a table from HBase.

To drop a table it should be enabled.

```
hbase(main):025:0> drop 'x'
ERROR: Table x is enabled. Disable it first.'

Here is some help for this command:
Drop the named table. Table must first be disabled. If table has
more than one region, run a major compaction on .META.: hbase>
major_compact ".META."
```

Follow the below steps to drop table 'x'

```
hbase(main):026:0> disable 'x'
0 row(s) in 2.0720 seconds

hbase(main):027:0> drop 'x'
0 row(s) in 1.2850 seconds
```

drop_all - Drops the tables matching the 'regex' given in the command.

Syntax:- drop_all '<<regex>>'

We are trying to delete all the tables starting with 'DUMMY'

```
hbase(main):004:0> drop_all 'DUMMY'
```

6.10.2.DML in HBASE

put - Puts a cell value at a specified column in a specified row in a particular table.

```
Syntax:- put '<<tablename>>' ,'<<rowkey>>'
,'<<columnfamily>:<columnqualifiername>>','<<value>>'hbase(main):009:0>
put 'SW_TECH','0001','CF_DB:SQL','MySQL'
0 row(s) in 0.0700 seconds

hbase(main):010:0> put 'SW_TECH','0002','CF_DB:SQL','Oracle'
0 row(s) in 0.0150 seconds

hbase(main):011:0> put 'SW_TECH','0003','CF_DB:SQL','Sybase'
0 row(s) in 0.0280 seconds
```

Tabular representation of above command execution result :- **SW_TECH**

	Column Family-DB (Database)	
RowKey	**CQN**-SQL	**CQN**-NoSQL
'0001	MySQL	Cassandra
'0002	Oracle	MongoDB
'0003	Sybase	HBASE

Let us check the content of the SW_TECH table for the inserted data using **scan**

```
hbase(main):012:0> scan 'SW_TECH'
ROW                                    COLUMN+CELL
 0001                                  column=CF_DB:SQL,
timestamp=1445672486706, value=MySQL
 0002                                  column=CF_DB:SQL,
timestamp=1445672512365, value=Oracle
 0003                                  column=CF_DB:SQL,
timestamp=1445672524619, value=Sybase
3 row(s) in 0.0600 seconds
```

get - Fetches the contents of row or a cell based on the attributes provided.

- To retrieve the data related to a specific row mention the rowkey.

Syntax:- get '<<tablename>>','<<rowkey>>'

- To retrieve the data related to a specific row and specific columnfamily mention the rowkey and columnfamily.

Syntax:- get '<<tablename>>','<<rowkey>>','<<columnfamily>>'

```
get 'SW_TECH','0002','CF_DB'
COLUMN                                CELL
 CF_DB:NoSQL                          timestamp=1445672741320,
value=MongoDB
 CF_DB:SQL timestamp=1445672512365, value=Oracle
2 row(s) in 0.1160 seconds
```

Tabular representation of above command :-**SW_TECH**

	Column Family-DB (Database)	
RowKey	**CQN**-SQL	**CQN**-NoSQL
'0001	MySQL	Cassandra
'0002	Oracle	MongoDB
'0003	Sybase	HBASE

To retrieve data of specific row and column data of specific version

Syntax:- get '<<tablename>>','<<rowkey>>', {

COLUMN=>'<<columnfamily>>:<<CQN>>',VERSIONS=><<versionnumber>>}

Tabular representation of above command :-**SW_TECH**

	Column Family-DB (Database)	
RowKey	**CQN**-SQL	**CQN**-NoSQL
'0001	MySQL	Cassandra
'0002	Oracle	MongoDB
'0003	Sybase	HBASE

```
hbase(main):052:0> get
'SW_TECH','0003',{COLUMN=>>'CF_DB:SQL',VERSIONS=>3}
COLUMN                                CELL
 CF_DB:SQL                            timestamp=1445675026048,
value=Sybase_0.5
 CF_DB:SQL                            timestamp=1445675023657,
value=Sybase_0.4
 CF_DB:SQL                            timestamp=1445675021083,
value=Sybase_0.3
3 row(s) in 0.0290 seconds
```

delete - Deletes a cell value in a table.

Lets scan the content of the 'SW_TECH'.We have 3 rows of data with 6 columns data.

```
hbase(main):026:0> scan 'SW_TECH'
ROW                                    COLUMN+CELL
 0001                                  column=CF_DB:NoSQL,
timestamp=1445672721490, value=Cassandra
 0001                                  column=CF_DB:SQL,
timestamp=1445672486706, value=MySQL
 0002                                  column=CF_DB:NoSQL,
timestamp=1445672741320, value=MongoDB
 0002                                  column=CF_DB:SQL,
timestamp=1445672512365, value=Oracle
 0003                                  column=CF_DB:NoSQL,
timestamp=1445672759483, value=HBase
 0003                                  column=CF_DB:SQL,
timestamp=1445672524619, value=Sybase
3 row(s) in 0.0930 seconds
```

Syntax:- delete '<<tablenam e>>','<<ColumnFamily>>:<<CQN>>'

```
hbase(main):028:0> delete 'SW_TECH','0003','CF_DB:SQL'
0 row(s) in 0.0120 seconds
```

Tabular representation of ab(ve command :-**SW_TECH**

	Column Family-DB (Database)	
RowKey	**CQN**-SQL	**CQN**-NoSQL
'0001	MySQL	Cassandra
'0002	Oracle	MongoDB
'0003	Sybase	HBASE

Now let's scan again and see the content of SW_TECH table.We have 3 rows with 5 columns data

```
hbase(main):029:0> scan 'SW_TECH'
ROW                                    COLUMN+CELL
 0001                                  column=CF_DB:NoSQL,
timestamp=1445672721490, value=Cassandra
 0001                                  column=CF_DB:SQL,
timestamp=1445672486706, value=MySQL
 0002                                  column=CF_DB:NoSQL,
timestamp=1445672741320, value=MongoDB
 0002                                  column=CF_DB:SQL,
timestamp=1445672512365, value=Oracle
 0003                                  column=CF_DB:NoSQL,
timestamp=1445672759483, value=HBase
3 row(s) in 0.0680 seconds
```

deleteall - Deletes all the cells in a given row.

```
hbase(main):004:0> put 'SW_TECH','0003','CF_DB:NoSQL','HBASE'
0 row(s) in 0.0430 seconds

hbase(main):005:0> scan 'SW_TECH'
ROW                    COLUMN+CELL
 0001                  column=CF_DB:NoSQL, timestamp=1445672721490,
value=Cassand
                       ra
 0001                  column=CF_DB:SQL, timestamp=1445672486706,
value=MySQL
 0002                  column=CF_DB:NoSQL, timestamp=1445672741320,
value=MongoDB
 0002                  column=CF_DB:SQL, timestamp=1445672512365,
value=Oracle
 0003                  column=CF_DB:NoSQL, timestamp=1445698805561,
value=HBASE
 0003                  column=CF_DB:SQL, timestamp=1445675026048,
value=Sybase_0.
                       5
3 row(s) in 0.0560 seconds
```

Syntax:- delete '<<tablename>>','<<rowkey>>'

```
hbase(main):006:0> deleteall 'SW_TECH','0003'
0 row(s) in 0.0200 seconds
```

Tabular Representation of the above command :-**SW_TECH**

	Column Family-DB (Database)	
RowKey	**CQN**-SQL	**CQN**-NoSQL
'0001	MySQL	Cassandra
'0002	Oracle	MongoDB
'0003	Sybase	HBASE

After delete lets scan the SW_TECH table. Since we have deleted rowkey 0003 we are left with 2 rows only.

```
hbase(main):007:0> scan 'SW_TECH'
ROW                    COLUMN+CELL
 0001                  column=CF_DB:NoSQL, timestamp=1445672721490,
value cassandra
 0001                  column=CF_DB:SQL, timestamp=1445672486706,
value=MySQL
 0002                  column=CF_DB:NoSQL, timestamp=1445672741320,
value=MongoDB
 0002                  column=CF_DB:SQL, timestamp=1445672512365,
value=Oracle
2 row(s) in 0.0380 seconds
```

scan - Scans and returns the table data.

Syntax:- scan '<<tablename>>'

```
hbase(main):007:0> scan 'SW_TECH'
ROW                    COLUMN+CELL
 0001                  column=CF_DB:NoSQL, timestamp=1445672721490,
value=Cassandra
 0001                  column=CF_DB:SQL, timestamp=1445672486706,
value=MySQL
 0002                  column=CF_DB:NoSQL, timestamp=1445672741320,
value=MongoDB
 0002                  column=CF_DB:SQL, timestamp=1445672512365,
value=Oracle
2 row(s) in 0.0380 seconds
```

count - Counts and returns the number of rows in a table.

Syntax: count '<<tablename>>'

```
hbase(main):005:0> count 'SW_TECH'
0 row(s) in 0.0280 seconds

hbase(main):006:0> count 'EMP_HBASE'
5 row(s) in 0.0480 seconds
```

truncate - Disables, drops, and recreates a specified table.

```
hbase(main):009:0> scan 'RK'
ROW                    COLUMN+CELL
 row1                  column=Hadoop:echosys1, timestamp=1357824008072,
value=MapReduce
 row2                  column=Hadoop:echosys2, timestamp=1357824029147,
value=Apache PIG
 row3                  column=Hadoop:echosys3, timestamp=1357824043192,
value=Hive
 row4                  column=Hadoop:echosys4, timestamp=1357824058518,
value=Sqoop
 row5                  column=Hadoop:echosys5, timestamp=1357824071877,
value=HBase
 row6                  column=Hadoop:echosys6, timestamp=1357824084803,
value=HDFS
6 row(s) in 0.1200 seconds
```

Syntax:- truncate '<<tablename>>'

```
hbase(main):010:0> truncate 'RK'
Truncating 'RK' table (it may take a while):
 - Disabling table...
 - Dropping table...
 - Creating table...
0 row(s) in 4.8160 seconds
```

Though we have truncated the table.HBASE re-creates the table with zero rows.

```
hbase(main):011:0> scan 'RK'
ROW                                    COLUMN+CELL
0 row(s) in 0.0450 seconds
hbase(main):012:0>
```

6.11.JAVA API for HBASE

Using Java Code we will try to perform CRUD operation of HBASE tables

We are going to use Eclipse IDE for developing the HBASE-Java API code.

1. In Eclipse File> New>>Java Project

   Provide the Project Name :- TestHBase

2. Click on Project Name>Src> New >Class

   Provide the Java Class Name :- HBaseCreateTable

3. Right click the src folder and **Build Path** > **Configure Build Path**

4. Using External Jar file tab ,provide the HBASE jar files and click **OK**

6.11.1.Create table in HBASE using java code

```
import java.io.IOException;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.HColumnDescriptor;
import org.apache.hadoop.hbase.HTableDescriptor;
import org.apache.hadoop.hbase.MasterNotRunningException;
import org.apache.hadoop.hbase.ZooKeeperConnectionException;
import org.apache.hadoop.hbase.client.HBaseAdmin;

public class HBaseCreateTable {
    public static void main(String[] args) throws IOException {
        //provide the Table Name
        String tableName = "DEPT_TABLE";
        // provide the Column Family Names
String[] familys = { "DEPT_DET", "DEPT_LOC", "DEPT_HIERARCHY" };
        // Configuration based on hbase-site.xml
  org.apache.hadoop.conf.Configuration hConf = HBaseConfiguration
                                .create();
        hConf.set("hbase.root.dir",
"localhost.localdomain:54310/hbase");
        hConf.set("hbase.zookeeper.quorum", "localhost");

        try {
        HBaseAdmin admin = new HBaseAdmin(hConf);
        //check if table already exists
        if (admin.tableExists(tableName)) {
        System.out.println("table already exists!");
        } else {
```

```
// if the table doesn't exist in the database,only then create the table
        HTableDescriptor tableDesc = new HTableDescriptor(tableName);
//iterate the ColumnFamily list provided above
        for (int i = 0; i < familys.length; i++) {
                tableDesc.addFamily(new HColumnDescriptor(familys[i]));
                }
                admin.createTable(tableDesc);
                System.out.println("create table " + tableName + " ok.");
                }
                }catch (MasterNotRunningException mnre) {
                        mnre.printStackTrace();
                } catch (ZooKeeperConnectionException zkce) {
                        zkce.printStackTrace();
                } catch (Exception e) {
                        e.printStackTrace();
                } finally {

                }

        }
}
```

Eclipse Output showing the Table got created

```
15/10/30 [illegible]
15/10/30 [illegible]
15/10/30 [illegible]
15/10/30 [illegible]
15/10/30 [illegible]
15/10/30 [illegible]
15/10/30 [illegible]
15/10/30 [illegible]
15/10/30 [illegible]
15/10/30 [illegible]
15/10/30 [illegible]
15/10/30 [illegible]
15/10/30 [illegible]
15/10/30 [illegible]
create table DEPT_TABLE ok.
```

Let us access HBASE shell and verify if the table DEPT_TABLE got created.

hbase> list

hbase > desc 'DEPT_TABLE'

```
hbase(main):021:0> list
TABLE
CUST_DET
DEPT_TABLE
EMP
EMPTAB
dept
emp
hem
7 row(s) in 0.0850 seconds

=> ["CUST_DET", "DEPT_TABLE", "EMP", "EMPTAB", "dept", "emp", "hem"]
hbase(main):022:0> desc 'DEPT_TABLE'
DESCRIPTION
 'DEPT_TABLE', {NAME => 'DEPT_DET', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICAT
 > '0', VERSIONS => '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL => 'FOREVER', KEEP_DELET
  'false', BLOCKSIZE => '65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}, {NAME => 'DEPT_HIER
 A_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', VERSIONS => '1', COM
  'NONE', MIN_VERSIONS => '0', TTL => 'FOREVER', KEEP_DELETED_CELLS => 'false', BLOCKSIZE => '655
 ORY => 'false', BLOCKCACHE => 'true'}, {NAME => 'DEPT_LOC', DATA_BLOCK_ENCODING => 'NONE', BLOOM
 ROW', REPLICATION_SCOPE => '0', VERSIONS => '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL
 R', KEEP_DELETED_CELLS => 'false', BLOCKSIZE => '65536', IN_MEMORY => 'false', BLOCKCACHE => 'tr
```

Delete a HBASE table using java code

Create table TEMP_TABLE in the HBASE so that we can delete this table using java code

> hbase shell

> create 'TEMP_TABLE','xmldata'

> desc 'TEMP_TABLE'

```
hbase(main):023:0> create 'TEMP_TABLE', 'xmldata'
0 row(s) in 0.4280 seconds

=> Hbase::Table - TEMP_TABLE
hbase(main):024:0> desc 'TEMP_TABLE'
DESCRIPTION                                                                                    ENA
 'TEMP_TABLE', {NAME => 'xmldata', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0',
  0', VERSIONS => '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL => 'FOREVER', KEEP_DELETED_CELLS =>
  'false', BLOCKSIZE => '65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}
1 row(s) in 0.0440 seconds

hbase(main):025:0> put 'TEMP_TABLE','0001','xmldata:hbase-site.xml', '<key>hbase.root.dir</key><value>localhost:1
0 row(s) in 0.0660 seconds

hbase(main):026:0> scan 'TEMP_TABLE'
ROW                                    COLUMN+CELL
 0001                                  column=xmldata:hbase-site.xml, timestamp=1446191140156, value=<key>
                                       ase.rootdir</key>
1 row(s) in 0.0160 seconds
```

We are going to use Eclipse IDE for developing the HBASE-Java API code.

1. In Eclipse File> New>>Java Project

Provide the Project Name :- TestHBase

2. Click on Project Name>Src> New >Class

   Provide the Java Class Name :- HBaseDeleteTable

3. Right click the src folder and **Build Path > Configure Build Path**

4. Using External Jar file tab ,provide the HBASE jar files and click **OK**

```
import java.io.IOException;

import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.MasterNotRunningException;
import org.apache.hadoop.hbase.ZooKeeperConnectionException;
import org.apache.hadoop.hbase.client.HBaseAdmin;
//This class will delete the HBASE table
public class HBaseDeleteTable {

public static void main(String[] args) throws IOException {
String tableName = "TEMP_TABLE";// mention the table name that
need to  be deleted
try {
            org.apache.hadoop.conf.Configuration hConf =
                    HBaseConfiguration.create();
      hConf.set("hbase.root.dir",
"localhost.localdomain:54310/hbase");
      hConf.set("hbase.zookeeper.quorum", "localhost");
      HBaseAdmin hadmin = new HBaseAdmin(hConf);
      // We need to disable table to make any structural changes
      hadmin.disableTable(tableName);
      // As table is in disable state we can execute DML statements
      hadmin.deleteTable(tableName);
      System.out.println("deleted table " + tableName);
      // We can write the code here to describe the table/ select from
table
      // which will throw execption as the table does not exist
            } catch (MasterNotRunningException mnre) {
                  mnre.printStackTrace();
            } catch (ZooKeeperConnectionException zkce) {
                  zkce.printStackTrace();
            } catch (Exception e) {
                  e.printStackTrace();
            }
      }
}
```

5. Check the HBASE for the 'TEMP_TABLE' using below command

6. desc 'TEMP_TABLE'

7. we will get ERROR unknow table TEMP_TABLE.

```
hbase(main):028:0> desc 'TEMP_TABLE'

ERROR: Unknown table TEMP_TABLE

Here is some help for this command:
Describe the named table. For example:
  hbase> describe 't1'
  hbase> describe 'ns1:t1'

Alternatively, you can use the abbreviated 'desc' for the same thing.
  hbase> desc 't1'
  hbase> desc 'ndsl:t1'

hbase(main):029:0>
```

6.11. 2.Scan Table for first 3 rows using Java Code

We are going to use Eclipse IDE for developing the HBASE-Java API code.

1. In Eclipse File> New>>Java Project

Provide the Project Name :- TestHBase

2. Click on Project Name>Src> New >Class

Provide the Java Class Name :- HBaseScanClient

3. Right click the src folder and Build Path > Configure Build Path

4. Using External Jar file tab ,provide the HBASE jar files and click OK

5. In the code provide the TableName,Column Family,CQN.

6. Compile the Java Code and Run as Java Application.

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.MasterNotRunningException;
import org.apache.hadoop.hbase.ZooKeeperConnectionException;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.hadoop.hbase.client.Result;
import org.apache.hadoop.hbase.client.ResultScanner;
import org.apache.hadoop.hbase.client.Scan;
import org.apache.hadoop.hbase.util.Bytes;

public class HBaseScanClient {

    public static void main(String[] arg) throws IOException {
        try {
        Configuration config = HBaseConfiguration.create();
        config.set("hbase.rootdir",
"localhost.localadmin:54310/hbase");
        config.set("hbase.zookeeper.quorum", "localhost");
            // provide the table name which we need to scan
```

```
            HTable custTable = new HTable(config, "CUST_DET");

            for (int i = 0; i < 3; i++) {
            // provide the ColumnFamilyName
            byte[] colfamily = Bytes.toBytes("CUST_NAME");
            // provide the ColumnQualifierName
            byte[] colqualname = Bytes.toBytes("FNAME");

            Scan scan = new Scan();
            scan.addColumn(colfamily, colqualname);
            ResultScanner rs = custTable.getScanner(scan);
            for (Result r = rs.next(); r != null; r = rs.next()) {
        // valueObj will have all the 3 rows related FirstName's of
        // the Customer
            byte[] valueObj = r.getValue(colfamily, colqualname);
            String value = new String(valueObj);
            System.out.println(value);
                        }
                    }

                    custTable.close();
            } catch (MasterNotRunningException e) {
                    e.printStackTrace();
            } catch (ZooKeeperConnectionException e) {
                    e.printStackTrace();
            } catch (Exception e) {
                    e.printStackTrace();
            }

        }
    }
```

Eclipse Output showing the Column "firstname" of the CUST_DET table :-

```
Krishna
Raju
Sandhya
Krishna
Raju
Sandhya
Krishna
Raju
Sandhya
```

6.11.3.Using Java Code scan and retrieve whole table content

We are going to use Eclipse IDE for developing the HBASE-Java API code.

1. In Eclipse File> New>>Java Project

   Provide the Project Name :- TestHBase

2. Click on Project Name>Src> New >Class

   Provide the Java Class Name :- HbaseScanTable

3. Right click the src folder and Build Path > Configure Build Path

4. Using External Jar file tab ,provide the HBASE jar files and click OK

5. Provide the Table Name only in the code

6. Compile the Java Code and Run as Java Application.

```
import java.io.IOException;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.KeyValue;
import org.apache.hadoop.hbase.MasterNotRunningException;
import org.apache.hadoop.hbase.ZooKeeperConnectionException;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.hadoop.hbase.client.Result;
import org.apache.hadoop.hbase.client.ResultScanner;
import org.apache.hadoop.hbase.client.Scan;
public class HBaseScanTable {
	public static void main(String[] args) throws IOException {
		try {
org.apache.hadoop.conf.Configuration hConf =
HBaseConfiguration.create();
hConf.set("hbase.root.dir", "localhost.localdomain:54310/hbase");
hConf.set("hbase.zookeeper.quorum", "localhost");
 // Provide the table name that we need to scan
HTable table = new HTable(hConf, "CUST_DET");
System.out.println("getting all records of Customer Information table
");
Scan s = new Scan();
ResultScanner rs = table.getScanner(s);
// we are retrieving the whole table data here i.e; all records of table
for (Result res : rs) {
for (KeyValue kv : res.raw()) {
	System.out.print(new String(kv.getRow())) + " ");
	System.out.print(new String(kv.getFamily()) + ":");
	System.out.print(new String(kv.getQualifier())) + " ");
	System.out.print(kv.getTimestamp() + " ");
	System.out.println(new String(kv.getValue())):
		}//for
	}//for
		} catch (MasterNotRunningException e) {
			e.printStackTrace();
		} catch (ZooKeeperConnectionException e) {
			e.printStackTrace();
		} catch (Exception e) {
			e.printStackTrace();
		}
	}
```

```
}
```

Eclipse Output Showing the content of the CUST_DET table

```
getting all records of Customer Information table
0001 CUST_NAME:FNAME 1446185368822 Krishna
0001 CUST_NAME:LNAME 1446185417652 Ch
0001 CUST_NAME:MNAME 1446185398998 Kanth
0002 CUST_NAME:FNAME 1446185441342 Raju
0002 CUST_NAME:LNAME 1446185455964 M
0003 CUST_NAME:FNAME 1446185487691 Sandhya
0003 CUST_NAME:LNAME 1446185529749 A
0003 CUST_NAME:MNAME 1446185500684 Rani
```

6.11.4.Using Java Code scan and retrieve Rowkey based table content

We are going to use Eclipse IDE for developing the HBASE-Java API code.

1. In Eclipse File> New>>Java Project

   Provide the Project Name :- TestHBase

2. Click on Project Name>Src> New >Class
3. Provide the Java Class Name :- HbaseGetRowData
4. Right click the src folder and **Build Path > Configure Build Path**
5. Using External Jar file tab ,provide the HBASE jar files and click **OK**
6. Provide the Table Name and Rowkey in the code
7. Compile the Java Code and Run as Java Application.

```
import java.io.IOException;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.KeyValue;
import org.apache.hadoop.hbase.MasterNotRunningException;
import
org.apache.hadoop.hbase.ZooKeeperConnectionException;
import org.apache.hadoop.hbase.client.Get;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.hadoop.hbase.client.Result;
import org.apache.hadoop.hbase.util.Bytes;
public class HBaseGetRowData {
    public static void main(String[] args) throws IOException {
```

```
            try {
org.apache.hadoop.conf.Configuration hConf =
HBaseConfiguration.create();
hConf.set("hbase.root.dir", "localhost.localdomain:54310/hbase");
hConf.set("hbase.zookeeper.quorum", "localhost");
// Provide the table name
HTable table = new HTable(hConf, "CUST_DET");
System.out.println("getting Customer Information of Row 0001");
// We want to retrieve the data pertaining to RowKey =0001
Get get = new Get(Bytes.toBytes("0001"));
Result rs = table.get(get);
for (KeyValue kv : rs.raw()) {
      System.out.print(new String(kv.getRow()) + " ");
      System.out.print(new String(kv.getFamily()) + ":");
      System.out.print(new String(kv.getQualifier()) + " ");
      System.out.print(kv.getTimestamp() + " ");
      System.out.println(new String(kv.getValue()));
                  }

            } catch (MasterNotRunningException e) {
                  e.printStackTrace();
            } catch (ZooKeeperConnectionException e) {
                  e.printStackTrace();
            } catch (Exception e) {
                  e.printStackTrace();
            }

      }
}
```

Eclipse Output showing the RowKey "0001" data of the CUST_DET table :-

```
getting Customer Information of Row 0001
0001 CUST_NAME:FNAME 1446185368822 Krishna
0001 CUST_NAME:LNAME 1446185417652 Ch
0001 CUST_NAME:MNAME 1446185398998 Kanth
```

6.11.5.Using Java code to retrieve table content for a specific ColumnFamily:ColumnQualifierName

We are going to use Eclipse IDE for developing the HBASE-Java API code.

1. In Eclipse File> New>>Java Project

   Provide the Project Name :- TestHBase

2. Click on Project Name>Src> New >Class

Provide the Java Class Name :- HbaseReadTableNew

3. Right click the src folder and Build Path > Configure Build Path
4. Using External Jar file tab ,provide the HBASE jar files and click OK
5. Provide the Table Name only in the code
6. Compile the Java Code and Run as Java Application.

```
import java.io.IOException;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.hadoop.hbase.client.Result;
import org.apache.hadoop.hbase.client.ResultScanner;
import org.apache.hadoop.hbase.client.Scan;
import org.apache.hadoop.hbase.util.Bytes;
public class HbaseReadTableNew {
    public static void main(String[] args) throws IOException {
org.apache.hadoop.conf.Configuration hConf =
HBaseConfiguration.create();
hConf.set("hbase.root.dir", "localhost.localdomain:54310/hbase");
hConf.set("hbase.zookeeper.quorum", "localhost");
// Provide the table name from which we want to retrieve the data
HTable table = new HTable(hConf, "CUST_DETAIL");
Scan scan = new Scan();
ResultScanner resultScanner = table.getScanner(scan);
System.out.println("Customer Information");
try {
for (Result rr = resultScanner.next(); rr != null; rr =
resultScanner.next()) {
//provide the ColumnFamilyName and ColumnQualifierName
String fname =
Bytes.toString(rr.getValue(Bytes.toBytes("CUST_NAME"),
Bytes.toBytes("FNAME")));
String mname =
Bytes.toString(rr.getValue(Bytes.toBytes("CUST_NAME"),
Bytes.toBytes("MNAME")));
String lname =
Bytes.toString(rr.getValue(Bytes.toBytes("CUST_NAME"),
Bytes.toBytes("LNAME")));
System.out.println(lname + " " + fname + " " + mname);
}
} finally {
        resultScanner.close();
```

```
            }

        }
    }
```

Compile the Java Code and Run as Java Application.

Eclipse Output showing the all the columns data of the ColumnFamily "CUST_NAME" of the CUST_DET table :-

```
Customer Information
Ch Krishna Kanth
M Raju null
A Sandhya Rani
```

6.11.6.Using Java code insert data into RowKey "0004" of the table "CUST_DET"

We are going to use Eclipse IDE for developing the HBASE-Java API code.

1. In Eclipse File> New>>Java Project

   Provide the ProjectName :- TestHBase

2. Click on Project Name>Src> New >Class

   Provide the Java Class Name :- HbaseInsertData

3. Right click the src folder and Build Path > Configure Build Path

4. Using External Jar file tab ,provide the HBASE jar files and click OK

5. Provide the Table Name , RowKey,Column Family, Column Qualifier Name and Value in the code

6. Compile the Java Code and Run as Java Application.

```
import java.io.IOException;
import org.apache.hadoop.hbase.HBaseConfiguration.
import org.apache.hadoop.hbase.MasterNotRunningException;
import org.apache.hadoop.hbase.ZooKeeperConnectionException;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.util.Bytes;
```

```
public class HBaseInsertData {
    public static void main(String[] args) throws IOException {
        try {
        org.apache.hadoop.conf.Configuration hConf =
HBaseConfiguration.create();
    hConf.set("hbase.root.dir",
"localhost.localdomain:54310/hbase");
    hConf.set("hbase.zookeeper.quorum", "localhost");
    // provide the table name where we want to insert data
    HTable table = new HTable(hConf, "CUST_DET");
    System.out.println("inserting Customer Information");
    // we are inserting rowkey "0004" into the CUST_DET table
    Put put = new Put(Bytes.toBytes("0004"));
    // provide the ColumnFamilyName and ColumnQualifierName
and Value
    put.add(Bytes.toBytes("CUST_NAME"),
Bytes.toBytes("FNAME"), Bytes.toBytes("RamKrishna"));
    table.put(put);
    System.out.println("inserted recored ");
        } catch (MasterNotRunningException e) {
            e.printStackTrace();
        } catch (ZooKeeperConnectionException e) {
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

HBASE Shell commands to view the content of 'CUST_DET'

Execute the command **scan 'CUST_DET'** in hbase shell

We can see that RowKey 0004 related data is inserted.

```
hem@localhost:~
File  Edit  View  Search  Terminal  Help
0003                          column=CUST NAME:FNAME, timestamp=1446185487691, value=Sandhya
0003                          column=CUST NAME:LNAME, timestamp=1446185529749, value=A
0003                          column=CUST NAME:MNAME, timestamp=1446185500684, value=Rani
3 row(s) in 0.0420 seconds

hbase(main):018:0> scan 'CUST DET'
ROW                           COLUMN+CELL
 0001                         column=CUST NAME:FNAME, timestamp=1446185368822, value=Krishna
 0001                         column=CUST NAME:LNAME, timestamp=1446185417652, value=Ch
 0001                         column=CUST NAME:MNAME, timestamp=1446185398998, value=Kanth
 0002                         column=CUST NAME:FNAME, timestamp=1446185441342, value=Raju
 0002                         column=CUST NAME:LNAME, timestamp=1446185455964, value=M
 0003                         column=CUST NAME:FNAME, timestamp=1446185487691, value=Sandhya
 0003                         column=CUST NAME:LNAME, timestamp=1446185529749, value=A
 0003                         column=CUST NAME:MNAME, timestamp=1446185500684, value=Rani
3 row(s) in 0.0220 seconds

hbase(main):019:0> scan 'CUST DET'
ROW                           COLUMN+CELL
 0001                         column=CUST NAME:FNAME, timestamp=1446185368822, value=Krishna
 0001                         column=CUST NAME:LNAME, timestamp=1446185417652, value=Ch
 0001                         column=CUST NAME:MNAME, timestamp=1446185398998, value=Kanth
 0002                         column=CUST NAME:FNAME, timestamp=1446185441342, value=Raju
 0002                         column=CUST NAME:LNAME, timestamp=1446185455964, value=M
 0003                         column=CUST NAME:FNAME, timestamp=1446185487691, value=Sandhya
 0003                         column=CUST NAME:LNAME, timestamp=1446185529749, value=A
 0003                         column=CUST NAME:MNAME, timestamp=1446185500684, value=Rani
 0004                         column=CUST NAME:FNAME, timestamp=1446187417888, value=Krishna
```

6.12.HBASE_MR Integration

Purpose of this document: -

The objective of this document is to process data provided in a file with tab separator using Map Reduce and store the processed data in HBASE.

Problem Statement:-

Read the student marks file content and using MapReduce code check for the students with highest score and gender and store them in the Scholarship table of HBase

Input File Content:-

Students_marks.log (tab separated data. Please find the input data file as attachment)

Vasantha	27	female	98
Krishna	38	male	63
Varma	27	male	87
Bharathi	32	female	82
Daniel	78	male	85
Chandrika	31	male	78

Expect Output:-

In MapReduce code,based on Gender find out highest scored student details .Store the output of the MapReduce into the tables .

Sample data in SCHOLARSHIP_DET TABLE in HBase

	STUDENT_DET			
RowKey	STUDENTS_NAME	AGE	GENDER	MARKS
001	Vasantha	27	Female	98
002	Varma	27	Male	87

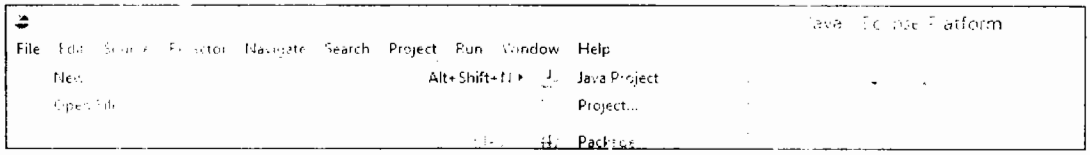
1.Open the Eclipse IDE

1.1 On Windows Environment Open Eclipse IDE using Eclipse Shortcut
Or
1.2 On Ubuntu Terminal enter eclipse command

```
gopal@ubuntu:~$ start-hbase.sh
localhost: starting zookeeper, logging to /usr/local/hbase-0.98.6.1-hadoop1/bin/../logs/hbase-gopal-zookeeper-ubun
starting master, logging to /usr/local/hbase-0.98.6.1-hadoop1/logs/hbase-gopal-master-ubuntu.out
localhost: starting regionserver, logging to /usr/local/hbase-0.98.6.1-hadoop1/bin/../logs/hbase-gopal-regionserve
gopal@ubuntu:~$ jps
5536 Jps
3449 DataNode
3774 JobTracker
5155 HQuorumPeer
3683 SecondaryNameNode
5479 HRegionServer
4608 TaskTracker
5217 HMaster
3215 NameNode
gopal@ubuntu:~$ eclipse
org.eclipse.m2e.logback.configuration: The org.eclipse.m2e.logback.configuration bundle was activated before the s
  Will retry after the state location is initialized.
org.eclipse.m2e.logback.configuration: Logback config file: /home/gopal/workspace/.metadata/.plugins/org.eclipse.m
ck.1.5.0.20140606-0033.xml
org.eclipse.m2e.logback.configuration: Initializing logback
java.lang.ClassCastException: org.eclipse.osgi.internal.framework.EquinoxConfiguration$1 cannot be cast to java.la
        at org.eclipse.m2e.logback.configuration.LogHelper.logJavaProperties(LogHelper.java:26)
        at org.eclipse.m2e.logback.configuration.LogPlugin.loadConfiguration(LogPlugin.java:189)
        at org.eclipse.m2e.logback.configuration.LogPlugin.configureLogback(LogPlugin.java:144)
        at org.eclipse.m2e.logback.configuration.LogPlugin.access$2(LogPlugin.java:107)
        at org.eclipse.m2e.logback.configuration.LogPlugin$1.run(LogPlugin.java:62)
        at java.util.TimerThread.mainLoop(Timer.java:555)
        at java.util.TimerThread.run(Timer.java:505)
```

2.Create New Java Project

Select FILE > New > Java Project from the Eclipse IDE Menu bar



3.Enter the Java Project Name as TestHBase

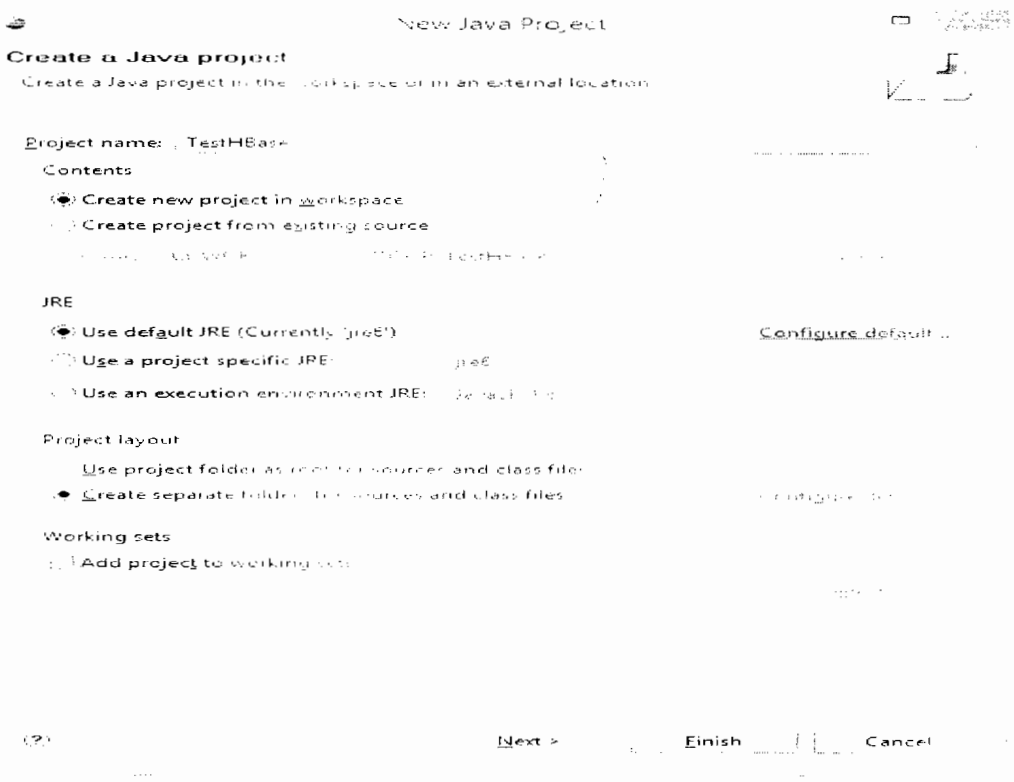
- **Under JRE section**

Select we are using default JRE

- **Under Project Layout**

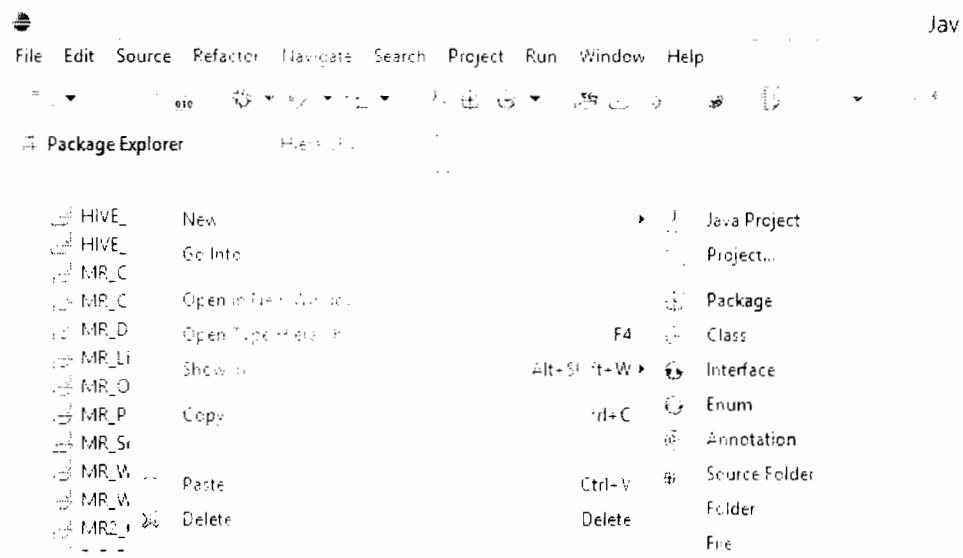
Select Create separate folders for sources and class files

- Click **Finish**



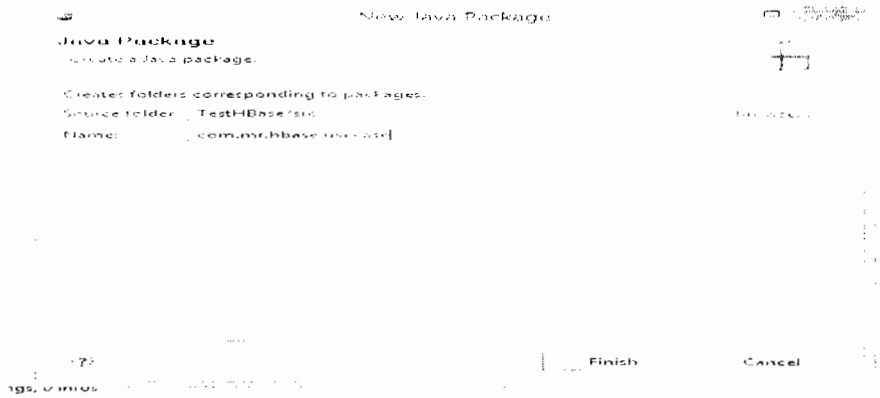
4.Create package

Select the Java Project TestHBase >Right Click > New > Package



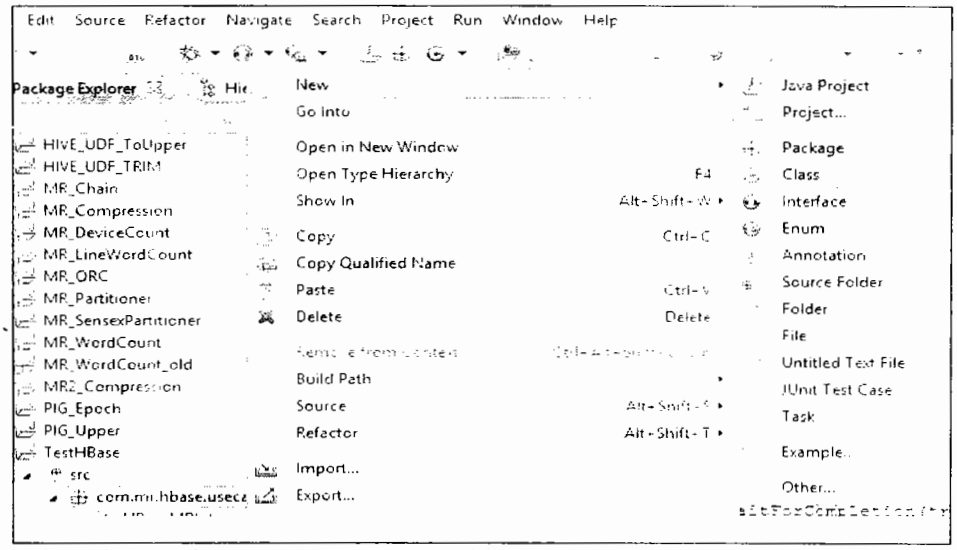
Provide the Package Name

com.mr.hbase.usecase and click **Finish**



5.1 We can view that package name under the Java Project

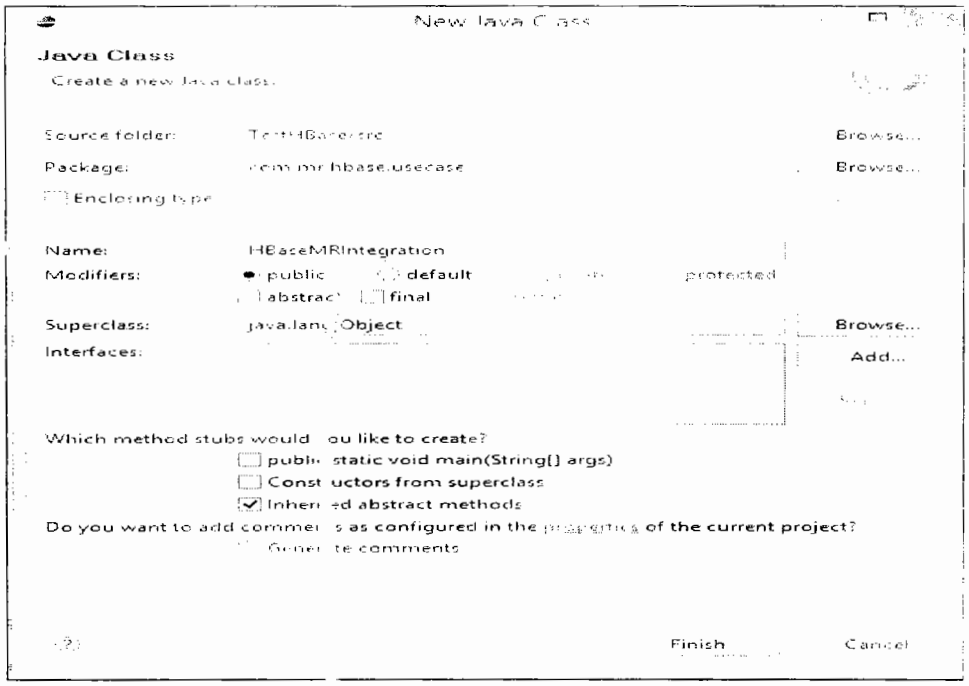
6.Create a Java file to Integrate MR with HBASE



Right click the package name "com.mr.hbase.usecase " > New > Class

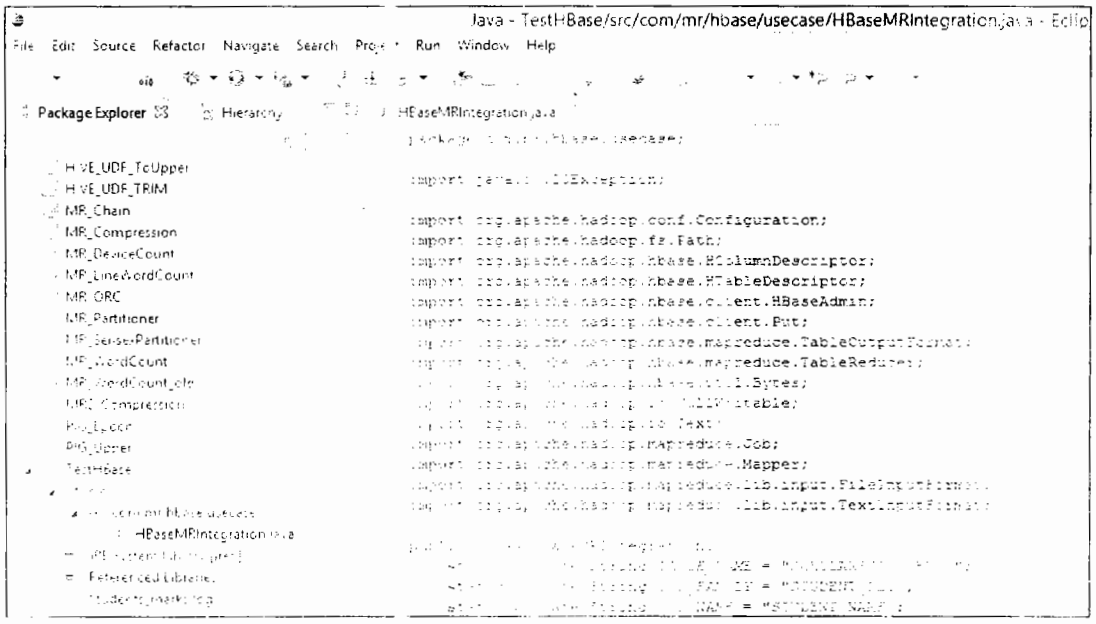
7. Provide the Java Class Name -HBaseMRIntegration

8.Write the Java Code



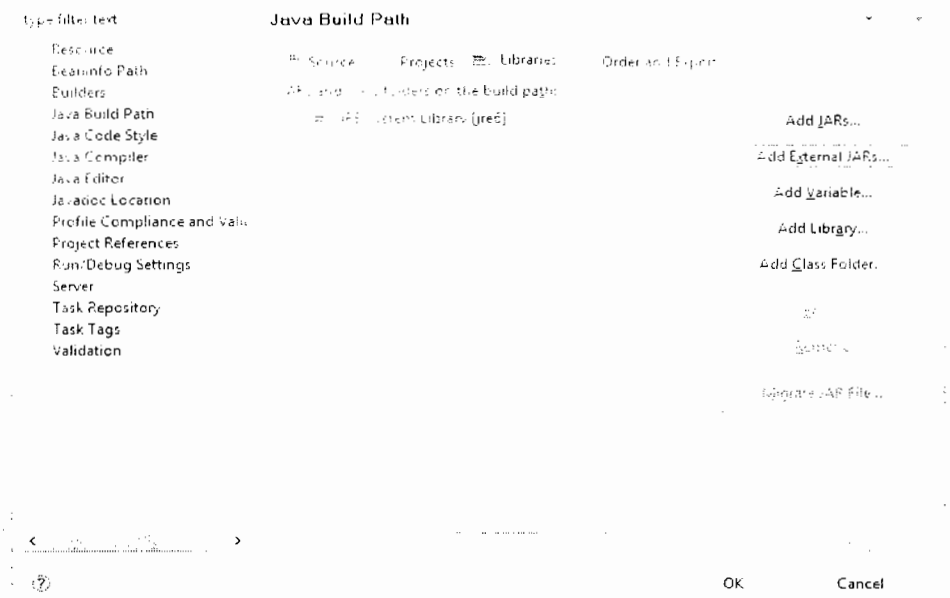
Copy the HbaseMRIntegration.java code provided in section 20 of this document and paste in the java file and save the code by clicking **Control+S**.

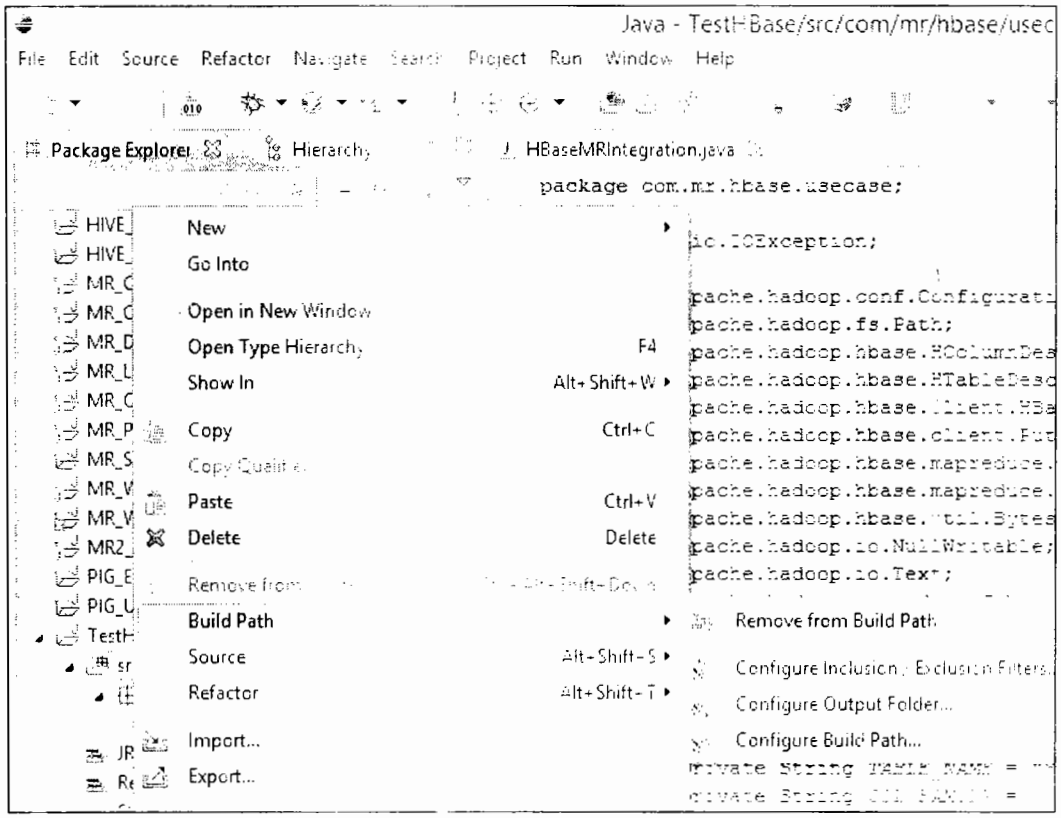
After saving the java code we can view lot of errors marked in red on the java code.



9.Fix the code error

In order to fix the error displayed on the java code





Select the src folder > Right Click > Build Path > Configure Build Path

10. Add External Jar file

Click the Libraries >> Add External Jars

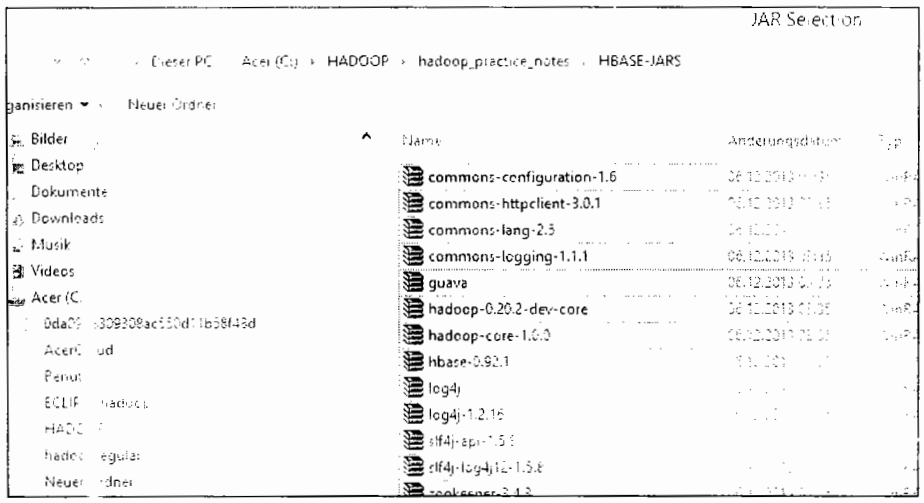
11.Select jar file location

Go the folder where the HBASE jar files are placed

or

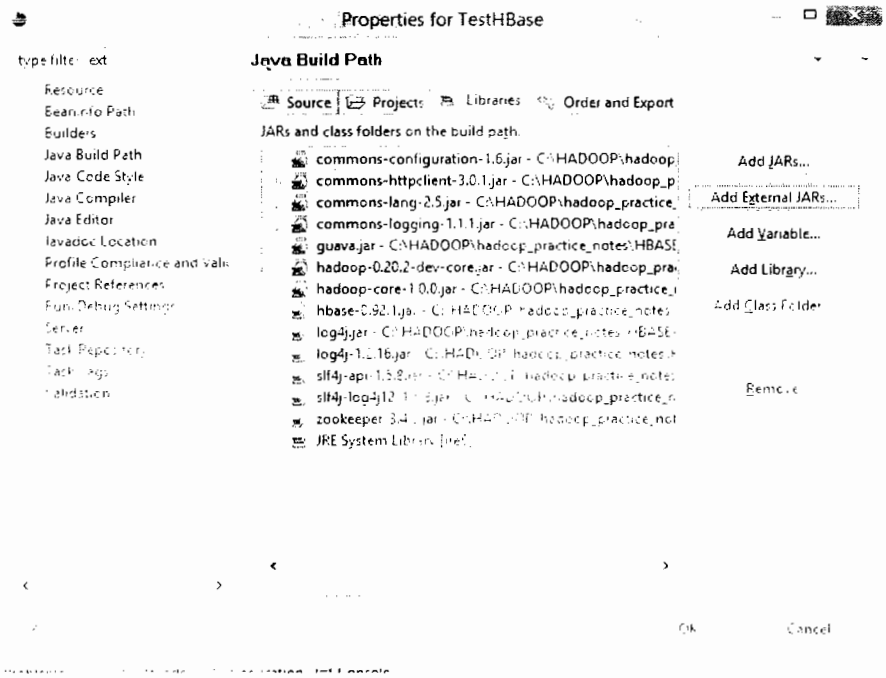
If you are using the eclipse in the ubuntu terminal

then select the jar files under the folder /usr/local/hadoop/hive/lib



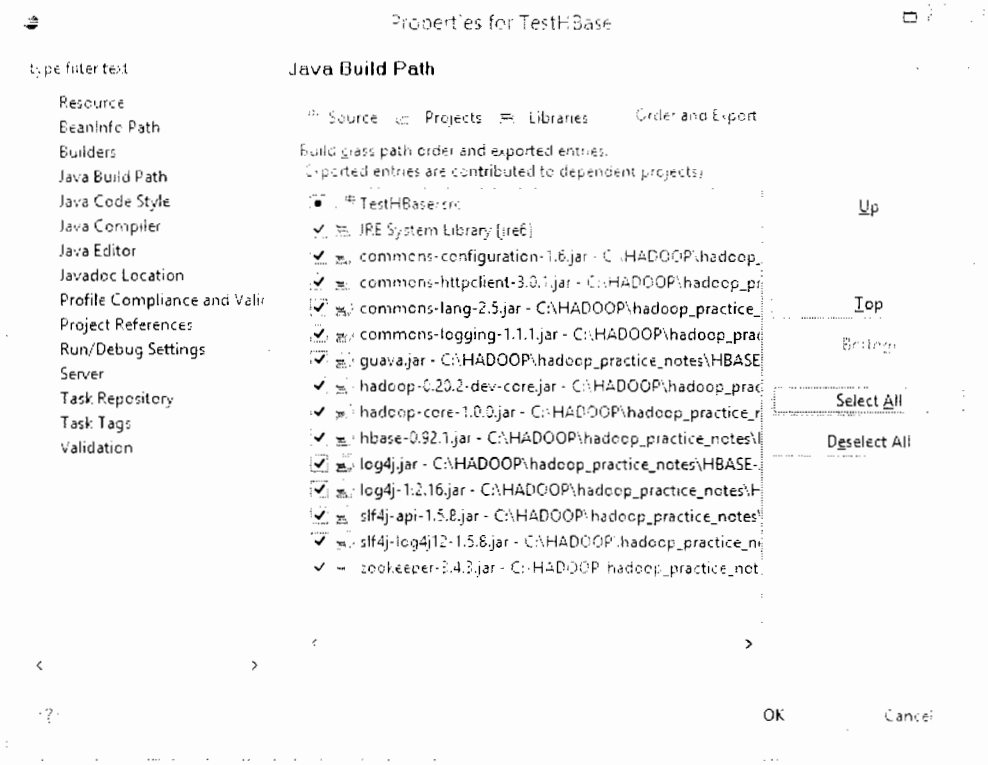
12.Add the jars to the source code

After all the selected jar files are add click **ok**



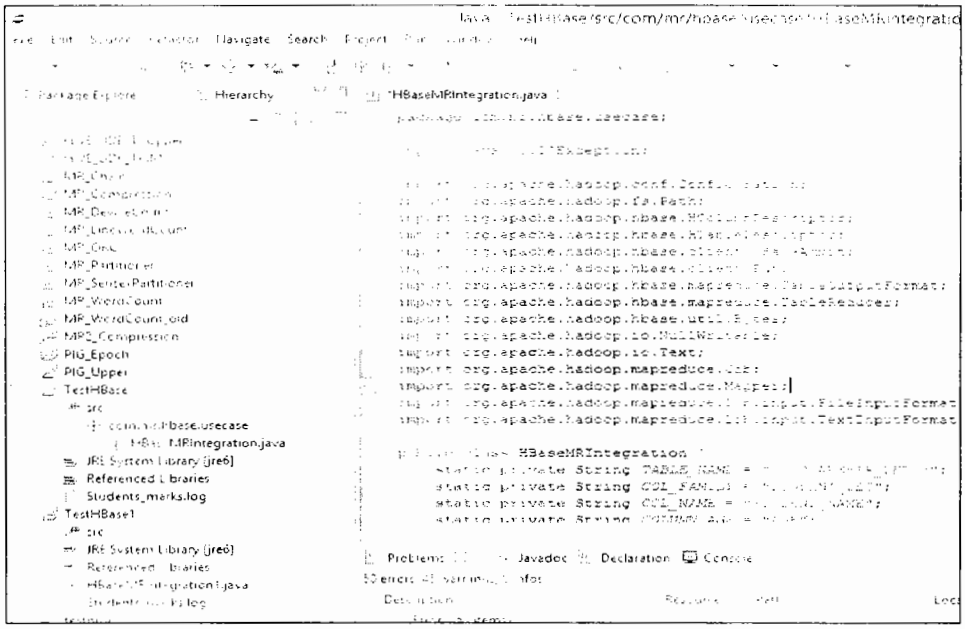
13.Select the Order and Export Tab

Select All and Click **OK**



14. View the Source code for any errors

Once we complete this task we can see that the java source code is without any errors .



15. Driver code important points

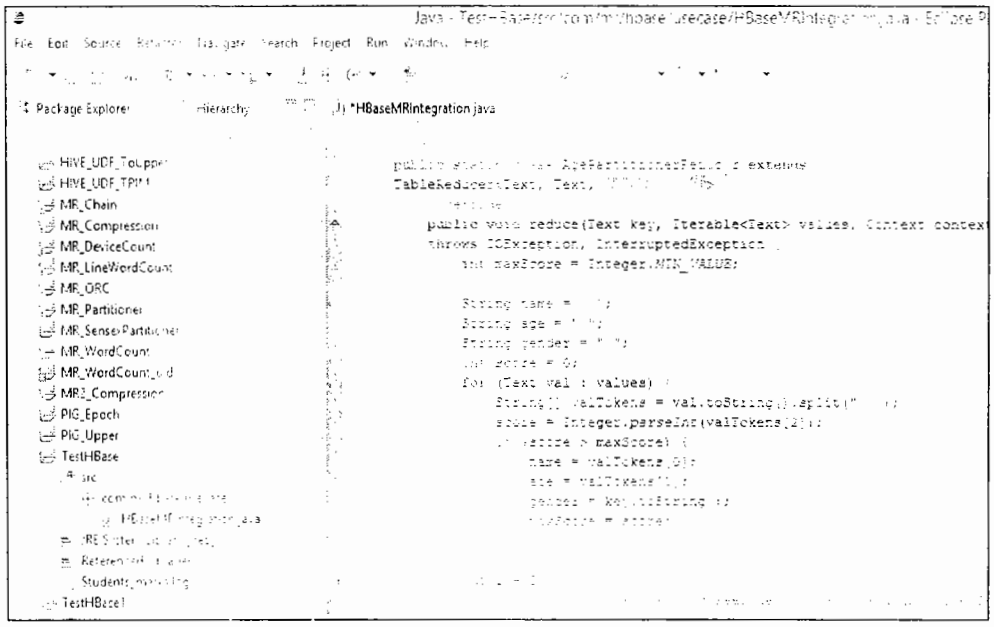
In the Driver Code ,ensure that the Configuration is set based on the **hbase-site.xml**

configuration setting for **hbase.root.dir** and **hbase.zookeeper.quorum** properties should be picked from the **hbase-site.xml**

```
public static void main(String args[]) throws Exception {
    if (args.length < 1) {
        System.err.println("Usage: HBaseAlbum <path> ");
        System.exit(-1);
    }

    Configuration conf = new Configuration();
    conf.set("hbase.rootdir", "[illegible]");
    conf.set("hbase.zookeeper.quorum", "[illegible]");
    [illegible] TableOutputFormat [illegible]
    conf.set(TableOutputFormat.OUTPUT_TABLE, TABLE_NAME);

    HBaseAdmin admin = new HBaseAdmin(conf);
```

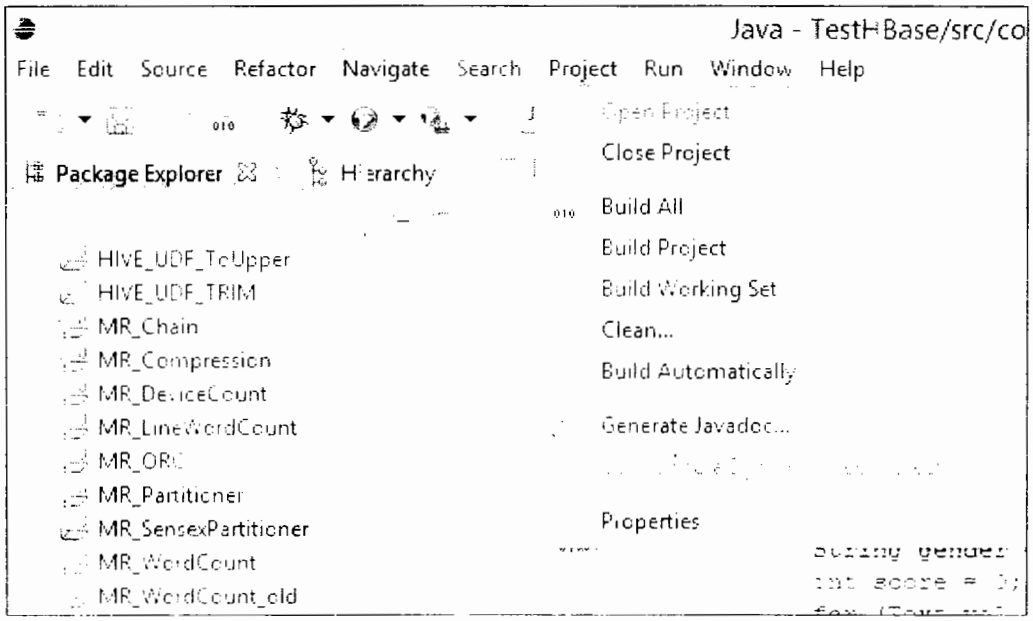
16. Reducer code

Ensure that the output parameter for the Reducer is set to NullWritable



17. Compile the source code

Eclipse IDE Menu ..**Project** > **Clean**

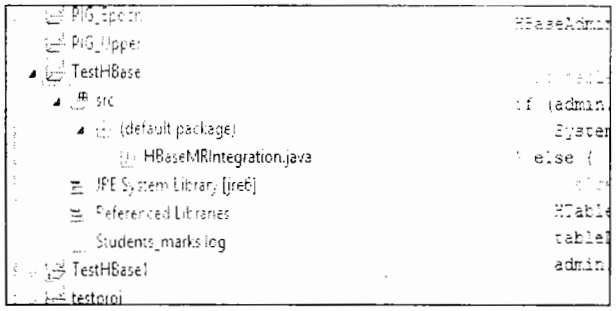


6.13.Execute the java code

We can test the java code in 2 methods
i) using Eclipse in ubuntu
ii) using windows Eclipse

6.13.1.Method 1:When using Eclipse in Ubuntu Terminal

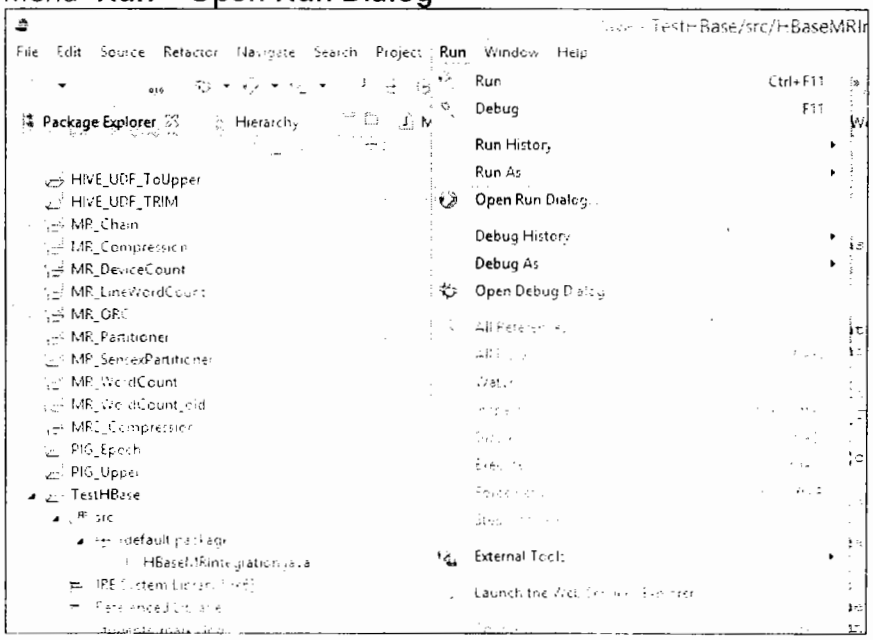
Place the Input file directly under the Java Project (not under the src folder)



2. Configure Input files

We have to provide the Input file while executing the java code

Eclipse IDE Menu **Run > Open Run Dialog**



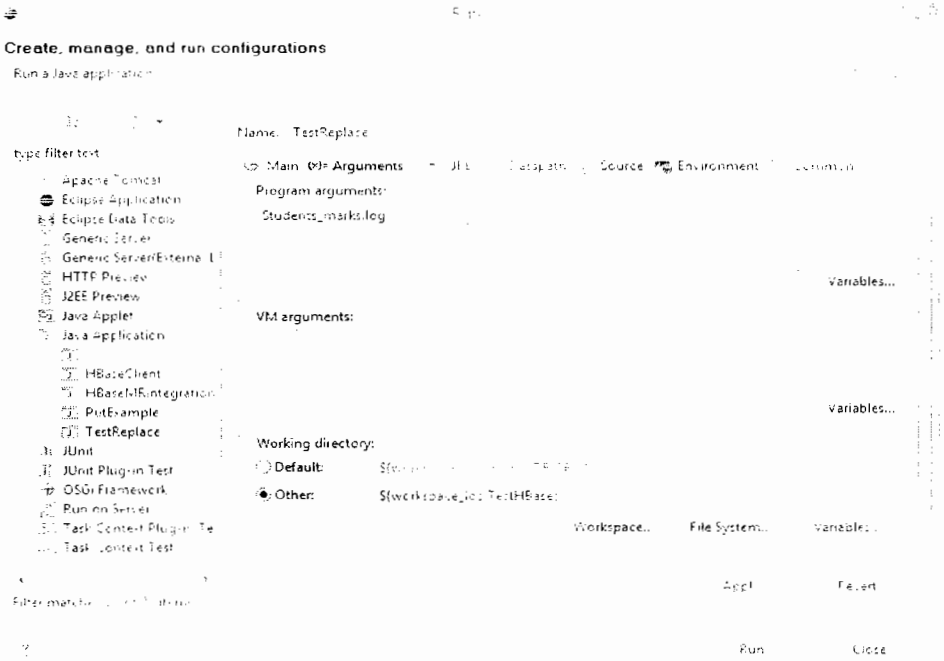
3 Provide input arguments to the java code

In the Arguments tab provide the Input file name **Students_marks.log**

Click Apply and Run

This is the same file that we have placed under the Java Project

After this steps ,the jave code gets executed and we can view the result in the Eclipse console



16.13.2.Method 2:If the java code is placed on Windows based Eclipse IDE

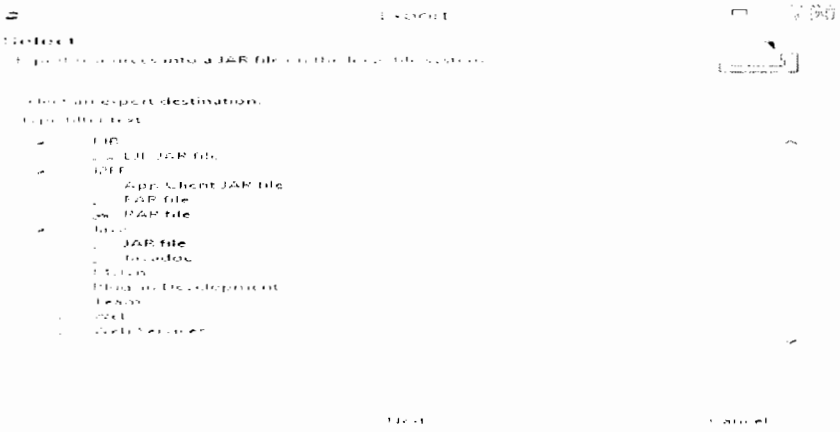
1 Create jar file

If Windows based Eclipse is used then

Right click the **project Name** > **Export**

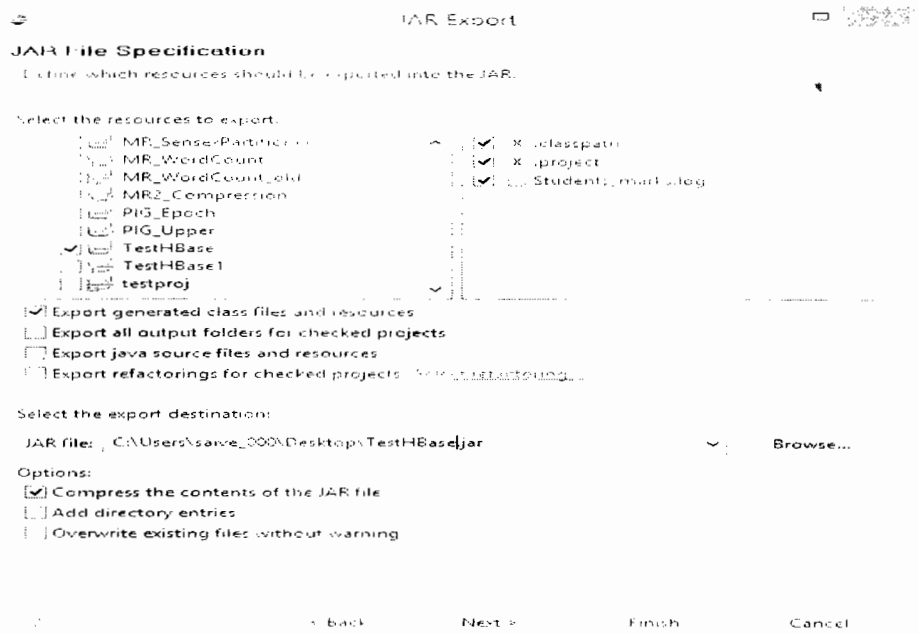
2 Export as a .jar file name

Provide a .jar file name for the project and click Next



3 Provide the jar file name

Provide a jar file name TestHBase.jar and click Finish



4 .Copy files to Ubuntu

Copy the TestHBase.jar and Students_marks.log file into the Ubuntu Shared folder

5.In the Ubuntu terminal copy the .jar file into the Hadoop LFS location

gopal@ubuntu:/usr/local/hadoop/sv_MR$ **cp /mnt/hgfs/SHARE/TestHBase.jar .**

6.In the Ubuntu terminal copy the input log file into the Hadoop LFS location

gopal@ubuntu:/usr/local/hadoop/sv_MR$ **cp /mnt/hgfs/SHARE/Students_marks.log .**

7. Create input folder in the HDFS

gopal@ubuntu:/usr/local/hadoop/sv_MR$ **hadoop fs -mkdir sv_MRINPUT**

8.Copy the Students_marks.log file into the MRINPUT HDFS folder

gopal@ubuntu:/usr/local/hadoop/sv_MR$ **hadoop fs -put Students_marks.log /sv_MRINPUT**

```
ivy                webapps
ivy.xml
gopal@ubuntu:/usr/local/hadoop$ cd sv_MR
gopal@ubuntu:/usr/local/hadoop/sv_MR$ cp /mnt/hgfs/SHARE/TestHBase.jar .
gopal@ubuntu:/usr/local/hadoop/sv_MR$ cp /mnt/hgfs/SHARE/Students_marks.log .
gopal@ubuntu:/usr/local/hadoop/sv_MR$ hadoop fs -mkdir sv_MRINPUT
gopal@ubuntu:/usr/local/hadoop/sv_MR$ hadoop fs -put Students_marks.log /sv_MRIN
PUT
gopal@ubuntu:/usr/local/hadoop/sv_MR$ hadoop jar TestHBase.jar com.mr.hbase.usecaseHBaseMRIntegration /sv_MRINPUT/Students_marks.l
```

10. Execute the hadoop jar command

Execute the hadoop jar command as follows

gopal@ubuntu:/usr/local/hadoop/sv_MR$ **hadoop jar TestHBase.jar com.mr.hbase.usecase.HBaseMRIntegration /sv_MRINPUT/Students_marks.log**

11. set the CLASSPATH

If we encounter Error "**java.lang.NoClassDefFoundError:**

org/apache/hadoop/hbase/client/HBaseAdmin"

then ensure to set the classpath and provide all the HBASE related .jar files in the

```
Exception in thread "main" java.lang.NoClassDefFoundError: org/apache/hadoop/hbase/client/HBaseAdmin
        at com.mr.hbase.usecase.HBaseMRIntegration.main(HBaseMRIntegration.java:94)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:606)
        at org.apache.hadoop.util.RunJar.main(RunJar.java:160)
Caused by: java.lang.ClassNotFoundException: org.apache.hadoop.hbase.client.HBaseAdmin
        at java.net.URLClassLoader$1.run(URLClassLoader.java:366)
        at java.net.URLClassLoader$1.run(URLClassLoader.java:355)
        at java.security.AccessController.doPrivileged(Native Method)
        at java.net.URLClassLoader.findClass(URLClassLoader.java:354)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:425)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:358)
```

classpath

To set the classpath:=

gopal@ubuntu:/usr/local/hadoop/sv_MR$ **export**

CLASSPATH=$CLASSPATH:/usr/local/hadoop/hive/lib/activation-

1.1.jar:/usr/local/hadoop/hive/lib/ant-1.9.1.jar

Ensure to include all the hbase jar files that are under the /usr/local/hadoop/hive/lib folder

```
gopal@ubuntu:/usr/local/hbase-0.98.6.1-hadoop1/lib$ ls
activation-1.1.jar                hadoop-core-1.2.1.jar             htrace-core-2.04.jar              jetty-util-6.1.26.jar
asm-3.1.jar                       hamcrest-core-1.3.jar             httpclient-4.1.3.jar              jruby-complete-1.6.8.jar
commons-beanutils-1.7.0.jar       hbase-client-0.98.6.1-hadoop1.jar httpcore-4.1.3.jar                jsp-2.1-6.1.14.jar
commons-beanutils-core-1.8.0.jar  hbase-common-0.98.6.1-hadoop1.jar jackson-core-asl-1.8.8.jar        jsp-api-2.1-6.1.14.jar
commons-cli-1.2.jar               hbase-common-0.98.6.1-hadoop1-tests.jar jackson-jaxrs-1.8.8.jar     jsr305-1.3.9.jar
commons-codec-1.7.jar             hbase-examples-0.98.6.1-hadoop1.jar jackson-mapper-asl-1.8.8.jar   junit-4.11.jar
commons-collections-3.2.1.jar     hbase-hadoop1-compat-0.98.6.1-hadoop1.jar jackson-xc-1.8.8.jar     libthrift-0.9.0.jar
commons-configuration-1.6.jar     hbase-hadoop-compat-0.98.6.1-hadoop1.jar jamon-runtime-2.3.1.jar   log4j-1.2.17.jar
commons-digester-1.8.jar          hbase-it-0.98.6.1-hadoop1.jar     jasper-compiler-5.5.23.jar        metrics-core-2.2.0.jar
commons-el-1.0.jar                hbase-it-0.98.6.1-hadoop1-tests.jar jasper-runtime-5.5.23.jar       netty-3.6.6.Final.jar
commons-httpclient-3.1.jar        hbase-prefix-tree-0.98.6.1-hadoop1.jar jaxb-api-2.2.2.jar           protobuf-java-2.5.0.jar
commons-io-2.4.jar                hbase-protocol-0.98.6.1-hadoop1.jar jaxb-impl-2.2.3-1.jar           ruby
commons-lang-2.6.jar              hbase-server-0.98.6.1-hadoop1.jar jersey-core-1.8.jar               servlet-api-2.5-6.1.14.jar
commons-logging-1.1.1.jar         hbase-server-0.98.6.1-hadoop1-tests.jar jersey-json-1.8.jar         slf4j-api-1.6.4.jar
commons-math-2.1.jar              hbase-shell-0.98.6.1-hadoop1.jar  jersey-server-1.8.jar             slf4j-log4j12-1.6.4.jar
commons-net-1.4.1.jar             hbase-testing-util-0.98.6.1-hadoop1.jar jettison-1.3.1.jar          xmlenc-0.52.jar
findbugs-annotations-1.3.9-1.jar  hbase-thrift-0.98.6.1-hadoop1.jar jetty-6.1.26.jar                   zookeeper-3.4.6.jar
guava-12.0.1.jar                  high-scale-lib-1.1.1.jar          jetty-sslengine-6.1.26.jar
gopal@ubuntu:/usr/local/hbase-0.98.6.1-hadoop1/lib$ export CLASSPATH=$CLASSPATH:/usr/local/hbase-0.98.6.1-hadoop1/activation-1.1.jar:/usr/local
hbase-0.98.6.1-hadoop1/hadoop-core-1.2.1.jar
```

12.Execute the hadoop jar command

Now once again execute the hadoop jar command

gopal@ubuntu:/usr/local/hadoop/sv_MR$ **hadoop jar TestHBase.jar**

com.mr.hbase.usecase.HBaseMRIntegration /sv_MRINPUT/Students_marks.log

13. Verify HBASE table creation and data

Let us check if the table got created in hbase and also if the MapReduce processed data is stored in the HBASE Table.

13.1 Access hbase shell

gopal@ubuntu:/usr/local/hadoop>hbase shell

gopal@ubuntu:/usr/local/hadoop> scan SCHOLARSHIP_DET13

14. Java Code – MapReduce HBase Integration code

MapReduce java code to create table and store the data in HBASE

```
hem@localhost:~/hbase_prac
File  Edit  View  Search  Terminal  Help
 Male                                   column=STUDENT_DET:AGE, timestamp=1446457188201, value=27
 Male                                   column=STUDENT_DET:GENDER, timestamp=1446457188201, value=Male
 Male                                   column=STUDENT_DET:MARKS, timestamp=1446457188201, value=87
 Male                                   column=STUDENT_DET:STUDENT_NAME, timestamp=1446457188201, value=[illegible]
2 row(s) in 0.1060 seconds

hbase(main):066:0> scan 'SCHOLARSHIP_DET11'
ROW                                     COLUMN+CELL
 Female                                 column=STUDENT_DET:AGE, timestamp=1446458424520, value=27
 Female                                 column=STUDENT_DET:GENDER, timestamp=1446458424520, value=Female
 Female                                 column=STUDENT_DET:MARKS, timestamp=1446458424520, value=98
 Female                                 column=STUDENT_DET:STUDENT_NAME, timestamp=1446458424520, value=[illegible]
 Male                                   column=STUDENT_DET:AGE, timestamp=1446458424520, value=27
 Male                                   column=STUDENT_DET:GENDER, timestamp=1446458424520, value=Male
 Male                                   column=STUDENT_DET:MARKS, timestamp=1446458424520, value=87
 Male                                   column=STUDENT_DET:STUDENT_NAME, timestamp=1446458424520, value=[illegible]
2 row(s) in 0.0800 seconds

hbase(main):007:0> scan 'SCHOLARSHIP_DET12'
ROW                                     COLUMN+CELL
 \x00\x00\x00\x00                       column=STUDENT_DET:AGE, timestamp=1446458634831, value=27
 \x00\x00\x00\x00                       column=STUDENT_DET:GENDER, timestamp=1446458634831, value=Male
 \x00\x00\x00\x00                       column=STUDENT_DET:MARKS, timestamp=1446458634831, value=87
 \x00\x00\x00\x06                       column=STUDENT_DET:STUDENT_NAME, timestamp=1446458634831, value=[illegible]
1 row(s) in 0.0550 seconds

hbase(main):008:0> scan 'SCHOLARSHIP_DET13'
ROW                                     COLUMN+CELL
 2787                                   column=STUDENT_DET:AGE, timestamp=1446458835584, value=27
 2787                                   column=STUDENT_DET:GENDER, timestamp=1446458835584, value=Male
 2787                                   column=STUDENT_DET:MARKS, timestamp=1446458835584, value=87
 2787                                   column=STUDENT_DET:STUDENT_NAME, timestamp=1446458835584, value=[illegible]
 2798                                   column=STUDENT_DET:AGE, timestamp=1446458835584, value=27
 2798                                   column=STUDENT_DET:GENDER, timestamp=1446458835584, value=Female
 2798                                   column=STUDENT_DET:MARKS, timestamp=1446458835584, value=98
 2798                                   column=STUDENT_DET:STUDENT_NAME, timestamp=1446458835584, value=[illegible]
2 row(s) in 0.1270 seconds
```

```
package com.mr.hbase.usecase;
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.hbase.HColumnDescriptor;
import org.apache.hadoop.hbase.HTableDescriptor;
import org.apache.hadoop.hbase.client.HBaseAdmin;
import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.mapreduce.TableOutputFormat;
import org.apache.hadoop.hbase.mapreduce.TableReducer;
import org.apache.hadoop.hbase.util.Bytes;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;

public class HBaseMRIntegration {
static private String TABLE_NAME = "SCHOLARSHIP_DET13";
static private String COL_FAMILY = "STUDENT_DET";
static private String COL_NAME = "STUDENT_NAME";
static private String COLUMN_AGE = "AGE";
static private String COLUMN_GENDER = "GENDER";
static private String COLUMN_MARKS = "MARKS";
public static class AgePartitionerMapper extends
Mapper<Object, Text, Text, Text> {
public void map(Object objkey, Text txtval, Context ctxt)
throws IOException, InterruptedException {

String[] tokens = txtval.toString().split("\t");
String strgender = tokens[2].toString();
String stroutval = tokens[0] + "\t" + tokens[1] + "\t" + tokens[3];
Text outkeytxt = new Text(strgender);
Text outvaltxt = new Text(stroutval);
```

```
ctxt.write(new Text(outkeytxt), new Text(outvaltxt));
}
// Here the 3rd argument is NullWritable
public static class AgePartitionerReducer extends
TableReducer<Text, Text, NullWritable> {
@Override
public void reduce(Text key, Iterable<Text> values, Context
context)throws IOException, InterruptedException {
int maxScore = Integer.MIN_VALUE;
String name = " ";
String age = " ";
String gender = " ";
int score = 0;
for (Text val : values) {
String[] valTokens = val.toString().split("\t");
score = Integer.parseint(valTokens[2]);
if (score > maxScore) {
name = valTokens[0];
age = valTokens[1];
gender = key.toString();
maxScore = score;
}// if
}// for
int i = 0;
//Based on the MapReduce processed data insert data into the Table
Put put = new Put(Bytes.toBytes(age.toString()
+ String.valueOf(maxScore)));
put.add(Bytes.toBytes(COL_FAMILY),Bytes.toBytes(COL_NAME),
Bytes.toBytes(String.valueOf(name)));
put.add(Bytes.toBytes(COL_FAMILY),Bytes.toBytes(COLUMN_AGE),
Bytes.toBytes(String.valueOf(age)));
put.add(Bytes.toBytes(COL_FAMILY),Bytes.toBytes(COLUMN_GENDE
R),Bytes.toBytes(String.valueOf(gender)));
put.add(Bytes.toBytes(COL_FAMILY),
```

```
Bytes.toBytes(COLUMN_MARKS),Bytes.toBytes(String.valueOf(maxSco
re)));
context.write(NullWritable.get(), put);
}
}
public static void main(String args[]) throws Exception {
if (args.length < 1) {
System.err.println("Usage: HBaseCalSum <path> ");
System.exit(-1);
}
Configuration conf = new Configuration();
conf.set("hbase.root.dir", "localhost.localdomain:54310/hbase");
conf.set("hbase.zookeeper.quorum", "localhost");
// set the TableOutputFormat provide the Table Name.
conf.set(TableOutputFormat.OUTPUT_TABLE, TABLE_NAME);
HBaseAdmin admin = new HBaseAdmin(conf);
//if table already exists provide an alert "table already exist"
if (admin.tableExists(TABLE_NAME)) {
System.out.println("table already exists!");
} else {
//else create new table with the provided Tablename,columnfamily
HTableDescriptor tableDesc = new HTableDescriptor(TABLE_NAME);
tableDesc.addFamily(new HColumnDescriptor(COL_FAMILY));
admin.createTable(tableDesc);
System.out.println("create table " + TABLE_NAME + " ok.");
}
String input = args[0];
Job job = new Job(conf, "HBASE_MR_Integration_" + input);
job.setJarByClass(HBaseMRIntegration.class);
job.setMapperClass(AgePartitionerMapper.class);
job.setReducerClass(AgePartitionerReducer.class);
job.setMapOutputKeyClass(Text.class);
job.setMapOutputValueClass(Text.class);
job.setInputFormatClass(TextInputFormat.class);
```

```
// The Output Format is Table
job.setOutputFormatClass(TableOutputFormat.class);
// We are providing only FileInputFormat and not FileOutputFormat
FileInputFormat.addInputPath(job, new Path(input));
int retJob = job.waitForCompletion(true) ? 0 : 1;
System.exit(retJob);
}
}
```

6.14.Client Side Buffering In Hbase

HBase:

HBase is an open source, non-relational, distributed database modeled after Google's. It runs on top of Hadoop distributed file system, providing BigTable-like capabilities for Hadoop.Hbase provides a way to store large amounts of data in a fault tolerant way.Hbase can be also termed as the hadoop database.

We can perform operations like insertion into HBase using Put method. Put operation is like an RPC that transfers data from client to server and back.

When the number of insertions are less, it is feasible but when we need to have like 1000 insertions per second, it's better that we go for buffering.

Client Side Buffering

HBase API has a built in client side buffer that can buffer all the puts and sent it in a single RPC call. The following methods are used for this purpose.

```
void setAutoFlush(boolean autoFlush)
boolean isAutoFlush()
```

We need to set the 'autoFlush' to 'false' to enable the client side buffer as by default it's not enabled.

```
table.setAutoFlush(false)
```

Once this has been done, we can check the state of the 'autoFlush' flag using the isAutoFlush() method. Now, all the put instances we created are stored in the memory in the client process. To force the data out of the buffer ,we need to call another function provided by API.

```
void flushCommits() throws IOException
```

This method flushes the data from buffer and ships it to the servers. Client batches the puts and sends them to the appropriate region servers in a single RPC call. You also have the option of setting the buffersize(client-side write buffer).In this case , API will track the size of data buffered and the required heap size for each instance. So it tracks the total

overhead of the data. When it goes beyond a certain limit , it will implicitly flush the data.

We can configure the max client side buffer size using the following:

void setWriteBufferSize(long writeBufferSize) throws IOException

You can retrieve the client side buffer size using getWriteBufferSize() method in the API. **The default size is 2MB.**

The data is hence flushed in 2 cases:

6.14.1.Implicit Flush :

It's triggered whenever put() or setWriteBufferSize() is called. This will cause the API to compare the current buffer size with the configured maximum buffer size. If it exceeds the limit, then it triggers the flush. If the client buffer is disabled then each invocation of the put() will cause a buffer flush.close() also triggers buffer flush.

6.14.2.Explicit Flush:

The flushCommits() is called explicitly to clear the buffer and sent the data to the server.

Here's an example of client side buffering.

```
HTable table = new HTable(config, "example_table");
table.setAutoFlush(false); //setting autoFlush to false to enable buffering.
Put put1 = new Put(Bytes.toBytes("row_1"));
put1.add(Bytes.toBytes("column_fam1"),  Bytes.toBytes("column_qual1"),
Bytes.toBytes("value_1"));
table.put(put1);
Put put2 = new Put(Bytes.toBytes("row_2   put1.add(Bytes.toBytes("column_fam2"),
Bytes.toBytes("column_qual2"),
Bytes.toBytes("value_2"));
table.put(put2);
Put put3 = new Put(Bytes.toBytes("row_3"));
put1.add(Bytes.toBytes("column_fam3"), Bytes.toBytes("column_qual3"),
Bytes.toBytes("value_3"));
table.put(put3);
Get get = new Get(Bytes.toBytes("row_1"));
Result res1 = table.get(get);
table.flushCommits();    //an explicit flush is performed here.
```

In this example, we are creating puts one by one and inserting into the table.

When you need to access the write buffer content .you can use

```
ArrayList<Put> getWriteBuffer()
```

This will return the internal list of the put instances that have been buffered so far by invoking table.

In the previous example, the put instances were being added one by one. We can also create a list of put instances and insert in one go using,

void put(List<Put> puts) throws IOException

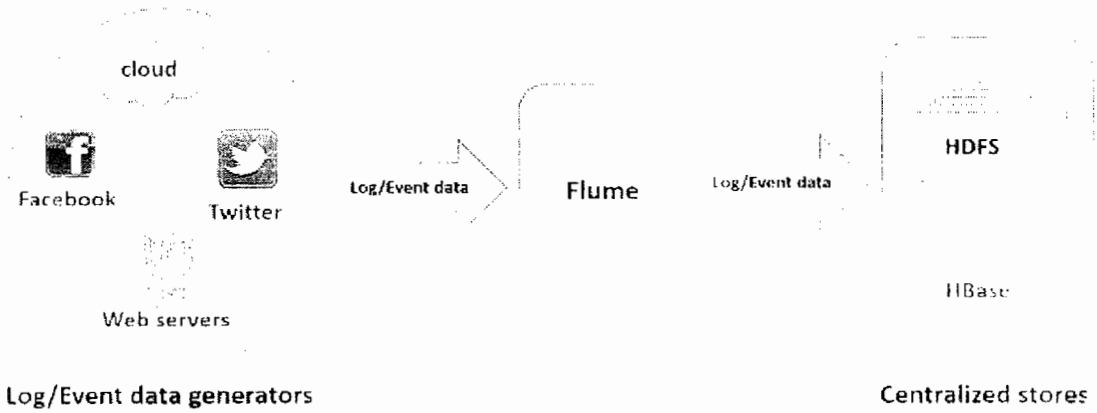
FLUME

7.1 What is Flume?

Apache Flume is a tool/service/data ingestion mechanism for collecting aggregating and transporting large amounts of streaming data such as log files. events (etc...) from various sources to a centralized data store.

Flume is a distributed system that gets logs from their source and aggregates them to where you want to process them.

Flume is a highly reliable, distributed, and configurable tool. It is principally designed to copy streaming data (log data) from various web servers to HDFS.



.2 Applications of Flume

Assume an e-commerce web application wants to analyze the customer behavior from a particular region. To do so, they would need to move the available log data in to Hadoop for analysis. Here, Apache Flume comes to our rescue.

Flume is used to move the log data generated by application servers into HDFS at a higher speed.

7.3 Advantages of Flume

Here are the advantages of using Flume:

- Using Apache Flume we can store the data in to any of the centralized stores (HBase. HDFS).
- When the rate of incoming data exceeds the rate at which data can be written to the destination, Flume acts as a mediator between data producers and the centralized stores and provides a steady flow of data between them.
- Flume provides the feature of **contextual routing**.
- The transactions in Flume are channel-based where two transactions (one sender and one receiver) are maintained for each message. It guarantees reliable message delivery.

7.4 Advantages over ad-hoc solutions

- Reliable, Scalable, Manageable, Customizable, High Performance
- Declarative, Dynamic Configuration
- Contextual Routing
- Feature Rich and Fully Extensible

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.

7.5 Features of Flume

Some of the notable features of Flume are as follows:

- Flume ingests log data from multiple web servers into a centralized store (HDFS, HBase) efficiently.
- Using Flume, we can get the data from multiple servers immediately into Hadoop.
- Along with the log files, Flume is also used to import huge volumes of event data produced by social networking sites like Facebook and Twitter, and e-commerce websites like Amazon and Flipkart.
- Flume supports a large set of sources and destinations types.
- Flume supports multi-hop flows, fan-in fan-out flows, contextual routing, etc.
- Flume can be scaled horizontally.

Apache Flume is a top level project at the Apache Software Foundation.
There are currently two release code lines available, versions 0.9.x and 1.x.

7.6 Core Concepts

7.6.1 Event

An Event is the fundamental unit of data transported by Flume from its point of origination to its final destination.
Event is a byte array payload accompanied by optional headers.

Header **Byte Payload**

Flume event

- Headers are specified as an unordered collection of string key-value pairs, with keys being unique across the collection
- Headers can be used for contextual routing

7.6.2 Client

An entity that generates events and sends them to one or more Agents.

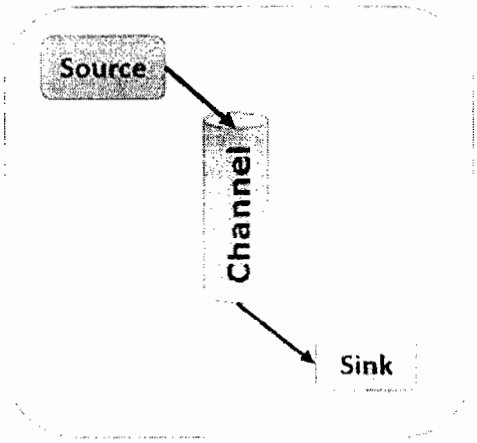
Example:-

Flume log4j Appender

- Custom Client using Client SDK (org.apache.flume.api)
- Decouples Flume from the system where event data is generated
- Not needed in all cases

7.6.3 Agent

A container for hosting Sources, Channels, Sinks and other components that enable the transportation of events from one place to another.



Flume Agent

- Fundamental part of a Flume flow
- Provides Configuration, Life-cycle
- Management, Monitoring Support for hosted components.
- Embedded Agent support for light-weight configurations

7.7 Streaming / Log Data

Generally, most of the data that is to be analyzed will be produced by various data sources like applications servers, social networking sites, cloud servers, and enterprise servers. This data will be in the form of **log files** and **events**.

7.7.1 Log file

In general, a log file is a **file** that lists events/actions that occur in an operating system. For example, web servers list every request made to the server in the log files.

On harvesting such log data, we can get information about:

- the application performance and locate various software and hardware failures.
- the user behavior and derive better business insights.

7.8 Data Transfer To HDFS

The traditional method of transferring data into the HDFS system is to use the **put** command. Let us see how to use the **put** command.

7.8.1 HDFS put Command

The main challenge in handling the log data is in moving these logs produced by multiple servers to the Hadoop environment.

Hadoop **File System Shell** provides commands to insert data into Hadoop and read from it. You can insert data into Hadoop using the **put** command as shown below.

$ Hadoop fs -put /path of the required file /path in HDFS where to save the file

7.8.1.1 Problem with put Command

We can use the **put** command of Hadoop to transfer data from these sources to HDFS. But, it suffers from the following drawbacks:

- Using **put** command, we can transfer **only one file at a time** while the data generators generate data at a much higher rate. Since the analysis made on older data is less accurate, we need to have a solution to transfer data in real time.

- If we use **put** command, the data is needed to be packaged and should be ready for the upload. Since the web servers generate data continuously, it is a very difficult task.

7.8.2 Problem with HDFS

In HDFS, the file exists as a directory entry and the length of the file will be considered as zero till it is closed. For example, if a source is writing data into HDFS and the network was interrupted in the middle of the operation (without closing the file), then the data written in the file will be lost.

Therefore we need a reliable, configurable, and maintainable system to transfer the log data into HDFS.

Note: In POSIX file system, whenever we are accessing a file (say performing write operation), other programs can still read this file (at least the saved portion of the file). This is because the file exists on the disc before it is closed.

7.8.3 Available Solutions

To send streaming data (log files, events etc...) from various sources to HDFS, we have the following tools available at our disposal:

7.8.3.1 Facebook's Scribe

Scribe is an immensely popular tool that is used to aggregate and stream log data. It is designed to scale to a very large number of nodes and be robust to network and node failures.

7.8.3.2 Apache Kafka

Kafka has been developed by Apache Software Foundation. It is an open-source message broker. Using Kafka, we can handle feeds with high-throughput and low-latency.

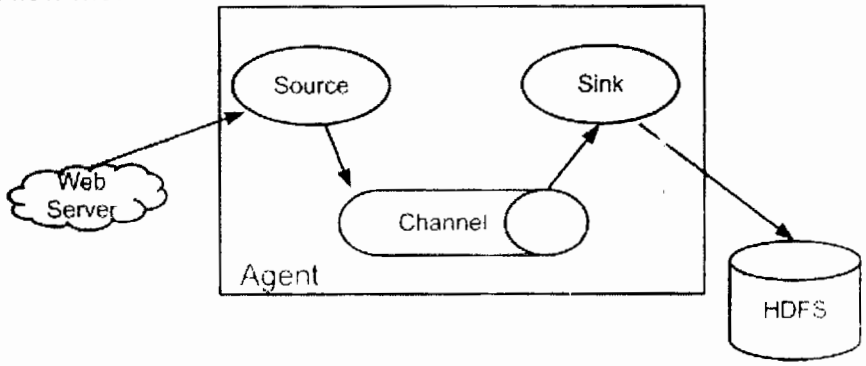
7.8.3.4 Apache Flume

Apache Flume is a tool/service/data ingestion mechanism for collecting aggregating and transporting large amounts of streaming data such as log data, events (etc...) from various web serves to a centralized data store.

It is a highly reliable, distributed, and configurable tool that is principally designed to transfer streaming data from various sources to HDFS.

7.9 Flume Architecture

Data flow model



7.9.1.Source

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. A similar flow can be defined using a Thrift Flume Source to receive events from a Thrift Sink or a Flume Thrift Rpc Client or Thrift clients written in any language generated from the Flume thrift protocol. When a Flume source receives an event, it stores it into one or more channels.

7.9.1.1 Different Source Types

- Specialized sources for integrating with well known systems. For example – Syslog, Netcat
- Auto-Generating Sources: Exec, SEQ
- IPC Sources for Agent-to-Agent Communication: Avro, Thrift
- Requires at least one channel to function.

7.9.2 Channel

A passive component that buffers the incoming events until they are drained by Sinks.

The channel is a passive store that keeps the event until it's consumed by a Flume sink. The file channel is one example – it is backed by the local file system.

Different Channels offer different levels of durability (Memory, File, and Database)

- Channels are fully transactional
- Provides weak ordering guarantees
- Can work with any number of Sources and Sinks

7.9.3 Sink

An active component that removes events from a Channel and transmits them to their next hop destination.

The sink removes the event from the channel and puts it into an external repository like HDFS (via Flume HDFS sink) or forwards it to the Flume source of the next Flume agent (next hop) in the flow. The source and sink within the given agent run asynchronously with the events staged in the channel.

7.9.3.1 Different types of Sinks

- Terminal Sinks that deposit events to their final destination. Example: HDFS, HBase
- Auto-Consuming sinks. Example: Null Sink
- IPC sink for Agent-to-Agent communication: Avro, Thrift
- Require exactly one channel to function

7.10 Additional Components of Flume Agent

What we have discussed above are the primitive components of the agent. In addition to this, we have a few more components that play a vital role in transferring the events from the data generator to the centralized stores.

7.10.1 Interceptors

Interceptors are used to alter/inspect flume events which are transferred between source and channel.

7.10.2 Channel Selectors

These are used to determine which channel is to be opted to transfer the data in case of multiple channels. There are two types of channel selectors:

- **Default channel selectors**: These are also known as replicating channel selectors they replicates all the events in each channel.

- **Multiplexing channel selectors**: These decides the channel to send an event based on the address in the header of that event.

7.10.3 Sink Processors

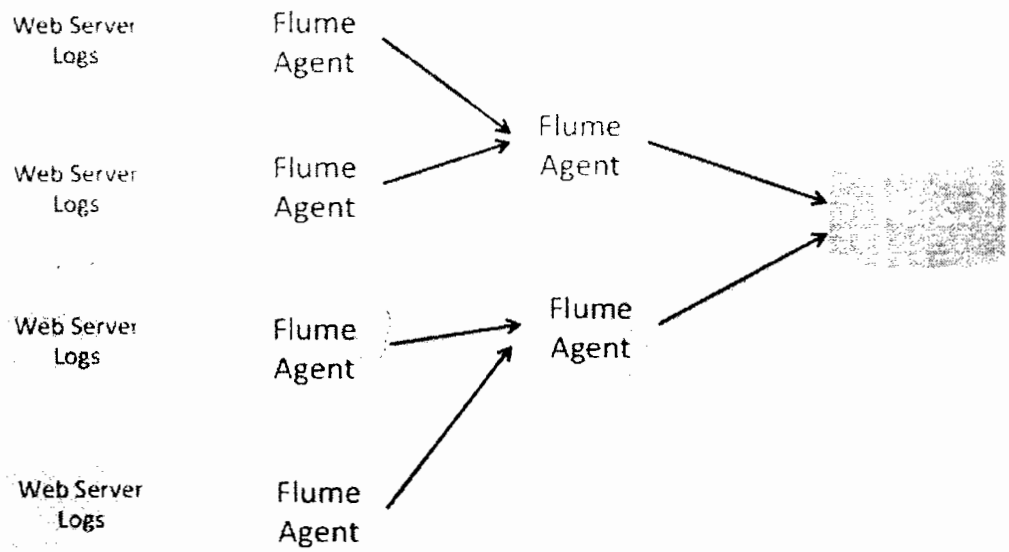
These are used to invoke a particular sink from the selected group of sinks. These are used to create failover paths for your sinks or load balance events across multiple sinks from a channel.

7.11 Flume Dataflow

Flume is a framework which is used to move log data into HDFS. Generally events and log data are generated by the log servers and these servers have Flume agents running on them. These agents receive the data from the data generators.

The data in these agents will be collected by an intermediate node known as **Collector**. Just like agents, there can be multiple collectors in Flume.

Finally, the data from all these collectors will be aggregated and pushed to a centralized store such as HBase or HDFS. The following diagram explains the data flow in Flume.



7.11.1 Multi-hop Flow

Within Flume, there can be multiple agents and before reaching the final destination, an event may travel through more than one agent. This is known as **multi-hop flow**.

7.11.2 Fan-out Flow

The dataflow from one source to multiple channels is known as **fan-out flow**. It is of two types:

- **Replicating:** The data flow where the data will be replicated in all the configured channels.

- **Multiplexing:** The data flow where the data will be sent to a selected channel which is mentioned in the header of the event.

7.11.3 Fan-in Flow

The data flow in which the data will be transferred from many sources to one channel is known as **fan-in flow**.

7.12 Failure Handling:

In Flume, for each event, two transactions take place: one at the sender and one at the receiver. The sender sends events to the receiver. Soon after receiving the data, the

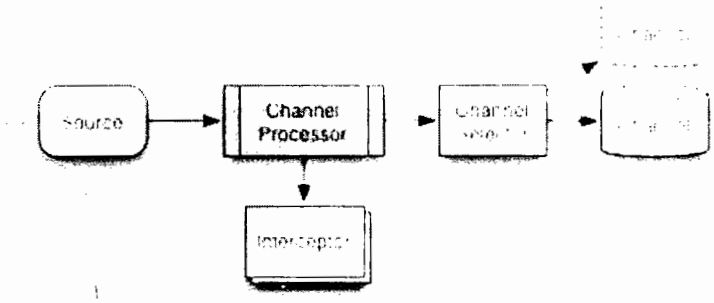
receiver commits its own transaction and sends a "received" signal to the sender. After receiving the signal, the sender commits its transaction. (Sender will not commit its transaction till it receives a signal from the receiver.)

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

7.13 .Agent Data Ingest



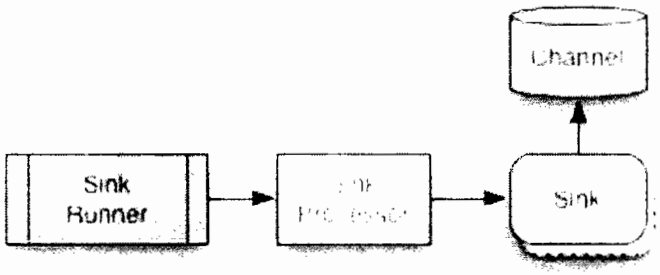
Clients send Events to Agents

• Source(s) operating with the Agent receive the Events

• Source(s) passes received Events through Interceptor(s) and if not filtered, puts them on Channel(s) identified by the configured Channel Selector.

- Event population in Channel is transactional

7.14 .Agent Data Drain



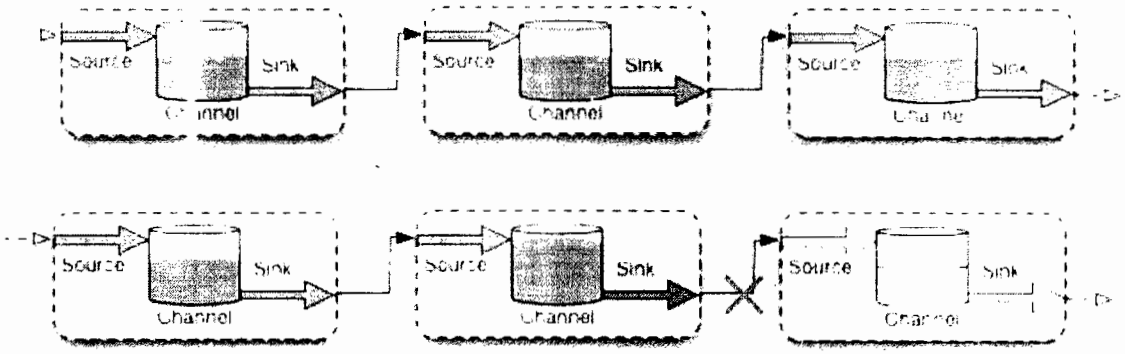
• Sink Processor identifies one of the configured Sinks and invokes it

• Invoked Sink takes Event(s) from its configured Channel and sends it to the next hop destination.

• Event Removal from Channel is transactional.

• If the transmission of Event(s) fail, the Sink Processor can take secondary action.

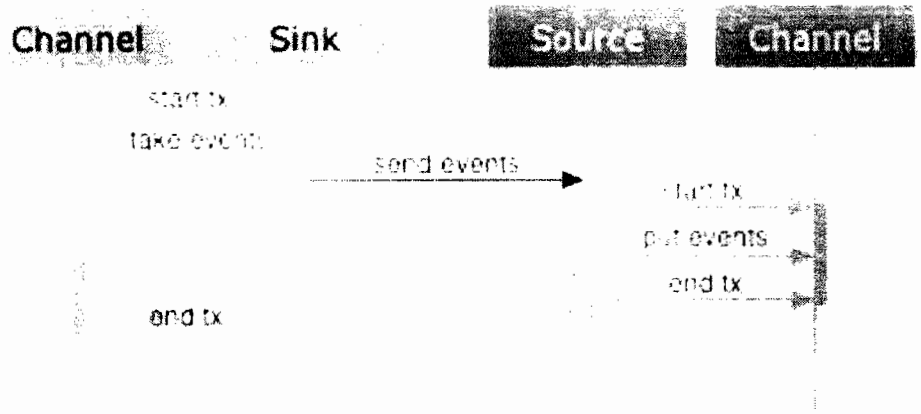
7.15. Agent Pipeline Flow



Agents transmit events to next hop agents

• In case of transmission failure the second agent retains the data in its channel(s)

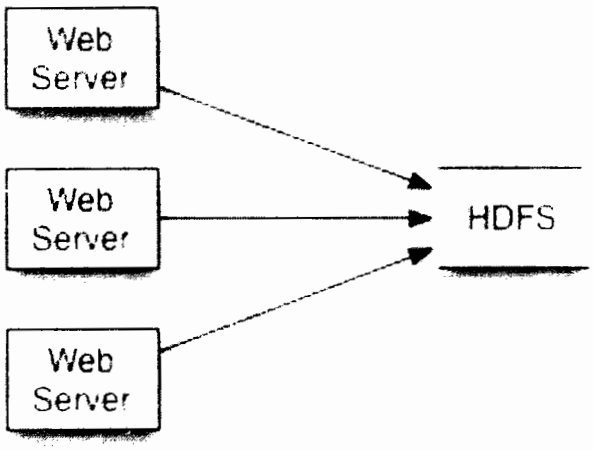
7.16.Delivery Guarantee



Agents use transactional exchange to guarantee delivery across hops

- Durable channels ensure that data is never lost in transit.

7.17.Aggregation Use-Case

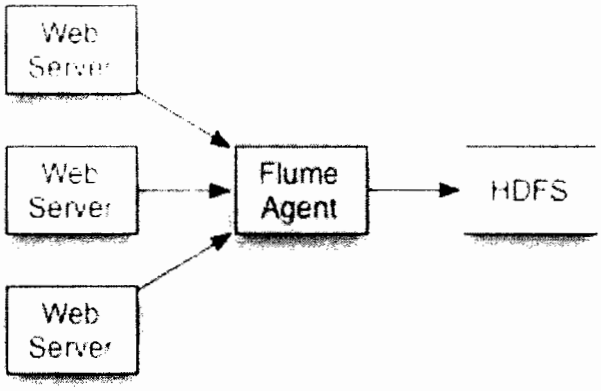


Collect Web Server logs

Ad-hoc Solutions

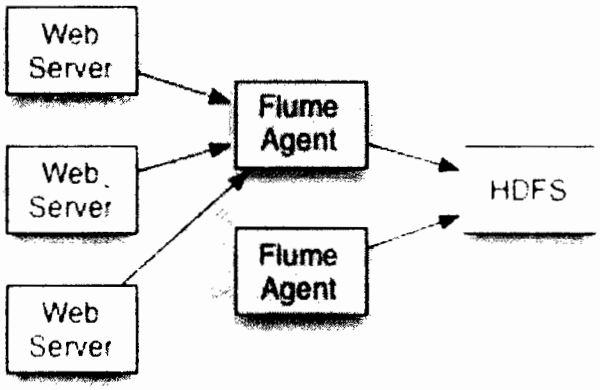
- Configuration Management – Output paths
- Complex Coding and Evolution
- Maintenance and upkeep
- Scalability

7.18.Using Single Flume Agent



- Insulation from destination downtime
- Managed impact on service
- Quick offload of events from Web Server machines
- Better network utilization

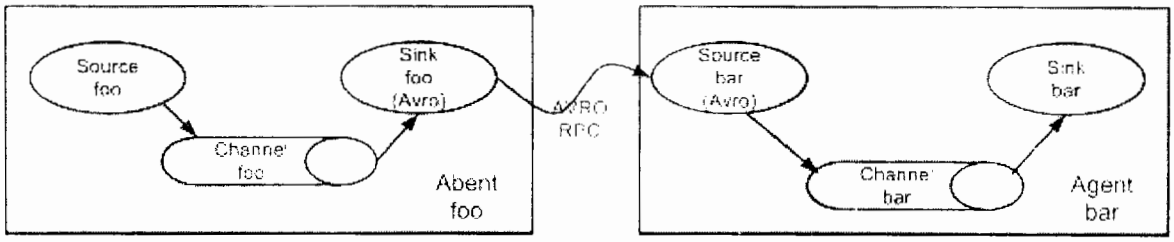
7.19.Using Multiple Flume Agents



- Redundancy is Availability
- Better handling of transient failures/down times with minimal hardware investment
- Automatic contextual routing for planned routing of event traffic.

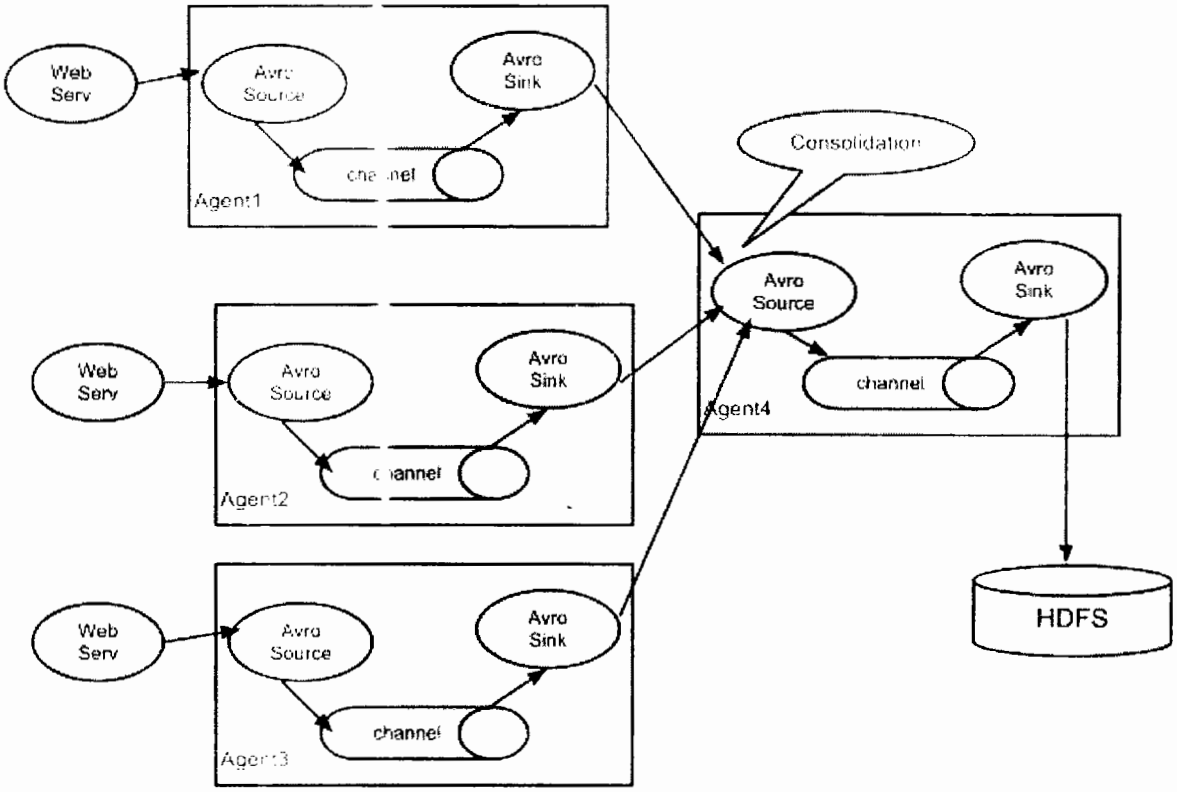
Setting multi-agent flow

In order to flow the data across multiple agents or hops. the sink of the previous agent and source of the current hop need to be avro type with the sink pointing to the hostname (or IP address) and port of the source.



7.20.Consolidation

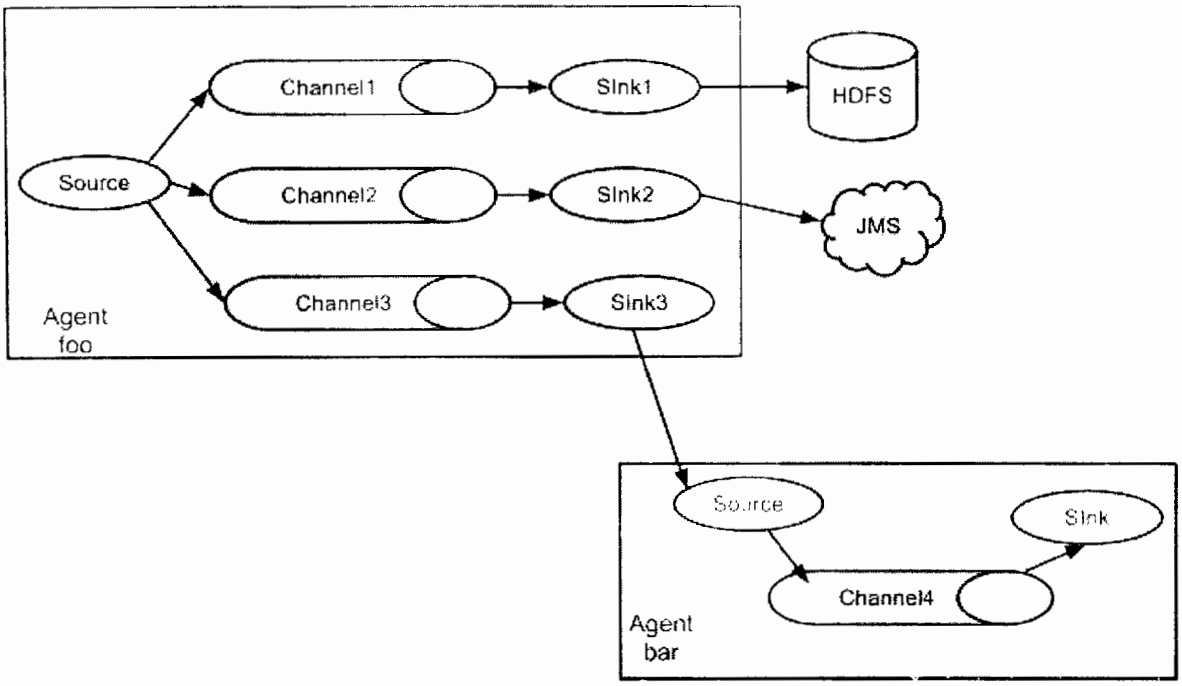
A very common scenario in log collection is a large number of log producing clients sending data to a few consumer agents that are attached to the storage subsystem. For example, logs collected from hundreds of web servers sent to a dozen of agents that write to HDFS cluster.



This can be achieved in Flume by configuring a number of first tier agents with an avro sink, all pointing to an avro source of single agent (Again you could use the thrift sources/sinks/clients in such a scenario). This source on the second tier agent consolidates the received events into a single channel which is consumed by a sink to its final destination.

7.21.Multiplexing the flow

Flume supports multiplexing the event flow to one or more destinations. This is achieved by defining a flow multiplexer that can replicate or selectively route an event to one or more channels.



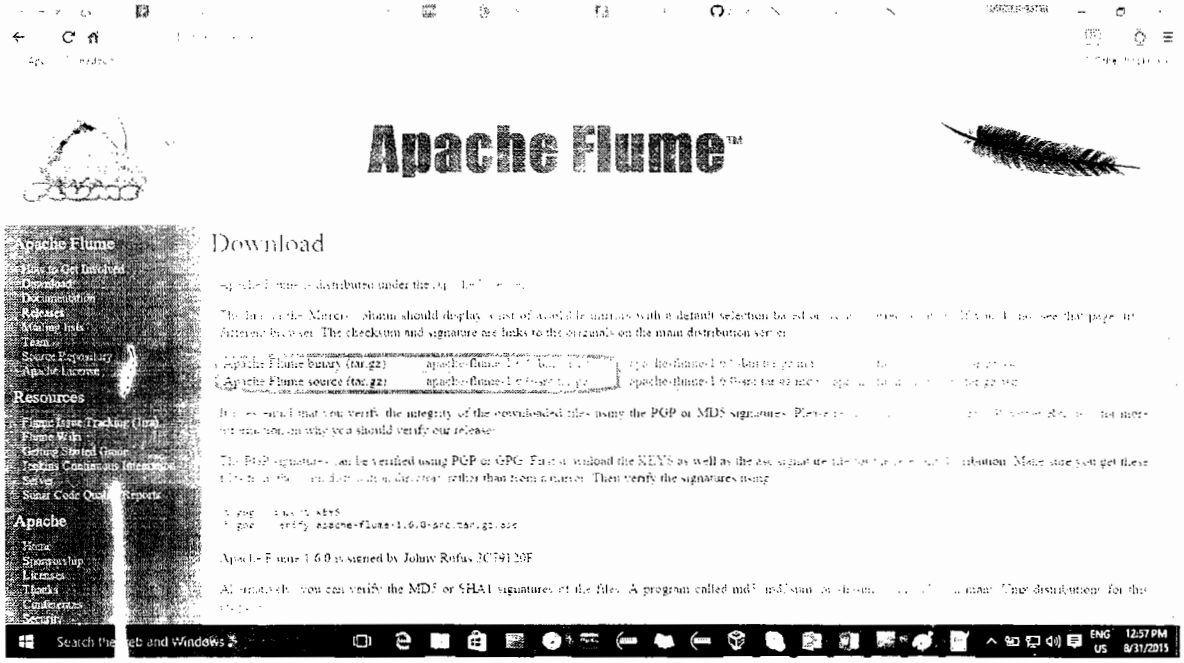
The above example shows a source from agent "foo" fanning out the flow to three different channels. This fan out can be replicating or multiplexing. In case of replicating flow, each event is sent to all three channels. For the multiplexing case, an event is delivered to a subset of available channels when an event's attribute matches a preconfigured value. For example, if an event attribute called "txnType" is set to "customer", then it should go to channel1 and channel3, if it's "vendor" then it should go to channel2, otherwise channel3. The mapping can be set in the agent's configuration file.

7.22.Flume Installation

First of all, download the latest version of Apache Flume software from the website https://flume.apache.org/.

Step 1

Open the website. Click on the **download** link on the left-hand side of the home page. It will take you to the download page of Apache Flume.



Step 2

In the Download page, you can see the links for binary and source files of Apache Flume. Click on the link apache-flume-1.6.0-bin.tar.gz

You will be redirected to a list of mirrors where you can start your download by clicking any of these mirrors. In the same way, you can download the source code of Apache Flume by clicking on apache-flume-1.6.0-src.tar.gz.

Step 3

Create a directory with the name Flume in the same directory where the installation directories of Hadoop, HBase, and other software were installed (if you have already installed any) as shown below.

$ mkdir Flume

Step 4

Extract the downloaded tar files as shown below.

$ cd Downloads/

$ tar zxvf apache-flume-1.6.0-bin.tar.gz

$ tar zxvf apache-flume-1.6.0-src.tar.gz

Step 5

Move the content of apache-flume-1.6.0-bin.tar file to the Flume directory created earlier as shown below. (Assume we have created the Flume directory in the local user named Hadoop.)

$ mv apache-flume-1.6.0-bin.tar/* /home/Hadoop/Flume/

7.22.1.Configuring Flume

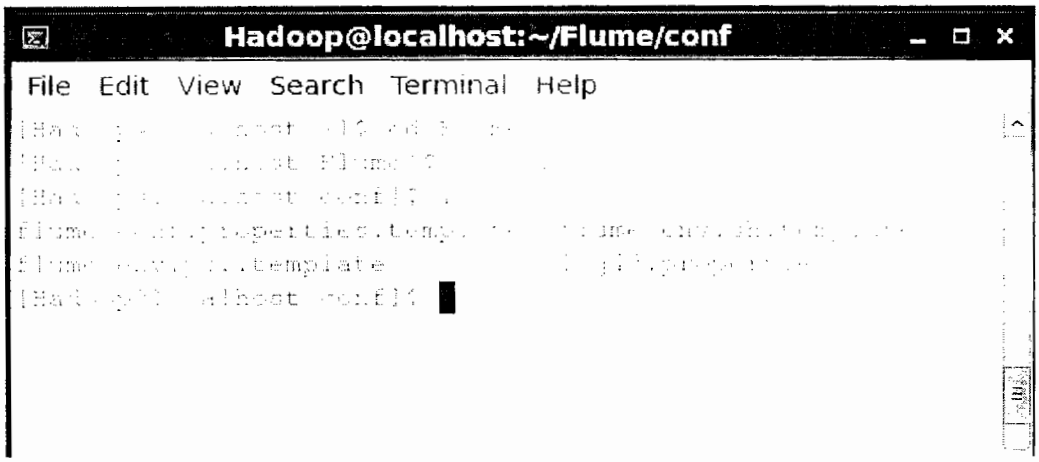
To configure Flume, we have to modify three files namely, flume-env.sh, flume-conf.properties, and bash.rc.
Setting the Path / Classpath

In the .bashrc file, set the home folder, the path, and the classpath for Flume as shown below.

7.22.2.conf Folder

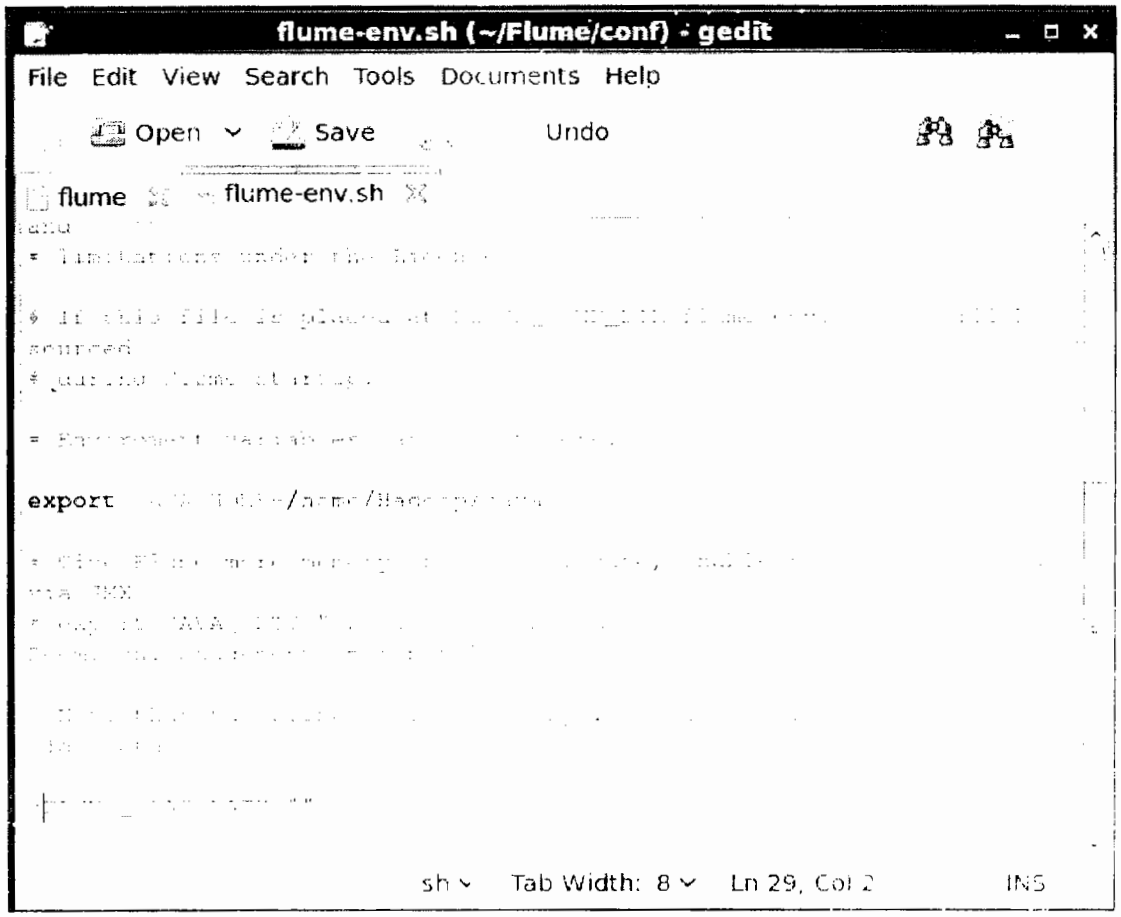
If you open the **conf** folder of Apache Flume, you will have the following four files:

- flume-conf.properties.template,
- flume-env.sh.template,
- flume-env.ps1.template, and
- log4j.properties



Now rename

- **flume-conf.properties.template** file as **flume-conf.properties** and
- **flume-env.sh.template** as **flume-env.sh**



7.22.3.flume-env.sh

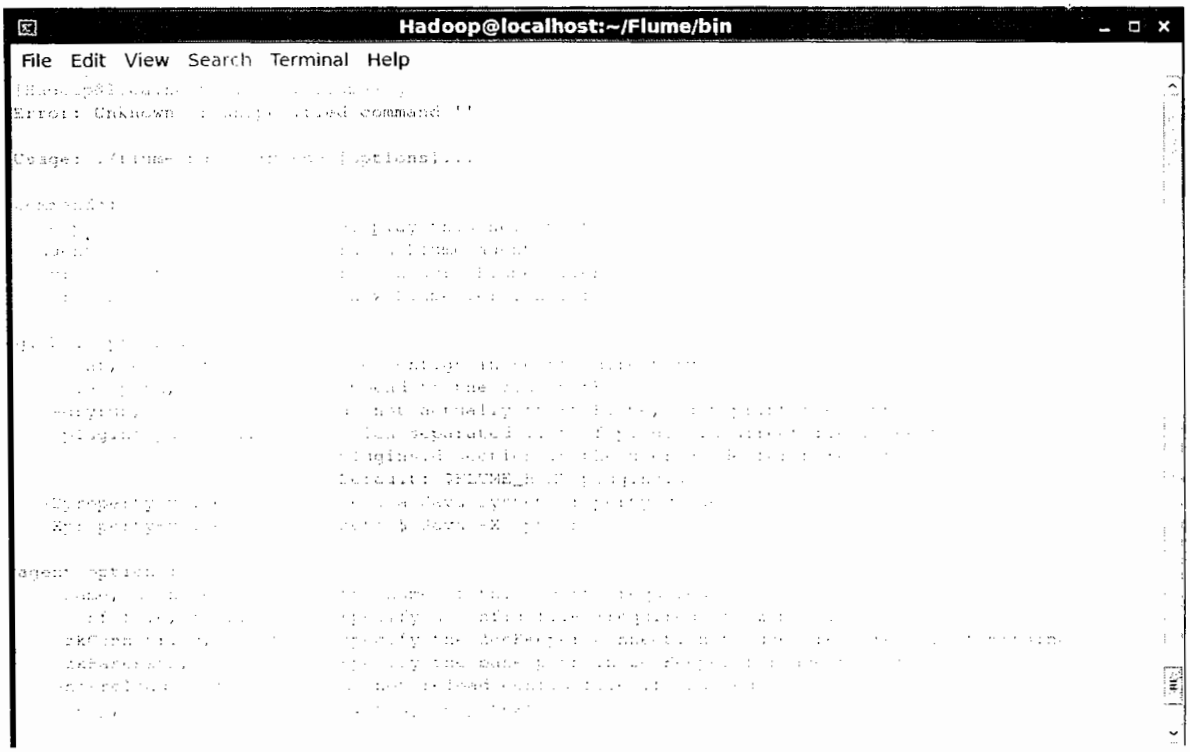
Open **flume-env.sh** file and set the **JAVA_Home** to the folder where Java was installed in your system.

7.23.Verifying the Installation

Verify the installation of Apache Flume by browsing through the **bin** folder and typing the following command.

$ ./flume-ng

If you have successfully installed Flume, you will get a help prompt of Flume as shown below.



7.24.Flume – Configuration

After installing Flume, we need to configure it using the configuration file which is a Java property file having **key-value pairs**. We need to pass values to the keys in the file.

In the Flume configuration file, we need to –

- Name the components of the current agent.
- Describe/Configure the source.
- Describe/Configure the sink.
- Describe/Configure the channel.
- Bind the source and the sink to the channel.

Usually we can have multiple agents in Flume. We can differentiate each agent by using a unique name. And using this name, we have to configure each agent.

7.24.1Naming the Components

First of all, you need to name/list the components such as sources, sinks, and the channels of the agent, as shown below.

agent_name.sources = source_name

agent_name.sinks = sink_name

agent_name.channels = channel_name

Describing the Source

Each source will have a separate list of properties. The property named "type" is common to every source, and it is used to specify the type of the source we are using.

Along with the property "type", it is needed to provide the values of all the **required** properties of a particular source to configure it, as shown below.

agent_name.sources. source_name.type = value

agent_name.sources. source_name.property2 = value

agent_name.sources. source_name.property3 = value

Describing the Sink

Just like the source, each sink will have a separate list of properties. The property named "type" is common to every sink, and it is used to specify the type of the sink we are using. Along with the property "type", it is needed to provide values to all the **required** properties of a particular sink to configure it, as shown below.

agent_name.sinks. sink_name.type = value

agent_name.sinks. sink_name.property2 = value

agent_name.sinks. sink_name.property3 = value

Describing the Channel

Flume provides various channels to transfer data between sources and sinks. Therefore, along with the sources and the channels, it is needed to describe the channel used in the agent.

To describe each channel, you need to set the required properties, as shown below.

agent_name.channels.channel_name.type = value

agent_name.channels.channel_name. property2 = value

agent_name.channels.channel_name. property3 = value

Binding the Source and the Sink to the Channel

Since the channels connect the sources and sinks, it is required to bind both of them to the channel, as shown below.

agent_name.sources.source_name.channels = channel_name

agent_name.sinks.sink_name.channels = channel_name

7.25.Starting a Flume Agent

After configuration, we have to start the Flume agent. It is done as follows:

$ bin/flume-ng **agent** --conf ./conf/ -f conf/**twitter.conf** -Dflume.root.logger=DEBUG,console -n TwitterAgent

where:

- **agent**: Command to start the Flume agent
- **--conf ,-c<conf>:** Use configuration file in the conf directory
- **-f<file>**: Specifies a config file path, if missing
- **--name, -n <name>** : Name of the twitter agent
- **-D property =value:** Sets a Java system property value

OOZIE

8.1.What is Apache Oozie?

Apache Oozie is a scheduler system to run and **manage Hadoop jobs** in a distributed environment. It allows to combine multiple complex jobs to be run in a sequential order to achieve a bigger task. Within a sequence of task, two or more jobs can also be programmed to run parallel to each other.

One of the main advantages of Oozie is that it is tightly integrated with Hadoop stack supporting various Hadoop jobs like **Hive, Pig, Sqoop** as well as system-specific jobs like **Java and Shell**.

Oozie is an **Open Source Java Web-Application** available under Apache license 2.0. It is responsible for triggering the workflow actions, which in turn uses the Hadoop execution engine to actually execute the task. Hence, Oozie is able to leverage the existing Hadoop machinery for load balancing, fail-over, etc.

Oozie detects completion of tasks through callback and polling. When Oozie starts a task, it provides a unique **callback HTTP URL** to the task, and notifies that URL when it is complete. If the task fails to invoke the callback URL, Oozie can poll the task for completion.

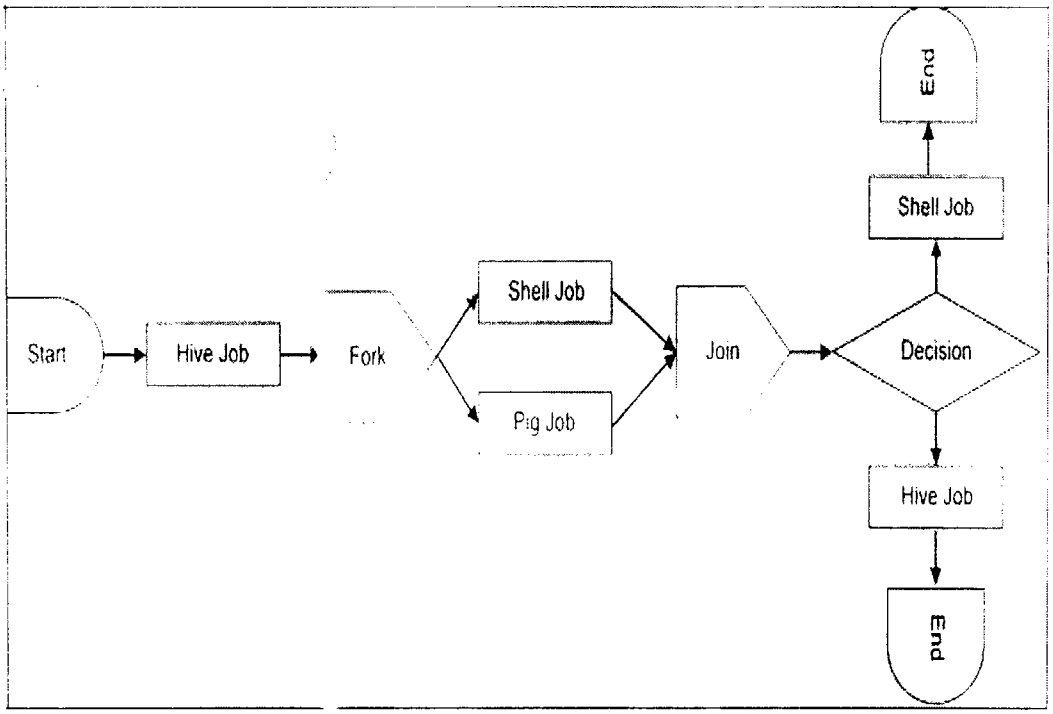
Following three types of jobs are common in Oozie:

- **Oozie Workflow Jobs** — These are represented as Directed Acyclic Graphs (DAGs) to specify a sequence of actions to be executed.

- **Oozie Coordinator Jobs** — These consist of workflow jobs triggered by time and data availability.

- **Oozie Bundle** — These can be referred to as a package of multiple coordinator and workflow jobs.

A sample workflow with Controls (Start, Decision, Fork, Join and End) and Actions (Hive, Shell, Pig) will look like the following diagram:



Workflow will always s art with a Start tag and end with an End tag.

8.2.Oozie Editors

Most of the time, you won't need an editor and will write the workflows using any popular text editors (like Notepad++, Sublime or Atom) as we will be doing in this tutorial.

But as a beginner it makes some sense to create a workflow by the drag and drop method using the edit r and then see how the workflow gets generated. Also, to map **GUI** with the act al **workflow.xml** created by the editor. This is the only section where we will d scuss about Oozie editors and won't use it in our tutorial.

The most popular among Oozie editors is **Hue**

8.3.Oozie Eclipse Plugin (OEP)

Oozie Eclipse plugin (OEP) is an Eclipse plugin for editing Apache Oozie workflows graphically. It is a graphical editor for editing Apache Oozie workflows inside Eclipse.

Composing Apache Oozie workflows is becoming much simpler. It becomes a matter of drag-and-drop, a matter of connecting lines between the nodes.

8.4.OOZIE WORKFLOW

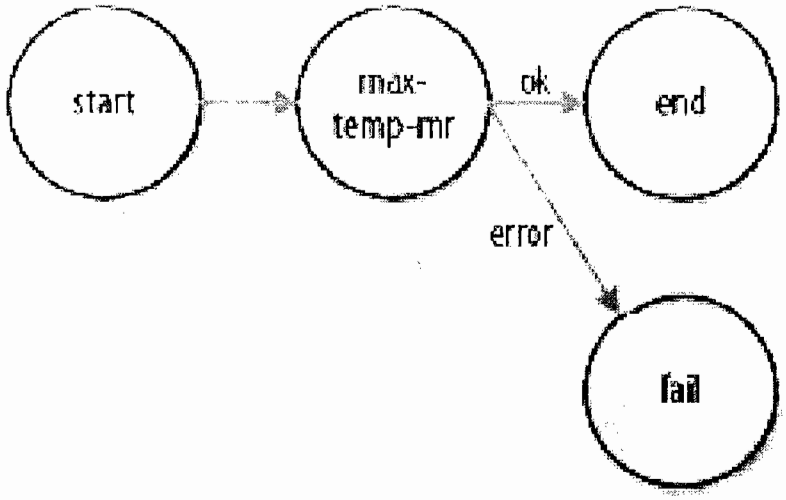
Workflow in Oozie is a sequence of actions arranged in a control dependency **DAG (Direct Acyclic Graph)**. The actions are in controlled dependency as the next action can only run as per the output of current action. Subsequent actions are dependent on its previous action. A workflow action can be a **Hive action, Pig action, Java action, Shell action**, etc. There can be decision trees to decide how and on which condition a job should run.

A fork is used to run multiple jobs in parallel. Oozie workflows can be parameterized (variables like **${nameNode}** can be passed within the workflow definition). These parameters come from a configuration file called as property file.

Oozie workflow definition : Max temperature MR Job

```
<workflow-app xmlns="uri:oozie:workflow:0.1" name="max-temp-workflow">
  <start to="max-temp-mr"/>
  <action name="max-temp-mr">
    <map-reduce>
      <job-tracker>${jobTracker}</job-tracker>
      <name-node>${nameNode}</name-node>
      <prepare>
        <delete path="${nameNode}/user/${wf:user()}/output"/>
      </prepare>
      <configuration>
        <property>
          <name>mapred.mapper.class</name>
          <value>OldMaxTemperature$OldMaxTemperatureMapper</value>
        </property>
        <property>
          <name>mapred.combiner.class</name>
          <value>OldMaxTemperature$OldMaxTemperatureReducer</value>
        </property>
        <property>
          <name>mapred.reducer.class</name>
          <value>OldMaxTemperature$OldMaxTemperatureReducer</value>
        </property>


        <property>
          <name>mapred.output.key.class</name>
          <value>org.apache.hadoop.io.Text</value>
        </property>
        <property>
          <name>mapred.output.value.class</name>
          <value>org.apache.hadoop.io.IntWritable</value>
        </property>
        <property>
          <name>mapred.input.dir</name>
          <value>/user/${wf:user()}/input/ncdc/micro</value>
        </property>
        <property>
          <name>mapred.output.dir</name>
          <value>/user/${wf:user()}/output</value>
        </property>
      </configuration>
    </map-reduce>
    <ok to="end"/>
    <error to="fail"/>
  </action>
  <kill name="fail">
    <message>MapReduce failed, error message [${wf:errorMessage(wf:lastErrorNode())}]
    </message>
  </kill>
  <end name="end"/>
</workflow-app>
```

Transition Diagram of Oozie workflow



Oozie workflow Application: Pack ging and Deploying

```
max-temp-workflow/
├── lib/
│   └── hadoop-examples.jar
└── workflow.wml
```

Oozie workflow Application : Running

export the OOZIE_URL environment variable to tell the oozie command which Oozie server to use

```
% export OOZIE_URL="http://localhost:11000/oozie"
```

config option specifies a local Java properties file containing definitions for the parameters in the workflow XML file

```
% oozie job -config ch05/src/main/resources/max-temp-workflow.properties -run
job: 0000009-120119174508294-oozie-tom-W
```

Example Workflow:

Consider we want to load a data from external hive table to an ORC Hive table.

Step 1: DDL for Hive external table (say **external.hive**)

```
Create external table external_table
(
```

```
name string,
 age int,
address string,
 zip int
)
row format delimited
fields terminated by ','
 stored as textfile
location '/test/abc';
```

Step 2: DDL for Hive ORC table (say orc.hive)

```
Create Table orc_table
(
  name string,      -- Concate value of first name and last name with space as Seperator
  yearofbirth int,
  age int,          -- Current year minus year of birth
address string,
zip int
)
STORED AS ORC
;
```

Step 3: Hive script to insert data from external table to ORC table (say Copydata.hql)

```
use ${database_name}; -- input from Oozie insert into table orc_table
select
```

```
concat(first_name,' ',last_name) as name,
yearofbirth,
year(from_unixtime) --yearofbirth as age,
address,
zip
from external_table ;
```

Step 4: Create a workflow to execute all the above three steps. (let's call it workflow.xml)

```
<!-- This is a comment -->
<workflow-app xmlns="uri:oozie:workflow:0.4" name="simple-
Workflow"> <start to="Create_External_Table" />
<!—Step 1 -->
<action name="Create_External_Table">
<hive xmlns="uri:oozie:hive-action:0.4">
<job-tracker>xyz.com:8088</job-tracker>
<name-node>hdfs://rootname</name-node>
<script>hdfs_path_of_script/external.hive</script>
</hive>
<ok to="Create_orc_Table" /> <error
to="kill_job" />
</action>

<!—Step 2 -->
<action name="Create_orc_Table">
<hive xmlns="uri:oozie:hive-action:0.4">
```

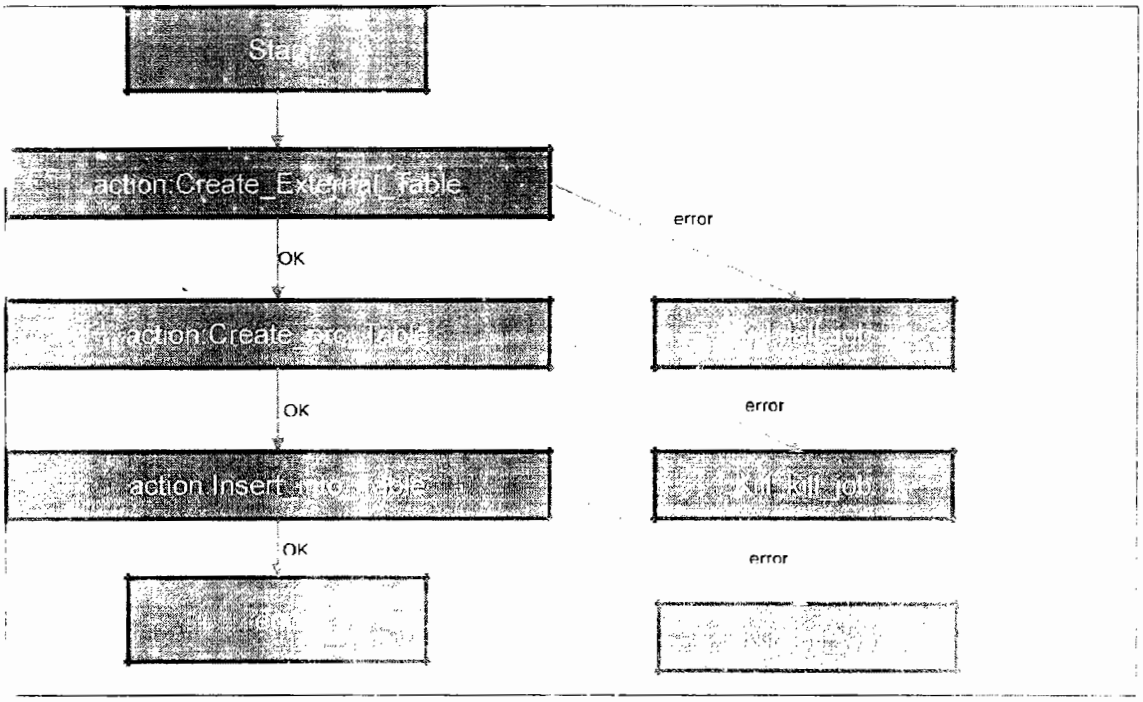
```
<job-tracker>xyz.com:8088</job-tracker>
<name-node>hdfs://rootname</name-node>
<script>hdfs_path_of_script/orc.hive</script>
</hive>
<ok  to="Insert_into_Table"  />  <error
to="kill_job" />
</action>

<!—Step 3 -->

<action name="Insert_into_Table">
<hive xmlns="uri:oozie:hive-action:0.4">
<job-tracker>xyz.com:8088</job-tracker>
<name-node>hdfs://rootname</name-node>
<script>hdfs_path_of_script/Copydata.hive</script>
<param>database_name</param
> </hive>
<ok to="end" />
<error to="kill_job" /> </action>
<kill name="kill_job"> <message>Job
failed</message>
</kill>
<end  name="end"  />
</workflow-app>
```

Explanation of the Above Example

Action Nodes in the above example defines the type of job that the node will run. Hive node inside the action node defines that the action is of type hive. This could also have been a **pig, java**, shell action, etc. as per the job you want to run.

Each type of action can have its own type of tags. In the above job we are defining the job tracker to us, name node details, script to use and the param entity. The Script tag defines the script we will be running for that hive action. The Param tag defines the values which we will pass into the hive script. (In this example we are passing database name in step 3).

The above workflow will translate into the following DAG.



Running the Workflow:

A topology runs in a distributed manner, on multiple worker nodes. Storm spreads the tasks evenly on all the worker nodes. The worker node's role is to listen for jobs and start or stop the processes whenever a new job arrives.

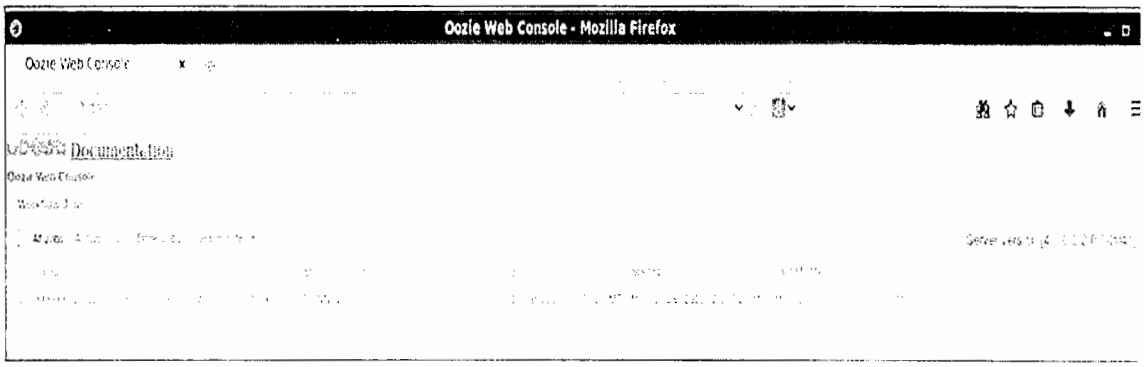
Note: *The workflow and hive scripts should be placed in HDFS path before running*

the workflow.

```
oozie job --oozie http://host_name:8080/oozie -D
oozie.wf.application.path=hdfs://namenodepath/pathof_workflow_xml/workflow.x
ml -run
```

This will run the workflow once.

To check the status of job you can go to Oozie web console http://host_name:8088/



By clicking on the job you will see the running job. You can also check the status using **Command Line Interface** (We will see this later). The possible states for workflow jobs are: PREP, RUNNING, SUSPENDED, SUCCEEDED, KILLED and FAILED.

In the case of an action start failure in a workflow job, depending on the type of failure, Oozie will attempt automatic retries. It will request a manual retry or it will fail the workflow job. Oozie can make HTTP callback notifications on action start/end/failure events and workflow end/failure events. In the case of a workflow job failure, the workflow job can be resubmitted skipping the previously completed actions. Before doing a resubmission the workflow application could be updated with a patch to fix a problem in the workflow application code.

8.5.Property File

Variables like **${nameNode}** can be passed within the workflow definition. The value of this variable will be replaced at the run time with the value defined in the '.properties' file

Following is an example of a property file we will use in our workflow example.

File name -- job1.properties

```
#proprties
nameNode=hdfs://rootn
ame
jobTracker=xyz.com:80
88
script_name_external=hdfs_path_of_script/external.hiv
e          script_name_orc=hdfs_path_of_script/orc.hive
```

```
script_name_copy=hdfs_path_of_script/Copydata.hive
database=database_name
```

Now to use this property file we will have to update the workflow and pass the parameters in
a workflow as shown in the following program.

```
<!-- This is a comment -->
<workflow-app      xmlns="uri:oozie:workflow:0.4"      name="simple-
Workflow"> <start to="Create_External_Table" />
<action name="Create_External_Table">
<hive xmlns="uri:oozie:hive-action:0.4">
<job-tracker>${jobTracker}</job-tracker>
<name-node>${nameNode}</name-node>
<script>${script_name_external}</script>
</hive>
<ok to="Create_orc_Table" /> <error
to="kill_job" />
</action>
<action name="Create_orc_Table">
```

```
<hive xmlns="uri:oozie:hive-action:0.4">
<job-tracker>${jobTracker}</job-tracker>
<name-node>${nameNode}</name-node>
<script>${script_name_orc}</script>
</hive>
<ok to="Insert_into_Table" /> <error
```

```
        to="kill_job" />
</action>
<action name="Insert_into_Table">
<hive xmlns="uri:oozie:hive-action:0.4">
<job-tracker>${jobTracker}</job-tracker>
<name-node>${nameNode}</name-node>
<script>${script_name_copy}</script>
<param>${database}</param>
</hive>
<ok to="end" />
<error to="kill_job" /> </action>
<kill name="kill_job">
 <message>Job failed</message>
</kill>
<end name="end" />
```

```
</workflow-app>
```

Now to use the property file in this workflow we will have to pass the **–config** while running the workflow.

```
oozie job --oozie http://host_name:8080/oozie --config edgenode_path/job1.properties -D oozie.wf.application.path=hdfs://Namenodepath/pathof_workflow_xml/workflow.xml
–run
```

Note: The property file should be on the edge node (not in HDFS), wherea the workflow and hive scripts will be in HDFS.

At run time, all the parameters in **${}** will be replaced by its corresponding value in the
.properties file.

Also a single property file can have more parameters than required in a single workflow and no error will be thrown. This makes it possible to run more than one workflow by using the same properties file. But if the property file does not have a parameter required by a workflow then an error will occur.

8.6.Coordinators

As done in the previous chapter for the workflow, let's learn concepts of coordinators with an example.

The first two hive actions of the workflow in our example creates the table. We don't need these step when we run the workflow in a coordinated manner each time with a given frequency. So let's modify the workflow which will then be called by our coordinator.

In a real life scenario, the external table will have a flowing data and as soon as the data is loaded in the external table, the data will be processed into ORC and from the file.

Modified Workflow:

```
<workflow-app      xmlns="uri:oozie:workflow:0.4"      name="simple-
Workflow"> <start to="Insert_into_Table" />
<action name="Insert_into_Table">
<hive xmlns="uri:oozie:hive-action:0.4">
<job-tracker>${jobTracker}</job-tracker>
<name-node>${nameNode}</name-node>
<script>${script_name_copy}</script>
<param>${database}</param>
</hive>
<ok to="end" />
<error to="kill_job" /> </action>
<kill name="kill_job">
<message>Job failed</message>
```

```
</kill>
<end name="end" />
</workflow-app>
```

Now let's write a simple coordinator to use this workflow.

```
<coordinator-app xmlns="uri:oozie:coordinator:0.2"
name="coord_copydata_from_external_orc" frequency="5 * * * *" start="2016-
00-18T01:00Z" end="2025-12-31T00:00Z"" timezone="America/Los_Angeles">
<controls>
<timeout>1</timeout>
<concurrency>1</concurrency>
<execution>FIFO</execution>
<throttle>1</throttle>
</controls>
<action>
<workflow>
<app-path>pathof_workflow_xml/workflow.xml</app-path>
</workflow>
</action>
</coordinator-app>
```

Definitions of the above given code is as follows:

- **start:** It means the start datetime for the job. Starting at this time the actions will be materialized.
- **end:** The end datetime for the job. When actions will stop being materialized.
- **timezone:** The timezone of the coordinator application.
- **frequency:** The frequency, in minutes, to materialize actions.

8.7. Control Information

- **timeout:** The maximum time, in minutes, that a materialized action will be waiting for the additional conditions to be satisfied before being discarded. A timeout of 0 indicates that at the time of materialization all the other conditions must be satisfied, else the action will be discarded. A timeout of 0 indicates that if all the input events are not satisfied at the time of action materialization, the action should timeout immediately. A timeout of -1 indicates no timeout, the materialized action will wait forever for the other conditions to be satisfied. The default value is -1.

- **concurrency:** The maximum number of actions for this job that can be running at the same time. This value allows to materialize and submit multiple instances of the coordinator app, and allows operations to catchup on delayed processing. The default value is 1.

- **execution:** Specifies the execution order if multiple instances of the coordinator job have satisfied their execution criteria. Valid values are:

1. FIFO (oldest first) default.
2. LIFO (newest first).
3. LAST_ONLY (discards all older materializations).

Above coordinator will run at a given frequency i.e. every 5th minute of an hour. (Similar to a cron job).

To run this coordinator, use the following command:

- **Oozie job** -- oozie http://host_name:8080/oozie --config edgenode_path/job1.properties

- **oozie.wf.application.path=hdfs**:

//Namenodepath/pathof_coordinator_xml/coordinator.xml -d "2 minute"` -run-d "2minute" will ensure that the coordinator starts only after 2 minutes of when the job was submitted.

The above coordinator will call the workflow which in turn will call the hive script. This script will insert the data from external table to hive the managed table.

8.8.Coordinator Job Status

Similar to the workflow, parameters can be passed to a coordinator also using the

.properties file. These parameters are resolved using the configuration properties of Job configuration used to submit the coordinator job.

If a configuration property used in the definitions is not provided with the job configuration used to submit a coordinator job, the value of the parameter will be undefined and the job submission will fail.

At any time, a coordinator job is in one of the following statuses: **PREP, RUNNING, PREPSUSPENDED, SUSPENDED, PREPPAUSED, PAUSED, SUCCEEDED, DONWITHERROR, KILLED, FAILED**.

Valid coordinator job status transitions are:

- **PREP** – PREPSUSPENDED | PREPPAUSED | RUNNING | KILLED
- **RUNNING** – SUSPENDED | PAUSED | SUCCEEDED | DONWITHERROR | KILLED | FAILED
- **PREPSUSPENDED** – PREP | KILLED
- **SUSPENDED** – RUNNING | KILLED
- **PREPPAUSED** – PREP | KILLED
- **PAUSED** – SUSPENDED | RUNNING | KILLED

When a coordinator job is submitted, Oozie parses the coordinator job XML. Oozie then creates a record for the coordinator with status **PREP** and returns a **unique ID**. The coordinator is also started immediately if the pause time is not set.

When a user requests to suspend a coordinator job that is in status **PREP**, Oozie puts the job in the status **PREPSUSPEND**. Similarly, when the pause time reaches for a coordinator job with the status **PREP**, Oozie puts the job in the status **PREPPAUSED**.

Conversely, when a user requests to resume a **PREPSUSPEND** coordinator job, Oozie puts the job in status **PREP**. And when the pause time is reset for a coordinator job and job status is **PREPPAUSED**, Oozie puts the job in status **PREP**.

When a coordinator job starts, Oozie puts the job in status **RUNNING** and starts materializing workflow jobs based on the job frequency.

When a user requests to kill a coordinator job, Oozie puts the job in status **KILLED** and it sends kill to all submitted workflow jobs. If any coordinator action finishes with not **KILLED**, Oozie puts the coordinator job into **DONEWITHERROR** .

When a user requests to suspend a coordinator job that is in status **RUNNING**, Oozie puts the job in status **SUSPEND** and it suspends all the submitted workflow jobs.

When pause time reaches for a coordinator job that is in status **RUNNING**, Oozie puts the job in status **PAUSED**.

Conversely, when a user requests to resume a **SUSPEND** coordinator job, Oozie puts the job in status **RUNNING**. And when pause time is reset for a coordinator job and job status is **PAUSED**, Oozie puts the job in status **RUNNING**.

A coordinator job creates workflow jobs (commonly coordinator actions) only for the duration of the coordinator job and only if the coordinator job is in **RUNNING** status. If the coordinator job has been suspended, when resumed it will create all the coordinator actions that should have been created during the time it was suspended, actions will not be lost, they will be delayed.

When the coordinator job materialization finishes and all the workflow jobs finish, Oozie updates the coordinator status accordingly. For example, if all the workflows are **SUCCEEDED**, Oozie puts the coordinator job into **SUCCEEDED** status. However, if any workflow job finishes with not **SUCCEEDED** (e.g. **KILLED** or **FAILED** or **TIMEOUT**), then Oozie puts the coordinator job into **DONEWITHERROR**. If all coordinator actions are **TIMEDOUT**, Oozie puts the coordinator job into **DONEWITHERROR**.

Parametrization of a Coordinator:

The workflow parameters can be passed to a coordinator as well using the **.properties** file These parameters are resolved using the configuration properties of Job configuration used to submit the coordinator job.

If a configuration property used in the definition is not provided with the job configuration used to submit a coordinator job, the value of the parameter will be undefined and the job submission will fail.

The **Oozie Bundle system** allows the user to define and execute a bunch of coordinator applications often called a data pipeline. There is no explicit dependency among the coordinator applications in a bundle. However, a user could use the data dependency of coordinator applications to create an implicit data application pipeline.

The user will be able to start/stop/suspend/resume/rerun in the bundle level resulting in a better and easy operational control.

8.9.Bundle

Let's extend our workflow and coordinator example to a bundle.

```
<bundle-app xmlns='uri:oozie:bundle:0.1'
name='bundle_copydata_from_external_o
rc'>
<controls>    <kick-off-time>${kickOffTime}</kick-off-
time>
</controls>
<coordinator name='coord_copydata_from_external_orc' >
<app-path>path of_coordinator_xml</app-path>
<configuration>
<property>
<name>startTime1</name>    <value>time
to start</value>
</property>
</configuration>
</coordinator>
</bundle-app>
```

Kick-off-time The time when a bundle should start and submit coordinator applications.

There can be more than one coordinator in a bundle.

Bundle Job Status:

At any time, a bundle job is in one of the following status: PREP, RUNNING, PREPSUSPENDED, SUSPENDED, PREPPAUSED, PAUSED, SUCCEEDED, DONWITHERROR, KILLED, FAILED.

Valid bundle job status transitions are:

- **PREP** – PREPSUSPENDED | PREPPAUSED | RUNNING | KILLED
- **RUNNING** – SUSPENDED | PAUSED | SUCCEEDED | DONWITHERROR | KILLED | FAILED
- **PREPSUSPENDED** – PREP | KILLED
- **SUSPENDED** – RUNNING | KILLED
- **PREPPAUSED** – PREP | KILLED
- **PAUSED** – SUSPENDED | RUNNING | KILLED

When a bundle job is submitted, Oozie parses the bundle job XML. Oozie then creates a record for the bundle with status **PREP** and returns a unique ID.

When a user requests to suspend a bundle job that is in **PREP** state, Oozie puts the job in status **PREPSUSPEND**. Similarly, when the pause time reaches for a bundle job with **PREP** status, Oozie puts the job in status **PREPPAUSED**.

Conversely, when a user requests to resume a PREPSUSPENDED bundle job, Oozie puts the job in status **PREP**. And when pause time is reset for a bundle job that is in **PREPPAUSED** state, Oozie puts the job in status **PREP**.

There are two ways a bundle job could be started. * If **kick-off-time** (defined in the bundle xml) reaches. The default value is null, which means starts coordinators NOW. * If user sends a start request to START the bundle.

When a bundle job starts, Oozie puts the job in status **RUNNING** and it submits the all coordinator jobs.

When a user requests to kill a bundle job, Oozie puts the job in status **KILLED** and it sends kill to all submitted coordinator jobs

When a user requests to suspend a bundle job that is not in **PREP** status, Oozie puts the job in status **SUSPEND** and it suspends all submitted coordinator jobs

When pause time reaches for a bundle job that is not in **PREP** status, Oozie puts the job in status **PAUSED**. When the paused time is reset, Oozie puts back the job

in status **RUNNING**.

When all the coordinator jobs finish, Oozie updates the bundle status accordingly. If all coordinators reach to the **same terminal state**, the bundle job status also moves to the same status. For example, if all coordinators are **SUCCEEDED**, Oozie puts the bundle job into **SUCCEEDED** status. However, if all coordinator jobs don't finish with the same status, Oozie puts the bundle job into **DONEWITHERROR**.

8.10.Command Line Tools

We have seen a few commands earlier to run the jobs of workflow, coordinator and bundle. Oozie provides a command line utility, Oozie, to perform job and admin tasks.

```
oozie version : show client version
```

Following are some of the other job operations:

```
oozie job <OPTIONS> :
-action <arg>        coordinator rerun on action ids (requires -rerun);
coordinator log retrieval on action ids (requires -log)
-auth <arg>          select authentication type [SIMPLE|KERBEROS]
-change <arg>        change a coordinator/bundle job
-config <arg>        job configuration file '.xml' or '.properties'
-D <property=value>  set/override value for given property
-date <arg>          coordinator/bundle rerun on action dates (requires -rerun)
-definition <arg>        job definition
-doas <arg>          doAs user, impersonates as the specified user
-dryrun              Supported in Oozie-2.0 or later versions ONLY - dryrun or
test run a coordinator job, job is not queued
-info <arg>          info of a job
-kill <arg>          kill a job
-len <arg>           number of actions (default TOTAL ACTIONS, requires -info)
```

-localtime use local time (default GMT)

-log <arg> job log

-nocleanup do not clean up output-events of the coordinator rerun actions (requires -rerun)

-offset <arg> job info offset of actions (default '1', requires -info)

-oozie <arg> Oozie URL

-refresh re-materialize the coordinator rerun actions (requires -rerun)

-rerun <arg> rerun a job (coordinator requires -action or -date; bundle requires -coordinator or -date)

-resume <arg> resume a job

-run run a job

-start <arg> start a job

-submit submit a job

-suspend <arg> suspend a job

-value <arg> new endtime/concurrency/pausetime value for changing a coordinator job; new pausetime value for changing a bundle job

-verbose verbose mode

To check the status of the job, following commands can be used.

-auth <arg> select authentication type [SIMPLE|KERBEROS]

-doas <arg> doAs user, impersonates as the specified user.

-filter <arg> user=<U>; name=<N>; group=<G>; status=<S>; ...

-jobtype <arg> job type ('Supported in Oozie-2.0 or later versions ONLY - coordinator' or 'wf' (default))

-len <arg> number of jobs (default '100')

-localtime use local time (default GMT)

-offset <arg> jobs offset (default '1')

-oozie <arg> Oozie URL

-verbose verbose mode

For example: To check the status of the Oozie system you can run the following

command:

oozie admin -oozie http://localhost:8080/oozie -status

Validating a Workflow XML –

oozie validate myApp/workflow.xml

It performs an XML Schema validation on the specified workflow XML file.

8.11.Action Extensions

We have seen hive extensions. Similarly, Oozie provides more action extensions few of them are as below:

Email Action

The email action allows sending emails in Oozie from a workflow application. An email action must provide to addresses, cc addresses (optional), a subject and a body. Multiple recipients of an email can be provided as comma separated addresses.

All the values specified in the email action can be parameterized (templated) using EL expressions.

Example:

```
<workflow-app name="sample-wf" xmlns="uri:oozie:workflow:0.1">
...
<action name="an-email">
<email xmlns="uri:oozie:email-action:0.1">
<to>julie@xyz.com,max@abc.com</to>
<cc>jax@xyz.com</cc>
<subject>Email notifications for ${wf:id()}</subject> <body>The wf
${wf:id()} successfully completed.</body>
</email>
<ok to="main_job"/> <error
to="kill_job"/>
</action>
</workflow-app>
```

Shell Action

The shell action runs a Shell command. The workflow job will wait until the Shell command completes before continuing to the next action.

To run the Shell job, you have to configure the shell action with the **=job-tracker=, name-node and Shell exec** elements as well as the necessary arguments and configuration. A shell action can be configured to create or delete HDFS directories before starting the Shell job.

The shell launcher configuration can be specified with a file, using the job-xml element, and inline, using the configuration elements.

Example:

How to run any shell script?

```
<workflow-app xmlns='uri:oozie:workflow:0.3' name='shell-wf'>
<start to='shell1' />
<action name='shell1'>
<shell xmlns="uri:oozie:shell-action:0.1"> <job-
tracker>${jobTracker}</job-tracker> <name-
node>${nameNode}</name-node>
<file>path_of_file_name</file>
                </shell>
<ok to="end" />
<error to="fail" />
</action>
<kill name="fail">
<message>Script failed, error
message[${wf:errorMessage(wf:lastErrorNode())}]</messa
ge>
</kill>
<end name='end' />
</workflow-app>
```

Similarly, we can have many more actions like **ssh, sqoop, java action**, etc.

MONGODB

9.1.MongoDB Introduction and Use

9.1.1.General database concept

Relationship database

In general database theory, database could be differentiated to two types: relationship database and non-relationship database. Usually what we talk about database is relationship database, such as Oracle, DB2 and Sql Server. Relationship databases usually have tabs to maintain the entities and the

relationship between entities, which offers strict constraints and multiple manipulations to data.

Non-Relationship database

Non-relationship database didn't offer that features which means usually it cannot provide full ACID(*atomicity, consistency, isolation, durability*). So it will not replace RDBMS in the future because of its weakness in business consistency. But it will show its value in such demand that require less consistency but more operation such like storing and retrieving great quantities of data

NoSQL

NoSQL is usually interpreted as **not only Sql**, it is recognized as non-relationship. NoSQL database systems gain scalability and performance for data model. NoSQL systems are often optimized for retrieving and appending operations since the storage architecture is record. The most popular record type of Nosql is key-value stores. NoSQL offers more storage operation performance due to its less constraint feature and flexible storage type.

9.2.What is MongoDB

MongoDB is one kind of NoSQL database which is popular among many enterprises such as SAP, Taobao. Mongo stores data using documents which is called BSON.BSON is a data format which is like JSON in javascript, making application using this data model easier and faster

9.3.Why MongoDB

Why we use MongoDB, here's some important feature of mongoDB

- Full Index Support : you can just use index like what you do in RDBMS
- Replication & High Availability : MongoDB supports asynchronous replication of data between servers for failover and redundancy
- Auto-Sharding : MongoDB scales horizontally by an auto-sharding (partitioning) architecture. Like many other database, you can add new machine without database down time. What's more failover is also a feature of them.
- Querying: if you have experience in javascript and such program language, then querying is easier to you
- Map/Reduce: useful for batch processing of data and aggregation operations, just like 'group by ' in SQL

- GridFS: MongoDB allows you to store big file without complicating your stack

9.4.How to use MongoDB

Here we are going to introduce how to setup a mongodb in server and use a third-party client to manipulate data, then we will show how to configure the driver in an IDE eclipse to let you program language know mongo.

MONGO SHELL

```
> show dbs
admin  0.000GB
local  0.000GB
```

```
> show collections
>
```

```
> use batch100
switched to db batch100
>db
batch100
```

```
> obj1 = {name:"gopal",addr:"HYD",state:"TS"}
{ "name" : "gopal", "addr" : "HYD", "state" : "TS" }

> obj2 = { name:"Hadoop", natureoftech:"BigData" }
{ "name" : "Hadoop", "natureoftech" : "BigData" }
```

```
>db.coll.save(obj1)
WriteResult({ "nInserted" : 1 })

>db.coll.save(obj2)
WriteResult({ "nInserted" : 1 })
```

```
>db.coll.find()
{ "_id" : ObjectId("58f052831467d39d12f642f3"), "name" : "gopal", "addr" :
"HYD", "state" : "TS" }
{ "_id" : ObjectId("58f052881467d39d12f642f4"), "name" : "Hadoop",
"natureoftech" : "BigData" }
```

```
> show dbs
admin    0.000GB
batch100  0.000GB
local    0.000GB
> show collections
coll
```

```
>

---

>db.coll.save( {Name:"Hadoop",TechOffer:"kelly" } )
WriteResult({ "nInserted" : 1 })

>db.coll.find()
{ "_id" : ObjectId("58f052831467d39d12f642f3"), "name" : "gopal", "addr" :
"HYD", "state" : "TS" }
{ "_id" : ObjectId("58f052881467d39d12f642f4"), "name" : "Hadoop",
"natureoftech" : "BigData" }
{ "_id" : ObjectId("58f0658febc3567b7213acab"), "Name" : "Hadoop",
"TechOffer" : "kelly" }

---

>db.coll.count()
3
>db.coll.find( {name:"gopal" } )
{ "_id" : ObjectId("58f052831467d39d12f642f3"), "name" : "gopal", "addr" :
"HYD", "state" : "TS" }

---

>db.coll.insert( {_id:1,name:"Kelly",TechOffer:"hadoop,sas,tableau" })
WriteResult({ "nInserted" : 1 })

>db.coll.count()
4

>db.coll.find()
{ "_id" : ObjectId("58f052831467d39d12f642f3"), "name" : "gopal", "addr" :
"HYD", "state" : "TS" }
{ "_id" : ObjectId("58f052881467d39d12f642f4"), "name" : "Hadoop",
"natureoftech" : "BigData" }
{ "_id" : ObjectId("58f0658febc3567b7213acab"), "Name" : "Hadoop",
"TechOffer" : "kelly" }
{ "_id" : 1, "name" : "Kelly", "TechOffer" : "hadoop,sas,tableau" }

---

>db.coll.insert( {_id:1,name:"Kelly",TechOffer:"hadoop,sas,tableau" })
WriteResult({
	"nInserted" : 0,
	"writeError" : {
		"code" : 11000,
		"errmsg" : "E11000 duplicate key error collection: batch100.coll
index: _id_ dup key: { : 1.0 }"
	}
})
```

```
>db.coll.update ( {name:"Kelly" } , { $set:{rating:1 } } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

>db.coll.find()
{ "_id" : ObjectId("58f052831467d39d12f642f3"), "name" : "gopal", "addr" :
"HYD", "state" : "TS" }
{ "_id" : ObjectId("58f052881467d39d12f642f4"), "name" : "Hadoop",
"natureoftech" : "BigData" }
{ "_id" : ObjectId("58f0658febc3567b7213acab"), "Name" : "Hadoop",
"TechOffer" : "kelly" }
{ "_id" : 1, "name" : "Kelly", "TechOffer" : "hadoop,sas,tableau", "rating" : 1 }
>
```

```
>db.coll.update ( {name:"Kelly" } , { $unset:{rating:1 } } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

>db.coll.find()
{ "_id" : ObjectId("58f052831467d39d12f642f3"), "name" : "gopal", "addr" :
"HYD", "state" : "TS" }
{ "_id" : ObjectId("58f052881467d39d12f642f4"), "name" : "Hadoop",
"natureoftech" : "BigData" }
{ "_id" : ObjectId("58f0658febc3567b7213acab"), "Name" : "Hadoop",
"TechOffer" : "kelly" }
{ "_id" : 1, "name" : "Kelly", "TechOffer" : "hadoop,sas,tableau" }
>
```

```
>db.coll.find()
{ "_id" : ObjectId("58f052831467d39d12f642f3"), "name" : "gopal", "addr" :
"HYD", "state" : "TS" }
{ "_id" : ObjectId("58f052881467d39d12f642f4"), "name" : "Hadoop",
"natureoftech" : "BigData" }
{ "_id" : ObjectId("58f0658febc3567b7213acab"), "Name" : "Hadoop",
"TechOffer" : "kelly" }
{ "_id" : 1, "name" : "Kelly", "TechOffer" : "hadoop,sas,tableau" }

>db.coll.remove( {name:"Hadoop"} )
WriteResult({ "nRemoved" : 1 })


>db.coll.find()
{ "_id" : ObjectId("58f052831467d39d12f642f3"), "name" : "gopal", "addr" :
"HYD", "state" : "TS" }
{ "_id" : ObjectId("58f0658febc3567b7213acab"), "Name" : "Hadoop",
"TechOffer" : "kelly" }
{ "_id" : 1, "name" : "Kelly", "TechOffer" : "hadoop,sas,tableau" }
>
```

```
> show collections
coll
>db.coll.drop()
true
> show collections
>exit
bye
```

9.4.Mongo java api

```
import com.mongodb.MongoClient;
import com.mongodb.MongoException;
import com.mongodb.WriteConcern;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.BasicDBObject;
import com.mongodb.DBObject;
import com.mongodb.DBCursor;
import com.mongodb.ServerAddress;
import java.util.Arrays;

public class MongoDBJDBC {
  public static void main( String args[] ){
    try{
          // To connect to mongodb server
      MongoClient mongoClient = new MongoClient( "localhost" , 27017 );
      // Now connect to your databases
      DB db = mongoClient.getDB( "test" );
          System.out.println("Connect to database successfully");
     // boolean auth = db.authenticate(myUserName, myPassword);
     //      System.out.println("Authentication: "+auth);

     DBCollection coll = db.getCollection("testCollection");

     BasicDBObject doc = new BasicDBObject("name", "MongoDB")
     .append("type", "database")
     .append("count", 1)
     .append("info", new BasicDBObject("x", 203).append("y", 102));
    coll.insert(doc);
}catch(Exception e){
        System.err.println( e.getClass().getName() + ": " + e.getMessage() );
      }
  }
}
```

```
gopal@ubuntu:~/mongo$
gopal@ubuntu:~/mongo$
gopal@ubuntu:~/mongo$ export CLASSPATH=".:/home/gopal/mongo/mongo-java-driver-2.10.1.jar"
```

```
gopal@ubuntu:~/mongo$
gopal@ubuntu:~/mongo$ javac MongoDBJDBC.java
```

```
gopal@ubuntu:~/mongo$ ll
total 408
drwxr-xr-x  2 gopal hadoop   4096 Feb  3 03:23 ./
drwxr-xr-x 28 gopal hadoop   4096 Feb  3 03:19 ../
-rw-r--r--  1 gopal hadoop   1665 Feb  3 03:23 MongoDBJDBC.class
-rw-r--r--  1 gopal hadoop   1229 Feb  3 03:20 MongoDBJDBC.java
-rw-------  1 gopal hadoop 398520 Feb  3 03:22 mongo-java-driver-2.10.1.jar
gopal@ubuntu:~/mongo$
```

```
gopal@ubuntu:~/mongo$ java MongoDBJDBC
Connect to database successfully
gopal@ubuntu:~/mongo$
```

```
>
> show dbs
local   (empty)
test    0.0625GB
> use test
switched to db test
> show collections
system.indexes
testCollection
> db.testCollection.find()
{ "_id" : ObjectId("58946850e4b04754bbbc9f38"), "name" : "MongoDB", "type" : "database", "count" : 1, "info" : { "x" : 203, "y" : 102 } }
{ "_id" : ObjectId("58946be8e4b04cceceb7f8b5"), "name" : "MongoDB", "type" : "database", "count" : 1, "info" : { "x" : 203, "y" : 102 } }
>
```

MAHOUT

10.1.What is Machine Learning?

Machine learning is a branch of artificial intelligence (AI) that focuses on the development of computer programs that can teach themselves to grow and change when exposed to new data. It is concerned with the design and development of algorithms that can take complex input data and can make intelligent decision based on the input data.

10.2.Types of machine learning

1.) **Supervised Learning** - This type of machine learning is concerned with associating some undefined data or unseen document to some predefined label of the training data in order to predict the value of any valid input. Common examples of supervised learning include classifying e-mail messages as spam. Mainly all classification algorithms in mahout for example, Naive bayes classification follows supervised learning.

2.) **Unsupervised Learning** - This type of machine learning is concerned with making sense out of complex hard to understand data by creating some similarity or some interesting patterns. No labels are associated with it. Mainly all clustering algorithms in mahout for example, kmeans, canopy etc follow unsupervised learning.

3.) **Semi-Supervised Learning** - Semi-supervised learning is concerned with defining an undefined data or unseen document in the presence of both labeled and unlabeled data. It is a combination of supervised and unsupervised learning. The main aim of semi-supervised learning is to demonstrate how combining both labeled and unlabeled data can change the learning behavior.

10.3.What is Mahout?

Apache Mahout is an open source project by the Apache Software Foundation (ASF) open to implementations of all kinds of machine learning techniques with the goal of creating scalable algorithms that are free to use under the Apache license.

In order to see the algorithms currently implemented in mahout type the following command in the terminal

i.) export MAHOUT_HOME= absolute path where mahout directory is present
ii.) cd $MAHOUT_HOME
iii.) bin/mahout

The output is shown as follows

```
BHUKRK262420D:mahout-distribution-0.5 Archana$ bin/mahout
Running on hadoop, using HADOOP_HOME=/Users/Archana/installations/hadoop-0.20.2-cdh3u3
No HADOOP_CONF_DIR set, using /Users/Archana/installations/hadoop-0.20.2-cdh3u3/src/conf
An example program must be given as the first argument.
Valid program names are:
  arff.vector: : Generate Vectors from an ARFF file or directory
  canopy: : Canopy clustering
  cat: : Print a file or resource as the logistic regression models would see it
  cleansvd: : Cleanup and verification of SVD output
  clusterdump: : Dump cluster output to text
  dirichlet: : Dirichlet Clustering
  eigencuts: : Eigencuts spectral clustering
  evaluateFactorization: : compute RMSE of a rating matrix factorization against probes in memory
  evaluateFactorizationParallel: : compute RMSE of a parallel matrix factorization against probes
  fkmeans: : Fuzzy K-means clustering
  fpg: : Frequent Pattern Growth
  itemsimilarity: : Compute the item-item-similarities for item-based collaborative filtering
  kmeans: : K-means clustering
  lda: : Latent Dirichlet Allocation
  ldatopics: : LDA Print Topics
  lucene.vector: : Generate Vectors from a Lucene index
  matrixmult: : Take the product of two matrices
  meanshift: : Mean Shift clustering
  parallelALS: : ALS-WR factorization of a rating matrix
  predictFromFactorization: : predict preferences from a factorization of a rating matrix
  prepare20newsgroups: : Reformat 20 newsgroups data
  recommenditembased: : Compute recommendations using item-based collaborative filtering
  rowid: : Map SequenceFile<Text,VectorWritable> to {SequenceFile<IntWritable,VectorWritable>, SequenceFile<IntWritable,Text>}
  rowsimilarity: : Compute the pairwise similarities of the rows of a matrix
  runlogistic: : Run a logistic regression model against CSV data
  seq2sparse: : Sparse Vector generation from Text sequence files
  seqdirectory: : Generate sequence files (of Text) from a directory
  seqdumper: : Generic Sequence File dumper
  seqwiki: : Wikipedia xml dump to sequence file
  spectralkmeans: : Spectral k-means clustering
  splitDataset: : split a rating dataset into training and probe parts
  ssvd: : Stochastic SVD
  svd: : Lanczos Singular Value Decomposition
  testclassifier: : Test Bayes Classifier
  trainclassifier: : Train Bayes Classifier
  trainlogistic: : Train a logistic regression using stochastic gradient descent
  transpose: : Take the transpose of a matrix
  vectordump: : Dump vectors from a sequence file to text
  wikipediaDataSetCreator: : Splits data set of wikipedia wit feature like country
  wikipediaXMLSplitter: : Reads wikipedia data and creates ch
```

All of these can be accessed via the $MAHOUT_HOME/bin/mahout command line driver. Each of these needs certain arguments as input to generate the output. The parameter that each algorithm requires can be known by typing the following command in the command line

$MAHOUT_HOME/bin/mahout name of the algorithm.

For example: - In order to know the parameters needed to run kmeans the command is

$MAHOUT_HOME/bin/mahout kmeans

The parameters are shown as follows

```
BHUKRK2624200:mahout-distribution-0.5 Archana3 bin/mahout kmeans
Running on hadoop, using HADOOP_HOME /Users/Archana/installations/hadoop-0.20.2-cdh3u3
No HADOOP_CONF_DIR set, using /Users/Archana/installations/hadoop-0.20.2-cdh3u3/src/conf
12/09/28 17:29:35 WARN driver.MahoutDriver: No kmeans.props found on classpath, will use command-line arguments only
12/09/28 17:29:35 ERROR common.AbstractJob: Missing required option --clusters
usage: <command> [Generic Options] [Job Specific Options]
Generic Options:
 -archives <paths>              comma separated archives to be unarchived
                                on the compute machines.
 -conf <configuration file>     specify an application configuration file
 -D <property=value>            use value for given property
 -files <paths>                 comma separated files to be copied to the
                                map reduce cluster
 -fs <local|namenode:port>      specify a namenode
 -jt <local|jobtracker:port>    specify a job tracker
 -libjars <paths>               comma separated jar files to include in
                                the classpath.
 -tokenCacheFile <tokensFile>   name of the file with the tokens
Missing required option --clusters
Usage:
 [--input <input> --output <output> --distanceMeasure <distanceMeasure>
--clusters <clusters> --numClusters <k> --convergenceDelta <convergenceDelta>
--maxIter <maxIter> --overwrite --clustering --method <method> --help --tempDir
<tempDir> --startPhase <startPhase> --endPhase <endPhase>]
--clusters (-c) clusters    The input centroids, as Vectors.  Must be a
                            SequenceFile of Writable, Cluster/Canopy.  If k is
                            also specified, then a random set of vectors will
                            be selected and written out to this path first
12/09/28 17:29:35 INFO driver.MahoutDriver: Program took 93 ms
```

10.4. Mahout algorithms are roughly divided into 4 major sections

- Collaborative filtering
- Clustering
- Categorization
- Mahout utilities

10.4.1. Collaborative filtering

In today's busy world online shopping is a boon. Gradually the entire society is moving towards online shopping. Thus it is neccesary to improvise this sector more and more so as to make it user friendly as well as a source of lucrative income. Hence the upgradation of technology is mandatary which brings in new features.One such important feature is generating recommendations so as to make it easy for the users to make their choice from almost unlimited items offered by these sites.

Collaborative filtering (CF) is a machine learning technique used for generating recommendations. It uses information's like ratings,user preferences etc to generate recommendations.

There are basically two ways of generating recommendations. They are

i.) **User-based**: Recommend items by finding similar users. For example if a user has purchased an iphone and a second user has even purchased a iphone along with other products then they are supposed to be similar users and the items

purchased by the second user other then the iphone are recommended to the 1st user. This is often harder to scale because of the dynamic nature of users.

ii.) **Item-based**: Item based recommendations calculates similarity between items and creates a similarity matrix from which recommendations are generated

Mahout provides a set of components from where we can create our own recommendation engine. These components are

i.) **DataModel** - A data model is an interface to inform about user preferences. It draws data from any source like databases.

ii.) **UserSimilarity** - It defines a notion of similarity between two users. This is a crucial part of a User Based recommendations engine. This is attached to a neighborhood implementation. Similar to User-Similarity we have Item-Similarity for Item Based Recommendations.

iii.) **User neighborhood** - It finds a neighborhood of similar user/items, for examples nearest 5 users and generates recommendations.

iv.) **Recommender** - It is the core component in mahout. Given a data model it produces recommendation.

Examples of Collaborative Filtering Algorithms :-

- Non-distributed recommenders ("Taste")
- Distributed Item-Based Collaborative Filtering
- Collaborative Filtering using a parallel matrix factorization

10.4.2. Clustering

Clustering is basically unsupervised learning that is useful to group together, or cluster, similar items automatically. The basic need of clustering algorithm is to make the complex data set or the hard to understand data set easier to comprehend by revealing some similarity or some some pattern.For instance. grouping the similar movies into a particular genre.

Clustering calculates the similarity between items in the collection and groups them together. However for calculating the similarity the algorithm requires data that is in a format suitable for processing i.e the data should be in vector representation sometimes called a feature vector. Once converted into vectors it calculates the distance between two items and groups the items that are close in distance.

Mahout provides us many utilities with the help of which we can convert any text or numeric data into vector form and then use it as an input for clustering algorithm.

The utilities are

i.) seqdirectory - This utility converts the text or numeric data in a sequence format

ii.) seq2sparse - This utility converts the sequence format to vector format.

After generating the vector form we can pass that data to any clustering algorithm

Examples of some clustering algorithm implemented by mahout

1.) Canopy: Creates initial seeds for other clustering algorithms.

2.) k-Means (and fuzzy k-Means): Clusters items into k clusters based on the distance the items are from the centroid, or center, of the previous iteration.

3.) Mean-Shift: This algorithm differs from kmeans in the sense that it does not require a prior knowledge about the number of clusters and can produce arbitrarily shaped clusters.

10.4.3.Categorization

Categorization (often also called classification) is a technique to categorize the unknown and unseen data into some predefined categories thus grouping them together. The basic concept of classification algorithms is assigning features to a document by calculating a variety of statistics and associating these features with a predefined label, thus creating a model that can be used later to classify unseen documents. For example, a simple approach to classification is keeping track of the words associated with a label, as well as the number of times those words are seen for a given label. Then, when a new document is classified, the words in the document are looked up in the model, probabilities are calculated, and the best result is output, usually along with a score indicating the confidence the result is correct.

Features for classification might include words, weights for those words (based on frequency, for instance), parts of speech, and so on.

Examples of categorization algorithms: -

- Logistic Regression (SGD)
- Bayesian
- Support Vector Machines (SVM)
- Neural Network
- Random Forests

- Restricted Boltzmann Machines
- Hidden Markov Models

10.4.4.Mahout utilities

There are some algorithms, which help in preparing content into formats for Mahout and are called MAHOUT UTILITIES. They are classified into 3 categories

1. Creating Vectors from Text - These utilities allow to produce Mahout Vector representations. There are two utilities for converting a directory of text documents into vector

- seqdirectory - converts the text or numeric data in a sequence format in a chunked manner
- seq2sparse - Converts the sequence files into mahout vector representations

2.) Creating text from vectors - These utilities allow to produce text from vectors. Utilities available for the same are

- seqdumper – dumps the vector files to text
- clusterdump – dumps the cluster output to text
- vectordumper - dumps vector from sequence file to text

3.) Viewing Result - These Utilities particularly is used for viewing the clusters generated by any clustering algorithm. Various utilities are present for various clustering algorithms. For example

- DisplayCanopy - uses Canopy clustering
- DisplayKMeans - uses k-Means clustering and so on

However there are many other algorithms, which do not specifically fall into the above category, belong to categories like

1. **Pattern Mining:** Example is Parallel FP Growth Algorithm (Also known as Frequent Itemset mining)

2. **Regression**: Examples are

- Locally Weighted Linear Regression
- Dimension reduction
- Singular value Decomposition

- Stochastic Singular Value Decomposition with PCA workflow
- Gaussian Discriminative Analysis (GDA)

3. **Vector Similarity**: Example is RowSimilarityJob

Reference(s)

1. http://mahout.apache.org

10.5.Mahout Classifier Demo Using Naive Bayes

Mahout is an open source project developed by Apache foundation which consists of the ML (Machine Learning) libraries

Introduction

This document gives an introduction about how to execute Mahout Classifier Example using naïve Bayes algorithm, along with the screenshots, using Hadoop on Linux platform.

Note: The volume of the dataset used for training is very small; therefore the accuracy of the result cannot be relied upon. The better the volume of the training dataset, the more accurate the result will be.

It should be noted that name of the directories (labels) to be created in both training

```
gopal@ubuntu:~/SCALA$ cat listPrg.scala
object listPrg
{
  def main(args: Array[String])
    {
      val tech = List.fill(5)("Spark") // Repeats Spark for 5 times.
      println( "Technology : " + tech )
      val num = List.fill(6)(2)  // Repeats num 2 for 6 times.
      println( "num : " + num )
    }
}
gopal@ubuntu:~/SCALA$
```

and testing folders should be same as the metric under consideration. Also, at any point there should be at least one file in each of the directories (labels).

Use cases:

- Categorizing Web pages automatically according to genre (sports, economy, war, and so on) or to mark e-mail messages as spam
- Predicting tumor cells as benign or malignant.

Steps for running a classifier engine using Mahout

- Prepare the datasets for training as well as testing. In the current example the datasets looks like as mentioned below.

Train data:

- We have considered two labels, namely – *GoodReview* and *BadReview*. First create a folder named **train**, and create 2 folders inside this folder with the metric under consideration. In this case *GoodReview* and *BadReview*

- There are 22 documents in each of the aforementioned labels.

Below document is one of the samples training dataset.

B_10023 - Notepad

File Edit Format View Help

```
Horrible Customer Service .
It is worst bank with worst experience.
```

There will be no. of documents in GoodReview as well as BadReview.Based on data, algorithm counts occurrence of each word and accordingly classified it as GoodReview or BadReview.

For e.g. in above document "Horrible" word has come, it will count no.of times this particular word has befallen in BadReview section. If it's more, then classified document that comprises "Horrible" word as BadReview.

Test data:

- Test data also consists of same labels as mentioned above with 11 and 15 documents in *GoodReview* and *BadReview* folders respectively.

- Create a folder named **test,** and create 2 folders inside this folder with the metric under consideration. In this case *GoodReview* and *BadReview*

- Below document is one of the sample data that will be used to test the system.

B_10011 - Notepad
File Edit Format View Help
i ar unhappy with my bank

- Based on training data system will test the data which is present in test section. For that it will used Naïve Bayes algorithm.We need to convert the train and test data into vector format as required by the trainclassifier and testclassifier, and it is accomplished using the steps mentioned below :

```
mahout prepare20newsgroups \
  -p /home/htrain02/Anuradha/mahout_5/Datasets/Test_data/train \
  -o /home/htrain02/Anuradha/mahout_5/Datasets/Test_data/train-input \
  -a org.apache.mahout.vectorizer.DefaultAnalyzer \
  -c UTF-8
mahout prepare20newsgroups \
  -p /home/htrain02/Anuradha/mahout_5/Datasets/Test_data/test \
  -o /home/htrain02/Anuradha/mahout_5/Datasets/Test_data/test-input \
  -a org.apache.mahout.vectorizer.DefaultAnalyzer \
  -c UTF-8
```

Explanation for the options used

- –p : Parent directory containing the newsgroups
- –o: The output directory, where the vector converted file will be saved.
- –a: The class name of the analyzer.
- The files created in steps 2 are copied to HDFS location

```
hadoop fs -put \
  /home/htrain02/Anuradha/mahout_5/Datasets/Test_data/test-input \
  ./ Mahout-Poc/bayes-test-input
hadoop fs -put \
  /home/htrain02/Anuradha/mahout_5/Datasets/Test_data/train-input \
  ./ Mahout-Poc/bayes-train-input
```

Generate a model for classification using "Bayes" algorithm, and train data as the input.

```
mahout trainclassifier \
-i ./Mahout-Poc/bayes-train-input \
```

```
-o ./ Mahout-Poc/bayes-model \
-type bayes \
-ng 1 \
-source hdfs
```

Explanation for the options used

- –i: The Directory on HDFS containing the collapsed, properly formatted files.
- –o: The location of the model on the HDFS.
- –type: The type of the classifier.
- –ng: Size of the n-gram. Default value is 1.
- –source: location of the model.
- The testclassifier along with the model generated in the step 4 is used to classify the test data into the predefined categories.

```
mahout testclassifier \
-m ./ Mahout-poc/bayes-model \
-d ./ Mahout-poc/bayes-test-input 2> inter_restults.txt \
-type cbayes \
-ng 1 \
-source hdfs \
-method mapreduce
```

Explanation for the options used

- –m: The path/location of the model on the HDFS.
- –d: The directory where the test documents resides in.
- –method : Method of classification

The following output was obtained by running a classifier with the aforementioned datasets details:

```
12/10/16 15:48:51 INFO bayes.BayesClassifierDriver: ==========================================
Confusion Matrix
-------------------------------------------------------
a       b       c       <--Classified as
9       2       0       | 11        a     = GoodReview
4       11      0       | 15        b     = BadReview
0       0       0       | 0         c     = unknown
Default Category: unknown: 2
```

The output result conveys that:

- Out of 11 documents in GoodReview folder 9 are classified as good review and 2 as bad review. i.e. we can say that **accuracy is 81.81%**
- Out of 15 documents in BadReview folder 4 are classified as good review and 11 as bad review; i.e. we can say that **accuracy in this case is 73.33%**

Conclusion:

This document illustrates the understanding of the usage of the Mahout classifier using the Naïve Bayes algorithm, along with the hands on demonstration of the steps required to run the classifier.

Reference(s)

1. http://hadoop.apache.org/
   http://mahout.apache.org/
2. Mahout in Action By SEAN OWEN, ROBIN ANIL,TED DUNNING,ELLEN FRIEDMAN,2012.

SCALA

11.1.Scala Closures

Scala Closure is a function, whose **return value depends** on the value of one or more variables **declared outside this function.**

```
scala> var fun1 = (i:Int) => i*10;
fun1: Int => Int = <function1>

scala> println(" Result is: " + fun1(2));
 Result is: 20

scala> println(" Result is: " + fun1(1));
 Result is: 10

scala> println(" Result is: " + fun1(0));
 Result is: 0

scala>
```

NOTE: Here "i" is defined as parameter to the function (**fun1()**)..depending on " i" value which we are passing result is varying ..

EXAMPLE 2:

```
scala> var fun2 = (x:String) => x + "Scala";
fun2: String => String = <function1>

scala> println("Result String Is: " + fun2(" Spark "));
Result String Is:  Spark Scala

scala> println("Result String Is: " + fun2(" Java,Phython and  "));
Result String Is:  Java,Phython and  Scala

scala> println("Result String Is: " + fun2(null));
Result String Is: nullScala

scala> println("Result String Is: " + fun2(" "));
Result String Is:  Scala

scala>
```

```
scala> println("Result String Is: " + fun2());
<console>:9: error: not enough arguments for method apply: (v1: String)String in trai
Function1.
Unspecified value parameter v1.
      println("Result String Is: " + fun2());
                                      ^

scala>
```

IN SCRIPT MODE

NOTE: Here in the above script, both "i" and "refVar" are the parameters.. "i" is a formal parameter , "refVar" is a outside reference parameter.

```
gopal@ubuntu:~/SCALA$ nano clousureTest.scala
gopal@ubuntu:~/SCALA$ scalac clousureTest.scala
gopal@ubuntu:~/SCALA$
gopal@ubuntu:~/SCALA$ scala clousureTest
gopalfun(1) value = 5
gopalfun(2) value = 10
gopal@ubuntu:~/SCALA$
```

1.2.FUNCTIONS IN SCALA

11.2.1.Anonymus Functions in Scala

Anonymous functions in source code are called **function literals** and at run time, function literals are instantiated into objects called **function values**.

```
gopal@ubuntu:~/SCALA$ cat anonymusFun.scala
object anonymusFun
{
    def main(args: Array[String])
    {
        var incr = (x:Int) => x+1;
        var multy = (a:Int,b:Int) => a*b;

        println(" INCR VALUE : " + incr(6));
        println(" MULTIPLIED VALUE  : " + multy(5,6));

        println(" NEW INCR VALUE : " + (incr(6)-2));
        println(" NEW MULTIPLIED VALUE  : " + (multy(5,6)+20));
    }
}
gopal@ubuntu:~/SCALA$
```

```
gopal@ubuntu:~/SCALA$ scalac anonymusFun.scala
gopal@ubuntu:~/SCALA$
gopal@ubuntu:~/SCALA$ scala anonymusFun
 INCR VALUE : 7
 MULTIPLIED VALUE  : 30
 NEW INCR VALUE : 5
 NEW MULTIPLIED VALUE  : 50
gopal@ubuntu:~/SCALA$
```

11.2.2.HIGHER ORDER FUNCTIONS IN SCALA

[foreach() , map() , reduce()]

1.Foreach() - > example

```
scala> val courses = List("Spark","Scala","Hadoop","Kafka");
courses: List[String] = List(Spark, Scala, Hadoop, Kafka)

scala> courses.foreach( (c:String) => (println(c)) );
Spark
Scala
Hadoop
Kafka

scala>
```

NOTE: foreach() takes a function (a procedure, to be accurate) and invokes it with every item in the listAlternative way of invoking every item in the list

```
scala> val courses = List("Spark","Scala","Hadoop","Kafka");
courses: List[String] = List(Spark, Scala, Hadoop, Kafka)

scala> courses.foreach( (c:String) => (println(c)) );
Spark
Scala
Hadoop
Kafka

scala> courses.foreach( (c) => (println(c)) );
Spark
Scala
Hadoop
Kafka

scala>
```

2.Map() -> Example

```
scala> val courses = List("Spark","Scala","Hadoop","Kafka");
courses: List[String] = List(Spark, Scala, Hadoop, Kafka)

scala> val cSize = courses.map( (c) => (c.size) );
cSize: List[Int] = List(5, 5, 6, 5)

scala> println(cSize);
List(5, 5, 6, 5)
```

NOTE: map() takes a function that converts a single list element to another value and/or type.

```
scala> val cSize = courses.map( (c) => (c.size) );
cSize: List[Int] = List(5, 5, 6, 5)

scala> println( courses(0)  + "\t" + cSize(0));
Spark    5

scala> println( courses  + "\t" + cSize);
List(Spark, Scala, Hadoop, Kafka)    List(5, 5, 6, 5)

scala>
```

3.Reduce() -> Example

```
scala> val marks = List(20,30,40,50,60);
marks: List[Int] = List(20, 30, 40, 50, 60)

scala> val totalMarks = marks.reduce( (a:Int,b:Int) => (a+b) );
totalMarks: Int = 200

scala> val totalMarks = marks.reduce( (a,b) => (a+b) );
totalMarks: Int = 200

scala>
```

NOTE: reduce () takes a function that combines two list elemen s into a single element

11.3.SCALA LOOPS

(while loop , do..while loop, for loop)

11.3.1.While loop

A while loop statement repeatedly executes a target statement as long as a given condition is true.

Syntax:

```
while(condition)
{
  statement(s);
}
```

Example:

```
object whileLoopPrac
{
    def main(args: Array[String])
    {
    var a =  0;// local variable
    // while  oop execution
    while( a < 20 )
    {
    println( 'Value of a: " + a );
    a = a + 1,
    }
 }
}
```

11.3.2.Do...While loop

Unlike while loop, which tests the loop condition at the top of the loop, the do...while loop checks its condition at the bottom of the loop.

A do...while loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

Syntax:

```
Do
{
    statemer t(s);
}while( condit ion );
```

Example:

```
object doWhileLoopPrac
{
    def main  ırgs: Array[String])
```

```
 {
     var a = 10;
     // Do while loop execution
     do
     {
     println( "Value of a: " + a );
     a = a + 1;
     } while (a > 20)
 }
}
```

11.3.3.for loop

A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

Various forms of for loop in Scala:

1. For Loop with Range
2. For Loop with Collections
3. For Loop with Filters
4. For Loop with Yield

1).For Loop with Range

```
for( var x <- Range )
 {
    statement(s);
 }
```

NOTE: Here Range is the number of numbers that reprasendted as I to J OR I unti J

NOTE: Here "<- " Is known as "Generator" command as this operator only generating individual values from range of numbers..

Example: using "I to J"

```
object forLoopPrac
{
 def main(args: Array[String])
 {
     var a = 0; //
```

```
    //for loop execution with a range ,      "<-" is a generator operator
    for( a <- 1 to 10)
    {
    println( "Value of a: " + a );
    }
  }
}
```

Example 2: using "I until J"

```
object forLoopPrac_Until
{
  def main(args: Array[String])
  {
    var a = 0;
    // until 10 means..10 is NOT included up to 9 only
    for( a <- 1 until 10)
    {
    println( "Value of a: " + a );
    }
  }
}
```

2.For Loop with Collection

```
for( var x <- List )
{
    statement(s);
}
```

NOTE: Here, the List variable is a collection type having a list of elements and for loop iterate through all the elements returning one element in x variable at a time

Example:

```
object forLoopPrac_Collction
```

```
{
 def main(args: Array[String])
 {
     var a = 0;
     val numList = List(1,2,3,4,5,6);

     for( a <- numList)
     {
     println( "Value of a: " + a );
     }
 }
}
```

3.For Loop with Filter

```
for( var x <- List
     if condition1; if condi :ion2...
  )
{
statement(s);
}
```

Example:

```
object forLoopPrac_Filter
{
 def main(args: Array[String])
 {
     var a = 0;
     val numList = List(1,2 3,4,5,6,7,8,9,10);
     for( a <- numList if a != 3; if a < 8 )
     {
     println( "Value of a: " + a
     }
```

```
 }
}
```

Output:

Value of a: 1 Value of a: 2 Value of a: 4 Value of a: 5 Value of a: 6 Value of a: 7

4.For Loop with Filter

To store return values from a for loop in a variable or can return through a function. To do so, you prefix the body of the for expression by the keyword yield as follows

```
var retVal = for{ var x <- List if condition1; if condition2...
} yield x
object forLoopPrac_Yield
{
  def main(args: Array[String])
      {
      var a = 0;
      val numList = List(1,2,3,4,5,6,7,8,9,10);
      // for loop execution with a yield
      var retVal = for{ a <- numList if a != 3; if a < 8 }yield a
      // Now print returned values using another loop.
      for( a <- retVal)
      {
      println( "Value of a: " + a );
      }
}
}
```

11.4.SCALA ARRAYS

To create an array of integers

scala> var arrayElements = Array(1,2,3,4,5,6,7,8,9,10) arrayElements: Array[Int] = Array(1, 2, 3, 4, 5. 6, 7, 8, 9, 10)

To Print the Elements of an array

```
scala> for( x <- arrayElements) { println(" Array Elements: " + x) }

scala> println(arrayElements.length)

10

scala> println(arrayElements(7)) 8
```

To make the SUM of Array Elements

```
scala> var total=0;for ( i <- 0 to (arrayElements.length - 1)) { total += arrayElements(i); }

total: Int = 55 scala>
```

To create an array of strings

```
scala> var arrayEle = Array("AAA","BBB","CCC","DDD","EEE") arrayEle: Array[String] = Array(AAA, BBB, CCC, DDD, EEE)

scala> println(arrayEle(2)) CCC

scala> println(arrayEle.length) 5
```

10.5.Scala Iterators

Scala Iterator is not a collection, but rather a way to access the elements of a collection one by one.

The two basic operations on an iterator are

1.next

2.hasNext

A call to it.next() will return the next element of the iterator and advance the state of the iterator. You can find out whether there are more elements to return using Iterator's it.hasNext method.

Example:

```
scala> val iteObj = Iterator("Spark","Scala","ApacheStorm","Kafka") iteObj: Iterator[String] = non-empty iterator

scala> while(iteObj.hasNext){ println("Element Is: " + iteObj.next()) } Element Is: Spark

Element Is: Scala Element Is: ApacheStorm Element Is: Kafka

scala> val iteObj = Iterator("Spark","Scala","ApacheStorm","Kafka") iteObj: Iterator[String] = non-empty iterator

scala> println(" SIZE OF ITERATOR OBJ : " + iteObj.size)

SIZE OF ITERATOR OBJ: 4

scala> println(" LENGTH OF ITERATOR OBJ : " + iteObj.length) LENGTH OF ITERATOR OBJ: 0
```

scala> val itObjNew = Iterator(10,20,30,40,50,60) itObjNew: Iterator[Int] = non-empty iterator

scala> println(" length of iterator obj : " + itObjNew.length)

length of iterator obj : 6

scala> println(" size of iterator obj : " + itobjnew.size)

size of iterator obj : 0

scala> val itObjNew = Iterator(10,20,30,40,50,60) itObjNew: Iterator[Int] = non-empty iterator

scala> println(" MAX OF ITERATOR OBJ : " + itObjNew.max)

MAX OF ITERATOR OBJ: 60

scala> val itObjNew = Iterator(10,20,30,40,50,60) itObjNew: Iterator[Int] = non-empty iterator

scala> println(" MAX OF ITERATOR OBJ : " + itObjNew.min) MAX OF ITERATOR OBJ: 10

10.6.SCALA COLLECTIONS

10.6.1.List

Scala Lists are quite similar to arrays which means, all the elements of a list have the same type, but there are two important differences. First, lists are immutable, which means elements of a list cannot be changed by assignment.

Second, lists represent a linked list whereas arrays are flat.

```
scala> val names: List[String] = List("Ram", "Lakshman", "Seetha");
names: List[String] = List(Ram, Lakshman, Seetha)

scala> val names: List[String] = List("RamChandra", "Lakshman", "Seetha");
names: List[String] = List(RamChandra, Lakshman, Seetha)

scala>  names = List("SriRamChandra", "Lakshman", "Seetha");
<console>:8: error: reassignment to val
        names = List("SriRamChandra", "Lakshman", "Seetha");
              ^
```

To Access the List Elements

```
scala> println(names(0));
RamChandra

scala> println(names(1));
Lakshman

scala> println(names(2));
Seetha

scala>
```

```
scala> val ranks: List[Int] = List(1, 2, 3, 4)
ranks: List[Int] = List(1, 2, 3, 4)

scala> val ranks = List(1,2,3,4)
ranks: List[Int] = List(1, 2, 3, 4)

scala>
```

Operations on Lists: head, tail & isEmpty

```
scala> val ranks = List(1,2,3,4)
ranks: List[Int] = List(1, 2, 3, 4)

scala> println(ranks.head);
1

scala> println(ranks.tail);
List(2, 3, 4)

scala> println(ranks.isEmpty);
false

scala>
```

All lists can be defined using two fundamental building blocks

1. a tail Nil

2. ::

This is pronounced cons Example:

val names = "Rama" :: ("Lakshman" :: ("Seetha" :: Nil))

```
scala> val names = "Rama" :: ("Lakshman" :: ("Seetha" :: Nil))
names: List[String] = List(Rama, Lakshman, Seetha)

scala>
```

To Concatenate the Two Lists (:::)

```
scala> val names = "Rama" :: ("Lakshman" :: ("Seetha" :: Nil))
names: List[String] = List(Rama, Lakshman, Seetha)

scala> val addresses = "Hyderabad" :: ("Bangalore" :: ("Chennai" :: Nil))
addresses: List[String] = List(Hyderabad, Bangalore, Chennai)
```

```
scala> var name_address = names ::: addresses
name_address: List[String] = List(Rama, Lakshman, Seetha, Hyderabad, Bangalore, Chenn
i)

scala>
```

```
scala> var name_address = List.concat(names,addresses)
name_address: List[String] = List(Rama, Lakshman, Seetha, Hyderabad, Bangalore, Chenn
i)

scala>
```

Alternative: (List.concat (list1, list2))

```
gopal@ubuntu:~/SCALA$ cat listPrg.scala
object listPrg
{
  def main(args: Array[String])
   {
     val tech = List.fill(5)("Spark") // Repeats Spark for 5 times.
     println( "Technology : " + tech )
     val num = List.fill(6)(2)  // Repeats num 2 for 6 times.
     println( "num : " + num )
   }
}
gopal@ubuntu:~/SCALA$
```

```
gopal@ubuntu:~/SCALA$ nano listPrg.scala
gopal@ubuntu:~/SCALA$ scalac listPrg.scala
gopal@ubuntu:~/SCALA$
gopal@ubuntu:~/SCALA$ scala listPrg
Technology : List(Spark, Spark, Spark, Spark, Spark)
num : List(2, 2, 2, 2, 2, 2)
gopal@ubuntu:~/SCALA$
```

List.fill() method creates a list consisting of zero or more copies of the same element as follows:

Partition: splits a list based on where it falls with respect to a predicate function.

```
scala> val ranks = List(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
ranks: List[Int] = List(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

scala> ranks.partition(_ % 2 == 0)
res13: (List[Int], List[Int]) = (List(2, 4, 6, 8, 10),List(1, 3, 5, 7, 9))

scala>
```

Other Operations on Lists:

```
scala> val data = List("Raj","Boby","anandbabu","Danya","rajamouli.s.s");
data: List[String] = List(Raj, Boby, anandbabu, Danya, rajamouli.s.s)

scala> println(data.sorted)
List(Boby, Danya, Raj, anandbabu, rajamouli.s.s)

scala> println(data.sortBy( a => a.length) )
List(Raj, Boby, Danya, anandbabu, rajamouli.s.s)

scala> println(data.sortWith( (a,b)  => a.length > b.length) )
List(rajamouli.s.s, anandbabu, Danya, Boby, Raj)

scala> println(data.sortWith( (a,b)  => a(0) > b(0)) )
List(rajamouli.s.s, anandbabu, Raj, Danya, Boby)

scala> val data = List("Raj","Boby","anandbabu","Danya","rajamouli.s.s");
data: List[String] = List(Raj, Boby, anandbabu, Danya, rajamouli.s.s)

scala> data.last
res8: String = rajamouli.s.s

scala> data.updated(0,"RajMohan")
res9: List[String] = List(RajMohan, Boby, anandbabu, Danya, rajamouli.s.s)

scala> data.updated(2,"Anand")
res10: List[String] = List(Raj, Boby, Anand, Danya, rajamouli.s.s)
```

ZIP Option:

```
scala> val list1 = List(10,20,30,40)
list1: List[Int] = List(10, 20, 30, 40)

scala> val list2 = List("AAA","BBB","CCC","DDD")
list2: List[String] = List(AAA, BBB, CCC, DDD)

scala> val newList = list1.zip(list2)
newList: List[(Int, String)] = List((10,AAA), (20,BBB), (30,CCC), (40,DDD))
```

```
scala> val newList = list2.zip(list1)
newList: List[(String, Int)] = List((AAA,10), (BBB,20), (CCC,30), (DDD,40))

scala> val newList = list1.zip(list1)
newList: List[(Int, Int)] = List((10,10), (20,20), (30,30), (40,40))

scala>
```

```
scala> newList(0)
res15: (Int, Int) = (10,10)

scala> newList(0)._1
res16: Int = 10

scala> newList(0)._2
res17: Int = 10

scala>
```

```
scala> val list1 = List(10,20,30,40)
list1: List[Int] = List(10, 20, 30, 40)

scala> val list2 = List("AAA","BBB")
list2: List[String] = List(AAA, BBB)

scala> val newList = list1.zip(list2)
newList: List[(Int, String)] = List((10,AAA), (20,BBB))

scala> newList.toMap
res18: scala.collection.immutable.Map[Int,String] = Map(10 -> AAA, 20 -> BBB)

scala>
```

```
scala> newList.foreach(println)
(10,AAA)
(20,BBB)

scala> newList.reverse.foreach(println)
(20,BBB)
(10,AAA)

scala> newList.map(a => a)
res21: List[(Int, String)] = List((10,AAA), (20,BBB))

scala>
```

Range

```
scala> 1 to 4
res22: scala.collection.immutable.Range.Inclusive = Range(1, 2, 3, 4)

scala> 1 until 4
res23: scala.collection.immutable.Range = Range(1, 2, 3)

scala> 1 until 4 by 3
res24: scala.collection.immutable.Range = Range(1)

scala>
```

Set: Set is a collection that contains no duplicate elements. There are two kinds of Sets, the immutable and the mutable. The difference between mutable and immutable objects is that when an object is immutable, the object itself can't be changed.

By default, Scala uses the immutable set

```
scala> val ranks = List(1,2,3,4,2,2,3,2)
ranks: List[Int] = List(1, 2, 3, 4, 2, 2, 3, 2)

scala> val ranks = Set(1,2,3,4,2,2,3,2)
ranks: scala.collection.immutable.Set[Int] = Set(1, 2, 3, 4)

scala>
```

```
scala> val ranks = Set("Gopal","Ravi","Gopal","Krishna")
ranks: scala.collection.immutable.Set[String] = Set(Gopal, Ravi, Krishna)

scala> val ranks = Set(100,"Gopal",12000)
ranks: scala.collection.immutable.Set[Any] = Set(100, Gopal, 12000)

scala>
```

```
gopal@ubuntu:~/SCALA$ cat setPrg.scala
object setPrg
 {
  def main(args: Array[String])
   {
    val technologies = Set("Spark", "Scala", "Phython")
    val ranks: Set[Int] = Set()
    println( "Head of Technologies : " + technologies.head )
    println( "Tail of Technologies : " + technologies.tail )
    println( " Is Technology empty? : " + technologies.isEmpty )
    println( " Is Ranks empty? : " + ranks.isEmpty )
   }
}
gopal@ubuntu:~/SCALA$
```

```
gopal@ubuntu:~/SCALA$ scalac setPrg.scala
gopal@ubuntu:~/SCALA$
gopal@ubuntu:~/SCALA$ scala setPrg
Head of Technologies : Spark
Tail of Technologies : Set(Scala, Phython)
 Is Technology empty? : false
 Is Ranks empty? : true
gopal@ubuntu:~/SCALA$
```

Concatenating Sets: can use either ++ operator or Set.++() method to concatenate two or more sets.

```
gopal@ubuntu:~/SCALA$ cat setConcatePrg.scala
object setConcatePrg
 {
   def main(args: Array[String])
   {
     val car1 = Set("Benz","Audi","BMW");
     val car2 = Set("i10","i20");
     val car3 = Set("Swift","Santro","Creta");

    // var carset = car1 ++ car2 ++ car3;
     var carset = car1.++(car2).++(car3)
     println("Selected Cars  " + carset);
   }
}
gopal@ubuntu:~/SCALA$
```

```
gopal@ubuntu:~/SCALA$ scalac setConcatePrg.scala
gopal@ubuntu:~/SCALA$
gopal@ubuntu:~/SCALA$ scala setConcatePrg
Selected Cars : Set(Audi, i20, Santro, BMW, Benz, Creta, i10, Swift)
gopal@ubuntu:~/SCALA$
```

10.6.2.Set Operations

```
gopal@ubuntu:~/SCALA$ cat setOperations.scala
object setOperations
 {
   def main(args: Array[String])
   {
      val ranks = Set(5,6,9,20,30,45);
      val ranksnew = Set(1,3,20,2,45,121);

      println("MAX Rank in Ranks : " + ranks.max);
      println("MIN Rank is Ranks : " + ranks.min);

      println( "ranks.intersect(ranksnew) : " + ranks.intersect(ranksnew) );
   }
}
gopal@ubuntu:~/SCALA$
```

```
gopal@ubuntu:~/SCALA$ nano setOperations.scala
gopal@ubuntu:~/SCALA$ scalac setOperations.scala
gopal@ubuntu:~/SCALA$
gopal@ubuntu:~/SCALA$ scala setOperations
MAX Rank in Ranks : 45
MIN Rank is Ranks : 5
ranks.intersect(ranksnew) : Set(20, 45)
gopal@ubuntu:~/SCALA$
```

10.6.3.Scala Maps

1. Scala map is a collection of key/value pairs.
2. Any value can be retrieved based on its key
3. eys are unique in the Map, but values need not be unique.
4. There are two kinds of Maps, the immutable and the mutable

```
gopal@ubuntu:~/SCALA$ cat mapPrg.scala
object mapPrg
{
   def main(args: Array[String])
   {
      val natureOfTech = Map("Hadoop" -> "BIGDATA",
                             "Spark" -> "IN-MEMORY",
                             "IoT" -> "AI" );
      println( "Keys in natureOfTech : " + natureOfTech.keys )
      println( "Values in natureOfTech : " + natureOfTech.values )
  }
}
gopal@ubuntu:~/SCALA$
```

```
gopal@ubuntu:~/SCALA$ scalac mapPrg.scala
gopal@ubuntu:~/SCALA$
gopal@ubuntu:~/SCALA$ scala mapPrg
Keys in natureOfTech : Set(Hadoop, Spark, IoT)
Values in natureOfTech : MapLike(BIGDATA, IN-MEMORY, AI)
gopal@ubuntu:~/SCALA$
```

```
scala> val nums = Map("one"->1,"two"->2,"three"->3);
nums: scala.collection.immutable.Map[String,Int] = Map(one -> 1, two -> 2, three
 -> 3)

scala> nums.get("two");
res0: Option[Int] = Some(2)

scala> nums.get("three");
res1: Option[Int] = Some(3)

scala>
```

11.7.Scala Tuples

Scala tuple combines a **fixed number of items together** so that they can be passed around as a whole. **Unlike an array or list, a tuple can hold objects with different types but they are also immutable**

```
scala> val b = new Tuple1(1,"Gopal",'g',123.44);
b: ((Int, String, Char, Double),) = ((1,Gopal,g,123.44),)

scala> val b = (1,"Gopal",'g',123.44);
b: (Int, String, Char, Double) = (1,Gopal,g,123.44)

scala>
```

```
scala> val x = (10,20,30,40);
x: (Int, Int, Int, Int) = (10,20,30,40)

scala> val sum = x._1+x._2+x._3+x._4;
sum: Int = 100

scala>
```

Tuple.productiterator() method to iterate over all the elements of a Tuple

```
scala> val x = (10,20,30,40);
x: (Int, Int, Int, Int) = (10,20,30,40)

scala> x.productIterator.foreach{ i =>println("Value = " + i )}
Value = 10
Value = 20
Value = 30
Value = 40

scala>
```

```
scala> val y = ("Gopal","Krishna");
y: (String, String) = (Gopal,Krishna)

scala> println(" Swapped Tuple is --> " + y.swap);
 Swapped Tuple is --> (Krishna,Gopal)

scala>
```

```
scala> var tuple1 = (10,20,30,40)
tuple1: (Int, Int, Int, Int) = (10,20,30,40)

scala> var tuple2 = ("gopal","krishna","niyathi")
tuple2: (String, String, String) = (gopal,krishna,niyathi)

scala> var tuple3 = (100,"GK",'V',true,123.44)
tuple3: (Int, String, Char, Boolean, Double) = (100,GK,V,true,123.44)

scala> tuple1._2
res12: Int = 20

scala> tuple2._1
res13: String = gopal

scala> tuple3._4
res14: Boolean = true
```

11.8.Flatten

flatten collapses one level of nested structure

```
scala> List(List(1, 2), List(3, 4)).flatten
res7: List[Int] = List(1, 2, 3, 4)

scala>
```

flatMap: flatMap is a frequently used combinator that combines mapping and flattening. flatMap takes a function that works on the **nested lists** and then concatenates the results back together.

```
scala>  val nestedNumbers = List(List(1, 2), List(3, 4))
nestedNumbers: List[List[Int]] = List(List(1, 2), List(3, 4))

scala> nestedNumbers.flatMap(x => x.map(_ * 2))
res11: List[Int] = List(2, 4, 6, 8)

scala> nestedNumbers.flatMap(x => x.map(_ + 10))
res12: List[Int] = List(11, 12, 13, 14)

scala>
```

11.9.Pattern Matching in Scala

A pattern match includes a sequence of alternatives, each starting with the keyword case. Each alternative includes a pattern and one or more expressions, which will be evaluated if the pattern matches.

=> Symbol separates the pattern from expressions

Example:

```
gopal@ubuntu:~/SCALA$ cat casePrg.scala
object casePrg
{
  def main(args: Array[String])
   {
      var days = List("Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun")
      for (day <- days) {
           println(day)
      day match {
        case "Mon" => println("First Day of the Week")
        case "Tue" => println("Second Day of the Week")
        case "Wed" => println("Third Day of the Week")
        case "Thu" => println("Fourth Day of the Week")
        case "Fri" => println("Fifth Day of the Week")
        case "Sat" => println("Sixth Day of the Week")
        case "Sun" => println("Seventh Day of the Week")
        case _     => println("Not a Day..")
    }

   }
}
}
```

```
gopal@ubuntu:~/SCALA$ scalac casePrg.scala
gopal@ubuntu:~/SCALA$
gopal@ubuntu:~/SCALA$ scala casePrg
Mon
First Day of the Week
Tue
Second Day of the Week
Wed
Third Day of the Week
Thu
Fourth Day of the Week
Fri
Fifth Day of the Week
Sat
Sixth Day of the Week
Sun
Seventh Day of the Week
gopal@ubuntu:~/SCALA$
```

Example 2:

```
gopal@ubuntu:~/SCALA$ cat numMatch_casePrg.scala
object numMatch_casePrg
 {
   def main(args: Array[String])
   {
     println(matchNum(3))
   }

   def matchNum(x: Int): String = x match
   {
      case 1 => "one"
      case 2 => "two"
      case _ => "Anything..Not a Number"  // if matching not found..
   }
}
gopal@ubuntu:~/SCALA$
```

```
gopal@ubuntu:~/SCALA$ scalac numMatch_casePrg.scala
gopal@ubuntu:~/SCALA$
gopal@ubuntu:~/SCALA$ scala numMatch_casePrg
Anything..Not a Number
gopal@ubuntu:~/SCALA$
```

Example 3:

```
gopal@ubuntu:~/SCALA$ cat techMatch_casePrg.scala
object techMatch_casePrg
{
  def main(args: Array[String])
   {
      var technologies = List("Linux","Windows","MacOs", "Eclipse", "NetBeans", "R",
Scala", "Java", "Phython")
      for (tech <- technologies) {
          println(tech)
      tech match {
      case."Windows" |"Linux"|"MacOs" => println("Operating System")
       case "Eclipse"|"NetBeans" => println("IDEs for Development")
       case "R"|"Scala"|"Java"|"Phython" => println("Programming Languages")
       case _  => println("Not a Techi Group Member")
    }

   }
}
```

```
gopal@ubuntu:~/SCALA$ scalac techMatch_casePrg.scala
gopal@ubuntu:~/SCALA$
gopal@ubuntu:~/SCALA$ scala techMatch_casePrg
Linux . .
Operating System
Windows . .
Operating System
MacOs
Operating System
Eclipse
IDEs for Development
NetBeans
IDEs for Development
R
Programming Languages
Scala
Programming Languages
Java
Programming Languages
```

Example 4:

```
gopal@ubuntu:~/SCALA$ cat letterMatch_casePrg.scala
object letterMatch_casePrg
{
def main(args: Array[String])
  {
    for (x <- 1 to 15) {
        println(x)
    x match {
      case x if x % 2 == 0 => println("Number is Even")
      case x if x % 2 == 1 => println("Number is Odd")
      case _               => println("Not An EvenOdd Number..")
    }
  }
}}
gopal@ubuntu:~/SCALA$
```

```
gopal@ubuntu:~/SCALA$ scala letterMatch_casePrg
1
Number is Odd
2
Number is Even
3
Number is Odd
4
Number is Even
5
Number is Odd
6
Number is Even
7
Number is Odd
8
Number is Even
9
Number is Odd
```

11.10.Some Other

scala> var myString = "GopalKrishna"

myString: String = GopalKrishna

scala>println(myString) GopalKrishna

WHEN U PRESS "objectName." And press TABSPACE - all the available methods will be displayed.

scala> myString.

+ contains isInstanceOf split asInstanceOf

contentEquals lastIndexOf startsWith

charAt endsWith length subSequence

codePointAt equalsIgnoreCase matches substring

codePointBefore getBytes offsetByCodePoints toCharArray

```
codePointCount      getChars        regionMatches      toLowerCase

compareTo           indexOf         replace            toString

compareToIgnoreCase intern          replaceAll         toUpperCase

concat              isEmpty         replaceFirst       trim

++++++++++++++++++++++++++++++++++++++++++++++

scala> var x  =10;

x: Int = 10

scala> x.

%   +  >    >>>          isInstanceOf  toDouble  toLong       unary_+  | &

-  >=  ^     toByte     toFloat       toShort    unary_-

*   /   >>  asInstanceOf  toChar          toInt      toString  unary_~

++++++++++++++++++++++++++++++++++++++++++++++++++++++

scala> var myList = List("aaa","bbb","ccc") myList:

List[String] = List(aaa, bbb, ccc)
```

11.21.CURRYING FUNCTION IN SCALA

Currying transforms a function that takes multiple parameters into a chain of functions, each taking a single parameter. Curried functions are defined with multiple parameter lists, as follows:

```
def strcat(s1: String)(s2: String) = s1 + s2
```

Alternatively, you can also use the following syntax to define a curried function:

```
def strcat(s1: String) = (s2: String) => s1 + s2
```

Following is the syntax to call a curried function:

```
scala> def gopalfunction(s1: String)(s2: String) = s1 + s2

gopalfunction: (s1: String)(s2: String)String

scala> gopalfunction("GOPAL")("KRISHNA")

res7: String = GOPALKRISHNA

scala> gopalfunction("GOPAL")("     KRISHNA")
```

```
res8: String = GOPAL  KRISHNA

cala>    gopalfunction("GOPAL")("\tKRISHNA")    res9:

String = GOPALKRISHNA

scala>  gopalfunction("GOPAL")("\nKRISHNA")  res10:

String =

GOPAL

 KRISHNA

scala> def abcFun(x:Int)(y:Int) = x * y abcFun:

(x: Int)(y: Int)Int

scala>  abcFun(4)(5)  res11:

Int = 20

scala> abcFun(4)(5)(6)

<console>:10:  error:  Int  does  not  take  parameters

abcFun(4)(5)(6)

scala> def abcFun(x:Int)(y:Int)(z:Int) = x * y + z abcFun: (x:

Int)(y: Int)(z: Int)Int

scala>  abcFun(2)(3)(4)  res13:

Int = 10

scala>  def  TestFun(x:Int)(y:Int)(z:String)  =  x  *  y  +  z

TestFun: (x: Int)(y: Int)(z: String)String

scala>       TestFun(2)(3)("AAA")

res14: String = 6AAA
```

11.12.Creating Mutable Collections from Immutable Ones

```
scala> val m = Map("SPARK" -> 597, "SCALA" -> 40)
```

```
m: scala.collection.immutable.Map[String,Int] = Map(SPARK -> 597, SCALA -> 40)
scala> m = Map("Java" -> 200)
<console>:8: error: reassignment to val m =

Map("Java" -> 200)

// To convert the above Immutable Map to Mutable Map

scala> val n = m.toBuffer

n: scala.collection.mutable.Buffer[(String, Int)] = ArrayBuffer((SPARK,597), (SCALA,40)) scala>

n += ("java" -> 200)

res0: n.type = ArrayBuffer((SPARK,597), (SCALA,40), (java,200)) To
```

Trim of certain Elements:

```
scala> n trimStart 1
scala> println(n)

ArrayBuffer((SCALA,40), (java,200))

//Below buffer of tuples is now an immutable map again

scala> val x = n.toMap

x: scala.collection.immutable.Map[String,Int] = Map(SCALA -> 40, java -> 200)
scala> println(n) ArrayBuffer((Java,201),

(Java,200))

scala> n += ("Java" -> 200)

res7: n.type = ArrayBuffer((Java,201), (Java,200), (Java,200))

scala> n += ("Java" -> 200)
res7: n.type = ArrayBuffer((Java,201), (Java,200), (Java,200))
scala> n += ("SCALA" -> 40)
res8: n.type = ArrayBuffer((Java,201), (Java,200), (Java,200), (SCALA,40))
```

11.13.Examples on Lists & Sets

```
scala> val l = n.toList
l: List[(String, Int)] = List((Java,201), (Java,200), (Java,200), (SCALA,40)) scala>
```

```
val s = n.toSet

s: scala.collection.immutable.Set[(String, Int)] = Set((Java,201), (Java,200), (SCALA,40))
```

NOTE: The list "l" and set "s" were created successfully, with the list containing the duplicated entries and the set restricted to contain only unique entries.

11.14.Scala Classes & Objects

```
gopal@ubuntu:~/SCALA$
cat classExample.scala class sample(val x:Int , val y:Int)
{
var a=x;
var b=y;
  def sum(xx:Int,yy:Int)
  {
            a=a+xx;
            b=b+yy;
    println("A value :" +a);
    println("B value :" +b);
}
}
object classExample
{
  def main(args: Array[String])
    {
        val s = new sample(10, 20);

        s.sum(10, 10);
    }
}
```

```
gopal@ubuntu:~/SCALA$ scalac classExample.scala
gopal@ubuntu:~/SCALA$
gopal@ubuntu:~/SCALA$ scala classExample
 A Value:20
B Value:30
gopal@ubuntu:~/SCALA$
```

11.15.SCALA Traits

A Scala trait encapsulates method and field definitions, which can then be reused by mixing them into classes. Unlike class inheritance, in which each class must inherit from just one superclass, a class can mix in any number of traits.

11.16.When to Choose "Object Oriented" Vs "Functional Programming" languages ?

- **Object-oriented** languages are good when you have a **fixed set of *operations* on *things***, and as your code evolves, you primarily add new things. This can be accomplished by adding new classes which implement existing methods, and the existing classes are left alone.
- **Functional languages** are good when you have a **fixed set of *things***, and as your code evolves, you primarily add new *operations* on existing things. This can be accomplished by adding new functions which compute with existing data types, and the existing functions are left alone

SPARK

12.1.Spark

Spark is an in-memory cluster computing framework for processing and analyzing large amounts of data.

It provides a simple programming interface, which enables an application developer to easily use the CPU, memory, and storage resources across a cluster of servers for processing large datasets

12.2.Key Features of Spark

1. Easy to Use
2. Fast
3. General Purpose
4. Scalable
5. Fault Tolerant

12.2.1.Easy to Use

Spark provides a simpler programming model than that provided by Map Reduce. Developing a distributed data processing application with Spark is a lot easier than developing the same application with Map Reduce.

Hadoop Map Reduce provides only two operations for processing the data like **"Map" & "Reduce"** where as Spark comes with **80 plus data processing operations** to work with big data applications.

Hadoop MapReduce requires every problem to be broken down into a sequence of map and reduce jobs and so its very difficult to implement some complex algorithms only with Map Reduce where as For Spark its much easier to implement the to do the complex data processing with its **in-memory processing** Spark development code is very concise compared to Hadoop MapReduce. What Map Reduce requires 50 lines of code for a specific application development, Spark only requires less than 10 lines of code.5 to 10 times of more productivity with Spark compared to MapReduce.

12.2.2.Fast

If data fits in the memory, Spark is 100 times faster than Map Reduce..Even if the data is fits in Memory also, Spark is 10 times faster than Map reduce.

Spark is faster than Hadoop Map Reduce for two reasons.

1. **It allows in-memory cluster computing**
2. **It implements an advanced execution engine.**

The sequential read throughput when reading data from memory compared to reading data from a hard disk is 100 times greater i.e. When the data volume is less, it may not be that much significant where as when we are reading/writing Terabytes of data

..it makes lot of difference in Network I/O.

Spark does not automatically cache input data in memory. A common misconception is that Spark cannot be used if input data does not fit in memory. It is not true. Spark can process terabytes of data on a cluster that may have only 100 GB total cluster memory.

It is up to an application to decide what data should be cached and at what point in a data processing pipeline that data should be cached.

12.2.3.General Purpose

Spark provides a unified integrated platform for different types of data processing jobs. It can be used for:

1. Batch processing
2. Interactive Processing
3. Stream Processing
4. Machine Learning
5. Graph Computing

With Map Reduce we can only perform the batch processing, and for driving Stream processing, Graph processing developer has to use different frameworks. Using different frameworks for different types of data processing jobs creates many challenges.

1. A developer has to learn multiple frameworks, each of which has a different interface. This reduces developer productivity
2. Each framework may operate in a silo. Therefore, data may have to be copied to multiple places.

Spark comes pre-packaged with an integrated set of libraries for batch processing, interactive analysis, stream processing, machine learning, and graph computing.

With Spark, you can use a single framework to build a data processing pipeline that involves different types of data processing tasks. There is no need to learn multiple frameworks or deploy separate clusters for different types of data processing jobs

12.2.4.Scalable

Spark is scalable. The data processing capacity of a Spark cluster can be increased by just adding more nodes to a cluster.You can start with a small cluster, and as your dataset grows, you can add more computing capacity. Thus, Spark allows you to scale economically. In addition, Spark makes this feature automatically available to an application. No code change is required when you add a node to a Spark cluster

12.2.5.Fault Tolerant

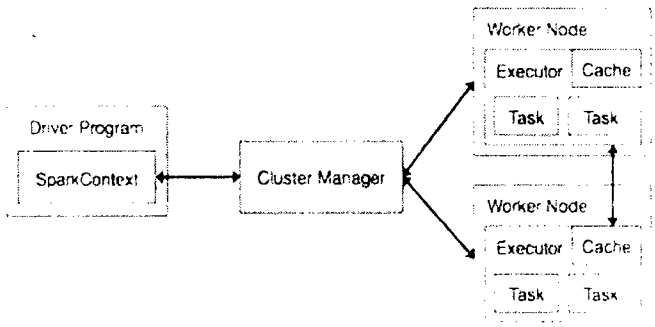
Spark is fault tolerant. In a cluster of a few hundred nodes, the probability of a node failing on any given day is high. The hard disk may crash or some other hardware problem may make a node unusable.Spark automatically handles the failure of a node

in a cluster. Failure of a node may degrade performance, but will not crash an application. Since Spark automatically handles node failures, an application developer does not have to handle such failures in his application. It simplifies application

12.3.Basic Difference between Map Reduce and Spark

	Map Reduce	Spark
Job/Task	Mapper /Reducer Task	DAG Task
Latency	High latency (Read /Write the temporary files to disk on local system)	Low latency(It will read/write the result in memory)
High throughput		Spark gives High throughput compares to Map reduce.
Iterative Algorithms	Difficult to implement	Easy to implement
Input /Output Types	Key Value pair	RDD's
Data Processing	Java code	Ready available Transformation and actions.

12.4.Spark Architecture



12.4.1.Spark Context

- Is the coordinator object which will run different spark applications.
- SparkContext is a class defined in the Spark library. It is the main entry point into the Spark library
- Spark Context will run in a program called **'Driver Program'** – the main program in spark.

- A Spark application must create an instance of the SparkContext class.
- Currently, an application can have only one active instance of SparkContext. An instance of the SparkContext class can be created as below:

val sc = new SparkContext()

NOTE: In this case, SparkContext gets configuration settings such as the address of the Spark master, application name, and other settings from system properties

If we want to mention the same programmatically:

val config = new SparkConf().setMaster("spark://host:port")

.setAppName("big app")

val sc = new SparkContext(config)

12.4.2.Cluster Manager

- Cluster Manager will allocate resources across applications of cluster
- to run on a cluster, the **Spark Context** can connect to several types

of ***cluster managers*** (either Spark's own standalone cluster manager or YARN).

2.4.3.Executors

Once resources connected, Spark acquires ***executors*** on nodes in the cluster, which are processes that run computations and store data for your application Next, cluster manager sends your **application code** (defined by JAR or Python files passed to SparkContext) to the executors. Finally, SparkContext sends ***tasks*** for the executors to run.

12.5.Different Cluster Managers In - Spark Architecture

As part of Spark Runtime architecture , there are 3 cluster managers

Standalone Cluster Manager (default)
Hadoop YARN
Apache Mesos

12.5.1.Apache Mesos

Apache Mesos is a general-purpose cluster manager that can run both analytics workloads and long-running services (e.g., web applications or key/value stores) on a cluster.

To use Spark on Mesos, pass a ***mesos://*** **URI to spark-submit:**
spark-submit --master mesos://masternode:5050 yourapp

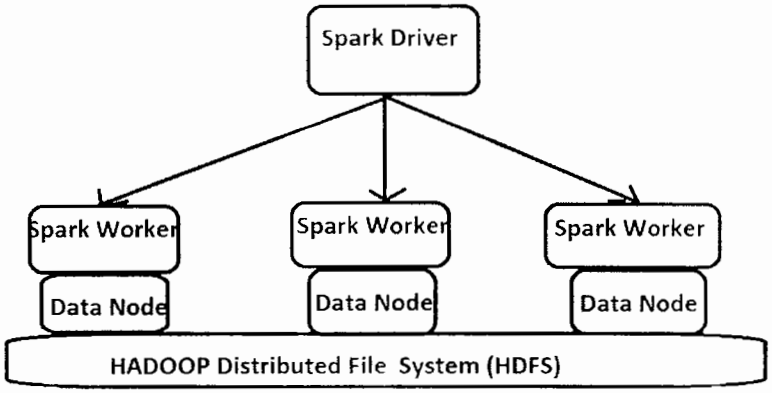
12.5.2.Hadoop YARN

YARN is a cluster manager introduced in Hadoop 2.0 that allows diverse data processing frameworks to run on a shared resource pool, and is typically installed on the same nodes as the Hadoop filesystem (HDFS).

Running Spark on YARN in these environments is useful because it lets Spark access HDFS data quickly, on the same nodes where the data is stored. To use Spark on Hadoop YARN use the below command

spark-submit --master yarn yourapp

12.6.Spark Architecture



12.7.Resilient Distributed Datasets (RDD)

RDD represents a collection of partitioned data elements that can be operated on in parallel. It is the primary data abstraction mechanism in Spark.

It is defined as an abstract class in the Spark library. Conceptually, RDD is similar to a Scala collection, except that it represents a distributed dataset and it supports lazy operations.

12.8.Key characteristics of an RDD

12.8.1.Immutable

An RDD is an immutable data structure.

Once created, it cannot be modified in-place. Basically, an operation that modifies an RDD returns a new RDD.

12.8.2.Partitioned

Data represented by an RDD is split into partitions.

These partitions are generally distributed across a cluster of nodes.

when Spark is running on a single machine, all the partitions are on that machine

12.9.Rdd Operations

Spark applications process data using the methods defined in the RDD class or classes derived from it. These methods are also referred to as **operations**.

RDD operations can be categorized into two types:

Transformations

Actions

12.9.1.Transformations

A transformation method of an RDD creates a new RDD by performing a computation on the source RDD.

RDD transformations are conceptually similar to Scala collection methods.

The key difference is that the Scala collection methods operate on data that can fit in the memory of a single machine, whereas RDD methods can operate on data distributed across a cluster of node RDD transformations are lazy, whereas Scala collection methods are strict.

1)Map

The map method is a higher-order method that takes a function as input and applies it to each element in the **source RDD** to create a **new RDD**

gopal@ubuntu:~$ hadoop fs -cat /SparkInputDir/Input.log

hadoop is one of the bigdata tool

bigdata is not only for hadoop

hadoop is ment for bigdata storage and processing hadoop is good for analytics also

thanks hadoop

gopal@ubuntu:~$

Example Code:

```
scala> val file = sc.textFile("hdfs://localhost:54310/SparkInputDir/Input.log")
scala> val fileLength = file.map(l => l.length)
scala> fileLength.collect
```

OUTPUT:

res0: Array[Int] = Array(33, 30, 49, 33, 13)

2)filter

The filter method is a higher-order method that takes a Boolean function as input and applies it to each element in the source RDD to create a new RDD.A Boolean function takes an input and returns true or false.

The filter method returns a new RDD formed by selecting only those elements for which the input Boolean function returned true.Thus, the new RDD contains a subset of the elements in the original RDD.

Example Code:

```
scala> val file = sc.textFile("hdfs://localhost:54310/SparkInputDir/Input.log")
scala> val fileLength = file.filter(l => l.length >= 33)
scala> fileLength.collect
```

OUTPUT:

Array[String] = Array(hadoop is one of the bigdata tool, hadoop is ment for bigdata storage and processing, hadoop is good for analytics also)

```
scala> val fileLength = file.filter(l => l.length == 30)
```

OUTPUT: Array[String] = Array(bigdata is not only for hadoop)

```
scala> val fileLength = file.filter(l => l.length != 33)
```

OUTPUT:

Array[String] = Array(bigdata is not only for hadoop, hadoop is ment for bigdata storage and processing, thanks hadoop)

```
scala> val fileLength = file.filter(l => l.contains("analytics"))
```

OUTPUT:

Array[String] = Array(hadoop is good for analytics also)

3)flatMap

The flatMap method is a higher-order method that takes an input function which returns a sequence for each input element passed to it.The flatMap method returns a new RDD formed by flattening this collection of sequence

Example Code

```
scala> val file = sc.textFile("hdfs://localhost:54310/SparkInputDir/Input.log")
scala> val fileWords = file.flatMap(l => l.split(" "))
scala> fileWords.collect
```

OUTPUT

res7: Array[String] = Array(hadoop, is, one, of, the, bigdata, tool, bigdata, is, not, only, for, hadoop, hadoop, is, ment, for, bigdata, storage, and, processing, hadoop, is, good, for, analytics, also, thanks, hadoop)

```
scala> println(fileWords.collect().mkString("\t"))
```

OUTPUT:

16/09/09 02:36:17 INFO DAGScheduler: Job 5 finished: collect at <console>:26, took 0.273978 s hadoop is one of the bigdata tool bigdata is not only for hadoop hadoop is ment for bigdata storage and processing hadoop is good for analytics also thanks hadoop

```
scala> println(fileWords.collect().mkString("\n"))
```

hadoop is one of the bigdata tool bigdata is not only for

hadoop hadoop is ment for bigdata storage and processing hadoop

is good

4)mapPartitions

The higher-order mapPartitions method allows you to process data at a partition level. Instead of passing one element at a time to its input function, **mapPartitions** passes a partition in the form of an iterator .The input function to the mapPartitions method takes an **iterator as input** and returns another **iterator as output.** The mapPartitions method returns new RDD formed by applying a user-specified function to each partition of the source RDD

Example Code:

```
scala> val file = sc.textFile("hdfs://localhost:54310/SparkInputDir/Input.log")
scala> val fileMapPar = file.mapPartitions(i => i.map( l => l.length))
fileMapPar: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[19] at mapPartitions at
<console>:23
scala> fileMapPar.collect
```

OUTPUT

```
res15: Array[Int] = Array(33, 30, 49, 33, 13)
```

5)Union

The union method takes an RDD as input and returns a new RDD that contains the union of the elements in the source RDD and the RDD passed to it as an input

INPUT DATA:

```
gopal@ubuntu:~$ pwd
/home/gopal
 gopal@ubuntu:~$ cat File1.txt
THIS IS LINE1
THIS IS LINE2
 gopal@ubuntu:~$ cat File2.txt
THIS IS LINE3
THIS IS LINE4
 gopal@ubuntu:~$
scala> val file1 = sc.textFile("/home/gopal/File1.txt")
 scala> val file2 = sc.textFile("/home/gopal/File2.txt")
scala> val unionFile = file1.union(file2)
unionFile: org.apache.spark.rdd.RDD[String] = UnionRDD[24] at union at <console>:25
```

```
scala> unionFile.collect
```

OUTPUT:

```
Array[String] = Array(THIS IS LINE1, THIS IS LINE2, THIS IS LINE3, THIS IS LINE4)
```

Example Code 2:

```
scala> val tech = sc.parallelize(List("Spark","Scala","Java"))
scala> val technew = sc.parallelize(List("Perl","Scala","Phython"))
scala> val unionTech = tech.union(technew)
scala> unionTech.collect
```

OUTPUT:

```
Array[String] = Array(Spark, Scala, Java, Perl, Scala, Phython)
```

6)Intersection

The intersection method takes an RDD as input and returns a new RDD that contains the intersection of the elements in the source RDD and the RDD passed to it as an input.

INPUT DATA:

```
gopal@ubuntu:~$ cat File1.txt
THIS IS LINE1
THIS IS LINE2
gopal@ubuntu:~$ cat File2.txt
THIS IS LINE2
THIS IS LINE4
gopal@ubuntu:~$
```

Example Code:

```
scala> val file1 = sc.textFile("/home/gopal/File1.txt")
scala> val file2 = sc.textFile("/home/gopal/File2.txt")
scala> val intersectFile = file1.intersection(file2)
scala> intersectFile.collect
```

OUTPUT:

```
Array[String] = Array(THIS IS LINE2)
```

Example Code 2:

```
scala> val tech = sc.parallelize(List("Spark","Scala","Java"))
scala> val technew = sc.parallelize(List("Perl","Scala","Phython"))
scala> val interTech = tech.intersection(technew)
scala> interTech.collect
```

OUTPUT:

```
Array[String] = Array(Scala)
```

7)Subtract

The subtract method takes an RDD as input and returns a new RDD that contains elements in the source RDD but not in the input RDD.

```
scala> val file1 = sc.textFile("/home/gopal/File1.txt")
scala> val file2 = sc.textFile("/home/gopal/File2.txt")
scala> val subtractFile = file1.subtract(file2)
scala> subtractFile.collect
```

OUTPUT:

```
Array[String] = Array(THIS IS LINE1)
```

Example Code 2:

```
scala> val tech = sc.parallelize(List("Spark","Scala","Java"))
scala> val technew = sc.parallelize(List("Perl","Scala","Phython"))
scala> val interTech = tech.subtract(technew)
scala> interTech.collect
```

OUTPUT:

```
Array[String] = Array(Java, Spark)
```

8)Parallelize

Parallelized collections are created by calling SparkContext's parallelize method on an existing collection in your driver program (a Scala Seq). The elements of the collection are copied to form a distributed dataset that can be operated on in parallel. Below is an example, to create a parallelized collection holding the numbers 1 to 5:

Spark can create distributed datasets from any storage source supported by Hadoop, including your local file system, HDFS, Cassandra, HBase,Amazon S3, etc. Spark supports text files, SequenceFiles, and any other Hadoop InputFormat

```
scala> val inputdata = Array(1, 2, 3, 4, 5)
inputdata: Array[Int] = Array(1, 2, 3, 4, 5)
scala> val distData = sc.parallelize(inputdata)
distData: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[49] at parallelize at
<console>:23
scala> distData.collect
```

OUTPUT:

```
Array[Int] = Array(1, 2, 3, 4, 5)
```

9)Distinct

The distinct method of an RDD returns a new RDD containing the distinct elements in the source RDD

Example Code:

```
scala> val tech = sc.parallelize(List("Spark","Spark","Spark","Scala","Java","Scala"))
scala> val distTech = tech.distinct s
cala> distTech.collect
```

OUTPUT:

Array[String] = Array(Java, Scala, Spark)

10)Cartesian

The cartesian method of an RDD takes an RDD as input and returns an RDD containing the Cartesian product of all the elements in both RDDs. It returns an RDD of ordered pairs, in which the first element comes from the source .RDD and the second element is from the input RDD. The number of elements in the returned RDD is equal to the product of the source and input RDD lengths.This method is similar to a cross join operation in SQL.

Example Code:

```
scala> val ratings = sc.parallelize(List(1,2,3))
scala> val tech = sc.parallelize(List("Spark","Scala","Java"))
scala> val cartRatTech = ratings.cartesian(tech)
scala> cartRatTech.collect
```

OUTPUT:

res0: Array[(Int, String)] = Array((1,Spark), (1,Scala), (1,Java), (2,Spark), (3,Spark), (2,Scala), (2,Java), (3,Scala), (3,Java))

```
scala> val cartRatTech = ratings.cartesian(ratings)
scala> cartRatTech.collect
```

OUTPUT:

res1: Array[(Int, Int)] = Array((1,1), (1,2), (1,3), (2,1), (3,1), (2,2), (2,3), (3,2), (3,3))

```
scala> val cartRatTech = tech.cartesian(tech)
scala> cartRatTech.collect
```

OUTPUT:

res2: Array[(String, String)] = Array((Spark,Spark), Spark,Scala), (Spark,Java), (Scala,Spark), (Java,Spark), (Scala,Scala), (Scala,Java), (Java,Scala), (Java,Java))

11)ZIP

The zip method takes an RDD as input and returns an RDD of pairs, where the first element in a pair is from the source RDD and second element is from the input RDD. Unlike the cartesian method, the RDD returned by zip has the same number of elements as the source RDD.Both the source RDD and the input RDD must have the same length. In

addition, both RDDs are assumed to have same number of partitions and same number of elements in each partition.

Example Code:

```
scala> val ratings = sc.parallelize(List(1,2,3))
scala> val tech = sc.parallelize(List("Spark","Scala","Java"))
scala> val zipRatTech = ratings.zip(tech)
scala> zipRatTech.collect
```

OUTPUT:

```
Array[(Int, String)] = Array((1,Spark), (2,Scala), (3,Java))
scala> val zipRatTech = ratings.zip(ratings)
scala> zipRatTech.collect
```

OUTPUT:

```
Array[(Int, Int)] = Array((1,1), (2,2), (3,3))
scala> val zipRatTech = tech.zip(tech)
scala> zipRatTech.collect
```

OUTPUT:

```
Array[(String, String)] = Array((Spark,Spark), (Scala,Scala), (Java,Java))
```

12)ZipWithIndex

The zipWithIndex method zips the elements of the source RDD with their indices and returns an RDD of pairs:

Example Code:

```
scala> val names = sc.parallelize(List("AAA","BBB","CCC","DDD"))
scala> val zipWithIndexNames = names.zipWithIndex
scala> zipWithIndexNames.collect
```

OUTPUT:Array[(String, Long)] = Array((AAA,0), (BBB,1), (CCC,2), (DDD,3))

13)GroupBy

The higher-order groupBy method groups the elements of an RDD according to a user specified criteria.It takes as input a function that generates a key for each element in the source RDD.It applies this function to all the elements in the source RDD and returns an RDD of pairs. In each returned pair, the first item is a key and the second item is a collection of the elements mapped to that key by the input function to the groupBy method.

Example Code:

```
scala> val names = sc.textFile("/home/gopal/names.csv")
scala> val namesRows = names.map( l => l.split(","))
scala> val yearName = namesRows.map( c => ( c(1),c(2) )
```

```
scala> yearName.groupByKey.collect
```

INPUTDATA:

```
gopal@ubuntu:~$ cat names.csv
2013,Raj,India,Male,25
2014,Ram,China,Male,34
2013,Raj,India,Male,25
2013,Ramya,USA,Female,34
2014,Rajesh,UK,Male,25
2014,Rajesh,AUS,Male,25
2014,Rajesh,UK,Fem le,34
gopal@ubuntu:~$
```

OUTPUT:

```
Array[(String, Iterable[String])] = Array((F am,CompactBuffer(China)),
(Ramya,CompactBuffer(USA)), (Raj,Con pactBuffer(India, India)),
(Rajesh,CompactBuffer(UK, AUS, UK)))
```

14)Sort By

The higher-order sortBy method return s an RDD with sorted elements from the source RDD. It takes two input parameters. Th first input is a function that generates a key for each element in the source RDD. The se cond argument allows you to specify ascending or descending order for sort.

```
scala> val numbers = sc.parallelize(List(2 1, 29, 41, 15, 50,22,42,31))
scala> val sorted = numbers.sortBy(x => x, true)
scala> sorted.collect
```

OUTPUT:

```
Array[Int] = Array(15, 21, 22, 29, 31, 41, 42, 50)
scala> val names = sc.parallelize(List("AAA","XXX","UUU","WWW","BBB","SSS","C"))
scala> val sorted = names.sortBy(x => x, true)
scala> sorted.collect
```

OUTPUT:

```
Array[String] = Array(AAA, BBB, C, SSS, JUU, WWW, XXX)
```

15)Coalesce

The coalesce method reduces the number of partitions in an RDD. It takes an integer input and returns a new RDD with the specified number of partitions.

Example Code:

```
scala> val numbers = sc.parallelize(1 to 100)
```

```
scala> val numbersWithTwoPartition = numbers.coalesce(2)
scala> numbersWithTwoPartition.collect
```

OUTPUT:

Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100)

16)Repartition

The repartition method takes an integer as input and returns an RDD with specified number of partitions. It is useful for increasing parallelism. It redistributes data, so it is an expensive operation. The coalesce and repartition methods look similar, but the first one is used for reducing the number of partitions in an RDD, while the second one is used to increase the number of partitions in an RDD

```
scala> val numbers = sc.parallelize(1 to 100).toList)
scala> val numbersWithOnePartition = numbers.repartition(2)
scala> numbersWithOnePartition.collect
```

OUTPUT:

Array[Int] = Array(1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63, 65, 67, 69, 71, 73, 75, 77, 79, 81, 83, 85, 87, 89, 91, 93, 95, 97, 99, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 98, 100)

17)Sample

The sample method returns a sampled subset of the source RDD. It takes three input parameters. The first parameter specifies the replacement strategy. The second parameter specifies the ratio of the sample size to source RDD size. The third parameter, which is optional, specifies a random seed for sampling.

```
scala> val numbers = sc.parallelize(1 to 9)
scala> val sampleNumbers = numbers.sample(true,.2)
scala>sampleNumbers.collect
```

OUTPUT:

res24: Array[Int] = Array(3, 9)

Transformations on RDD of key-value Pairs

18)Keys

The keys method returns an RDD of only the keys in the source RDD

```
scala> val stuNameMarks = sc.parallelize(List(("Raj", 70), ("Ramesh", 89), ("Rakesh", 79)))
scala> val stuNames = stuNameMarks.keys
scala> stuNames.collect
```

OUTPUT:

Array[String] = Array(Raj, Ramesh, Rakesh)

19)Values

The values method returns an RDD of only the values in the source RDD

```
scala> val stuNameMarks = sc.parallelize(List(("Raj", 70), ("Ramesh", 89), ("Rakesh", 79)))
scala> val stuMarks = stuNameMarks.values
scala> stuMarks.collect
```

OUTPUT:

Array[Int] = Array(70, 89, 79)

20)MapValues

The mapValues method is a higher-order method that takes a function as input and applies it to each value in the source RDD.It returns an RDD of key-value pairs.It is similar to the map method, except that it appliesthe input function only to each value in the source RDD, so the keys are not changed.The returned RDD has the same keys as the source RDD

Example Code:

```
scala> val stuNameMarks = sc.parallelize(List(("Raj", 70), ("Ramesh", 89), ("Rakesh", 79)))
scala> val stuMarksFiltered = stuNameMarks.mapValues { x => 2*x}
scala> stuMarksFiltered.collect
```

OUTPUT:

Array[(String, Int)] = Array((Raj,140), (Ramesh,178), (Rakesh,158))

21)Join

The join method takes an RDD of key-value pairs as input and performs an inner join on the source and input RDDs.It returns an RDD of pairs, where the first element in a pair is a key found in both source and input RDD and the second element is a tuple containing values mapped to that key in the source and input RDD.

Example Code:

```
scala> val data1 = sc.parallelize(List( ("Spark",10),("Scala",20),("Java",30) ))
scala> val data2 = sc.parallelize(List(
("Phython","PaltformDep"),("Scala","JVM") ("Spark","Cache") ))
scala> val joinData = data1.join(data2)
```

```
scala> joinData.collect
```

OUTPUT:

```
Array[(String, (Int, String))] = Array((Scala,(20,JVM)), (Spark,(10,Cache)))
scala> val joinData = data2.join(data1)
scala> joinData.collect
```

OUTPUT:

```
Array[(String, (String, Int))] = Array((Scala,(JVM,20)), (Spark,(Cache,10)))
scala> val joinData = data1.join(data1)
scala> joinData.collect
```

OUTPUT:

```
Array[(String, (Int, Int))] = Array((Java,(30,30)), (Scala,(20,20)), (Spark,(10,10)))
```

22)LeftOuterJoin

The leftOuterJoin method takes an RDD of key-value pairs as input and performs a left outer join on the source and input RDD.It returns an RDD of key-value pairs, where the first element in a pair is a key fromSource RDD and the second element is a tuple containing value from source RDD and optional value from the input RDD . An optional value from the input RDD is represented with Option type.

Example Code:

```
scala> val data1 = sc.parallelize(List( ("Spark",10),("Scala",20),("Java",30) ))
scala> val data2 = sc.parallelize(List(
("Phython","PaltformDep"),("Scala","JVM"),("Spark","Cache") ))
scala> val joinData = data1.leftOuterJoin(data2)
scala> joinData.collect
```

OUTPUT:

```
Array[(String, (Int, Option[String]))] = Array((Java,(30,None)), (Scala,(20,Some(JVM))),
(Spark,(10,Some(Cache))))
```

23)RightOuterJoin

The rightOuterJoin method takes an RDD of key-value pairs as input and performs a right outer join on the source and input RDD. It returns an RDD of key-value pairs, where the first element in a pair is a key from input RDD and the second element is a tuple containing optional value from source.RDD and value from input RDD.An optional value from the source RDD is represented with the Option type.

Example Code:

```
scala> val data1 = sc.parallelize(List( ("Spark",10),("Scala",20),("Java",30) ))
```

```
scala> val data2 = sc.parallelize(List(
("Phython","PaltformDep"),("Scala","JVM"),("Spark","Cache") ))
scala> val joinData = data1.rightOuterJoin(data2)
scala> joinData.collect
```

OUTPUT:

```
Array[(String, (Option[Int], String))] = Array((Phython,(None,PaltformDep)),
(Scala,(Some(20),JVM)), (Spark,(Some(10),Cache)))
```

24 FullOuterJoin

The full OuterJoin method takes an RDD of key-value pairs as input and performs a full outer join on the source and input RDD. It returns an RDD of key-value pairs.

Example Code:

```
scala> val data1 = sc.parallelize(List( ("Spark",10),("Scala",20),("Java",30) ))
scala> val data2 = sc.parallelize(List(
("Phython","PaltformDep"),("Scala","JVM"),("Spark","Cache") ))
scala> val joinData = data1.fullOuterJoin(data2)
scala> joinData.collect
```

OUTPUT:

```
Array[(String, (Option[Int], Option[String]))] = Array((Java,(Some(30),None)),
(Phython,(None,Some(PaltformDep))), (Scala,(Some(20),Some(JVM))),
(Spark,(Some(10),Some(Cache))))
```

25 GroupByKey

The groupByKey method returns an RDD of pairs, where the first element in a pair is a key from the source RDD and the second element is a collection of all the values that have the same key. It is similar to the groupBy method. The difference is that groupBy is a higher-order,method that takes a input a function that returns a key for each element in the source RDD.The groupByKey method operates on an RDD of key-value pairs, so a key generator function is not required as input

Example Code:

```
scala> val data2 = sc.parallelize(List( (10,1),(20,2),(10,3),(10,4),(20,8),(40,9),(20,5),(10,6) ))
scala> val groupByKeyData = data2.groupByKey()
scala> groupByKeyData.collect
```

OUTPUT:

```
Array[(Int, Iterable[Int])] = Array((40,CompactBuffer(9)), (20,CompactBuffer(2,
8, 5)), (10,CompactBuffer(1, 3, 4, 6)))
```

Example Code

```
scala> val iniData = sc.parallelize(List( ("Spark",1),("Scala",2),("Spark",3),("Spark",4),("Spark",8),("Java",9),("Scala",5),("Spark",6) ))
scala> val groupByKeyData = iniData.groupByKey()
scala> groupByKeyData.collect
```

OUTPUT:

```
Array[(String, Iterable[Int])] = Array((Java,CompactBuffer(9)), (Scala,CompactBuffer(2, 5)), (Spark,CompactBuffer(1, 3, 4, 8, 6)))
```

26)ReduceByKey

The higher-order reduceByKey method takes an associative binary operator as input and reduces values with the same key to a single value using the specified binary operator. A binary operator takes two values as input and returns a single value as output.An associative operator returns the same result regardless of the grouping of theoperands. The reduceByKey method can be used for aggregating values by key. For example, it can be used forCalculating sum, Calculating product, Calculating minimum or maximum of all the values mapped to the same key

Example Code :

```
scala> val iniData = sc.parallelize(List( ("Spark",1),("Scala",2),("Spark",3),("Spark",4),("Spark",8),("Java",9),("Scala",5),("Spark",6) ))
scala> val sumOfIniData = iniData.reduceByKey( (x,y) => (x+y) )
scala> sumOfIniData.collect
```

OUTPUT:

```
Array[(String, Int)] = Array((Java,9), (Scala,7), (Spark,22))
scala> val minByKeyData = iniData.reduceByKey((x,y) => if (x < y) x else y)
scala> minByKeyData.collect
```

OUTPUT:

```
Array[(String, Int)] = Array((Java,9), (Scala,2), (Spark,1))
scala> val maxByKeyData = iniData.reduceByKey((x,y) => if (x > y) x else y)
scala> minByKeyData.collect
```

OUTPUT:

```
Array[(String, Int)] = Array((Java,9), (Scala,5), (Spark,8))
```

12.9.2.Actions

Actions are RDD methods that return a value to a driver program.

1)Collect

The collect method returns the elements in the source RDD as an array.This method should be used with caution since it moves data from all the worker nodes to the driver program.

It can crash the driver program if called on a very large RDD.

Example Code :

```
scala> val rdd = sc.parallelize((1 to 10000).toList)
scala> val filteredRdd = rdd.filter { x => (x % 1000) == 0 }
scala>val filterResult = filteredRdd.collect
```

OUTPUT:

Array[Int] = Array(1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000)

2)Count

The count method returns a count of the elements in the source RDD.

Example Code:

```
scala> val rdd = sc.parallelize((1 to 10000).toList)
scala> val totalCount = rdd.count
```

OUTPUT:

totalCount: Long = 10000

Example 2:

```
scala> val data = sc.parallelize(List ("AAA","BBB","CCCC","DD","EEEEE","FFF"))
scala> println(data.count)
```

OUTPUT:

6

Example 3: INPUTD TA:

```
gopal@ubuntu:~/SPARK$ pwd /home/gopal/SPARK
gopal@ubuntu:~/SPARK$ cat Input.log
spark is in-memory cluster  omputing system spark is ment for spark processing
spark is having diff modulesspark is based on rdd processing only thanks spark hadoop
and spark are going to lead bigdata marketspark is in-memory cluster computing
framework spark is ment for spark processing spark is having diff modules spark is based
on rdd processing only thanks spark hadoop and spark are going to lead bigdata market
 gopal@ubuntu:~/SPARK$
```

CODE:

```
scala> val file = sc textFile("/home/gopal/SPARK/Input.log") // Base RDD
```

```
scala> val filData = file.filter(x => x.contains("spark")) // Transformed RDD
scala> println(filData.count)// ActionRDD or ComputedRDD
```

OUTPUT:

10

3)Count By Value

The countByValue method returns a count of each unique element in the source RDD.It returns an instance of the Map class containing each unique element and its count as a key-value pair.

Example Code:

```
scala> val data = sc.parallelize(List
("AAA","BBB","CCCC","DD","AAA","BBB","AAA","BBB","AAA")) scala> val
countByValueResult = data.countByValue
```

OUTPUT:

```
countByValueResult: scala.collection.Map[String,Long] = Map(DD -> 1, CCCC -> 1, BBB ->
3, AAA -> 4)
```

4)first

The first method returns the first element in the source RDD.

```
scala> val data = sc.parallelize(List
("AAA","BBB","CCCC","DD","AAA","BBB","AAA","BBB","AAA"))
scala> val firstElement = data.first
```

OUTPUT:

```
firstElement: String = AAA
```

Example 2: INPUTDATA:

```
gopal@ubuntu:~/SPARK$ pwd
/home/gopal/SPARK
gopal@ubuntu:~/SPARK$ cat Input.log
spark is in-memory cluster computing system spark is ment for spark processing
spark is having diff modules spark is based on rdd processing only thanks saprk
hadoop and spark are going to lead bigdata market spark is in-memory cluster computing
framework spark is ment for spark processing
spark is having diff modules spark is based on rdd processing only thanks saprk
hadoop and spark are going to lead bigdata market
gopal@ubuntu:~/SPARK$
```

CODE:

```
scala> val file = sc.textFile("/home/gopal/SPARK/Input.log") // Base RDD
```

```
scala> val filData = file.filter(x => x.contains("spark")) // Transformed RDD
scala> println(filData.first) // ActionRDD
```

OUTPUT:

spark is in-memory cluster computing system

5)max

The max method returns the largest element in an RDD.

```
scala> val rdd = sc.parallelize(List (45,98,73,61,55,85,91,10))
scala> rdd.max
```

OUTPUT: Int = 98

```
scala> val rdd = sc.parallelize(List("Spark","Spark&Scala","A","Java"))
scala> rdd.max
```

OUTPUT:

String = Spark&Scala

6)min

The min method returns the smallest element in an RDD

```
scala> val rdd = sc.parallelize(List(45,98,73,61,55,85,91,10))
scala> rdd.min
```

OUTPUT: Int = 10

```
scala> val rdd = sc.parallelize(List("Spark","Spark&Scala","A","Java"))
scala> rdd.min
```

OUTPUT: String = A

7)Top

The top method takes an integer N as input and returns an array containing the N largest elements in the source RDD.

```
scala> val rdd = sc.parallelize(List(45,98,73,61,55,85,91,10))
scala> rdd.top(3)
```

OUTPUT:

Array[Int] = Array(98, 91, 85)

```
scala> val rdd = sc.parallelize(List("Spark","Spark&Scala","A","Java"))
scala> rdd.top(2)
```

OUTPUT:

Array[String] = Array(Spark&Scala, Spark)

8)reduce

The higher-order reduce method aggregates the elements of the source RDD using an associative and commutative binary operator provided to it.

```
scala> val data = sc.parallelize(List(10,20,10,30,50,70))
scala> val sum = data.reduce( (x,y) => (x + y) )
```

OUTPUT:

```
sum: Int = 190
scala> val multipliedProduct = data.reduce( (x,y) => (x * y) )
```

OUTPUT:

```
multipliedProduct: Int = 210000000
```

12.9.3.Actions on RDD of key-value Pairs

1)CountByKey

The countByKey method counts the occurrences of each unique key in the source RDD. It returns a Map of key-count pairs.

```
scala> val iniData = sc.parallelize(List( ("Spark",1),("Scala",2),("Spark",3),
("Spark",4),("Spark",8), ("Java",9),("Scala",5),("Spark",6) ))scala> val countOfKeys =
iniData.countByKey()
```

OUTPUT:

```
countOfKeys: scala.collection.Map[String,Long] = Map(Java -> 1, Scala -> 2, Spark -> 5)
```

2)lookup

The lookup method takes a key as input and returns a sequence of all the values mapped to that key in the source RDD.

```
scala> val iniData = sc.parallelize(List(
("Spark",1),("Scala",2),("Spark",3),("Spark",4),("Spark",8),("Java",9),("Scala",5),("Spark",
6) ))
scala> val countOfKeys = iniData.lookup("Spark")
```

OUTPUT:

```
countOfKeys: Seq[Int] = WrappedArray(1, 3, 4, 8, 6)
//create a RDD called "lines"
scala> var lines = sc.textFile("/home/hadoop/DeviceInput.log");
```

```
scala> var lines = sc.textFile("/home/hadoop/DeviceInput
```

```
//Count the Num of lines in the same file called //"DeviceInput.log"
scala> lines.count()
```

```
scala> lines.count()
```

```
//to see the first line of the Given Input File
```

```
scala> lines.first()
```

```
res1: String = Cisco is one of the telecom custormer
```

12.10.HOW TO LOGIN TO SCALA SHELL IN SPARK

At command prompt (any path) , type spark-shell

```
gopal@ubuntu:~$ pwd
/home/gopal
gopal@ubuntu:~$ spark-shell
log4j:WARN No appenders could be found for logger (org.apache.hadoop.conf.Configuration).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
15/12/23 04:36:16 INFO SecurityManager: Changing view acls to: gopal
15/12/23 04:36:16 INFO SecurityManager: Changing modify acls to: gopal
15/12/23 04:36:16 INFO SecurityManager: SecurityManager: authentication disabled; ui acls
ith view permissions: Set(gopal); users with modify permissions: Set(gopal)
15/12/23 04:36:18 INFO HttpServer: Starting HTTP Server
15/12/23 04:36:18 INFO Utils: Successfully started service 'HTTP class server' on port 347
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.4.1
      /_/

Using Scala version 2.10.4 (OpenJDK Server VM, Java 1.7.0_79)
```

Jps –changes

```
gopal@ubuntu:~$ jps
3064 DataNode
2814 NameNode
4237 Jps
3418 JobTracker
3330 SecondaryNameNode
4090 SparkSubmit
3653 TaskTracker
gopal@ubuntu:~$
```

HOW TO LOGIN TO PHYTHON SHELL

At command prompt (any path) , type pyspark

```
hadoop@ubuntu:~$ pyspark
Python 2.7.6 (default, Mar 22 2014, 22:59:56)
[GCC 4.8.2] on linux2
```

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.4.1
      /_/

Using Python version 2.7.6 (default, Mar 22 2014 22:59:5
SparkContext available as sc, HiveContext available as s
>>>
```

Above Example In Phython

#Create an RDD called "lines"

lines = sc.textFile("/home/hadoop/DeviceInput.log")

```
>>> lines = sc.textFile("/home/hadoop/DeviceInput.log")
```

#Count the Num of lines in the same file called #"DeviceInput.log"

```
>>> lines.count()
```

pythonLines = lines.filter(lambda line: "Spark" in line)

NOTE: Python supports the creation of anonymous functions (i.e. functions that are not bound to a name) at runtime, using a construct called "lambda".

lambda definition does not include a "return" statement -- it always contains an expression which is returned

#to see the first line of the Given Input File

```
>>> lines.first()
```

Output

```
'Cisco is one of the telecom custormer'
```

Practical Usage Of "Transformations" and "Actions"

```
scala> var lines = sc.textFile("/home/hadoop/DeviceInput.log")
```

```
lines: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[7] at textFile at <console>:21

scala>
```

//In the above line we are taking a RDD called "lines" . Opening a TextFile called //"DeviceInput.log"

Once any RDD created, RDDs offer two types of operations:

Transformations: Transformations construct a new RDD from a previous one

Example: For example, one common transformation is filtering data that matches a predicate.

```
scala> val filteredName = lines.filter(line=>line.contains("Cisco"))
filteredName: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[10] at filter at <console>:23

scala>
```

// In the above we are calling the "filter" transformation

Actions: Actions, on the other hand, compute a result based on an RDD, and either return it to the driver program or save it to an external storage system (e.g., HDFS).

Example: One example of an action is first (), which returns the first element in an RDD

```
scala> filteredName.first()
```

// In the above we are using the "first()" action . to get the First Line of the //Input File

Output:

```
res2: String = Cisco is one of the telecom custormer
```

12.11.Why Spark is "Lazy Evaluated" System?

Transformations and actions are different because of the way Spark computes RDDs. Although you can define new RDDs any time, Spark computes them only in a lazy fashion—that is, the first time they are used in an action. This approach might seem unusual at first, but makes a lot of sense when you are working with Big Data.For instance, consider where we defined a text file and then filtered the lines that include "Cisco" client name If Spark were to load and store all the lines in the file as soon as we wrote

lines = sc.textFile(...),

it would waste a lot of storage space, given that we then immediately filter out many lines. Instead, once Spark sees the whole chain of transformations, it can compute just the data needed for its result. In fact, for the first() action, Spark scans the file only until it finds the first matching line; it doesn't even read the whole file.

12.12.How RDDs are Fault Tolerant?

RDD is designed to be fault tolerant. An RDD represents data distributed across a cluster of nodes.The probability of a node failing is proportional to the number ofnodes in a cluster. The larger a cluster, the higher the probability that some node will fail on any given day.

RDD automatically handles node failures. When a node fails, and partitions stored on that nodebecome inaccessible, Spark reconstructs the lost RDD partitions on another node. Spark stores lineage (hierarchy) information for each RDD. Using this lineage information, it can recover parts of an RDD or even an entire RDD in the event of node failures.

RDD.persist()

Spark's RDDs are by default recomputed each time you run an action on them. If you would like to reuse an RDD in multiple actions, you can ask Spark to persist it using RDD.persist().

```
scala> filteredName.persist()
res3: filteredName.type = MapPartitionsRDD[10] at filter at <console>:23

scala>
```

To summarize, every Spark program and shell session will work as follows:

Create some input RDDs from external data.

Transform them to define new RDDs using transformations like filter().

Ask Spark to persist() any intermediate RDDs that will need to be reused.

Launch actions such as count() and first() to kick off a parallel computation, which is then optimized and executed by Spark.

NOTE:

cache() is the same as calling persist() with the default storage level

12.13.How to Initialize Spark Context

```
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
val conf = new SparkConf().setMaster("local").setAppName("My App")
val sc = new SparkContext(conf)
```

NOTE:

A *cluster URL*, namely local in these examples, which tells Spark how to connect to a cluster. local is a special value that runs Spark on one thread on the local machine, without connecting to a cluster.

An *application name*, namely My App in these examples. This will identify your application on the cluster manager's UI if you connect to a cluster.

12.14.Parallelized collections

Parallelized collections are created by calling SparkContext's parallelize method on an existing collection in your driver program (a Scala Seq). The elements of the collection are copied to form a distributed dataset that can be operated on in parallel.

```
scala> val iniData = Array(10,20,30,40,50,60)
scala> val disData = sc.parallelize(iniData)
scala> val lengthDisData = disData.map(line=>line/2)
scala> println(lengthDisData.collect().mkString("\t"))
```

OUTPUT:

16/01/31 05:23:49 INFO DAGScheduler: Job 0 finished: collect at <console>:28, took 1.390161 s

5 10 15 20 25 30

```
scala> val sumDisData = disData.reduce( (a,b) => a+b )
```

16/01/31 05:27:36 INFO DAGScheduler: Job 1 finished: reduce at <console>:25, took 0.060768 s

sumDisData: Int = 210

SQUARING the Values in RDD

```
scala> val nums = sc.parallelize(List(10,20,30,40))
nums: org.apache.spark.rdd.RDD[Int] = ParallelCollection
RDD[2] at parallelize at <console>:21

scala>
```

```
scala> val result = nums.map(x => x * x)
result: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD

scala> println(result.collect().mkString(","))
```

Output

```
15/08/21 17:56:28 INFO DAGScheduler: Job 1 finished: col
lect at <console>:26, took 0.117266 s
100,400,900,1600
```

splitting lines into multiple words→ flatMap()

Sometimes we want to produce multiple output elements for each input element. The operation to do this is called flatMap().

```
scala> val lines = sc.parallelize(List("hadoop is one of the bigdata tool","Spark and Scala"))
scala> val words = lines.flatMap(line => line.split(" "))
```

scala> words.collect()

scala> words.first()

```
scala> val lines = sc.parallelize(List("hadoop is one of the bigdata tool","Spark and Scala"))
lines: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[5] at parallelize at <console>:21

scala> val words = lines.flatMap(line => line.split(" "))
words: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[6] at flatMap at <console>:23
```

```
scala> words.first()
```

Output is :

```
res4: String = hadoop
```

```
scala> words.collect()
```

How to get the Splited words from Given Line

```
res7: Array[String] = Array(hadoop, is, one, of, the, bigdata, tool, Spark, and, Scala)
```

UNION

scala> val data1 = sc.textFile("DeviceInput.log")

scala> val data2 = sc.textFile("DeviceInput.log")

scala> val uniData = data1.union(data2)

Output

```
 pool
15/08/21 18:28:00 INFO DAGScheduler: Job 6 finished: collect at <console>:28, took 0.223923 s
res8: Array[String] = Array(Cisco is one of the telecom customer, Cisco is having lot of departmen
ts as part of customer service, customers like Cisco services always, ONA is one of the department
of Cisco, MI is another department of Cisco, SA is smart analytics dept of Cisco, DFP is dealing w
th fault prediction of Cisco, Cisco 360DF view will give customer view, Cisco is one of the telecom
 customer, Cisco is having lot of departments as part of customer service, customers like Cisco se
rvices always, ONA is one of the department of Cisco, MI is another department of Cisco, SA is sma
t analytics dept of Cisco, DFP is dealing with fault prediction of Cisco, Cisco 360DF view will gi
e customer view)

scala>
```

12.15.pairRDD

scala> val pairRDD = sc.parallelize(List(("hadoop",2), ("CLOUD", 3),("BI",3), ("SAS", 4),("SALEFORCE", 12), ("SAP", 12), ("HADOOP", 2)), 2)

```
scala> def myfunc(index: Int, iter: Iterator[(String, Int)]) : Iterator[String] = {
    |  iter.toList.map(x => "[partID:" +  index + ", val: " + x + "]").iterator
    | }
myfunc: (index: Int, iter: Iterator[(String, Int)])Iterator[String]
scala>
```

```
scala> val pairRDD = sc.parallelize(List( ("hadoop",2), ("CLOUD", 3),("BI",3), ("SAS", 4),("SALEFO
CE", 12), ("SAP", 12), ("HADOOP", 2)), 2)
pairRDD: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[14] at parallelize at <co
sole>:21

scala> def myfunc(index: Int, iter: Iterator[(String, Int)]) : Iterator[String] = {
    | iter.toList.map(x => "[partID:" +  index + ", val: " + x + "]").iterator
    | }
myfunc: (index: Int, iter: Iterator[(String, Int)])Iterator[String]

scala> pairRDD.mapPartitionsWithIndex(myfunc).collect
```

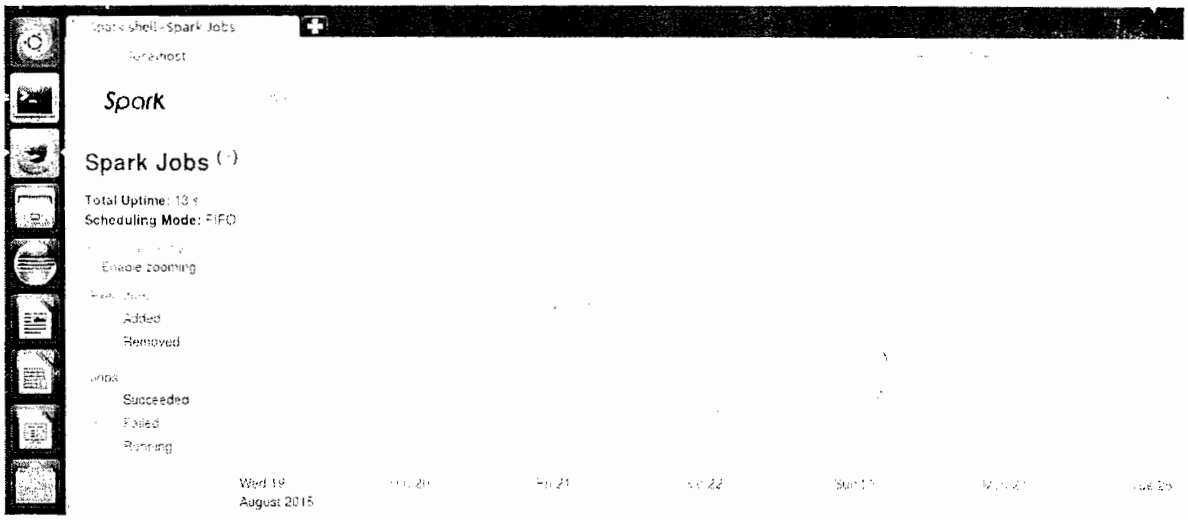
```
scala> pairRDD.mapPartitionsWithIndex(myfunc).collect
res9: Array[String] = Array([partID:0, val: (hadoop,2)], [partID:0, val: (CLOUD,3)], [partID:0, va
: (BI,3)], [partID:1, val: (SAS,4)], [partID:1, val: (SALEFORCE,12)], [partID:1, val: (SAP,12)], [
artID:1, val: (HADOOP,2)])

scala>
```

NOTE:

mapPartitionsWithIndex()→ Integer of partition number, and Iterator of the elements in that partition

12.16.HOW TO ACCESS SPARK GUI

We can access the Spark GUI at *http://localhost:4040*



OTHER EXAMPLES:

```
Scala>val data = 1 to 10000
//create a RDD based on "data"
Scala>val distData = sc.parallelize(data)
// to select the values less than 10
Scala>distData.filter(_ < 10).collect()
// base RDD!
val lines = sc.textFile("hdfs://localhost:54310/ErrorLog.txt")!!
// transformed RDDs!
val errors = lines.filter(_.startsWith("ERROR"))!
val messages = errors.map(_.split("\t")).map(r => r(1))!
messages.cache()!!
// action 1!
messages.filter(_.contains("mysql")).count()!!
// action 2!
messages.filter(_.contains("php")).count()
```

Spark Essentials: RDD

Scala

```
scala> val data = Array(1, 2, 3, 4, 5)!
data: Array[Int] = Array(1, 2, 3, 4, 5)!
scala> val distData = sc.parallelize(data)!
distData: spark.RDD[Int] = spark.ParallelCollection@10d13e3e
```

Phython

```
>>> data = [1, 2, 3, 4, 5]!
>>> data
```

```
[1, 2, 3, 4, 5]!
>>> distData = sc.parallelize(data)!
>>> distData!
ParallelCollectionRDD[0] at parallelize at PythonRDD.scala:229
```

12.17.Word Count Use Case Using Spark Context – In Scala

```
// Create a Scala Spark Context.
val conf = new SparkConf().setAppName("wordCount")
val sc = new SparkContext(conf)
// Load our input data.
val input = sc.textFile(inputFile)
// Split it up into words.
val words = input.flatMap(line => line.split(" "))
// Transform into pairs and count.
val counts = words.map(word => (word, 1)).reduceByKey{case (x, y) => x + y}
// Save the word count back out to a text file, causing evaluation.
counts.saveAsTextFile(outputFile)
```

12.18.spark sql

HiveContext provides a superset of the functionality provided by SQLContext. The parser that comes with HiveContext is more powerful than the SQLContext parser . It can execute both HiveQL and SQL queries. It can read data from Hive tables. It also allows applications to access Hive UDFs (user-defined functions).if we want to process existing Hive tables, add hive-site.xml file to Spark's classpath, since HiveContext reads Hive configuration from the hive-site.xml file.

Data Frames:DataFrame is Spark SQL's primary data abstraction. It represents a distributed collection of rows organized into named columns.Conceptually, it is similar to a table in a relational database.

Spark SQL is a Spark module for structured data processing. It provides a programming abstraction called DataFrames and can also act as distributed SQL query engine.

1. A DataFrame is a distributed collection of dataorganized into named columns

2.It is conceptually equivalent to a table in a relational database or a data frame in R/Python.

3.DataFrames can be constructed from a wide array of sources such as: structured data files, tables in Hive, external databases, or existing RDDs Converting "RDD" to "DataFrames [DF]"

Spark SQL provides an implicit conversion method named toDF, which creates a DataFrame

from an RDD of objects represented by a case class[To Define Schema Only we are using Case Class] When this technique is used, Spark SQL infers the schema of a dataset.The toDF method is not defined in the RDD class, but it is available through an implicit conversion.To convert an RDD to a DataFrame using toDF, we need to import the implicit methods

defined in the implicits object

Examples

```
scala> val data = sc.parallelize(1 to 10)
data: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at <console>:21
scala> val newData = data.map(I => (I , I+10) )
newData: org.apache.spark.rdd.RDD[(Int, Int)] = MapPartitionsRDD[1] at map at <console>:23
scala> val resultData = newData.toDF("original","transformed")
resultData: org.apache.spark.sql.DataFrame = [original: int, transformed: int]
scala> resultData.printSchema
root
|-- original: integer (nullable = false)
|-- transformed: integer (nullable = false)
scala> resultData.show

+--------+-----------+
|original|transformed|
+--------+-----------+
| 1| 11|
| 2| 12|
|
3| 13|
| 4| 14|
| 5| 15|
| 6| 16|
| 7| 17|
| 8| 18|
```

```
| 9| 19|
| 10| 20|
+--------+-----------+
scala> val data = List("Spark","Scala","Java","Phython")
data: List[String] = List(Spark, Scala, Java, Phython)
scala> val newData = data.map( l => (l , l.length))
newData: List[(String, Int)] = List((Spark,5), (Scala,5), (Java,4), (Phython,7))
scala> val resultData = newData.toDF("Name" , "Length")
resultData: org.apache.spark.sql.DataFrame = [Name: string, Length: int]
scala> resultData.printSchema
root
|-- Name: string (nullable = true)
|-- Length: integer (nullable = false)
scala> resultData.show
+-------+------+
| Name|Length|
+-------+------+
| Spark| 5|
| Scala| 5|
| Java| 4|
|Phython| 7|
+-------+------+
scala> val numList = List(1,2,3,4,5)
numList: List[Int] = List(1, 2, 3, 4, 5)
scala> val numRDD = sc.parallelize(numList)
numRDD: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[77] at parallelize at
<console>:26
scala> val numDF = numRDD.toDF
numDF: org.apache.spark.sql.DataFrame = [_1: int]
scala> numDF.show
+--+
|_1|
+--+
| 1|
| 2|
```

```
| 3|
| 4|
| 5|
+--+
scala> val data = sc.parallelize(List( ("Spark", 1),("Scala",2),("Phython",3),("R",4) ))
data: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[7] at parallelize at
<console>:21
scala> val resultData = data.toDF("Technology","Rank")
resultData: org.apache.spark.sql.DataFrame = [Technology: string, Rank: int]
scala> resultData.printSchema
root
|-- Technology: string (nullable = true)
|-- Rank: integer (nullable = false)
scala> resultData.show
+----------+----+
|Technology|Rank|
+----------+----+
| Spark| 1
|
| Scala| 2
|
| Phython| 3
|
| R| 4
|
+----------+----+
scala> resultData.registerTempTable("techtab")
scala>
scala> val tabList = sqlContext.sql("show tables")
scala> tabList.show
+---------+-----------+
|tableName|isTemporary|
+---------+-----------+
|
techtab| true
```

```
|
| dept| false|
| dept1| false|
| depttab| false|
| emp| false|
| emptab| false|
| testtab| false|
+---------+-----------+
scala> val bestTech = sqlContext.sql("select * from techtab")
16/09/26 00:02:56 INFO ParseDriver: Parsing command: select * from techtab
16/09/26 00:02:58 INFO ParseDriver: Parse Completed
bestTech: org.apache.spark.sql.DataFrame = [Technology: string, Rank: int]
scala> bestTech.show
+----------+----+
|Technology|Rank|
+----------+----+
| Spark| 1|
| Scala| 2|
| Phython| 3|
| R| 4|
+----------+----+
scala> val aTab = sqlContext.sql("select * from techtab where Technology like '%a%'")
scala> aTab.show
+----------+----+
|Technology|Rank|
+----------+----+
| Spark| 1|
| Scala| 2|
+----------+----+
scala> case class Line(x:Int,y:String)
defined class Line
scala> val data = sqlContext.createDataFrame( (1 to 100).map( l => Line( l , s"val_$l")) )
data: org.apache.spark.sql.DataFrame = [x: int, y: string]
scala> data.registerTempTable("LineTab")
scala> sqlContext.sql("select * from linetab").sho )
```

```
+--+------+
| x| y|
+--+------+
| 1| val_1|
| 2| val_2|
| 3| val_3|
| 4| val_4|
| 5| val_5|
| 6| val_6|
| 7| val_7|
| 8| val_8|
| 9| val_9|
|10|val_10|
|11|val_11|
|12|val_12|
|13|val_13|
|14|val_14|
|15|val_15|
|16|val_16|
|17|val_17|
|18|val_18|
|19|val_19|
|20|val_20|
+--+------+
scala> val data = sqlContext.createDataFrame( (1 to 100).map( l => Line( l , (l+2).toString())) )
data: org.apache.spark.sql.DataFrame = [x: int, y: string]
scala> data.registerTempTable("LineTab")
scala> sqlContext.sql("select * from linetab").show()
+--+--+
| x| y|
+--+--+
| 1| 3|
| 2| 4|
| 3| 5|
```

```
| 4| 6|
| 5| 7|
| 6| 8|
| 7| 9|
| 8|10|
| 9|11|
|10|12|
|11|13|
|12|14|
|13|15|
|14|16|
|15|17|
|16|18|
|17|19|
|18|20|
```

NOTE

:

show command will only display the first 20 values.

```
scala> sqlContext.sql("select * from linetab WHERE y > 15").show()
+--+--+
| x| y|
+--+--+
|14|16|
|15|17|
|16|18|
|17|19|
|18|20|
|19|21|
|20|22|
|21|23|
|22|24|
|23|25|
|24|26|
|25|27|
|26|28|
```

```
|27|29|
|28|30|
|29|31|
|30|32|
|31|33|
|32|34|
|33|35|
+--+--+
scala> sqlContext.sql("select * from linetab WHERE y > 15 AND y < 22").show()
+--+--+
| x| y|
+--+--+
|14|16|
|15|17|
|16|18|
|17|19|
|18|20|
|19|21|
+--+--+
```

To get all the values we have to use foreach()..

```
scala> sqlContext.sql("select * from linetab WHERE x >15").collect().foreach(println)
```

INPUTDATA:

```
gopal@ubuntu:~/SparkSQLInput$ cat EmpDetails.txt
1000 Gopal
12000
1001
Krishna 14000
1002
Ravi
16000
1002 Ramya 18000
1003
Rakesh
20000
1004 Rajesh 22000
```

```
1005
Mounika
24000
1006
Sravya 26000
1007 Siya
28000
1008
Dixitha
30000
1009 Trinath 34000
scala> val empdata = sc.textFile("/home/gopal/SparkSQLInput/EmpDetails.tx ")
scala> case class Employee(id:Int,name:String,sal:Int)
defined class Employee
scala> val newdata = empdata.map(_.split("\t"))
newdata: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[26] a  map
scala> val resultdata = newdata.map( e => Employee( e(0).trim.toInt , e(1) e(2).trim.toInt
)).toDF()
resultdata: org.apache.spark.sql.DataFrame = [id: int, name: string, sal: int]
scala> resultdata.printSchema
root
|-- id: integer (nullable = false)
|-- name: string (nullable = true)
|-- sal: integer (nullable = false)
scala> resultdata.show
+----+-------+-----+
| id| name| sal|
+----+-------+-----+
|1000| Gopal|12000|
|1001|Krishna|14000|
|1002| Ravi|16000|
|1002| Ramya|18000|
|1003| Rakesh|20000|
|1004| Rajesh|22000|
|1005|Mounika|24000|
```

```
|1006| Sravya|26000|
|1007| Siya|28000|
|1008|Dixitha|30000|
|1009|Trinath|34000|
+----+-------+-----+
scala> resultdata.registerTempTable("employeetab")
scala> val entrylevelemp = sqlContext.sql("select * from employeetab WHERE sal > 15000
AND sal <
30000").show
+----+-------+-----+
| id| name| sal|
+----+-------+-----+
|1002| Ravi|16000|
|1002| Ramya|18000|
|1003| Rakesh|20000|
|1004| Rajesh|22000|
|1005|Mounika|24000|
|1006| Sravya|26000|
|1007| Siya|28000|
+----+-------+-----+
```

INPUT DATA:

```
gopal@ubuntu:~/SparkSQLInput$ cat AccountDetails.txt
10101,Gopal,Savings,ICICI,50000
10201,Krishna,Current,SBI,56000
10301,Ravi,Savings,HDFC,67000
20201,Ramya,Savings,SBI,89000
30303,Radha,Fixed,ICICI,560000
50607,Sruthi,Current,AXIS,45000
70007,Nalini,Fixed,HDFC,66000
33333,Raj,Savings,AXIS,59500
20201,Ramya,Savings,SBI,89000
55303,Rakhee,Fixed,ICICI,560000
50607,Santhosh,Current,AXIS,45000
88007,Nandha,Fixed,HDFC,65900
44333,Raj,Savings,SBI,79500
```

```
gopal@ubuntu:~/SparkSQLInput$
scala> import sqlContext.implicits._
import sqlContext.implicits._
scala> val accdetails = sc.textFile("/home/gopal/SparkSQLInput/AccountDetails.txt")
scala> case class
Account(acc_id:Int,acc_holder:String,acc_type:String,bankname:String,amt:Int)
defined class Account
scala> val newAccData = accdetails.map(_.split(",")).map(a =>
Account(a(0).trim.toInt,a(1),a(2),a(3),a(4).trim.toInt)).toDF()
newAccData: org.apache.spark.sql.DataFrame = [acc_id: int, acc_holder: string, acc_type:
string,
bankname: string, amt: int]
scala> newAccData.registerTempTable("accounttab")
scala> newAccData.show
+------+----------+--------+--------+------+
|acc_id|acc_holder|acc_type|bankname| amt|
+------+----------+--------+--------+------+
| 10101| Gopal| Savings| ICICI| 50000|
| 10201| Krishna| Current| SBI| 56000|
| 10301| Ravi| Savings| HDFC| 67000|
| 20201| Ramya| Savings| SBI| 89000|
| 30303| Radha| Fixed| ICICI|560000|
| 50607| Sruthi| Current| AXIS| 45000|
| 70007| Nalini| Fixed| HDFC| 66000|
| 33333| Raj| Savings| AXIS| 59500|
| 20201| Ramya| Savings| SBI| 89000|
| 55303| Rakhee| Fixed| ICICI|560000|
| 50607| Santhosh| Current| AXIS| 45000|
| 88007| Nandha| Fixed| HDFC| 65900|
| 44333| Raj| Savings| SBI| 79500|
+------+----------+--------+--------+------+
scala> newAccData.groupBy("acc_type").count().show
+--------+-----+
|acc_type|count|
+--------+-----+
```

```
| Savings| 6
|
| Current| 3
|
| Fixed| 4
|
+--------+-----+
scala> sqlContext.sql("select bankname , count(*) from accounttab group by bankname").show
+--------+---+
|bankname|_c1|
+--------+---+
| AXIS| 3
|
| HDFC| 3
|
| SBI| 4
|
| ICICI| 3
|
+--------+---+
```

12.19.DIFF WAYS OF "LOADING" & "SAVING" DATA

USING SPARK SQL

Spark SQL provides a unified interface for creating a DataFrame from a variety of data sources

Same API can be used to create a DataFrame from a Parquet, JSON, ORC or CSV file on local

file system, HDFS or S3 Spark SQL provides a class named DataFrameReader, which defines the interface for reading data from a data source. It allows you to specify different options for reading data.

NOTE: The default data source is Parquet , if NO Other format is specified. PFB Example for

the same:

```
scala> val data = sqlContext.read.load("/home/gopal/Input.log")
java.io.IOException: Could not read footer: java.lang.RuntimeException:
```

file:/home/gopal/Input.log is not a

Parquet file

Parquet Files:

12.20.Parquet

is a columnar format that is supported by many other data processing systems. Spark SQL provides support for both reading and writing Parquet files that automatically preserves the

schema of the original data.Parquet is a popular column-oriented storage format that can store records with nested fields efficiently. It is often used with tools in the Hadoop ecosystem, and it supports all of the data types in Spark SQL. Spark SQL provides methods for reading data directly to and from Parquet files.

Apache Parquet is a columnar storage format for the Hadoop ecosystem. Since its inception about 2 years ago, Parquet has gotten very good adoption due to the highly efficient compression and encoding schemes used that demonstrate significant performance benefits. Its ground-up design allows it to be used regardless of any data processing framework, data model, and programming language used in Hadoop ecosystem. A variety of tools and frameworks including MapReduce, Hive, Impala, and Pig provided the ability to work with Parquet data and a number of data models such as AVRO, Protobuf, and Thrift have been expanded to be used with Parquet as storage format. Parquet is widely adopted by a number of major companies including tech giants such as Twitter and Netflix. To Save the File as a Parquet file , use the below method.people.saveAsParquetFile("people.parquet")

Examples On Parquet File

```
scala> val parquetFile =
sqlContext.parquetFile("/home/gopal/SparkSQLInput/users.parquet")
parquetFile: org.apache.spark.sql.DataFrame = [name: string, favorite_color:
string, favorite_numbers: array<int>]
scala> parquetFile.registerTempTable("parquetFile")
scala> parquetFile.printSchema
root
|-- name: string (nullable = false)
|-- favorite_color: string (nullable = true)
|-- favorite_numbers: array (nullable = false)
| |-- element: integer (containsNull = false)
scala> val selectedPeople = sqlContext.sql("SELECT name FROM parquetFile")
```

```
scala> selectedPeople.map(t => "Name: " + t(0)).collect().foreach(println)
```

OUTPUT:

```
Name: Alyssa
Name: Ben
```

Alternative Way:

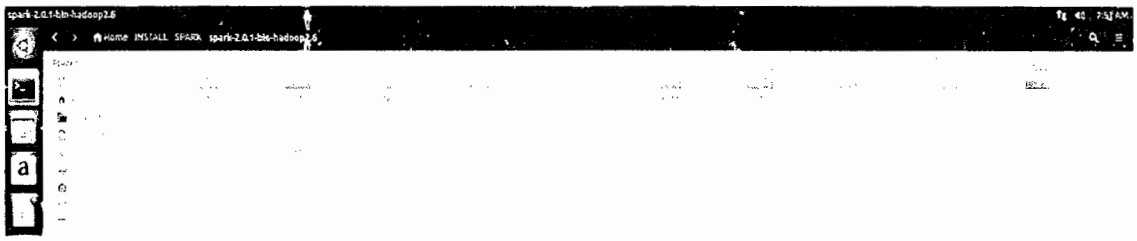
```
scala> val selectedPeople = sqlContext.sql("SELECT name FROM parquetFile").show
+------+
| name|
+------+
|Alyssa|
| Ben|
+------+
```

How to Save the Data in a "ParquetFile" format

```
scala> val sqlContext = new org.apache.spark.sql.SQLContext(sc)
sqlContext: org.apache.spark.sql.SQLContext =
org.apache.spark.sql.SQLContext@fb0ff
scala> val df = sqlContext.read.load("/home/gopal/SparkSQLInput/users.parquet")
df: org.apache.spark.sql.DataFrame = [name: string, favorite_color: string,
favorite_numbers: array<int>]
df.select("name", "favorite_color").write.save("namesAndFavColors.parquet")
```

12.21.SPARK SQL – 2.X EXAMPLES WITH "JDBC" INTRACTION

copy "Spark_Custom.jars" foder to "$SPARK_HOME" folder



Rename "$SPARK_HOME/conf/spark-defaults.conf.template" file TO "$SPARK_HOME/conf/spark-defaults.conf"

Add the below lines to "$SPARK_HOME/conf/spark-defaults.conf" file

```
spark.driver.memory      5g
spark.jars   /home/gopalkrishna/INSTALL/SPARK/spark-2.0.1-bin-
hadoop2.6/Spark_Custom.jars/guava-
```

16.0.1.jar,/home/gopalkrishna/INSTALL/SPARK/spark-2.0.1-bin-hadoop2.6/Spark_Custom.jars/spark-cassandra-connector_2.11-2.0.0-M3.jar,/home/gopalkrishna/INSTALL/SPARK/spark-2.0.1-bin-hadoop2.6/Spark_Custom.jars/cassandra-driver-core-3.0.2.jar,/home/gopalkrishna/INSTALL/SPARK/spark-2.0.1-bin-hadoop2.6/Spark_Custom.jars/jackson-core-2.5.2.jar,/home/gopalkrishna/INSTALL/SPARK/spark-2.0.1-bin-hadoop2.6/Spark_Custom.jars/jackson-databind-2.5.2.jar,/home/gopalkrishna/INSTALL/SPARK/spark-2.0.1-bin-hadoop2.6/Spark_Custom.jars/jackson-module-scala_2.11-2.5.2.jar,/home/gopalkrishna/INSTALL/SPARK/spark-2.0.1-bin-hadoop2.6/Spark_Custom.jars/jsr166e-1.1.0.jar,/home/gopalkrishna/INSTALL/SPARK/spark-2.0.1-bin-hadoop2.6/Spark_Custom.jars/mysql-connector-java-5.1.38.jar,/home/gopalkrishna/INSTALL/SPARK/spark-2.0.1-bin-hadoop2.6/Spark_Custom.jars/phoenix-4.7.0-HBase-1.1-client.jar,/home/gopalkrishna/INSTALL/SPARK/spark-2.0.1-bin-hadoop2.6/Spark_Custom.jars/phoenix-spark-4.7.0-HBase-1.1.jar

start the `spark-shell

SPARK SQL CODE EXAMPLES

```
scala> import org.apache.spark._
import org.apache.spark._
scala> import org.apache.spark.sql._
import org.apache.spark.sql._
```

NOTE: To Connect to RDBMS , we MUST require the below 4 values

To connect any RDBMS:

1. connection url
2. username & password
3. driver class name
4. client jar

FROM MySQL SIDE

```
mysql> create database gkdb;
Query OK, 1 row affected (0.00 sec)
mysql> use gkdb;
Database changed
```

```
mysql> INSERT INTO emptab VALUES(100,'RAJ',12000);
mysql> INSERT INTO emptab VALUES(101,'RAJU',14000);
mysql> INSERT INTO emptab VALUES(102,'RAM',16000);
mysql> INSERT INTO emptab VALUES(103,'RAVI',18000);
mysql> INSERT INTO emptab VALUES(104,'RAMYA',20000);
mysql> INSERT INTO emptab VALUES(105,'RAKESH',22000);
mysql> INSERT INTO emptab VALUES(106,'KAVITHA',24000);
mysql> INSERT INTO emptab VALUES(107,'KAVYA',26000);
scala> val prop = new java.util.Properties
prop: java.util.Properties = {}
scala> prop.setProperty("driver","com.mysql.jdbc.Driver")
res0: Object = null
scala> prop.setProperty("user","root")
res1: Object = null
scala> prop.setProperty("password","root")
res2: Object = null
scala> val jdbcDF = spark.read.jdbc("jdbc:mysql://localhost:3306/gkdb", "emptab", prop)
jdbcDF: org.apache.spark.sql.DataFrame = [id: int, name: string ... 1 more field]
scala> jdbcDF.show
+---+-------+-----+
| id|  name| sal|
+---+-------+-----+
|100|   RAJ|12000|
|101|  RAJU|14000|
|102|   RAM|16000|
|103|  RAVI|18000|
|104| RAMYA|20000|
|105|RAKESH|22000|
|106|KAVITHA|24000|
|107| KAVYA|26000|
+---+-------+-----+
scala> val jdbcDF = spark.read.jdbc("jdbc:mysql://localhost:3306/gkdb", "emptab",
Array("sal > 24000") , prop)
jdbcDF: org.apache.spark.sql.DataFrame = [id: int, name: string ... 1 more field]
scala> jdbcDF.show
```

```
+---+-----+-----+
| id| name|  sal|
+---+-----+-----+
|107|KAVYA|26000|
+---+-----+-----+
```

TO WRITE THE DATA FRAME DATA IN "JSON" FORMAT

scala> jdbcDF.write.json("/home/gopalkrishna/PRAC/emp.json")

TO WRITE THE DATA FRAME DATA IN "PARQUET" FORMAT

scala> jdbcDF.write.parquet("/home/gopalkrishna/PRAC/emp.parquet")

TO WRITE THE DATA FRAME DATA IN "ORC" FORMAT

scala> jdbcDF.write.orc("/home/gopalkrishna/PRAC/emp.orc")

TO WRITE THE DATA FRAME DATA IN "CSV" FORMAT

scala> jdbcDF.write.csv("/home/gopalkrishna/PRAC/emp.csv")

scala> jdbcDF.write.mode("append").csv("/home/gopalkrishna/PRAC/emp.csv")

scala> jdbcDF.write.mode("overwrite").csv("/home/gopalkrishna/PRAC/emp.csv")

TO WRITE THE SAME DATAFRAME DATA TO HIVE TABLES

To Have Spark – Hive Integration→ copy the $HIVE_HOME/conf/hive-site.xml and place it in $SPARK_HOME/conf/

scala> jdbcDF.write.saveAsTable("emptabb")

NOTE: Go to Hive Shell , and check the above table "emptabb" in default database

scala> jdbcDF.write.saveAsTable("gkdb.emptabb")

NOTE: Go to Hive Shell , and check the above table "emptabb" in gkdb database

TO WRITE THE SAME DATAFRAME DATA TO RDBMS

scala> jdbcDF.write.jdbc("jdbc:mysql://localhost:3306/gkdb", "exptab", prop)

scala> jdbcDF.write. mode("overwrite")jdbc("jdbc:mysql://localhost:3306/gkdb", "exptab", prop)

scala> jdbcDF.write. mode("append").jdbc("jdbc:mysql://localhost:3306/gkdb", "exptab", prop)

12.22.Spark Streaming

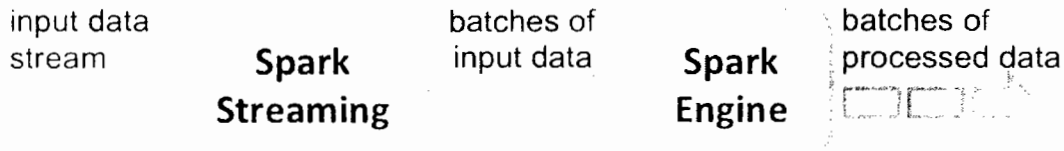
Spark Streaming is Spark's module for real time applications (for ex: track statistics about page views in real time, automatically detect anomalies). It lets users write streaming applications using a very similar API to batch jobs, and thus reuse a lot of the skills and even code they built for those.

Spark Streaming is a distributed data stream processing framework. It makes it easy to develop distributed applications for processing live data streams in near real time. It not

only provides a simple programming model but also enables an application to process high-velocity stream data. It also allows the combining of data streams and historical data for processing.

Spark Streaming is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams. Data can be ingested from many sources like Kafka, Flume, Twitter, ZeroMQ, Kinesis or TCP sockets can be processed using complex algorithms expressed with high-level functions like map, reduce, join and window. Finally, processed data can be pushed out to file systems, databases, and live dashboards

12.22.1.Process Flow in Spark Streaming



Spark Streaming works in the below fashion:

Spark Streaming receives live input data streams and divides the data into batches.
Spark Engine will process the same data
Once processing is done Spark engine will generate the final stream of results in batches.

12.22.2.High-Level Architecture

Spark Streaming processes a data stream in micro-batches.

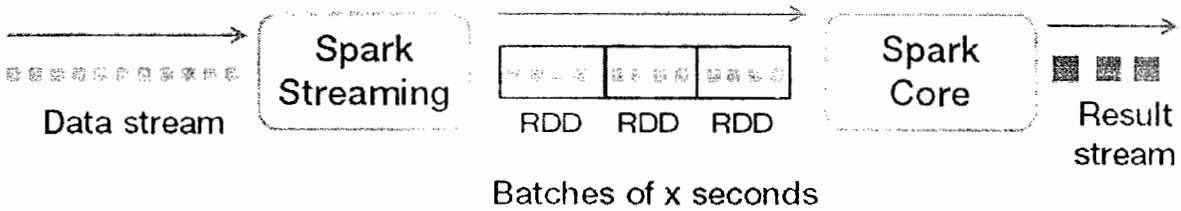
It splits a data stream into batches of very small fixed-sized time intervals.

Data in each micro-batch is stored as an RDD, which is then processed using Spark core

Any RDD operation can be applied to an RDD created by Spark Streaming.

The results of the RDD operations are streamed out in batches

.



12.22.3.StreamingContext

StreamingContext, a class defined in the Spark Streaming library, is the main entry point into the Spark Streaming library.

It allows a Spark Streaming application to connect to a Spark cluster.

It also provides methods for creating an instance of the data stream abstraction provided by Spark Streaming.

Every Spark Streaming application must create an instance of this class.

```
import org.apache.spark._
import org.apache.spark.streaming._
val config = new SparkConf().setMaster("spark://host:port")
                                .setAppName("big streaming app")
val batchInterval = 10
val ssc = new StreamingContext(conf, Seconds(batchInterval))
```

NOTE: The batch size can be as small as 500 milliseconds. The upper bound for the batch size is determined by the latency requirements of your application and the available memory

12.22.4.Starting Stream Computation

The start method begins stream computation. Nothing really happens in a Spark Streaming application until the start method is called on an instance of the StreamingContext class.

A Spark Streaming application begins receiving data after it calls the start method.

```
ssc.start()
```

Waiting for Stream Computation to Finish

The awaitTermination method in the StreamingContext class makes an application thread wait for stream computation to stop.

It's syntax is:

ssc.awaitTermination()

12.23.5.DStreams or discretized streams

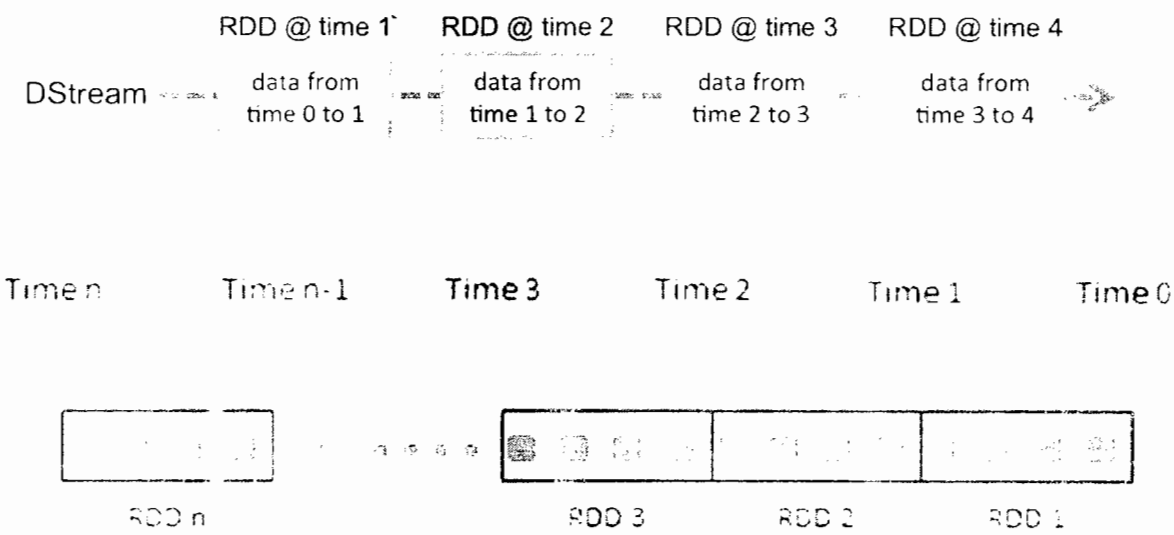
Spark is built on the concept of RDDs, Spark Streaming provides an abstraction called DStreams, or discretized streams.

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.

A DStream is a sequence of data arriving over time. Internally, each DStream is represented as a sequence of RDDs arriving at each time step (hence the name "discretized").

Spark Streaming provides a high-level abstraction called discretized stream or DStream, which represents a continuous stream of data

DStream is represented by a continuous series of RDDs, which is Spark's abstraction of an immutable, distributed dataset. Each RDD in a DStream contains data from a certain interval, as shown in the following figure.



DStreams offer two types of operations:

Transformations: which yield a new DStream.

Output operations: which write data to an external system

NOTE: DStreams provide many of the same operations available on RDDs, plus new operations related to time, such as sliding windows

Unlike batch programs, Spark Streaming applications need additional setup in order to operate 24/7.

Check pointing: main mechanism Spark Streaming provides for streaming purpose, which lets it store data in a reliable file system such as HDFS.

NOTE: In Spark 1.1, Spark Streaming is available only in Java and Scala. Experimental Python support was added in Spark 1.2

Transform

The transform method returns a DStream by applying an RDD => RDD function to each RDD in the source DStream. It takes as argument a function that takes an RDD as argument and returns an RDD. Thus, it gives us direct access to the underlying RDDs of a DStream.

This method allows you to use methods provided by the RDD API, but which do not have equivalent operations in the DStream API. For example, sortBy is a transformation available in the RDD API, but not in the DStream API. If you want to sort the elements within each RDD of a DStream, you can use the transform

Method as shown in the following example.

```
val lines = ssc.socketTextStream("localhost", 9999)
val words = lines.flatMap{line => line.split(" ")}
val sorted = words.transform{rdd => rdd.sortBy((w)=> w)}
```

NOTE: The transform method is also useful for applying machine and graph computation algorithms to data streams. The machine learning and graph processing libraries provide classes and methods that operate at the RDD level. Within the transform method, you can use the API provided by these libraries

12.23.6.Spark Streaming Use Case

We will receive a stream of newline-delimited lines of text from a server running at port 7777, filter only the lines that contain the word error, and print them.

Maven coordinates for Spark Streaming

groupId = org.apache.spark

artifactId = spark-streaming_2.10

version = 1.2.0

Scala streaming imports

```
import org.apache.spark.streaming.StreamingContext
import org.apache.spark.streaming.StreamingContext._
import org.apache.spark.streaming.dstream.DStream
import org.apache.spark.streaming.Duration
import org.apache.spark.streaming.Seconds
```

StreamingContext:

Main entry point for streaming functionality.

This also sets up an underlying SparkContext that it will use to process the data.

It takes as input a batch interval specifying how often to process new data... which we set to 1 second.

socketTextStream()

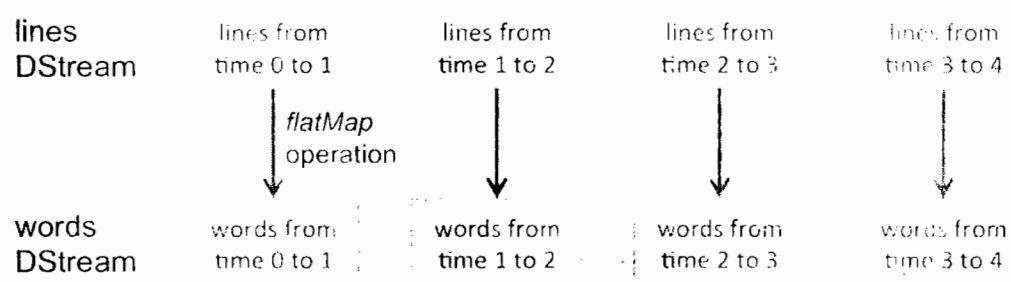
We use socketTextStream() to create a DStream based on text data received on port 7777 of the local machine.

Then we transform the DStream with filter() to get only the lines that contain error

Finally, we apply the output operation print() to print some of the filtered lines.

```
// Create a StreamingContext with a 1-second batch size from a SparkConf
val ssc = new StreamingContext(conf, Seconds(1))
// Create a DStream using data received after connecting to port 7777 on the local machine
val lines = ssc.socketTextStream("localhost", 7777)
// Filter our DStream for lines with "error"
val errorLines = lines.filter(_.contains("error"))
// Print out the lines with errors
errorLines.print()
```

For example, in the above example of converting a stream of lines to words, the flatMap operation is applied on each RDD in the lines DStream to generate the RDDs of the words DStream. This is shown in the following figure.



Input DStreams and Receivers

nput DStreams are DStreams representing the stream of input data received from streaming sources. In the above example of converting streaming of lines into words, lines was an input DStream as it represented the stream of data received from the netcat server

Every input DStream is associated with a Receiver (Scala doc, Java doc) object which receives the data from a source and stores it in Spark's memory for processing.

Spark Streaming provides two categories of built-in streaming sources.

Basic sources: Sources directly available in the StreamingContext API. Example: file systems, socket connections, and Akka actors.

Advanced sources: Sources like Kafka, Flume, Kinesis, Twitter, etc. are available through extra utility classes. These require linking against extra dependencies.

Basic Sources

File Streams: For reading data from files on any file system compatible with the HDFS API (that is, HDFS, S3, NFS, etc.), a DStream can be created as:

streamingContext.fileStream[KeyClass, ValueClass, InputFormatClass](dataDirectory)

Spark Streaming will monitor the directory dataDirectory and process any files created in that directory (files written in nested directories not supported). Note that

The files must have the same data format.

The files must be created in the data Directory by atomically moving or renaming them into the data directory.

Once moved, the files must not be changed. So if the files are being continuously appended, the new data will not be read.

For simple text files, there is an easier method streamingContext.textFileStream(dataDirectory). And file streams do not require running a receiver, hence does not require allocating cores.

Advanced Sources

This category of sources requires interfacing with external non-Spark libraries, some of them with complex dependencies (e.g., Kafka and Flume).

Hence, to minimize issues related to version conflicts of dependencies, the functionality to create DStreams from these sources have been moved to separate libraries

Example:

If we want to create a DStream using data from Twitter's stream of tweets, you have to do the following.

Linking: Add the artifact spark-streaming-twitter_2.10 to the SBT/Maven project dependencies

Programming: Import the TwitterUtils class and create a DStream with TwitterUtils.createStream as shown below

Some of these advanced sources are as follows.

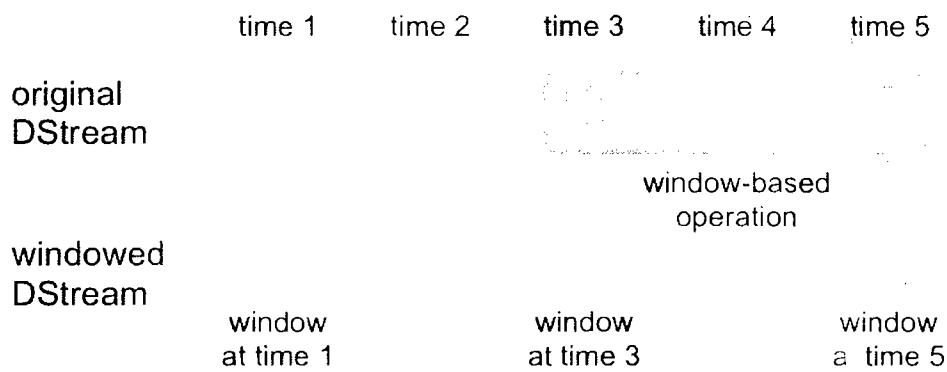
Twitter: Spark Streaming's TwitterUtils uses Twitter4j 3.0.3 to get the public stream of tweets using Twitter's Streaming API. Authentication information can be provided by any of the methods supported by Twitter4J library. You can either get the public stream, or get the filtered stream based on a keywords.

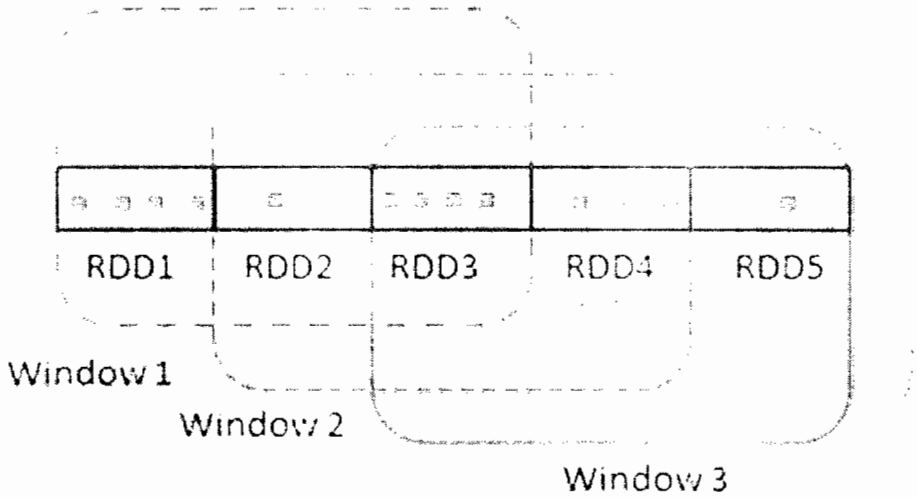
Flume: Spark Streaming 1.2.0 can received data from Flume 1.4.0 .

Kafka: Spark Streaming 1.2.0 can receive data from Kafka 0.8.0.

Window Operations

Spark Streaming also provides windowed computations, which allow you to apply transformations over a sliding window of data. This following figure explaines this sliding window.





Scala Code

// Reduce last 30 seconds of data, every 10 seconds

val windowedWordCounts = pairs.reduceByKeyAndWindow((a:Int,b:Int) => (a + b), Seconds(30), Seconds(10))

NOTE: Above Code generating word counts over last 30 seconds of data, every 10 seconds. To do this, we have to apply the reduceByKey operation on the pairs DStream of (word, 1) pairs over the last 30 seconds of data. This is done using the operation reduceByKeyAndWindow.

Performance Considerations

Batch and Window Sizes

The most common question is what minimum batch size Spark Streaming can use. In general, 500 milliseconds has proven to be a good minimum size for many applications.

The best approach is to start with a larger batch size (around 10 seconds) andwork your way down to a smaller batch size. If the processing times reported in the Streaming UI remain consistent, then you can continue to decrease the batch size.

Level of Parallelism

A common way to reduce the processing time of batches is to increase the parallelism. There are three ways to increase the parallelism.

Increasing the number of receivers

Explicitly repartitioning received data

explicitly repartitioning the input stream (or the union multiple streams) using DStream.repartition.

Increasing parallelism in aggregation

For operations like reduceByKey(), you can specify the parallelism as a second parameter

Spark-Streaming-Twitter-Example

```
// scalastyle:off println
package org.apache.spark.examples.streaming
import org.apache.spark.streaming.{Seconds, StreamingContext}
import org.apache.spark.SparkContext._
import org.apache.spark.streaming.twitter._
import org.apache.spark.SparkConf
/**
 * Calculates popular hashtags (topics) over sliding 10 and 60 second windows from a Twitter
 * stream. The stream is instantiated with credentials and optionally filters supplied by the
 * command line arguments.
 * Run this on your local machine as
 */
object TwitterPopularTags {
  def main(args: Array[String]) {
    if (args.length < 4) {
      System.err.println("Usage: TwitterPopularTags <consumer key> <consumer secret> " +
        "<access token> <access token secret> [<filters>]")
      System.exit(1)
    }
    StreamingExamples.setStreamingLogLevels()
    val Array(consumerKey, consumerSecret, accessToken, accessTokenSecret) = args.take(4)
    val filters = args.takeRight(args.length - 4)
    // Set the system properties so that Twitter4j library used by twitter stream
    // can use them to generat OAuth credentials
    System.setProperty("twitter4j.oauth.consumerKey", consumerKey)
    System.setProperty("twitter4j.oauth.consumerSecret", consumerSecret)
    System.setProperty("twitter4j.oauth.accessToken", accessToken)
    System.setProperty("twitter4j.oauth.accessTokenSecret", accessTokenSecret)
    val sparkConf = new SparkConf().setAppName("TwitterPopularTags")
    val ssc = new StreamingContext(sparkConf, Seconds(2))
```

```
    val stream = TwitterUtils.createStream(ssc, None, filters)
    val hashTags = stream.flatMap(status => status.getText.split(" ").filter(_.startsWith("#")))

    val topCounts60 = hashTags.map((_, 1)).reduceByKeyAndWindow(_ + _, Seconds(60))
                .map{case (topic, count) => (count, topic)}
                .transform(_.sortByKey(false))
    val topCounts10 = hashTags.map((_, 1)).reduceByKeyAndWindow(_ + _, Seconds(10))
                .map{case (topic, count) => (count, topic)}
                .transform(_.sortByKey(false))
    // Print popular hashtags
    topCounts60.foreachRDD(rdd => {
      val topList = rdd.take(10)
      println("\nPopular topics in last 60 seconds (%s total):".format(rdd.count()))
      topList.foreach{case (count, tag) => println("%s (%s tweets)".format(tag, count))}
    })
    topCounts10.foreachRDD(rdd => {
      val topList = rdd.take(10)
      println("\nPopular topics in last 10 seconds (%s total):".format(rdd.count()))
      topList.foreach{case (count, tag) => println("%s (%s tweets)".format(tag, count))}
    })
    ssc.start()
    ssc.awaitTermination()
  }
}
// scalastyle:on println
package org.apache.spark.examples.streaming
import org.apache.log4j.{Level, Logger}
import org.apache.spark.Logging

/** Utility functions for Spark Streaming examples. */
object StreamingExamples extends Logging {
  /** Set reasonable logging levels for streaming if the user has not configured log4j. */
  def setStreamingLogLevels() {
    val log4jInitialized = Logger.getRootLogger.getAllAppenders.hasMoreElements
    if (!log4jInitialized) {
```

```
    // We first log something to initialize Spark's default logging, then we override the
    // logging level.
    logInfo("Setting log level to [WARN] for streaming example." +
      " To override add a custom log4j.properties to the classpath.")
    Logger.getRootLogger.setLevel(Level.WARN)
  }
}
}
```

STORM

13.1.Introduction

Apache Storm, in simple terms, is a distributed framework for real time processing of Big Data like Apache Hadoop is a distributed framework for batch processing.

13.2.Advantages of Storm

1) Fault tolerance - where if worker threads die, or a node goes down, the workers are automatically restarted;

2) Scalability - where throughput rates of even one million 100 byte messages per second per node can be achieved and

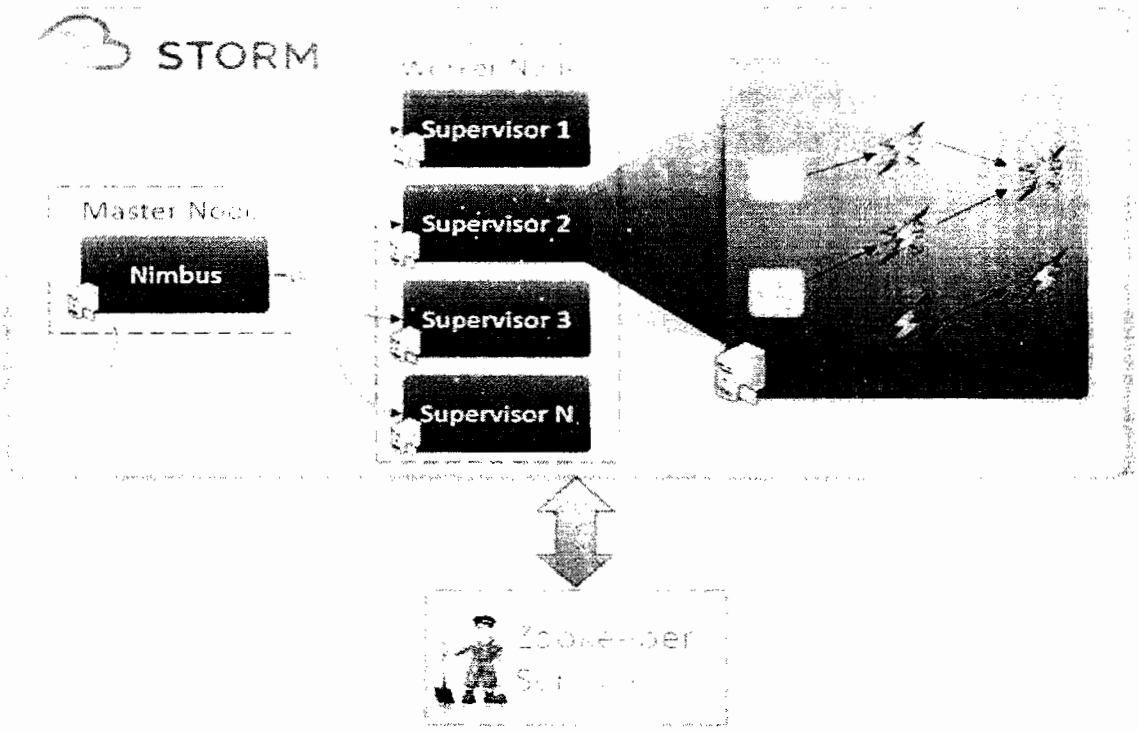
3) Ease of use in deploying and operating the system.

13.3.Need of Storm

First of all , we should understand why we needed storm when we already had hadoop.Hadoop is intelligently designed to process huge amount of data in distributed fashion on commodity hardwares but the way it processes does not suit the use case of Real time processing. MR jobs are disk intensive, every time it runs, it fetches data from disk,processes in memory in mapper/reducer and then write back to disk ; the job is finished then and there. For next batch of data, a new job will be created again and whole cycle will repeat.

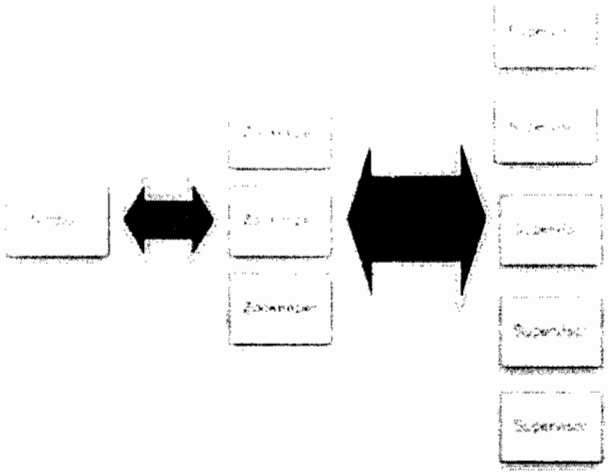
However in real time processing, data will be coming continuously and you have to keep getting data,process it and keep writing the output as well. And of course , all these have to be done parallel and fast in distributed fashion with no data loss. This is where Storm pitches in. Storm is fast: a benchmark clocked it at over a million tuples processed per second per node.

13.4.Architecture of Storm



Note 1:

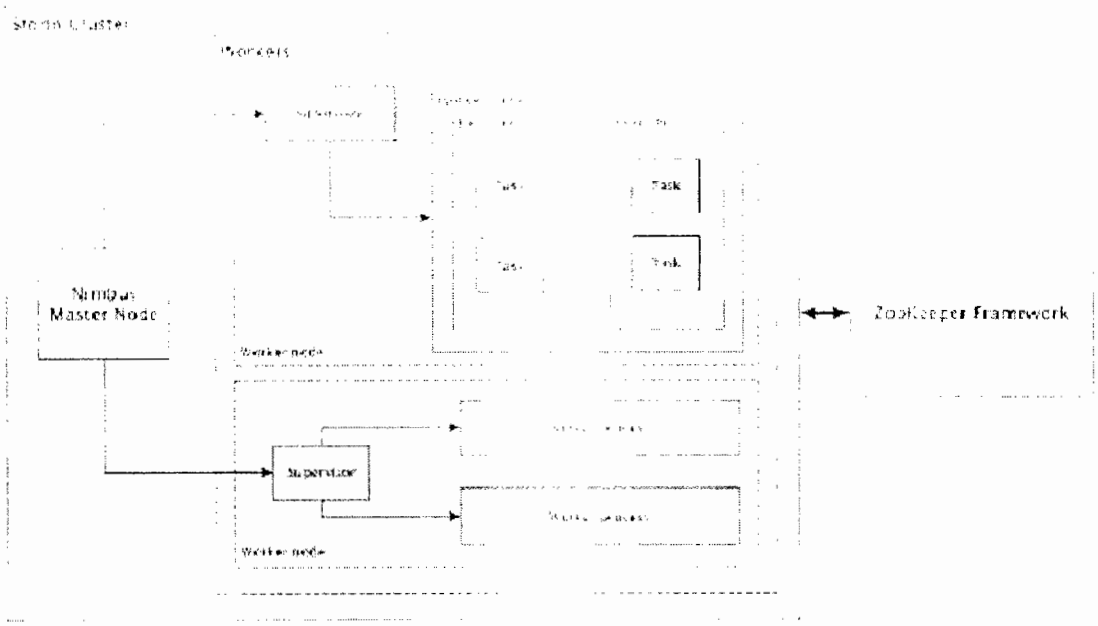
Apache Storm does not have its own state managing capabilities. Instead, It uses Apache ZooKeeper to manage the cluster state. All coordination between Nimbus and the Supervisors such as message acknowledgements, processing status etc is done through a Zookeeper cluster. Nimbus daemon and Supervisor daemons are stateless; all state is kept in Zookeeper or on local disk (property storm.local.dir) . This enables Storm to start right from where it left even after the restart. Even if we kill -9 Nimbus or the Supervisors , they'll start back like nothing happened.



Note 2:

Another important point to be aware of is that Storm makes use of zeromq library for inter process communication(between different worker processes) but after it was adopted as an apache project, storm developers replaced zeromq with Netty .

13.5.Cluster Architectural flow



13.6.Explanation of the Components

Components	Description
Nimbus	Nimbus is a master node of Storm cluster. All other nodes in the cluster are called as **worker nodes**. Master node is responsible for distributing data among all the worker nodes, assign tasks to worker nodes and monitoring failures.
Supervisor	The nodes that follow instructions given by the nimbus are called as Supervisors. A **supervisor** has multiple worker processes and it governs worker processes to complete the tasks assigned by the nimbus.
Worker process	A worker process will execute tasks related to a specific topology. A worker process will not run a task by itself, instead it creates **executors** and asks them to perform a particular task. A worker process will have multiple executors.
Executor	An executor is nothing but a single thread spawn by a worker process. An executor runs one or more tasks but only for a specific spout or bolt.
Task	A task performs actual data processing. So, it is either

a spout or a bolt.

ZooKeeper framework

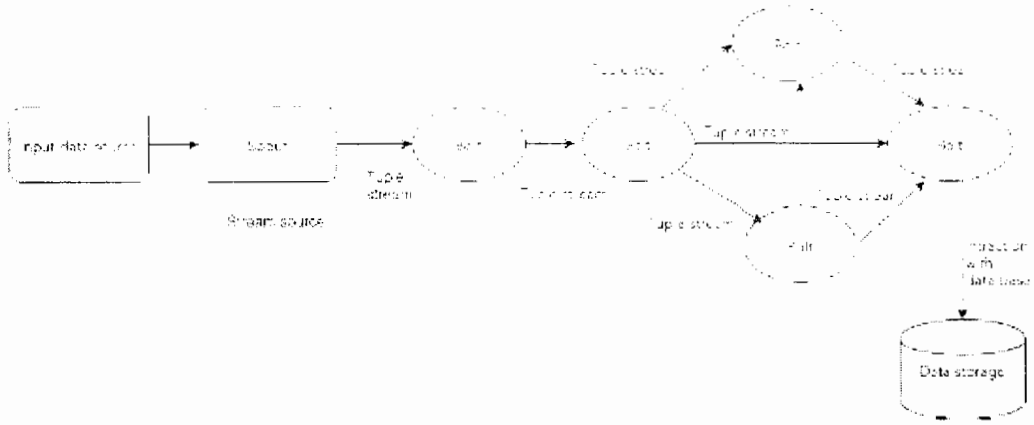
Apache ZooKeeper is a service used by a cluster (group of nodes) to coordinate between themselves and maintaining shared data with robust synchronization techniques. Nimbus is stateless, so it depends on ZooKeeper to monitor the working node status.

ZooKeeper helps the supervisor to interact with the nimbus. It is responsible to maintain the state of nimbus and supervisor.

13.7.Difference between Hadoop cluster and Storm cluster

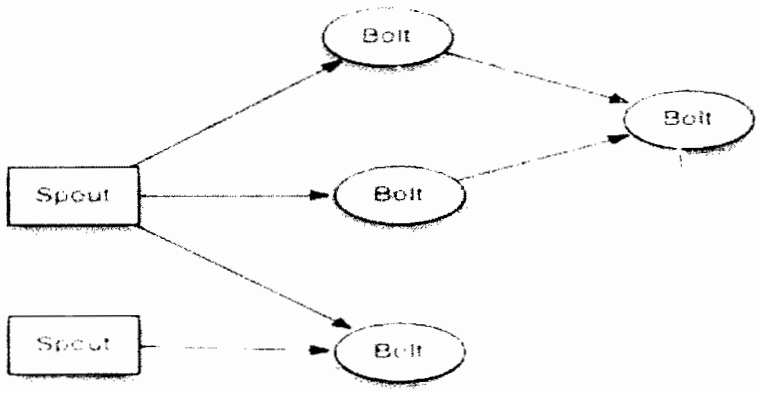
Hadoop Cluster	Storm Cluster
We run MR jobs	We run Topologies
MR job starts,processes and ends eventually	Topology once started,is intended to keep on processing live data forever which it keeps on getting from data sources like zmq,kafka,etc (until we wish to kill it)
Master Node runs a daemon call Job Tracker	Master Node runs a daemon called Nimbus
Hadoop MR job runs 2 tasks in a job 1 mapper task and 1 reducer task . This is restricting, you can only have 2 tasks and they have to be only mapper and reducer only. Not all problems can be solved using this MR paradigm.	Storm in contrast runs 2 tasks : Spouts and Bolts . In a topology, spout will act as data receiver from external sources and creator of Stream for bolts for actual processing. Bolts can be chained serially or in parallel depending on what kind of processing we want to do.

13.8.Internal Process of the data in the Apache Storm



13.9.Components of Storm

13.9.1. Topology



In simple words, Topology is a network of spouts and bolts as in above figure. It is analogous to a MR Job in Hadoop. It is a graph of computation consisting of spouts and bolts. Spouts as data stream source tasks and Bolts as actual processing tasks.
Each node in the graph contains some processing logic and links in the graph shows how the data will be passed and processing will happen among nodes.

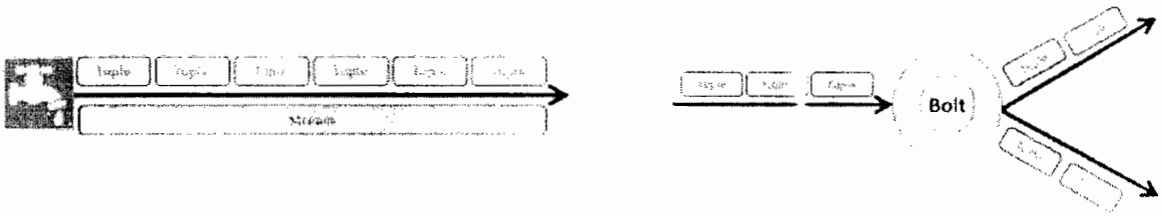
When a topology is submitted to a Storm cluster, Nimbus service on master node consults the supervisor services on different worker nodes and submits the topology. Each supervisor, creates one or more worker processes, each having its own separate jvm .

Each process runs within itself threads which we call Executors. The thread/executor processes the actual computational tasks : Spout or Bolt .

Topology is submitted to a storm cluster through a command :

storm jar storm-topology-code.jar chandan.storm.MyTopology arg1 arg2

13.9.2. Stream



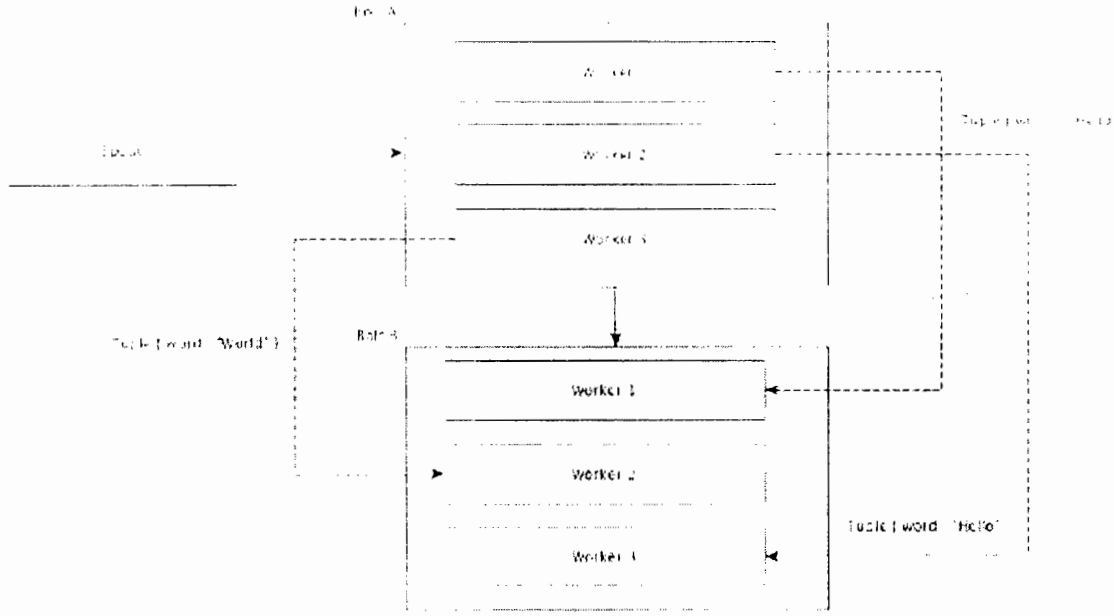
Stream is the core abstraction in Storm. A stream is an unbounded sequence of tuples (collection of key value pairs).Tuple is the most basic data structure in storm . It is a named list of values. Each field in the values can be an object of any serializable type.

Stream of data flows from spouts to bolts or from one bolt to another bolt. Stream grouping controls how the tuples are routed in the topology and helps us to understand the tuples flow in the topology .

There are four in-built stream groupings:

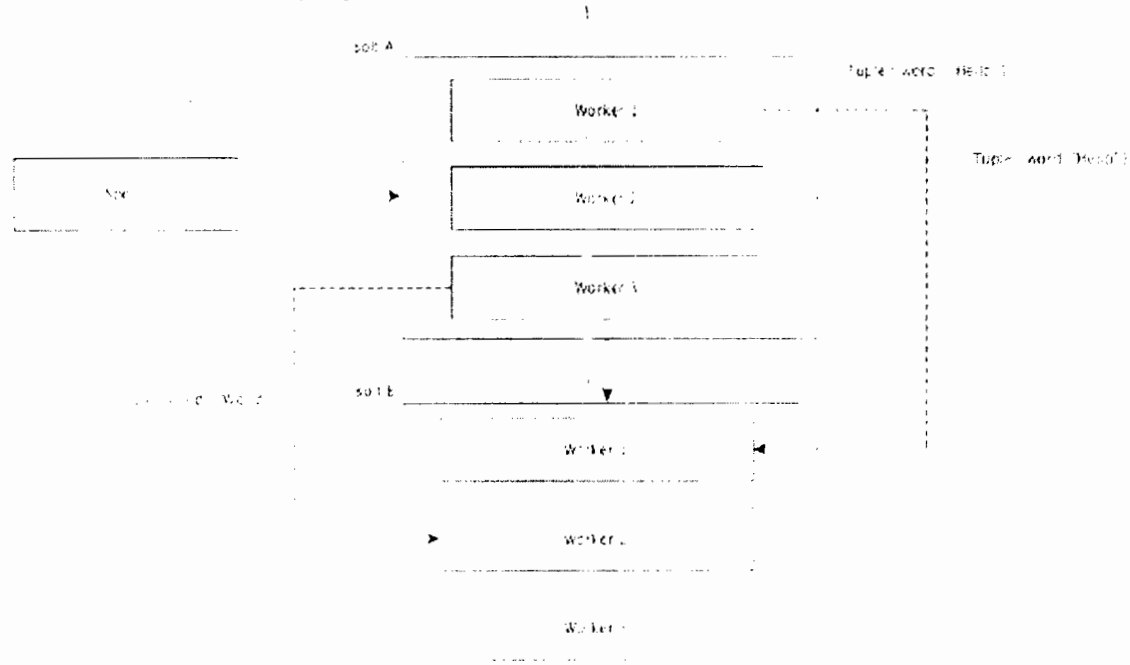
2.1 Shuffle Grouping

In shuffle grouping, an equal number of tuples is distributed randomly across all of the workers executing the bolts. The following diagram depicts the structure.



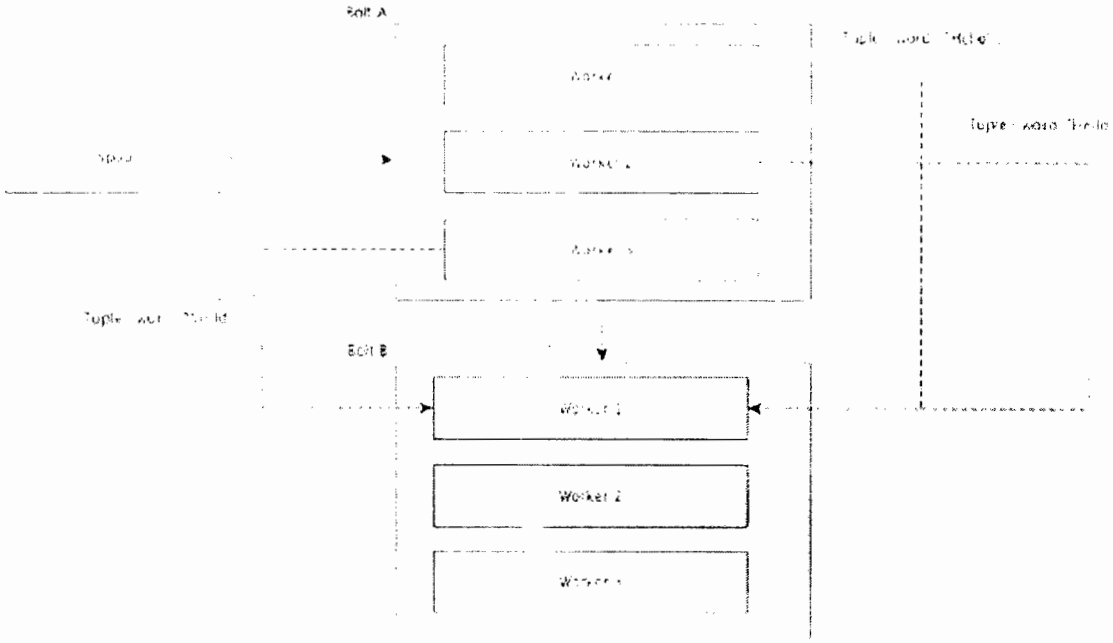
2.2 Field Grouping

The fields with same values in tuples are grouped together and the remaining tuples kept outside. Then, the tuples with the same field values are sent forward to the same worker executing the bolts. For example, if the stream is grouped by the field "word", then the tuples with the same string, "Hello" will move to the same worker. The following diagram shows how Field Grouping works.



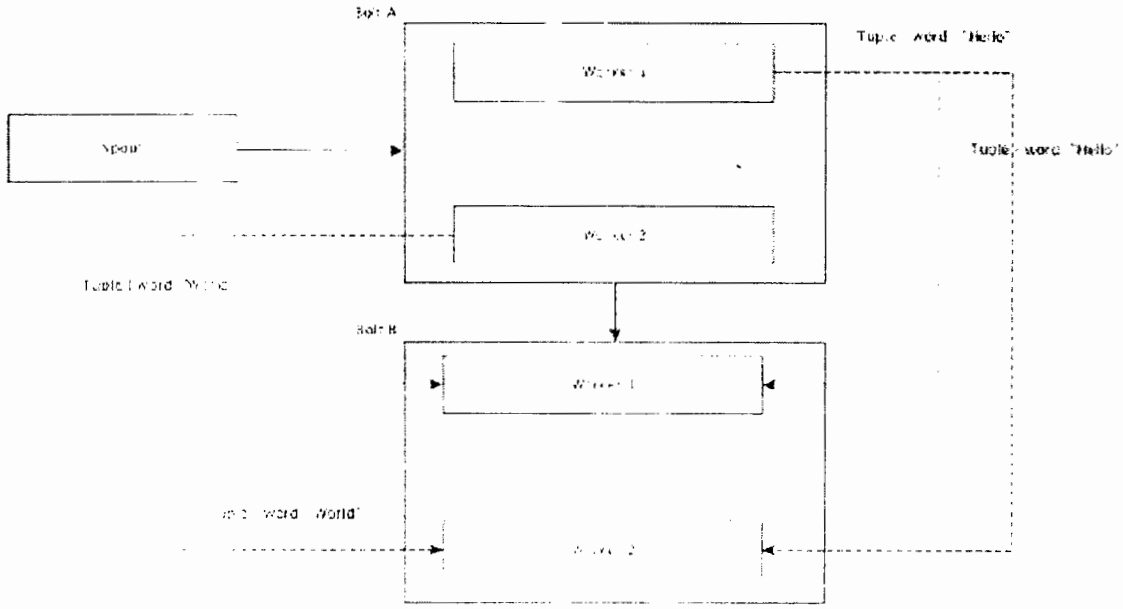
2.3 Global Grouping

All the streams can be grouped and forward to one bolt. This grouping sends tuples generated by all instances of the source to a single target instance (specifically, pick the worker with lowest ID).



2.4 All Grouping:

All Grouping sends a single copy of each tuple to all instances of the receiving bolt. This kind of grouping is used to send signals to bolts. All grouping is useful for join operations.



3.9.3.Spout

Spout is the entry point in a storm topology. It is the source of streams in the topology. A spout connects to the actual data source such as a message queue as Kafka , gets continuous data . converts the actual data into stream of tuples . emits them to bolts for actual processing. Spouts run as tasks in worker processes by Executor thread

3.9.4.Bolt

Bolt contains the actual processing logic. It works only on streams and can emit streams too for further processing downstream by other bolts or can export/save data for persistent storage. It receives stream from either one or more spouts or some other bolts. For example in simple word count example(see diagram above), map and reduce tasks will be executed as 2 different bolts executed in serial fashion. Bolts can do anything from run functions, filter tuples, do streaming aggregations, do streaming joins, talk to databases, and more.

13.10.Differences between Apache Spark, Apache Storm and Apache Kafka

Apache Spark	Apache Storm	Apache Kafka
Apache Spark is an in-memory distributed data analysis platform--primarily targeted at speeding up batch analysis jobs, iterative machine learning jobs, interactive query and graph processing.	**Apache Storm** is focused on stream processing or what some call complex event processing.	**Apache Kafka** is a log-based publish subscribe messaging system
Apache Spark operates on data at rest. Its real time nature is due to its ability to perform computations on data (RDD) in real time, these are still batch computations like Hadoop.	**Apache Storm** operates on data in motion (continuous stream of data). The real time nature is due to its ability to operate on streaming data (data flowing through a set of queries).	**Apache Kafka** gets the data from producers, stores it and allows consumers to retrieve it. It does not do any data processing.
Spark performs **Data-Parallel computations**	Storm performs **Task-Parallel computations.**	
	Apache Storm is a distributed real-time computation system for processing fast, large streams of data. Storm is an architecture based on master-workers paradigma.	Apache Kafka is a distributed publish-subscribe system for processing large amounts of streaming data..

KAFKA

14.1.Distributed Messaging Architecture: Apache Kafka

Apache Kafka is the messaging system originally developed at LinkedIn for processing LinkedIn s activity stream. Let look at the issues with the traditional messaging system which caused people to seek an alternative.

Any messaging system has three major components; the message producer, the message consumer, and the message broker. Message producers and message consumer use message queue for asynchronous inter-process communication. Any messaging supports

both point-to-point as well as publish/subscribe communication. In point-to-point paradigm, the producer of the messages sends the messages to a queue. There can be multiple consumers associated with the queue, but only one consumer can consume a given message from a queue. In publish/subscribe paradigm, there can be multiple producers producing messages for a given entity called Topic, and there can be multiple consumers subscribed for that topic. Each subscription receives the copy of each message sent for that topic. This differs from point-to-point communication, where one message is consumed by only one consumer. The message broker is the heart of the whole messaging system, which bridges the gap between the producer and consumer.

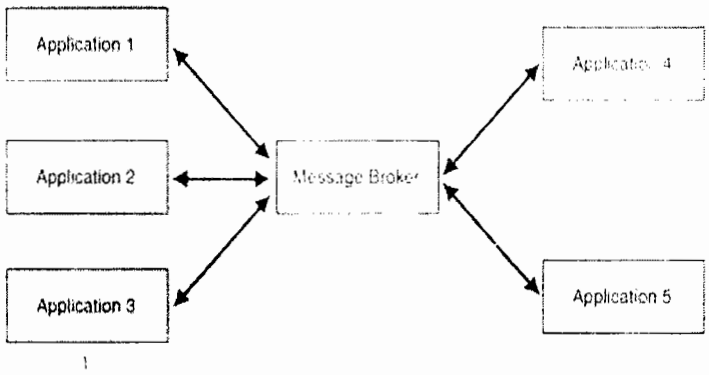


Figure 3

In Figure 3, the applications running either side of the broker can communicate with each other over point-to-point or publish/ subscribe way. The broker acts as a mediation layer between applications which provides some basic messaging properties, such as message durability and message persistency.

A "durable message" is a message where the broker will hold on to a message if the subscriber is temporarily unavailable. So the durability is defined by the relationship between a "Topic Subscriber" and the "Broker". Durability is applicable only to the publish/subscribe paradigm.

A "persistent message" is a message that defines the relationship between a "Message Producer" and the "Broker". This can be established for both point-to-point and publish/subscribe. This has to do with the guaranteed delivery of the message by persisting the message after it has been received from the message producer.

Both message durability and message persistency come with a cost. Keeping the state of the message (whether message is consumed or not) is a tricky problem. The traditional messaging brokers keep track of the consumer state. It uses metadata about the messages

and stores this metadata in broker. Storing metadata about billions of messages creates large overhead for broker. On top of that, the relational database for storing the message metadata does not scale very well. However, the broker tries to keep the metadata size small by deleting messages which are already consumed. The challenging problem arises about how Broker and Consumer conclude that a given message is consumed. Can a broker mark a message consumed as and when it put the message in the network for delivery? What will happen if the Consumer is down by the time the message reaches the consumer? To solve this, most messaging systems keep an acknowledgement system. When a message is delivered by a broker, it is marked as "sent", and when the consumer consumes the message and sends an acknowledgement, the broker marks it as "consumed". But what will happen if the consumer actually consumed the message, but a network failure occurred before the acknowledgement reached the broker? The broker will still keep the state as "sent" not "consumed". If the broker resends the message, the message will be consumed twice. The major problem arises around the performance of the broker. Now, the broker must keep track of each and every message. Imagine the overhead of the broker in cases when thousands of messages are being produced every second. This is a major reason why the traditional messaging system is not able to scale beyond a certain limit.

Another problem for traditional messaging systems is the storage data structure they use. Almost all messaging systems use Relational Database for storing messages. A relational database uses BTree for storing the indexes for faster retrieval. BTrees are the most versatile data structure available, and make it possible to support a wide variety of transactional and non-transactional semantics in the messaging system. But they come with a fairly high cost: BTrees operations are O (log N). Normally O (log N) is considered essentially equivalent to constant time, but this is not true for disk operations. Disk seeks come at 10 ms a pop, and each disk can perform only one seek at a time so parallelism is limited. Hence, even a handful of disk seeks lead to very high overhead. Furthermore, BTrees require a very sophisticated page or row-locking implementation to avoid locking the entire tree on each operation. The implementation must pay a high price for row-locking or else effectively serialize all reads

14.2.The following is how Kafka solved these problems

14.2.1.Messaging System: The Kafka Way

Figure 4 shows how different types of producers can communicate to different types of consumers through the Kafka Broker.

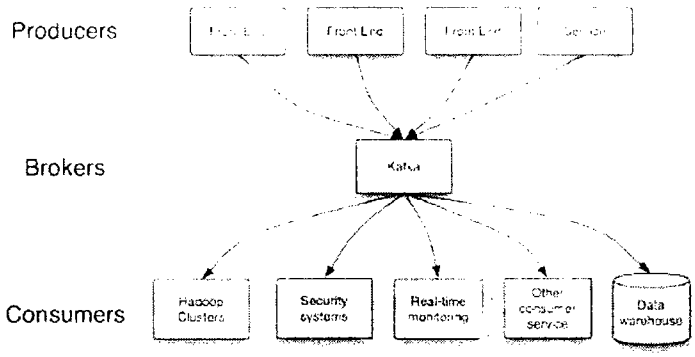
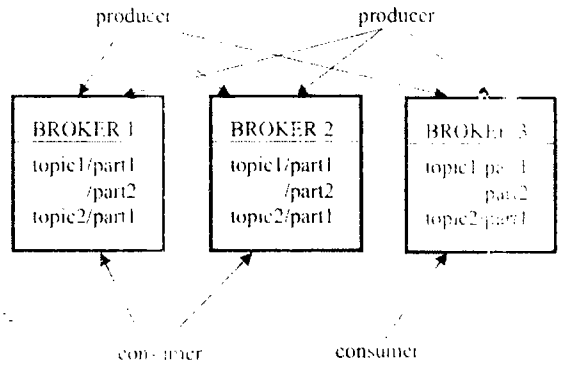


Figure 4

Kafka is explicitly distributed—producers, consumers, and brokers can all be run on a cluster of machines that co-operate as a logical group.

This happens fairly naturally for brokers and producers, but consumers require particular support. Each consumer process belongs to a consumer group and each message is delivered to exactly one process within every consumer group. Hence, a consumer group allows many processes or machines to logically act as a single consumer. The concept of consumer group is very powerful and can be used to support the semantics of either a queue or topic as found in JMS. To support queue semantics, we can put all consumers in a single consumer group, in which case each message will go to a single consumer. To support topic semantics, each consumer is put in its own consumer group, and then all consumers will receive each message. Kafka has the added benefit in the case of large data that no matter how many consumers a topic has, a message is stored only a single time.



The overall architecture of Kafka is shown in Figure 5. Kafka is distributed in nature; a Kafka cluster typically consists of multiple brokers. To balance load, a topic is divided into multiple partitions and each broker stores one or more of those partitions. Multiple producers and consumers can publish and retrieve messages at the same time[2]

Figure 5

Kafka relies heavily on the file system for storing and caching messages. There is a general perception that "disks are slow" which makes people skeptical that a persistent structure can offer competitive performance. In fact, disks are both much slower and much faster than people expect depending on how they are used; a properly designed disk structure can often be as fast as the network.

The key fact about disk performance is that the throughput of hard drives has been diverging from the latency of a disk seek for the last decade. As a result, the performance of linear writes on a 6 7200rpm SATA RAID-5 array is about 300MB/sec but the performance of random writes is only about 50k/sec—differences of nearly 10000X! These linear reads and writes are the most predictable of all usage patterns, and hence the one detected and optimized best by the operating system using read-ahead and write-behind techniques.

Kafka is designed around Operating System Page Cache. In computing, page cache—sometimes ambiguously called disk cache—is a "transparent" buffer of disk-backed pages kept in main memory (RAM) by the operating system for quicker access. Page cache is typically implemented in kernels with the paging memory management, and is completely transparent to applications. Any modern OS will happily divert *all* free memory to disk caching with little performance penalty when the memory is reclaimed. All disk reads and writes will go through this unified cache. If your disk usage favors linear reads, then read-ahead is effectively pre-populating this cache with useful data on each disk read. This suggests a design which is very simple: rather than maintain as much as possible in-memory and flush to the file system only when necessary, Kafka inverts that. All data is immediately written to a persistent log on the file system without any call to flush the data. In effect, this just means that it is transferred into the kernel's page cache where the OS can flush it later.

Kafka has a very simple storage layout. Each partition of a topic corresponds to a logical log. Physically, a log is implemented as a set of segment files of approximately the same size (e.g., 1GB). Every time a producer publishes a message to a partition, the broker simply appends the message to the last segment file. For better performance, Kafka flush the segment files to disk only after a configurable number of messages have been published or a certain amount of time has elapsed. A message is only exposed to the consumers after it is flushed.

Unlike typical messaging systems, a message stored in Kafka doesn't have an explicit message ID. Instead, each message is addressed by its logical offset in the log. This avoids the overhead of maintaining auxiliary, seek-intensive random-access index structures that map the message IDs to the actual message locations.

If the messaging system is designed around this kind of design of read ahead and write behind, how does Kafka support the Consumer State problem we defined earlier, i.e. how does Kafka keep track of which messages are being "sent" or "consumed"? Fact of the matter is, Kafka broker never keeps track of this. In Kafka, it is consumer which keeps track of the messages it consumed. Consumer is maintaining something like a watermark which tells which offset in the log segment is consumed. A consumer always consumes messages from a particular partition sequentially. If the consumer acknowledges a particular message offset, it implies that the consumer has received all messages prior to that offset in the partition. This provides flexibility to the consumer to consume older message by lowering the watermark. Typically, Consumer stores the state information in Zookeeper which is used for Distributed consensus service. Otherwise, Consumer can maintain this watermark level in any data structure it wishes, which depends on the Consumer. For instance, if Hadoop is consuming messages from Kafka, it can store the watermark value into HDFS.

Along with the architectural details mentioned above, Kafka also has many advanced configurations such as Topic Partitioning, Automatic Load Balancing, and so on.

This design of Kafka makes it highly scalable, able to process millions of messages per second. The producer of real time data can write messages into Kafka cluster, and the real time consumer can read the messages from Kafka cluster for parallel processing.

Offer Courses

Big Data hadoop Development & Administration

Spark & Scala

DATA SCIENCE | DEVOPS

Cloud Computing Salesforce CRM Development • Admin • Integration • Realtime Projects

(AWS) amazon web services

Windows Azure

python | tableau | QlikView | SAS | MSBI

ROBOTIC PROCESS AUTOMATION

Cyber Security

MuleSoft

(IoT) INTERNET of THINGS

DIGITAL MARKETING | talend | mongoDB cassandra