Module 2

Hadoop Distributed File System Basics

1. Explain the Hadoop Distributed File system design features?

- The Hadoop Distributed file system(HDFS) was designed **for Big Data processing**.
- Although capable of supporting many users simultaneously, HDFS is not designed as a true parallel file system. Rather, the design assumes a large file **write-once/read-many model**
- HDFS rigorously restricts data writing to one user at a time.
- Bytes are always appended to the end of a stream, and **byte streams** are guaranteed tobe

  stored in the order written
- The design of HDFS is based on the design of **the Google File System(GFS).**
- HDFS is designed for data streaming where **large amounts** of data are **read from disk in bulk**.
- **The HDFS block size is typically 64MB or 128MB**. Thus, this approach is unsuitable for standard POSIX file system use.
- Due to sequential nature of data, there is **no local caching mechanism**. The large block and file sizes makes it more efficient to reread data from HDFS than to try to cache the data.
- A principal design aspect of Hadoop MapReduce is the emphasis on moving the computation to the data rather than moving the data to the computation.
- In other high performance systems, a parallel file system will exist on hardware separate from computer hardware. Data is then moved to and from the computer components via high-speed interfaces to the parallel file system array.
- Finally, Hadoop clusters assume node failure will occur at some point. To deal with this situation, it has a redundant design that can tolerate system failure and still provide the data needed by the compute part of the program.
- The following points are important aspects of HDFS:
  - ❖ The **write-once/read-many design** is intended to facilitate streaming reads.
  - ❖ Files may be appended, but **random seeks are not permitted**. There is **no caching** of data.
  - ❖ Converged **data storage and processing** happen on the **same server nodes.**

- <**Moving computation is cheaper than moving data**.=
- A reliable file **system maintains multiple copies of data** across the cluster.

Consequently, **failure of a single will not bring down the file system**.

- ❖ A specialized file system is used, which **is not designed for general use**.

2. Discuss the various system roles in an HDFS components or deployment?

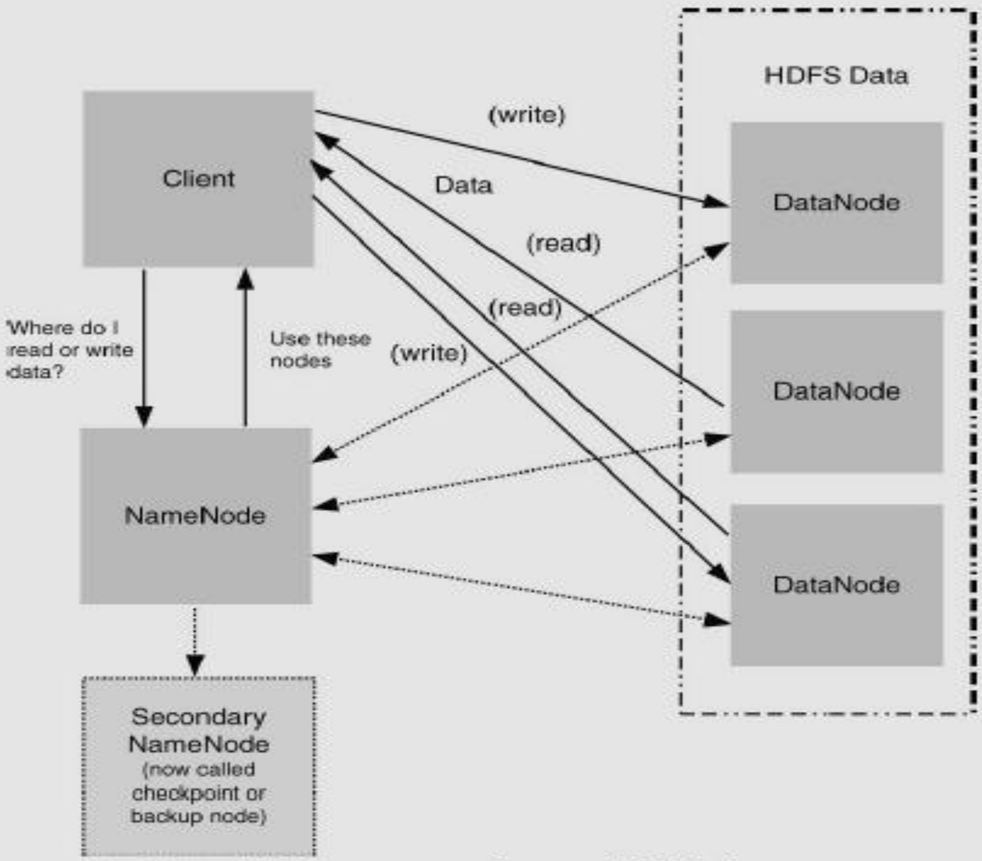

Figure 3.1 Various system roles in an HDFS deployment

- The design of HDFS is based on two types of nodes**: NameNode and multiple DataNodes.**
- In a basic design, **NameNode manages all the metadata** needed to store and **retrieve the actual data from the DataNode**s. No data is actually stored on the NameNode.
- The design is a **Master/Slave architecture** in which **master(NameNode)** manages the **file system namespace and regulates access to files by clients**.
- File system namespace operations such as opening, closing and renaming files and directories are all managed by the NameNode.
- The **NameNod**e also determines the **mapping of blocks to DataNodes** and **handles Data Node failures.**

- The **slave(DataNodes) are responsible for serving read and write requests** from the file system to the clients. The **NameNode manages block creation, deletion and replication**.
- When a client writes data, it first communicates with the NameNode and requests to create a file. The NameNode determines how many blocks are needed and provides the client with the DataNodes that will store the data.
- As part of the storage process, the data blocks are replicated after they are written to the assigned node.
- Depending on how many nodes are in the cluster, the NameNode will attempt to write replicas of the data blocks on nodes that are in other separate racks. If there is only one rack, then the replicated blocks are written to other servers in the same rack.
- After the Data Node acknowledges that the file block replication is complete, the client closes the file and informs the NameNode that the operation is complete.
- Note that the **NameNode does not write any data directly to the DataNodes**. It does, however, give the client a limited amount of time to complete the operation. **If it does not complete in the time period, the operation is cancelled.**
- The client requests a file from the NameNode, which returns the best DataNodes from which to read the data. The client then access the data directly from the DataNodes.
- Thus, once the metadata has been delivered to the client, the NameNode steps back and lets the conversation between the client and the DataNodes proceed. While data transfer is progressing, the NameNode also monitors the DataNodes by listening for heartbeats sent from DataNodes.
- The **lack of a heartbeat signal indicates a node failure**. Hence the NameNode will route around the failed Data Node and begin re-replicating the now-missing blocks.
- The mappings b/w data blocks and physical DataNodes are not kept in persistent storage on the NameNode. The NameNode stores all metadata in memory.
- In almost all Hadoop deployments, there is a SecondaryNameNode(Checkpoint Node). It is not an active failover node and cannot replace the primary NameNode in case of it failure.
- Thus the various important roles in HDFS are:
  - ❖ HDFS uses a **master/slave model** designed for large file reading or streaming.
  - ❖ The **NameNode** is a **metadata server or “Data traffic cop”.**

- ❖ HDFS provides a **single namespace** that is **managed by the NameNode**.
- ❖ **Data is redundant**ly stored **on DataNodes**; there is **no data on NameNode.**

SecondaryNameNode performs checkpoints of NameNode file system's state but is not a failover node.

3. Describe HDFS block replication.

- When HDFS writes a file, it is represented across the cluster. The **amount of replication** is based on the **value of dfs.replication in the hdfs-site.xml file**.

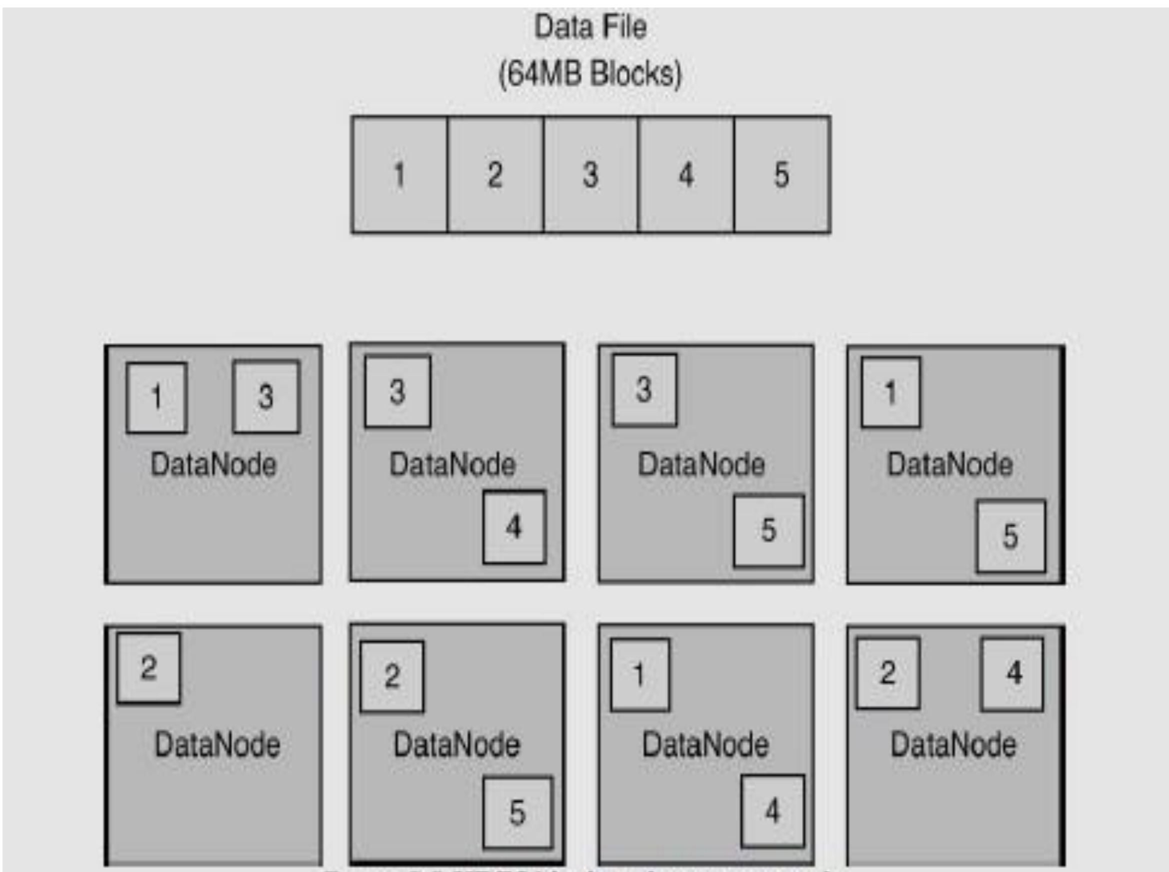

Figure 3.2 HDFS block replication example

- **Hadoop clusters** containing **more than eight DataNodes, the replication value is usually set to 3**. In a Hadoop cluster of **fewer DataNodes** but more than one DataNode , a **replication factor of 2** is adequate. For a single machine ,like pseudo-distributed the replication factor is set to 1.
- If several machines must be involved in the serving o a file ,then a file could be rendered unavailable by the loss of any one of those machines. HDFS solves this problem by replicating

each block across a number of machines.

- **HDFS default block size is often 64MB**.In a typical OS , the block size is 4KB or 8KB. However, if a 20KB file is written to HDFS, it will create a block that is approximately 20KB in size. If a file size 80MB is written to HDFS, a 64MB block and a 16MB block will be created.
- The HDFS blocks are based on size, while the splits are based on a logical partitioning of the data.
- For instance, if a file contains discrete records, logical split ensures that a record is not split physically across two separate servers during processing. Each HDFS block consist of one or more splits.
- The figure above provides an example of how a file is broken into blocks and replicated across the cluster. In this case replication factor of 3 ensures that any one DataNode can fail and the replicated blocks will be available on other nodes and subsequently re-replicated on other DataNodes.

4. Explain HDFS safe mode and rack awareness.

HDFS safe mode:

- When the NameNode starts, it enters a read-only safe node where blocks cannot be replicated
  or deleted.
- Safe Mode enables the NameNode to perform two important processes:
  - ❖ The previous **file system state is reconstructed** by loading the **fsimage file into memory and replaying the edit log**.
  - ❖ The **mapping between blocks and data nodes** is created by waiting for enough of the DataNodes to register so that at least one copy of the data is available. Not all DataNodes are required to register before HDFS exits from Safe Mode .The registration process may continue for some time.
- HDFS may also enter safe mode for maintenance using the HDFS **dfsadmin-safemode** command or when there is a file system issue that must be addressed by the administrator.

Rack Awareness:

- It deals with **data locality**.
- One of the main design goals Hadoop MapReduce is to move the computation to the data. Assuming that most data center networks do not offer full bisection bandwidth, a typical Hadoop cluster will exhibit three levels of data locality:

- ❖ Data resides on **the local machine(best).**
- ❖ Data resides in **the same rack(better**).
- ❖ Data resides in a **different rack(good).**

- When the YARN scheduler is assigning MapReduce containers to work as mappers, it will try to place the container first on the local machine, then on the same rack, and finally on another rack.

- In addition NameNode tries to replace replicated data blocks on multiple racks for improved fault tolerance. In such a case, an entire rack failure will not cause data loss or stop HDFS from working.

- **HDFS can be made rack-aware by using a user-derived script** that enables the master node to map the network topology of the cluster. **A default Hadoop installation assumes all the nodes belong to the same rack.**

5. Discuss the HDFS high availabilitydesign.

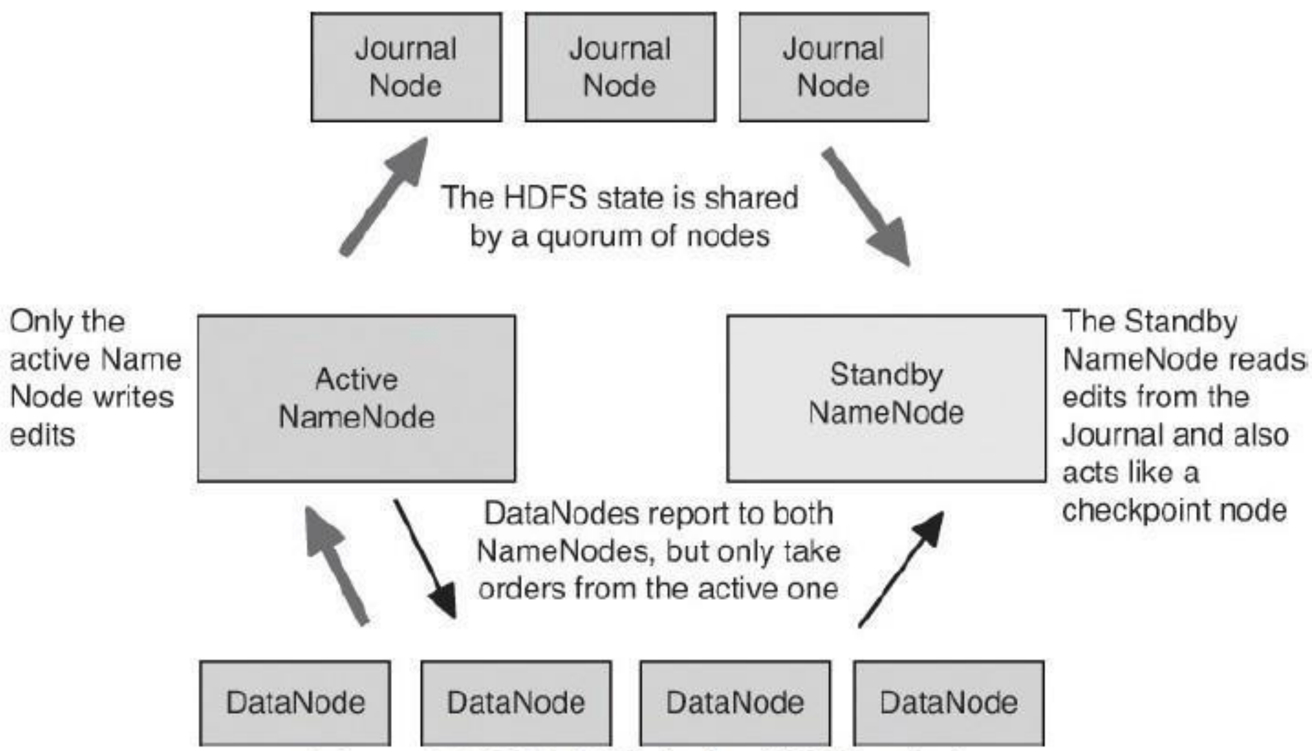

Figure 3.3 HDFS High Availability design

- High Availability(HA) hadoop cluster has two **(or more) separate NameNode** machines. Each machine is **configured with exactly the same software**.

- **One of the NameNode** machines is **in the Active state**, and the **other is in the Standby state**.
- **Active NameNode** is responsible for **all client HDFS operations** in the cluster. The **Standby NameNode** maintains enough **state to provide a fast failover** (if required).
- **To guarantee the file system state is preserved**, **both the Active and Standby Name Nodes receive block reports from the DataNodes**.
- The Active node also sends all file system edits to a quorum of Journal nodes. **At least three physically separate JournalNode daemons are required, because edit log modifications must be written to a majority of the JournalNodes**. This design will enable the system to tolerate the failure of a single JournalNode machine.
- The Standby node continuously reads the edits from the JournalNodes to ensure its namespace is synchronized with that of the Active node.
- In the event of an Active NameNode failure, the Standby node reads all remaining edits from the JournalNodes before promoting itself to the Active state.
- To prevent confusion between NameNodes, the JournalNodes allow only one NameNode to be a writer at a time.
- During failover, the NameNode that is chosen to become active takes over the role of writing to the JournalNodes.
- A **Secondary NameNode is not required in the HA configuration** because the **Standby node also performs the tasks of the Secondary NameNode.**
- **Apache Zookeeper is used to monitor the NameNode health**.
- Zookeeper is a highly available service for maintaining small amounts of coordination data, notifying clients of changes in that data, and monitoring clients for failures.
- **HDFS failover relies on Zookeeper for failure detection and for Standby to Active NameNode election.**

6. Explain the HDFS Name Node federation with example.
   - Another **important feature of HDFS** is NameNode Federation.
   - **Older versions of HDFS** provided a **single namespace for the entire cluster managed by a single NameNode.** Thus, the resources of a single NameNode determined the size of the namespace.

- **Federation addresses this limitation** by adding **support for multiple NameNodes /** namespaces to the HDFS file system.
- **The key benefits are as follows:**
  - ❖ ***Namespace scalability***: HDFS cluster storage scales horizontally without placing a burden on the NameNode.
  - ❖ ***Better performance***: Adding more NameNodes to the cluster scales the file system read/write operations throughput by separating the total namespace.
  - ❖ ***System isolation***: Multiple NameNodes enable different categories of applications to be distinguished, and users can be isolated to different namespaces.
- In Fig 3.4 NameNode1 manages the /research and /marketing namespaces, and NameNode2 manages the /data and /project namespaces.
- The NameNodes do not communicate with each other and the DataNodes <just store data block= as directed by either NameNode.

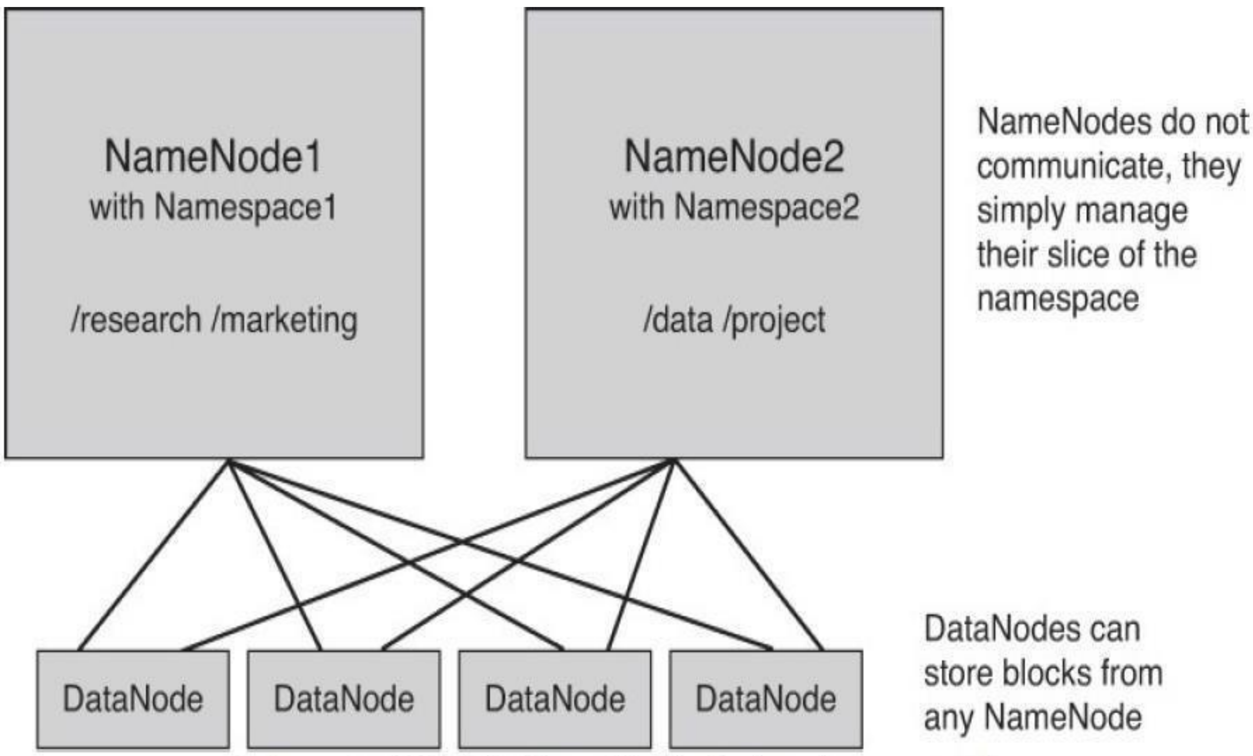

Figure 3.4 HDFS NameNode Federation example

7. Discuss various HDFS user commands.

- **List Files in HDFS**

- To list the files in the root HDFS directory, enter the following command: Syntax: $ hdfs dfs -ls /

  Output:

  Found 2 items
  drwxrwxrwx - yarn hadoop 0 2015-04-29 16:52 /app-logs
  drwxr-xr-x - hdfs hdfs 0 2015-04-21 14:28 /apps

- To list files in your home directory, enter the following command:
  Syntax: $ hdfs dfs -ls

  Output:

  Found 2 items
  drwxr-xr-x - hdfs hdfs 0 2015-05-24 20:06 bin
  drwxr-xr-x - hdfs hdfs 0 2015-04-29 16:52 examples

- **Make a Directory in HDFS**
  - To make a directory in HDFS, use the following command. As with the -ls command, when no path is supplied, the user's home directory is used

    Syntax: $ hdfs dfs -mkdir stuff

- **Copy Files to HDFS**
  - To copy a file from your current local directory into HDFS, use the following command. If a full path is not supplied, your home directory is assumed. In this case, the file test is placed in the directory stuff that was created previously.

    Syntax: $ hdfs dfs -put test stuff

  - The file transfer can be confirmed by using the -ls command: Syntax: $ hdfs dfs -ls stuff

    Output:

    Found 1 items
    -rw-r--r-- 2 hdfs hdfs 12857 2015-05-29 13:12 stuff/test

- **Copy Files from HDFS**
  - Files can be copied back to your local file system using the following command.
  - In this case, the file we copied into HDFS, test, will be copied back to the current local directory with the name test-local.

    Syntax: $ hdfs dfs -get stuff/test test-local

- **Copy Files within HDFS**

- The following command will copy a file in HDFS:
  Syntax: $ hdfs dfs -cp stuff/test test.hdfs

- **Delete a File within HDFS**
  - The following command will delete the HDFS file test.dhfs that was
    Syntax: $ hdfs dfs -rm test.hdfs

8. Explain MapReduce model

- **Hadoop** version 2 maintained the **MapReduce capability** and also made other processing models available to users. Virtually all the **tools** developed for Hadoop, such as **Pig and Hive**, will **work seamlessly on top of the Hadoop** version 2 MapReduce.
- There are **two stages: a mapping stage and a reducing stage**. In the **mapping stage**, a mapping procedure is applied **to input data**. The map is usually some kind of filter or **sorting process.**
- The **mapper inputs a text file** and then **outputs data in a (key, value**) pair (token-name,count) format.
- The **reducer script takes these key–value pairs** and **combines the similar tokens** and counts the total number of instances. The result is a new key–value pair (token-name, sum).
- Simple Mapper Script

```
#!/bin/bash
while read line ; do
for token in $line; do
if [ "$token" = "Kutuzov" ] ; then echo
"Kutuzov,1"
elif [ "$token" = "Petersburg" ] ; then
echo "Petersburg,1"
fi
done
done
```

- Simple Reducer Script

```
#!/bin/bas
h kcount=0
pcount=0
while read line ; do
```

```
if [ "$line" = "Kutuzov,1" ] ; then
let kcount=kcount+1
elif [ "$line" = "Petersburg,1" ] ;then
let pcount=pcount+1
fi
done
echo "Kutuzov,$kcount"
echo "Petersburg,$pcount"
```

- The mapper and reducer functions are both defined with respect to data structured in (key,value) pairs. The mapper takes one pair of data with a type in one data domain, and returns a list of pairs in a different domain:

  Map(key1,value1) → list(key2,value2)
- The reducer function is then applied to each key–value pair, which in turn produces a collection of values in the same domain:

  Reduce(key2, list (value2)) → list(value3)

  Each reducer call typically produces either one value (value3) or an empty response.

- Thus, the MapReduce framework transforms a list of (key, value) pairs into a list of values.
- The MapReduce model is inspired by the map and reduce functions commonly used inmany functional programming languages.
- The functional nature of MapReduce has some important properties:
  - **Data flow is in one direction** (map to reduce). It is possible to use the output of a reduce step as the input to another MapReduce process.
  - As with functional programing, **the input data are not changed**. By applying the mapping and reduction functions to the input data**, new data are produced**.
  - Because there is no dependency on how the mapping and reducing functions are applied to the data, the mapper and reducer data flow can be **implemented in any number of ways** to provide **better performance**.
- In general, the **mapper process** is **fully scalable** and can be applied to any subset of the input data. Results from multiple parallel mapping functions are then combined in the reducer phase.
- Hadoop accomplishes parallelism by using a distributed file system (HDFS) to slice and **spread data over multiple servers.**

9. Elaborate the Apache Hadoop Parallel MapReduce data flow.

- The programmer must provide a mapping function and a reducing function. Operationally, however, the Apache Hadoop parallel MapReduce data flow can be quite complex.
- Parallel execution of MapReduce requires other steps in addition to the mapper and reducer processes.
- The basic steps are as follows:

1. Input Splits *:*

- ❖ **HDFS distributes and replicates data over multiple servers**. The default data chunk or block size is 64MB. Thus, a **500MB file would be broken into 8 blocks and written to different machines in the cluster**.
- ❖ The data are also replicated on multiple machines (typically three machines). **These data slices are physical boundaries** determined by HDFS and have nothing to do with the data in the file. Also,while not considered part of the MapReduce process, the time required to load and distribute data throughout HDFS servers can be considered part of the total processing time.
- ❖ **The input splits** used by MapReduce **are logical boundaries** based on the input data.

1. Map Step :

- ❖ The mapping process is where the parallel nature of Hadoop comes into play. For large amounts of data, **many mappers** can be operating **at the same time**.
- ❖ The user provides the specific mapping process. **MapReduc**e will try to execute the **mapper on the machines** where the **block resides**. Because the file is replicated in HDFS, the least busy node with the data will be chosen.
- ❖ If all nodes holding the data are too busy, MapReduce will try to pick a node that is closest to the node that hosts the data block (a characteristic called rack-awareness). The last choice is any node in the cluster that has access to HDFS.

3. Combiner Step :

- ❖ It is possible **to provide an optimization or pre-reduction** as part of the map stage where key–value pairs are combined prior to the next stage. The combiner stage is optional.

4. Shuffle Step :

- ❖ Before the parallel reduction stage can complete, **all similar keys must be combined** and **counted by the same reducer process.**
- ❖ Therefore, **results of the map** stage must be collected by key–value pairs and shuffled to the **same reducer process.**
- ❖ If only a single reducer process is used, the shuffle stage is not needed.

5. Reduce Step :

- ❖ The final step is the actual reduction. In this stage, the **data reduction** is performed as **per the programmer's design**.
- ❖ The **reduce step is also optional**. The results are written to HDFS. Each reducer will write an output file.
- ❖ For example, a MapReduce job running four reducers will create files called part-0000, part-0001, part-0002, and part-0003.

- Figure 5.1 is an example of a simple Hadoop MapReduce data flow for a word count program. The map process counts the words in the split, and the reduce process calculates the total for each word. Where as, the actual computation of the map and reduce stages are up to the programmer.
- The input to the MapReduce application is the following file in HDFS with three lines of text. The goal is to count the number of times each word is used.

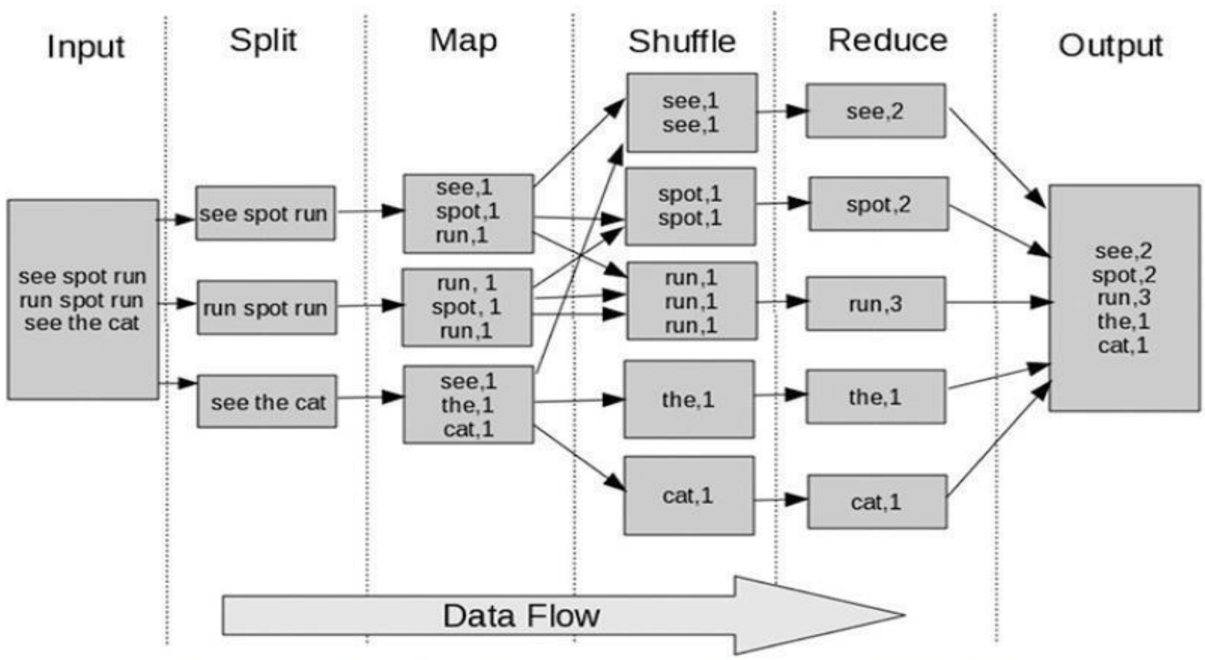

Figure 5.1 Apache Hadoop parallel MapReduce data flow

- The first thing MapReduce will do is create the data splits. For simplicity, each line will be one split. Since each split will require a map task, there are three mapper processes that count the number of words in the split.
- On a cluster, the results of each map task are written to local disk and not to HDFS. Next, similar keys need to be collected and sent to a reducer process. The shuffle step requires data movement and can be expensive in terms of processing time.
- Depending on the nature of the application, the amount of data that must be shuffled throughout the cluster can vary from small to large.
- Once the data have been collected and sorted by key, the reduction step can begin. In some cases, a single reducer will provide adequate performance. In other cases, multiple reducers may be required to speed up the reduce phase. The number of reducers is a tunable option for many applications. The final step is to write the output to HDFS.
- A combiner step enables some pre-reduction of the map output data. For instance, in the previous example, one map produced the following counts:

  (run,1)

  (spot,1)

  (run,1).

10. Design MapReduce WordCount Program.

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

public class WordCount {
        public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable>{
                private final static IntWritable one = new IntWritable(1);
                private Text word = new Text();

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

public static void main(String[] args) throws Exception { Configuration
        conf = new Configuration();
        String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
        if (otherArgs.length != 2) {
          System.err.println("Usage: wordcount <in> <out>");
          System.exit(2);
        }
        Job job = new Job(conf, "word count");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class);
        job.setReducerClass(IntSumReducer.class);
        job.setOutputKeyClass(Text.class);
```

```
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
        FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

For Compilation using Hadoop v3.1.1:

Step1: Compile

$ javac -classpath word_count.java

Step2: To create JAR file

$ jar -cvf wordcount.jar *.class

Step3: Next place that new jar file inside /usr/local/hadoop

$ sudo cp wordcount.jar /usr/local/hadoop/

Step 4: Switch to hduser and go to Hadoop Home

$ su hduser

$ cd /usr/local/hadoop/

Step5: To start namenode, datanode etc command is

$ sbin/start-all.sh

Step6: To check whether they started or not

$ jps

Step7: Copy that input folder into hadoop distributed file system.

$ hdfs dfs -copyFromLocal /home/hp/wordcount_input.txt

/user/hduser/ Step 8: To verify it's there

$ bin/hadoop dfs -ls /user/hduser/

Step9: Now we can run our program in hadoop:

```
$ bin/hadoop jar wordcount.jar WordCount /user/hduser/wordcount_input.txt/user/hduser
/word_output
```

Step10: To display the output on command promt:

```
$ hdfs dfs -cat /user/hduser/word_output/part-r-00000
```

Step11: To stop namenode, datanode etc command is:

```
$ sbin/stop-all.sh
```

Input in wordcount_input.txt:

do as i say not as i do

Final Output:

```
as     2
do     2
i      2
not    1
say    1
```

Essential Hadoop Tools

1 Discuss the usage of Apache pig.

- **Apache Pig** is a high-level language that enables programmers to write complex Map Reduce transformations using a simplescripting language.
- Pig's simple SQL-like scripting language is called **Pig Latin**, and appeals to developers already familiar with scriptinglanguages and SQL.
- Pig Latin (the actual language) defines a set of transformations on a data set such as aggregate, join, and sort.
- Pig is often used to extract, transform, and load (ETL) data pipelines, quickresearch on raw data.
- Apache Pig has several usage modes. The first is a **local mode** in which all processing is done on thelocal machine. The **non-local (cluster) modes** are Map Reduce and Tez.
- These modes execute the job on the cluster using either the Map Reduce engine or the optimized Tez engine.
- There are also interactive and batch modes available; they enable Pig applications to be developed locally in interactive modes, using small amounts of data, and then run at scale on the cluster in a production mode. The modes are in below fig.

	Local Mode	Tez Local Mode	MapReduce Mode	Tez Mode
Interactive Mode	Yes	Experimental	Yes	Yes
Batch Mode	Yes	Experimental	Yes	Yes

Table 7.1 **Apache Pig Usage Modes**

Pig Example Walk-Through:

- Working knowledge of Pig through the hand-on experience of creating pig scripts to carry out essential data operations and tasks.
- Apache Pig is also installed as part of the Horton works HDP Sandbox.
- In this simple example, Pig is used to extract user names from the **/etc/passwd file.**
- The following example assumes the user is hdfs , but any valid user with access to HDFScan run the example.
- To begin first, copy the passwd file to a working directory for local Pig operation: **$cp/etc/passwd.**
- Next, copy the data file into HDFS for Hadoop Map Reduceoperation: $
  - **hdfs dfs –put passwd passwd.**
- To confirm the file is in HDFS by entering the following command:
  - **hdfs dfs –ls passwd**
  - **-rw-r--r- 2 hdfs hdfs 2526 2015-03-17 11:08 passwd.**
- In local Pig operation, all processing is done on the local machine (Hadoop is not used). First, the interactive command linestarted: **$ pig -x local.**
- If Pig starts correctly, you will seea **grunt>** prompt.
- And also see a bunch of INFO messages. Next, enter the commands to load thepasswd file and then grab the user name and dump it to the terminal.
- Pig commands must end with a semicolon (;).
  - **grunt> A = load 'passwd' using Pig Storage(':');**
  - **grunt> B = foreach A generate $0 asid;**
  - **grunt> dumpB;**
- The processing will start and a list of user names will be printed to the screen.

- To exit the interactive session, enter the command quit.
  - **$ grunt> quit.**

2 Explain Apache Sqoop to Acquire Relational data with an example.

- Sqoop is a tool designed to transfer data between Hadoop and relational databases.
- Sqoop is used to
  -import data from a relational database management system (RDBMS) into the Hadoop Distributed File System(HDFS),
  - transform the data in Hadoop and
  - export the data back into an RDBMS.

Sqoop import method :

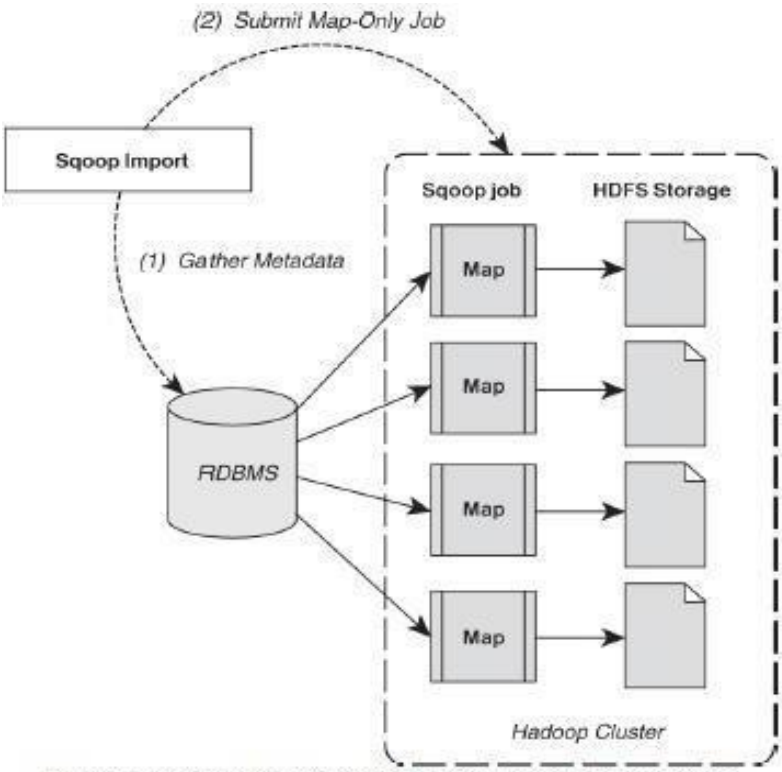

Figure 7.1 Two-step Apache Sqoop data import method (Adapted from Apache Sqoop Documentation)

The data import is done in two steps :

1) Sqoop examines the database to gather the necessary metadata for the data to be imported.
2) Map-only Hadoop job : Transfers the actual data using the metadata.

- The imported data are saved in an HDFS directory.
- Sqoop will use the database name for the directory, or the user can specify any alternative directory where the files should be populated. By default, these files contain comma delimited fields, with new lines separating different records.

Sqoop Export method :

Data export from the cluster works in a similar fashion. The export is done in two steps :

1) examine the database for metadata.
2) Map-only Hadoop job to write the data to the database.

Sqoop divides the input data set into splits, then uses individual map tasks to push the splits to the database.

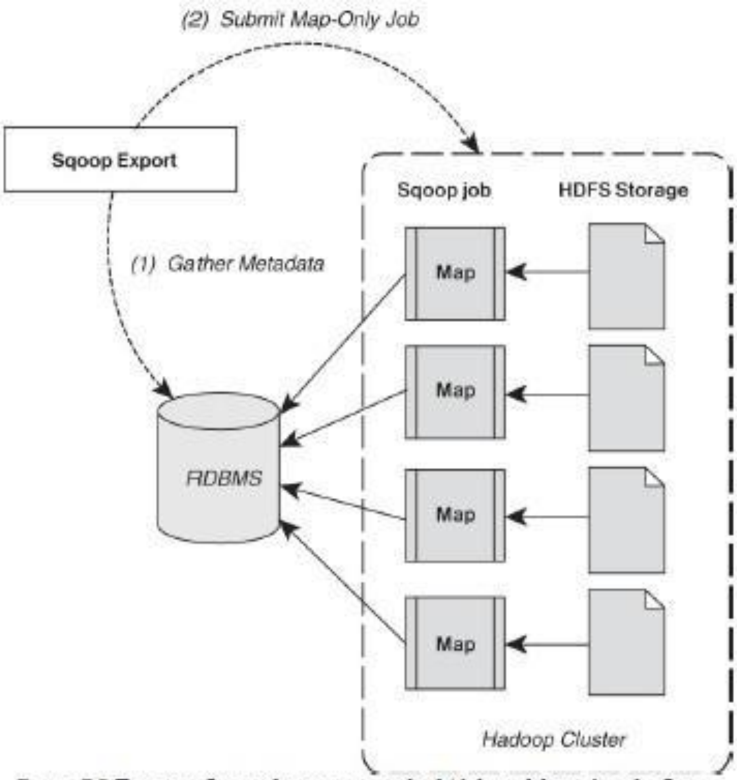

Figure 7.2 Two-step Sqoop data export method (Adapted from Apache Sqoop

Example: The following example shows the use of sqoop:

Steps:

1. Download Sqoop.
2. Download and load sample MySQL data.
3. Add Sqoop user permissions for the local machine and cluster.
4. Import data from MySQL to HDFS.
5. Export data from HDFS to MySQL.

Step 1: Download Sqoop and Load Sample MySQL Database

To install sqoop,

```
# yum install sqoop sqoop-metastore
```

To download database,

```
$ wget http : //downloads.mysql.com/docs/world_innodb.sql.gz
```

Step 2: Add Sqoop User Permissions for the Local Machine and Cluster.

In MySQL, add the following privileges for user sqoop to MySQL.

```
mysql> GRANT ALL PRIVILEGES ON world.* To 'sqoop'@'limulus'
IDENTIFIED BY 'sqoop';
mysql> GRANT ALL PRIVILEGES ON world.* To 'sqoop'@'10.0.0.%'
IDENTIFIED BY 'sqoop';
mysql> quit
```

Step 3: Import Data Using Sqoop

To import data, we need to make a directory in HDFS:

```
$ hdfs dfs -mkdir sqoop-mysql-import
```

The following command imports the Country table into HDFS. The option -table signifies the table to import, --target-dir is the directory created previously, and -m 1 tells Sqoop to use one map task to import the data.

```
$ sqoop import --connect jdbc:mysql://limulus/world --username sqoop
--password sqoop --table Country -m 1 --target-dir
/user/hdfs/sqoopmysql- import/country
```

The file can be viewed using the hdfs dfs -cat command:

Step 4: Export Data from HDFS to MySQL

Sqoop can also be used to export data from HDFS. The first step is to create tables for exported data.

Then use the following command to export the cities data into MySQL:

```
sqoop --options-file cities-export-options.txt --table CityExport --
```

staging-table CityExportStaging --clear-staging-table -m 4 --exportdir /user/hdfs/sqoop-mysql-import/city

3. Discuss Apache Flume to acquire data streams

- Apache Flume is an independent agent designed to collect, transport, and store data into HDFS.
- Data transport involves a number of Flume agents that may traverse a series of machines and locations.
- Flume is often used for log files, social media-generated data, email messages, and just about any continuous data source.

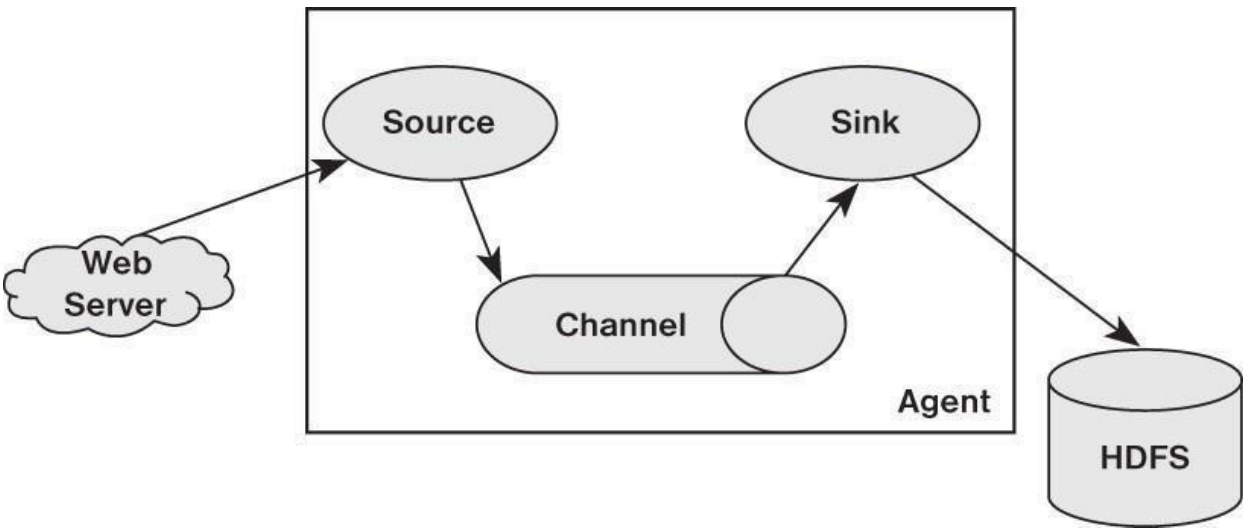

Figure 7.1 Flume agent with source, channel, and sink

- Flame agent is composed of three components.
  - Source: The source component receives data and sends it to a channel. It can send the data tomore than one channel.

- ○ Channel: A channel is a data queue that forwards the source data to the sink destination.
- ○ Sink: The sink delivers data to destination such as HDFS, a local file, or another Flume agent.

- A Flume agent must have all three of these components defined. Flume agent can have several source, channels, and sinks.
- Source can write to multiple channels, but a sink can take data from only a single channel.
- Data written to a channel remain in the channel until a sink removes the data.
- By default, the data in a channel are kept in memory but may be optionally stored on disk to prevent data loss in the event of a network failure.

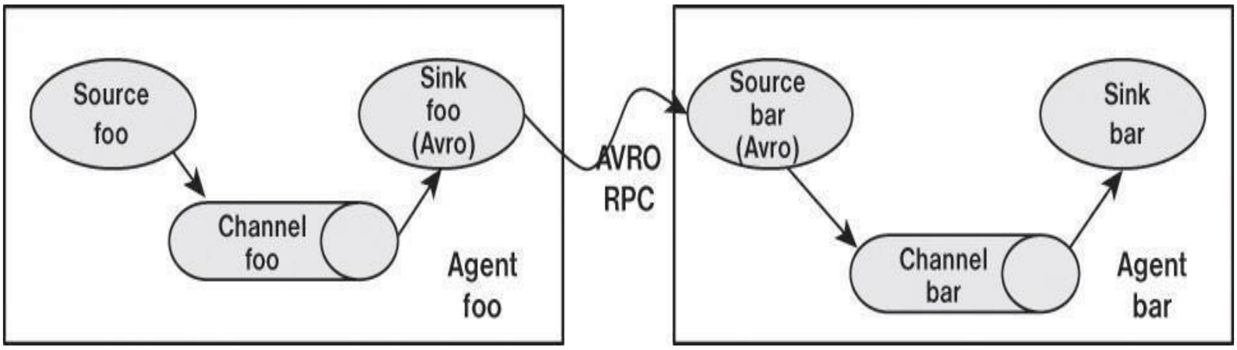

Figure 7.2 Pipeline created by connecting Flume agents

- As shown in the above figure, Sqoopagents may be placed in a pipeline, possibly to traverse several machines or domains.
- In this Flume pipeline, the sink from one agent is connected to the source of another.
- The data transfer normally used by Flume, which is called Apache Avro.

- Avro is a data serialization/deserialization system that uses a compact binary format.
- The scheme is sent as part of the data exchange and is defined using JSON.
- Avro also uses remote procedure calls (RPCs) to send data.

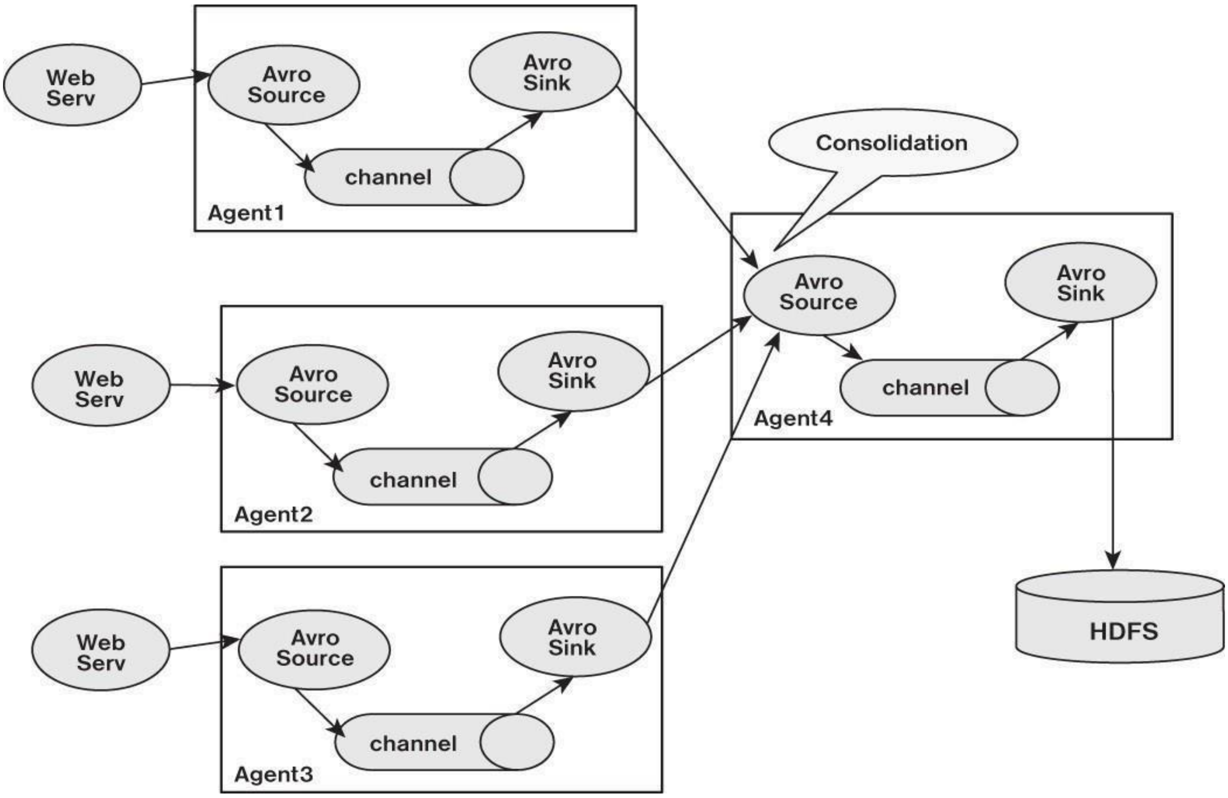

Figure 7.3 A Flume consolidation network.

4 Demonstrate the working of Hive with Hadoop

- Apache Hive is a data warehouse infrastructure built on top of Hadoop for providing data summarization, ad hoc queries, and the analysis of large data sets using a SQL-like language called HiveQL.
- Hive is considered the de facto standard for interactive SQL queries over petabytes of data using Hadoop.

- Some essential features:
- Tools to enable easy data extraction, transformation, and loading (ETL)
- A mechanism to impose structure on a variety of data formats
- Access to files stored either directly in HDFS or in other data storage systems such as HBase
- Query execution via MapReduce and Tez (optimized MapReduce)
- Hive is also installed as part of the Hortonworks HDP Sandbox.
- To work in Hive with Hadoop, user with access to HDFS can run the Hive queries.
- Simply enter the hive command. If Hive start correctly,it get a hive> prompt.

```
$ hive
(some messages may show up here)
hive>
```

- Hive command to create and drop the table. That Hive commands must end with a semicolon (;).

```
hive> CREATE TABLE pokes (foo INT, bar STRING); OK
Time taken: 1.705 seconds
```

- To see the table is created,

```
hive> SHOW TABLES;
OK
pokes
```

Time taken: 0.174 seconds, Fetched: 1 row(s)

- To drop the table,

  hive> DROP TABLE pokes;

  OK

  Time taken: 4.038 seconds

- The first step is to Creation of table can be developed using a web server log file:

  hive> CREATE TABLE logs(t1 string, t2 string, t3 string, t4 string, t5 string, t6 string, t7 string) ROW FORMAT DELIMITED FIELDS TERMINATED BY ' ';

- Next, to load the data from the sample.log file, the file is found in the local directory and not in HDFS.

  hive> LOAD DATA LOCAL INPATH 'sample.log' OVERWRITE INTO TABLE logs;

- Finally, the select step that this invokes a Hadoop MapReduce operation. The results appear at the end of the output.
  - hive> SELECT t4 AS sev, COUNT(*) AS cnt FROM logs WHERE t4 LIKE '[%'
  - GROUP BY t4;
  - Total jobs = 1
  - 2015-03-27 13:00:17,399 Stage-1 map = 0%, reduce = 0%
  - 2015-03-27 13:00:26,100 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 2.14 sec
  - 2015-03-27 13:00:34,979 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 4.07 sec

- Total MapReduce CPU Time Spent: 4 seconds 70 msec
- OK
- [DEBUG] 434
- [ERROR] 3
- [FATAL] 1
- [INFO] 96
- [TRACE] 816
- [WARN] 4
- Time taken: 32.624 seconds, Fetched: 6 row(s)

- To exit Hive, simply type exit;
  - hive> exit;

5. Explain yarn application framework with a neat diagram?

YARN presents a resource management platform, which provides services such as scheduling, fault monitoring, data locality, and more to MapReduce and other frameworks. Below figure illustrates some of the various frameworks that will run under YARN.

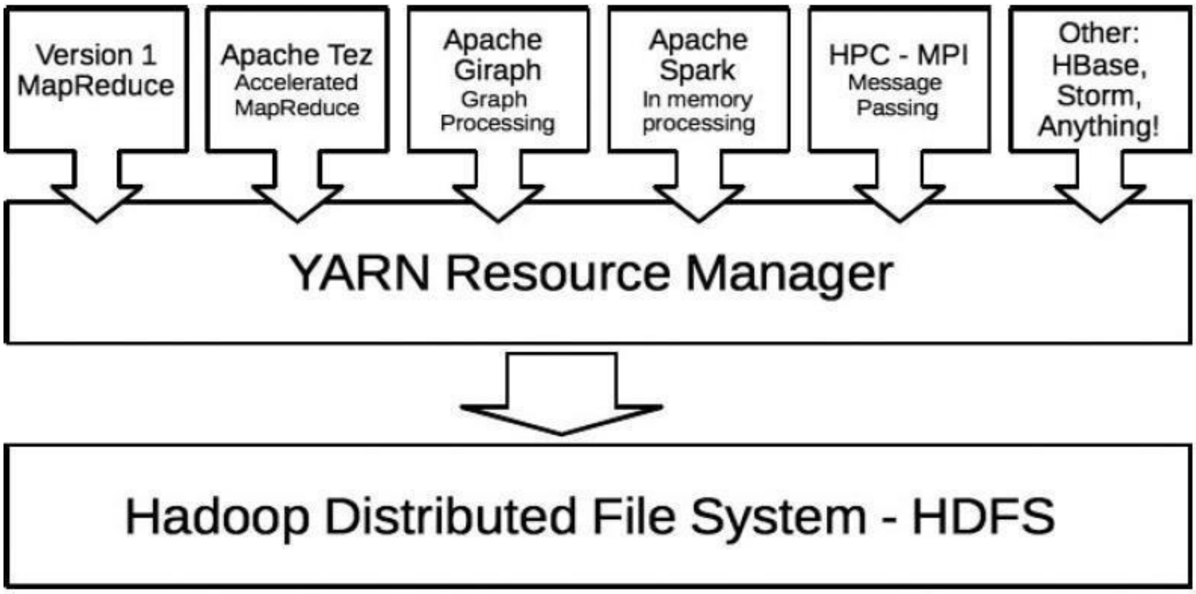

Figure 8.2 Example of the Hadoop version 2 ecosystem. Hadoop version 1 supports batch MapReduce applications only.

Distributed-Shell

- Distributed-Shell is an example application included with the Hadoop core components that demonstrates how to write applications on top of YARN.
- It provides a simple method for running shell commands and scripts in containers in parallel on a Hadoop YARN cluster.

Hadoop MapReduce

- MapReduce was the first YARN framework and drove many of YARN's requirements. It is integrated tightly with the rest of the Hadoop ecosystem projects, such as Apache Pig, Apache Hive, and Apache Oozie.

Apache Tez:

- Many Hadoopjobs involve the execution of a complex directed acyclic graph (DAG) of tasks using separate MapReduce stages. Apache Tez generalizes this process and enables these tasks to be spread across stages so that they can be run as a single,all-encompassing job.

- Tez can be used as a MapReduce replacement for projects such as Apache Hive and Apache Pig. No changes are needed to the Hive or Pig applications.

Apache Giraph

- Apache Giraph is an iterative graph processing system built for high scalability.
- Facebook, Twitter, and LinkedIn use it to create social graphs of users.
- Giraph was originally written to run on standard Hadoop V1 using the MapReduce framework, but that approach proved inefficient and totally unnatural for various reasons.
- The native Giraph implementation under YARN provides the user with an iterative processing model that is not directly available with MapReduce.
- In addition, using the flexibility of YARN, the Giraph developers plan on implementing their own web interface to monitor job progress.

Hoya: HBase on YARN

- The Hoya project creates dynamic and elastic Apache HBase clusters on top of YARN.
- A client application creates the persistent configuration files, sets up the HBase cluster XML files, and then asks YARN to create an ApplicationMaster.

- YARN copies all files listed in the client's application-launch request from HDFS into the local file system of the chosen server, and then executes the command to start the Hoya ApplicationMaster.
- Hoya also asks YARN for the number of containers matching the number of HBase region servers it needs.

Dryad on YARN

- Similar to Apache Tez, Microsoft's Dryad provides a DAG as the abstraction of execution flow. This framework is ported to run natively on YARN and is fully compatible with its non-YARN version.
- The code is written completely in native C++ and C# for worker nodes and uses a thin layer of Java within the application.

Apache Spark

- Spark was initially developed for applications in which keeping data in memory improves performance, such as iterative algorithms, which are common in machine learning, and interactive data mining.
- Spark differs from classic MapReduce in two important ways.
- First, Spark holds intermediate results in memory, rather than writing them to disk.
- Second, Spark supports more than just MapReduce functions; that is, it greatly expands the set of possible analyses that can be executed over HDFS data stores.
- It also provides APIs in Scala, Java, and Python.

Apache Storm

- This framework is designed to process unbounded streams of data in real time. It can be used in any programming language.
- The basic Storm use-cases include real-time analytics,online machine learning, continuous computation, distributed RPC (remote procedure calls), ETL (extract, transform, and load), and more.
- Storm provides fast performance, is scalable, is fault tolerant, and provides processing guarantees.
- It works directly under YARN and takes advantage of the common data and resource management substrate.

6. Explain Apache oozie workflow for Hadoop Architecture with a neat diagram.

- Oozie is a workflow director system designed to run and manage multiple related Apache Hadoop jobs.
- Oozie is designed to construct and manage these workflows.
- Oozie is not a substitute for the YARN scheduler.That is, YARN manages resources for individual Hadoop jobs, and Oozie provides a way to connect and control Hadoop jobs on the cluster.
- Oozie workflow jobs are represented as directed acyclic graphs (DAGs) of actions. (DAGs are basically graphs that cannot have directed loops.) Three types of Oozie jobs are permitted:
  1. Workflow—a specified sequence of Hadoop jobs with outcome-based decision points and control dependency. Progress from

one action to another cannot happen until the first action is complete.

2. Coordinator—a scheduled workflow job that can run at various time intervals or when data become available.
3. Bundle—a higher-level Oozie abstraction that will batch a set of coordinator jobs.

Oozie also provides a CLI and a web UI for monitoring jobs.

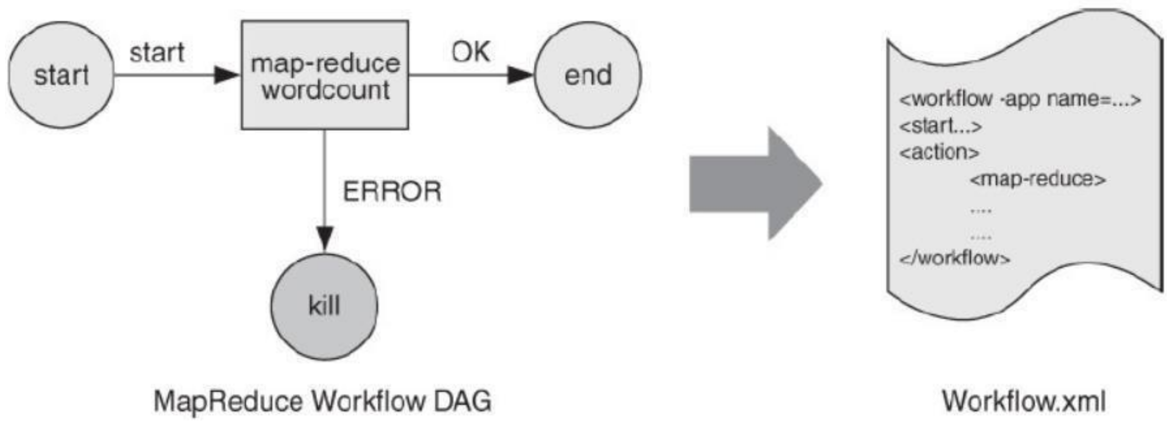

Figure : simple oozie DAG workflow

Oozie workflow definitions are written in hPDL. Such workflows contain several types of nodes:

a) **Control flow nodes** define the beginning and the end of a workflow. They include start, end, and optional fail nodes.

b) **Action nodes** are where the actual processing tasks are defined. When an action node finishes, the remote systems notify Oozie and the next node in the workflow is executed. Action nodes can also include HDFS commands.

c) **Fork/join nodes** enable parallel execution of tasks in the workflow. The fork node enables two or more tasks to run at the same time. A join node represents a rendezvous point that must wait until all forked tasks complete.

d) Control flow nodes enable decisions to be made about the previous task. Control decisions are based on the results of the previous action (e.g., file size or file existence). Decision nodes are essentially switch-case statements that use JSP EL (Java Server Pages—Expression Language) that evaluate to either true or false.

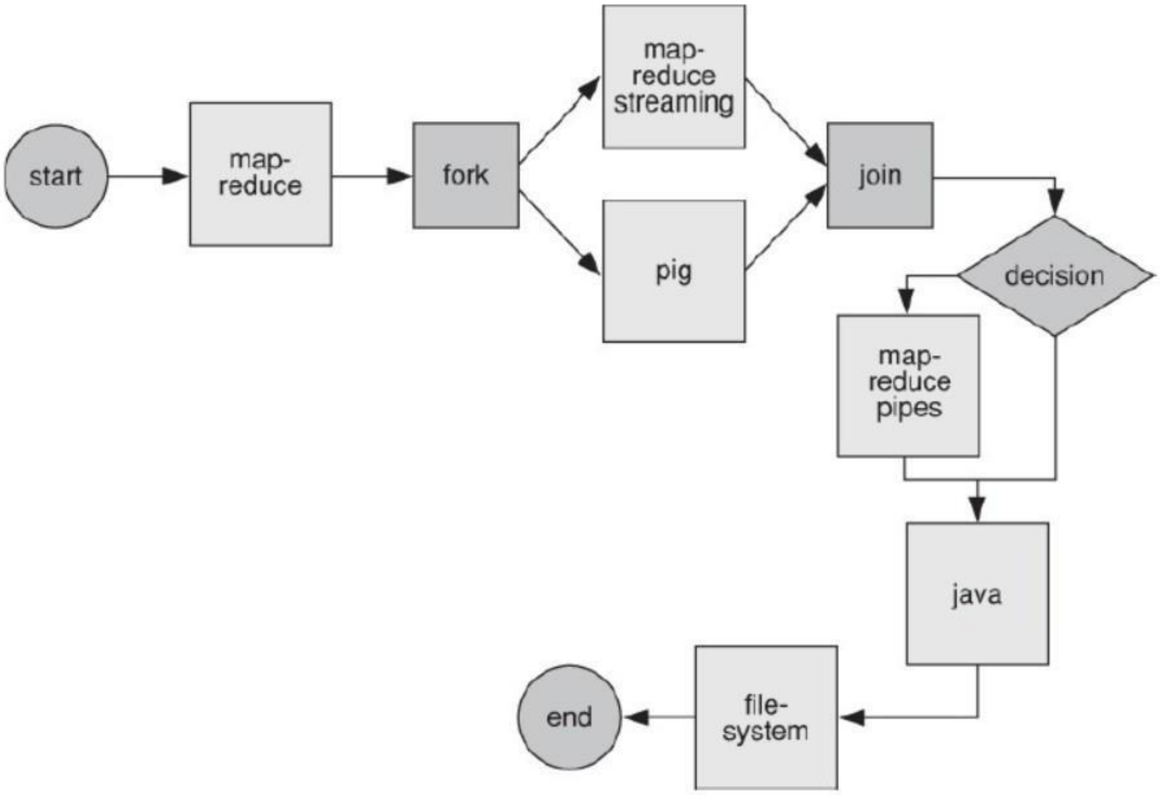

Figure: A more complex Oozie DAG workflow