|  |  |
| --- | --- |
| **TITLE** | Hadoop code to count number of occurrences |
| **PROBLEM STATEMENT/ DEFINITION** | Write a code in JAVA for a simple Word Count application that counts the number of occurrences of each word in a given input set using the Hadoop Map-Reduce framework on local-standalone set-up. |
| **OBJECTIVE** | ● Learn Map reduce for counting occurrences using Hadoop ● Learn to setup Hadoop environment |
| **S/W PACKAGES AND HARDWARE APPARATUS USED** | 1. Operating System : 64-bit Open source Linux or its derivative  2. Programming Language: JAVA  3. Hadoop Environment |
| **REFERENCES** | ● Tom White, “HADOOP The Definitive Guide”, O’REILLY  ● Donald Miner & Adam Shook, “MapReduce Design Patterns”, O’REILLY |
| **STEPS** | **Refer to theory, algorithm, test input, test output.** |
| **INSTRUCTIONS FOR WRITING JOURNAL** | 1. Date  2. Assignment no.  3. Problem definition  4. Learning objective  5. Learning outcome  6. Related Mathematics  7. Concepts related Theory  8. Test cases  9. Program code with proper documentation.  10. Output of program.  11. Conclusion and applications (the verification and testing of outcomes) |

Head of Department Subject Co-ordinator

**(Dr. M.S.Takalikar) (Dr. S.S.Sonawane)**

P:F:-LTL-UG / 03 / R1

**Assignment No. 11**

**● Aim**: Hadoop code to count number of occurrences.

**● Problem Statement / Definition**: Write a code in JAVA for a simple Word Count application that counts the number of occurrences of each word in a given input set using the Hadoop Map-Reduce framework on local-standalone set-up.

**● Prerequisites** : JAVA Programming

**● Learning Objectives**

* + Learn MapReduce using Hadoop
  + Learn to setup Hadoop environment

● Learning Outcome:

Students will be able to decompose problem into subproblems and to learn how to implement counting using Hadoop.

**● Related Mathematics :**

**Mathematical Model**

Let S be the system set:

S = {s; e; X; Y; Fme;DD;NDD; Fc; Sc}

s=start state ,e=end state

X=set of inputs X = {X1,X2,X3,X4}

where

X1 = Word count

X2 = U

X3 = Occurrences

X4 = Timestamp

Y= set of outputs Y = {Y1,Y2}

Y1 = Lines

Y2 = Average rating

Fme is the set of main functions

Fme = {f1,f2}

where

DD= Deterministic Data Text Data

NDD=Non-deterministic data No non deterministic data

Fc =failure case: No failure case identified for this application

**● Theory**

**Hadoop**

Hadoop is an open source distributed processing framework that manages data processing and storage for big data applications running in clustered systems. It is at the center of a growing ecosystem of big data technologies that are primarily used to support advanced analytics initiatives, including predictive analysis, data mining and machine learning applications. Hadoop can handle various forms of structured and unstructured data, giving users more flexibility for collecting, processing and analyzing data than relational databases and data warehouse provide.

**MapReduce**

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

Diagram

Description automatically generated

sequence of the name MapReduce implies, the reduce task is always performed after the map job.

●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.

**Mapper Class**

The first stage in Data Processing using MapReduce is the Mapper Class. Here, RecordReader processes each Input record and generates the respective key-value pair. Hadoop’s Mapper store saves this intermediate data into the local disk.

* Input Split

It is the logical representation of data. It represents a block of work that contains a single map task in the MapReduce Program.

Record Reader

It interacts with the Input split and converts the obtained data in the form of Key-Value Pairs.

**Reducer Class**

The Intermediate output generated from the mapper is fed to the reducer which processes it and generates the final output which is then saved in the HDFS.

**Driver Class**

The major component in a MapReduce job is a Driver Class. It is responsible for setting up a MapReduce Job to run-in Hadoop. We specify the names of Mapper and Reducer Classes long with data types and their respective job names.

**How to run Hadoop Program:**

1.start hadoop. start-all.sh

2.Check all components of Hadoop whether it is ready or not jps

3.Assuming environment variables are set as follows:

export JAVA\_HOME=/usr/java/default

export PATH=${JAVA\_HOME}/bin:${PATH}

export HADOOP\_CLASSPATH=${JAVA\_HOME}/lib/tools.jar

4.copy the code of to the home directory

5.Compile code

javac -classpath <hadooop-core.jar file> -d <Your New Directory>/ <sourceCode.java>

6.Create JAR file for: a.Mapper Class b.Driver Class c.Reducer Class

jar -cvf <File you have to create> -C <Directory you have obtained in previous command>

7.Run code on Hadoop Framework hadoop fs -put <source file path> /input

8.Now run program using ur Jar file

hadoop jar <your jar file> <directory name without /> /input/<your file name> /output/<output file name>

9.Read Output file

hadoop fs -cat /output/<your file>/part-r-00000

Link : <http://www.pavanjaiswal.com/2015/07/hadoop-260-single-node-setup-on-fedora.html>

* **Test data:**

Normal text file.

**Code: Count.java**

package demo;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

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;

import org.apache.hadoop.util.GenericOptionsParser;

public class count

{

public static void main(String[] args) throws IOException, ClassNotFoundException, InterruptedException

{

Configuration c=new Configuration(); String[] files=new

GenericOptionsParser(c,args).getRemainingArgs();

Path input=new Path(files[0]); Path output=new Path(files[1]); Job j=new Job(c,"wordcount"); j.setJarByClass(count.class); j.setMapperClass(mapper.clas; j.setReducerClass(reduce.clas); j.setOutputKeyClass(Text.clas;

j.setOutputValueClass(LongWritable.class); FileInputFormat.addInputPath(j, input); FileOutputFormat.setOutputPath(j, output);

System.exit(j.waitForCompletion(true)?0:1);

}

}

**Conclusion:**

* Demonstrates how applications can access configuration parameters in the setup method of the Mapper (and Reducer) implementations.
* Demonstrates the utility of the GenericOptionsParser to handle generic Hadoop command-line options.
* Demonstrates how applications can use Counters and how they can set application-specific status information passed to the map (and reduce) method.