|  |
| --- |
| Topic : **Custom\_Input\_FileFormat\_Example**  Need to implement : FileInputFormat , LogRecordReader Classes (Reading Logfiles data)  Output : **Hostname and Response Size** |
| Topic : **Custom\_Input\_FileFormat\_Assignment**  Need to implement : FileInputFormat , EmpRecordReader Classes (Reading Employee data)  Output : **Agegroup and Max Salary** |
| Topic : **Custom\_Output\_FileFormat\_Example**  Need to implement : FileOutputFormat(custom xml fileformat ) , CustomRecordWriter Classes  Output : **Hostname and Sum of responsesizes** |
| Topic : **Custom\_Output\_FileFormat\_Assignment**  Need to implement : FileOutputFormat (custom xml fileformat ), CustomRecordWriter Classes  Output : **Employeename and Salary in XML** |

**Custom-FileInputFormat Example**

**Sample Output**:

130.99.52.153 400

134.7.115.71 800

ad12-023.compuserve.com 400

ad14-021.compuserve.com 400

alyssa.prodigy.com 2520

**Driver Program – LogProcessor.java**

import java.net.URI;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.conf.Configured;

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.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.jobcontrol.ControlledJob;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.util.Tool;

import org.apache.hadoop.util.ToolRunner;

public class LogProcessor extends Configured implements Tool {

public static void main(String[] args) throws Exception {

int res = ToolRunner.run(new Configuration(), new LogProcessor(), args);

System.exit(res);

}

@Override

public int run(String[] args) throws Exception {

if (args.length < 3) {

System.err.println("Usage: <input\_path> <output\_path> <num\_reduce\_tasks>");

System.exit(-1);

}

/\* input parameters \*/

String inputPath = args[0];

String outputPath = args[1];

int numReduce = Integer.parseInt(args[2]);

Job job = new Job(getConf(), "log-analysis");

job.setJarByClass(LogProcessor.class);

job.setMapperClass(LogProcessorMap.class);

job.setReducerClass(LogProcessorReduce.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

job.setInputFormatClass(LogFileInputFormat.class);// specifying InputFormatClass here

FileInputFormat.setInputPaths(job, new Path(inputPath));

FileOutputFormat.setOutputPath(job, new Path(outputPath));

job.setNumReduceTasks(numReduce);

int exitStatus = job.waitForCompletion(true) ? 0 : 1;

return exitStatus;

}

}

**LogWritable.java**:

import java.io.DataInput;

import java.io.DataOutput;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.io.WritableComparable;

public class LogWritable implements WritableComparable<LogWritable> {

private Text userIP, timestamp, request;

private IntWritable responseSize, status;

public LogWritable() {

this.userIP = new Text();

this.timestamp = new Text();

this.request = new Text();

this.responseSize = new IntWritable();

this.status = new IntWritable();

}

public void set (String userIP, String timestamp, String request, int bytes, int status)

{

this.userIP.set(userIP);

this.timestamp.set(timestamp);

this.request.set(request);

this.responseSize.set(bytes);

this.status.set(status);

}

@Override

public void readFields(DataInput in) throws IOException {

userIP.readFields(in);

timestamp.readFields(in);

request.readFields(in);

responseSize.readFields(in);

status.readFields(in);

}

@Override

public void write(DataOutput out) throws IOException {

userIP.write(out);

timestamp.write(out);

request.write(out);

responseSize.write(out);

status.write(out);

}

@Override

public int compareTo(LogWritable o) {

if (userIP.compareTo(o.userIP) == 0) {

return timestamp.compareTo(o.timestamp);

} else

return userIP.compareTo(o.userIP);

}

public int hashCode()

{

return userIP.hashCode();

}

public Text getUserIP() {

return userIP;

}

public Text getTimestamp() {

return timestamp;

}

public Text getRequest() {

return request;

}

public IntWritable getResponseSize() {

return responseSize;

}

public IntWritable getStatus() {

return status;

}

}

**LogProcessorMap.java**:

import java.io.File;

import java.io.IOException;

import java.net.URL;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class LogProcessorMap extends Mapper<Object, LogWritable, Text, IntWritable >

{

public void map(Object key, LogWritable value, Context context)

throws IOException, InterruptedException

{

context.write(value.getUserIP(),value.getResponseSize());

}

}

**LogProcessorReducer.java**:

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class LogProcessorReduce 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);

}

}

**LogFileRecordReader.java**:

import java.io.IOException;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.mapreduce.InputSplit;

import org.apache.hadoop.mapreduce.RecordReader;

import org.apache.hadoop.mapreduce.TaskAttemptContext;

import org.apache.hadoop.mapreduce.lib.input.LineRecordReader;

public class LogFileRecordReader extends RecordReader<LongWritable, LogWritable>{

LineRecordReader lineReader;

LogWritable value;

@Override

public void initialize(InputSplit inputSplit, TaskAttemptContext attempt)

throws IOException, InterruptedException {

lineReader = new LineRecordReader();

lineReader.initialize(inputSplit, attempt);

}

@Override

public boolean nextKeyValue() throws IOException, InterruptedException

{

if (!lineReader.nextKeyValue())

{

return false;

}

String logEntryPattern = "^(\\S+) (\\S+) (\\S+) \\[([\\w:/]+\\s[+\\-]\\d{4})\\] \"(.+?)\" (\\d{3}) (\\d+)";

Pattern p = Pattern.compile(logEntryPattern);

Matcher matcher = p.matcher(lineReader.getCurrentValue().toString());

if (!matcher.matches()) {

return nextKeyValue();

}

String userIP = matcher.group(1);

String timestamp = matcher.group(4);

String request = matcher.group(5);

int status = Integer.parseInt(matcher.group(6));

int bytes = Integer.parseInt(matcher.group(7));

value = new LogWritable();

value.set(userIP, timestamp, request,status, bytes);

return true;

}

@Override

public LongWritable getCurrentKey() throws IOException,

InterruptedException {

return lineReader.getCurrentKey();

}

@Override

public LogWritable getCurrentValue() throws IOException,

InterruptedException {

return value;

}

@Override

public float getProgress() throws IOException, InterruptedException {

return lineReader.getProgress();

}

@Override

public void close() throws IOException {

lineReader.close();

}

}

**LogFileInputFormat.java**:

import java.io.IOException;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.mapreduce.InputSplit;

import org.apache.hadoop.mapreduce.RecordReader;

import org.apache.hadoop.mapreduce.TaskAttemptContext;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

public class LogFileInputFormat extends FileInputFormat<LongWritable, LogWritable>

{

@Override

public RecordReader<LongWritable, LogWritable> createRecordReader(InputSplit arg0,

TaskAttemptContext arg1) throws IOException, InterruptedException

{

return new LogFileRecordReader();

}

}

**Custom-FileInputFormat Example**

**SampleInput:**

1001,RAMESH, 9000, A, CSE

1002, SOMESH, 8000, A, ECE

1003,UMESH,9500,A,CSE

1004,RAKESH,15000,A,CSE

1005,RAJU,20000,A,IT

1006,RAMU,30000,A,CSE

1007,SHYAM,36000,A,CSE

1008,RANI,38000,A,CSE

**Sample Output**:

A 40000

B 72000

C 75000

D 150000

**EmpProcessor.java**:

import java.net.URI;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.conf.Configured;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.\*;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.jobcontrol.ControlledJob;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.util.Tool;

import org.apache.hadoop.util.ToolRunner;

public class EmpProcessor extends Configured implements Tool

{

public static void main(String[] args) throws Exception {

int res = ToolRunner.run(new Configuration(), new EmpProcessor(), args);

System.exit(res);

}

@Override

public int run(String[] args) throws Exception

{

if (args.length < 3) {

System.err.println("Usage: <input\_path> <output\_path> <num\_reduce\_tasks>");

System.exit(-1);

}

/\* input parameters \*/

String inputPath = args[0];

String outputPath = args[1];

int numReduce = Integer.parseInt(args[2]);

@SuppressWarnings("deprecation")

Job job = new Job(getConf(), "employee-data-analysis");

job.setJarByClass(EmpProcessor.class);

job.setMapperClass(EmpProcessorMap.class);

job.setReducerClass(EmpProcessorReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(LongWritable.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(LongWritable.class);

job.setInputFormatClass(EmpFileInputFormat.class);// specifying InputFormatClass

FileInputFormat.setInputPaths(job, new Path(inputPath));

FileOutputFormat.setOutputPath(job, new Path(outputPath));

job.setNumReduceTasks(numReduce);

int exitStatus = job.waitForCompletion(true) ? 0 : 1;

return exitStatus;

}

}

**EmpWritable.java**:

import java.io.DataInput;

import java.io.DataOutput;

import java.io.IOException;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.io.WritableComparable;

public class EmpWritable implements WritableComparable<EmpWritable>

{

private LongWritable empID = new LongWritable();

private Text empName = new Text();

private LongWritable salary = new LongWritable();

private Text ageGroup = new Text();

private Text empDepartment = new Text();

public EmpWritable() {

this.empID = new LongWritable();

this.empName = new Text();

this.salary = new LongWritable();

this.ageGroup = new Text();

this.empDepartment = new Text();

}

public void set (long empID, String empName, long salary,String ageGroup,String empDepartment)

{

this.empID.set(empID);

this.empName.set(empName);

this.salary.set(salary);

this.ageGroup.set(ageGroup);

this.empDepartment.set(empDepartment);

}

@Override

public void readFields(DataInput in) throws IOException {

empID.readFields(in);

empName.readFields(in);

salary.readFields(in);

ageGroup.readFields(in);

empDepartment.readFields(in);

}

@Override

public void write(DataOutput out) throws IOException {

empID.write(out);

empName.write(out);

salary.write(out);

ageGroup.write(out);

empDepartment.write(out);

}

@Override

public int compareTo(EmpWritable o) {

return empDepartment.compareTo(o.empDepartment);

}

public int hashCode()

{

return empDepartment.hashCode();

}

public LongWritable getEmpID() {

return empID;

}

public Text getEmpName() {

return empName;

}

public LongWritable getSalary() {

return salary;

}

public Text getAgeGroup() {

return ageGroup;

}

public Text getEmpDepartment() {

return empDepartment;

}

}

**EmpProcessorMap.java**:

import java.io.File;

import java.io.IOException;

import java.net.URL;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.\*;

import org.apache.hadoop.mapreduce.Mapper;

public class EmpProcessorMap extends Mapper<Object, EmpWritable, Text, LongWritable >

{

public void map(Object key, EmpWritable empValue, Context context)

throws IOException, InterruptedException

{

context.write(empValue.getAgeGroup(),empValue.getSalary());

}

}

**EmpProcessorReducer.java**:

import java.io.IOException;

import org.apache.hadoop.io.\*;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class EmpProcessorReducer extends Reducer<Text, LongWritable, Text, LongWritable>

{

private LongWritable result = new LongWritable();

public void reduce(Text key, Iterable<LongWritable> values, Context context)

throws IOException, InterruptedException

{

long max = 0;long number=0;

for (LongWritable val : values)

{

number =val.get();

if(number>max)

{

max=number;//update max variable if you found input value is greater than max

}

}

result.set(max);

context.write(key, result);

}

}

**EmpFileRecordReader.java**:

import java.io.IOException;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.mapreduce.InputSplit;

import org.apache.hadoop.mapreduce.RecordReader;

import org.apache.hadoop.mapreduce.TaskAttemptContext;

import org.apache.hadoop.mapreduce.lib.input.LineRecordReader;

public class EmpFileRecordReader extends RecordReader<LongWritable, EmpWritable>

{

LineRecordReader lineReader;

EmpWritable empValue;

@Override

public void initialize(InputSplit inputSplit, TaskAttemptContext attempt)

throws IOException, InterruptedException {

lineReader = new LineRecordReader();

lineReader.initialize(inputSplit, attempt);

}

@Override

public boolean nextKeyValue() throws IOException, InterruptedException

{

if (!lineReader.nextKeyValue())

{

return false;

}

String empEntryPattern = "^(\\S+),(\\S+),(\\S+),(\\S+),(\\S+)";

Pattern p = Pattern.compile(empEntryPattern);

Matcher matcher = p.matcher(lineReader.getCurrentValue().toString());

if (!matcher.matches()) {

return nextKeyValue();

}

long empID = Long.parseLong(matcher.group(1));

String empName = matcher.group(2);

long salary = Long.parseLong(matcher.group(3));

String ageGroup = matcher.group(4);

String empDepartment = matcher.group(5);

empValue = new EmpWritable();

empValue.set(empID, empName, salary, ageGroup,empDepartment );

return true;

}

@Override

public LongWritable getCurrentKey() throws IOException, InterruptedException

{

return lineReader.getCurrentKey();

}

@Override

public EmpWritable getCurrentValue() throws IOException, InterruptedException {

return empValue;

}

@Override

public float getProgress() throws IOException, InterruptedException {

return lineReader.getProgress();

}

@Override

public void close() throws IOException {

lineReader.close();

}

}

**EmpFileInputFormat.java**:

import java.io.IOException;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.mapreduce.InputSplit;

import org.apache.hadoop.mapreduce.RecordReader;

import org.apache.hadoop.mapreduce.TaskAttemptContext;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

public class EmpFileInputFormat extends FileInputFormat<LongWritable, EmpWritable>

{

@Override

public RecordReader<LongWritable, EmpWritable> createRecordReader(InputSplit arg0,

TaskAttemptContext arg1) throws IOException, InterruptedException

{

return new EmpFileRecordReader();

}

}

**Custom – FileOutFormat Example**

**Sample Output**:

-<results>

**-**<result>

<key>**130.99.52.153**</key>

<value>**61716**</value>

</result>

[**-**](file:///E:\py-DataScience\Hadoop\Day_6_Hadoop\Custom_Output_FileFormat_Example\Datasets\Custom_Output_FileFormat_Example_Output_Correct.xml) <result>

<key>**130.99.52.153**</key>

<value>**1121554**</value>

</result>

[**-**](file:///E:\py-DataScience\Hadoop\Day_6_Hadoop\Custom_Output_FileFormat_Example\Datasets\Custom_Output_FileFormat_Example_Output_Correct.xml) <result>

<key>**134.7.115.71**</key>

<value>**64939**</value>

</result>

**LogProcessor**:

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.conf.Configured;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.io.\*;

import org.apache.hadoop.mapreduce.Job;

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 LogProcessor extends Configured implements Tool

{

public static void main(String[] args) throws Exception {

int res = ToolRunner.run(new Configuration(), new LogProcessor(), args);

System.exit(res);

}

@Override

public int run(String[] args) throws Exception

{

if (args.length < 3) {

System.err.println("Usage: <input\_path> <output\_path> <num\_reduce\_tasks>");

System.exit(-1);

}

/\* input parameters \*/

String inputPath = args[0];

String outputPath = args[1];

int numReduce = Integer.parseInt(args[2]);

Configuration conf = new Configuration();

Job job = new Job(conf, "log-analysis");

job.setJarByClass(LogProcessor.class);

job.setMapperClass(LogProcessorMap.class);

job.setReducerClass(LogProcessorReduce.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(LongWritable.class);

FileInputFormat.setInputPaths(job, new Path(inputPath));

FileOutputFormat.setOutputPath(job, new Path(outputPath));

job.setNumReduceTasks(numReduce);

job.setOutputFormatClass(XmlOutputFormat.class);//Set Output File Format

int exitStatus = job.waitForCompletion(true) ? 0 : 1;

return exitStatus;

}

}

**LopProcessorMap.java**:

import java.io.IOException;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class LogProcessorMap extends Mapper<Object, Text, Text, LongWritable> {

private Text userHostText = new Text();

private LongWritable responseSize = new LongWritable();

public void map(Object key, Text value, Context context)

throws IOException, InterruptedException

{

String logEntryPattern = "^(\\S+) (\\S+) (\\S+) \\[([\\w:/]+\\s[+\\-]\\d{4})\\] \"(.+?)\" (\\d{3}) (\\d+)";

Pattern p = Pattern.compile(logEntryPattern);

Matcher matcher = p.matcher(value.toString());

if (!matcher.matches()) {

return;

}

String userHost = matcher.group(1);

userHostText.set(userHost);

Long size = Long.parseLong(matcher.group(7));

responseSize.set(size);

context.write(userHostText, responseSize);

}

}

**LopProcessorReducer.java**:

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.io.Writable;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.io.\*;

public class LogProcessorReduce extends Reducer<Text,LongWritable,Text,LongWritable>

{

public void reduce(Text key, Iterable<LongWritable> values, Context context)

throws IOException, InterruptedException

{

for ( LongWritable value : values) {

context.write(key,value);

}

}

}

**CustomRecordWriter.java**:

import java.io.\*;

import org.apache.hadoop.mapreduce.\*;

import org.apache.hadoop.io.\*;

import org.apache.hadoop.mapreduce.lib.input.\*;

import org.apache.hadoop.mapreduce.lib.output.\*;

import org.apache.hadoop.fs.\*;

public class CustomRecordWriter extends RecordWriter<Text, LongWritable> {

private DataOutputStream out;

public CustomRecordWriter(DataOutputStream stream) {

out = stream;

try {

out.writeBytes("<results>\n");

}

catch (Exception ex) {

}

}

public void close(TaskAttemptContext arg0) throws IOException, InterruptedException {

//close our file

try {

out.writeBytes("</results>\n");

} finally {

// even if writeBytes() fails, make sure we close the stream

out.close();

}

}

public void write(Text arg0, LongWritable arg1) throws IOException, InterruptedException {

//write out our key

out.writeBytes("<result>");

out.writeBytes("\r\n<key>");

out.writeBytes(arg0.toString());

out.writeBytes("</key>");

out.writeBytes("\r\n<value>");

out.writeBytes(arg1.toString());

out.writeBytes("</value>\r\n");

out.writeBytes("</result>");

out.writeBytes("\r\n");

}

}

**XMLOutputFormat.java**:

import java.io.\*;

import org.apache.hadoop.mapreduce.\*;

import org.apache.hadoop.io.\*;

import org.apache.hadoop.mapreduce.lib.input.\*;

import org.apache.hadoop.mapreduce.lib.output.\*;

import org.apache.hadoop.fs.\*;

public class XmlOutputFormat extends FileOutputFormat<Text, LongWritable>

{

public RecordWriter<Text, LongWritable> getRecordWriter(TaskAttemptContext arg0)

throws IOException, InterruptedException

{

//get the current path

Path path = FileOutputFormat.getOutputPath(arg0);

//create the full path with the output directory plus our filename

Path fullPath = new Path(path, "result.xml");

//create the file in the file system

FileSystem fs = path.getFileSystem(arg0.getConfiguration());

FSDataOutputStream fileOut = fs.create(fullPath, arg0);

//create our record writer with the new file

return new CustomRecordWriter(fileOut);

}

}

**Custom FileOutputFormat Assignment**

**SampleInput:**

1001,RAMESH, 9000, A, CSE

1002, SOMESH, 8000, A, ECE

1003,UMESH,9500,A,CSE

1004,RAKESH,15000,A,CSE

1005,RAJU,20000,A,IT

1006,RAMU,30000,A,CSE

1007,SHYAM,36000,A,CSE

1008,RANI,38000,A,CSE

**SampleOutput**:

[**-**](file:///E:\py-DataScience\Hadoop\Day_6_Hadoop\Custom_Output_FileFormat_Assignment\Datasets\Custom_Output_FileFormat_Assignment_Output_Correct.xml) <results>

[**-**](file:///E:\py-DataScience\Hadoop\Day_6_Hadoop\Custom_Output_FileFormat_Assignment\Datasets\Custom_Output_FileFormat_Assignment_Output_Correct.xml) <result>

<key>**AKASH**</key>

<value>**75000**</value>

</result>

[**-**](file:///E:\py-DataScience\Hadoop\Day_6_Hadoop\Custom_Output_FileFormat_Assignment\Datasets\Custom_Output_FileFormat_Assignment_Output_Correct.xml) <result>

<key>**ANANTH**</key>

<value>**150000**</value>

</result>

[**-**](file:///E:\py-DataScience\Hadoop\Day_6_Hadoop\Custom_Output_FileFormat_Assignment\Datasets\Custom_Output_FileFormat_Assignment_Output_Correct.xml) <result>

<key>**ASHOK**</key>

<value>**80000**</value>

</result>

**EmpProcessor.java**:

import java.net.URI;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.conf.Configured;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.\*;

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.jobcontrol.ControlledJob;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.util.Tool;

import org.apache.hadoop.util.ToolRunner;

public class EmpProcessor extends Configured implements Tool {

public static void main(String[] args) throws Exception {

int res = ToolRunner.run(new Configuration(), new EmpProcessor(),

args);

System.exit(res);

}

@Override

public int run(String[] args) throws Exception {

if (args.length < 3) {

System.err.println("Usage: <input\_path> <output\_path> <num\_reduce\_tasks>");

System.exit(-1);

}

/\* input parameters \*/

String inputPath = args[0];

String outputPath = args[1];

int numReduce = Integer.parseInt(args[2]);

@SuppressWarnings("deprecation")

Job job = new Job(getConf(), "employee-data-analysis");

job.setJarByClass(EmpProcessor.class);

job.setMapperClass(EmpProcessorMap.class);

job.setReducerClass(EmpProcessorReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(LongWritable.class);

FileInputFormat.setInputPaths(job, new Path(inputPath));

FileOutputFormat.setOutputPath(job, new Path(outputPath));

job.setNumReduceTasks(numReduce);

job.setOutputFormatClass(XmlOutputFormat.class);//Set Output File Format

int exitStatus = job.waitForCompletion(true) ? 0 : 1;

return exitStatus;

}

}

**EmpProcessorMap.java**:

import java.io.File;

import java.io.IOException;

import java.net.URL;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.\*;

import org.apache.hadoop.mapreduce.Mapper;

public class EmpProcessorMap extends Mapper<Object, Text, Text, LongWritable >

{

private Text empName = new Text();

private LongWritable salary = new LongWritable();

public void map(Object key, Text value, Context context)

throws IOException, InterruptedException

{

String empEntryPattern = "^(\\S+),(\\S+),(\\S+),(\\S+),(\\S+)";

Pattern p = Pattern.compile(empEntryPattern);

Matcher matcher = p.matcher(value.toString());

if (!matcher.matches()) {

return;

}

String name = matcher.group(2);

empName.set(name);

long sal = Long.parseLong(matcher.group(3));

salary.set(sal);

context.write(empName,salary);

}

}

**EmpProcessorReducer.java**:

import java.io.IOException;

import org.apache.hadoop.io.\*;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class EmpProcessorReducer extends Reducer<Text,LongWritable,Text,LongWritable>

{

private IntWritable result = new IntWritable();

public void reduce(Text key, Iterable<LongWritable> values, Context context)

throws IOException, InterruptedException {

int sum = 0;

for (LongWritable val : values) {

context.write(key,val);

}

}

}

**CustomRecordWriter.java**:

import java.io.\*;

import org.apache.hadoop.mapreduce.\*;

import org.apache.hadoop.io.\*;

import org.apache.hadoop.mapreduce.lib.input.\*;

import org.apache.hadoop.mapreduce.lib.output.\*;

import org.apache.hadoop.fs.\*;

public class CustomRecordWriter extends RecordWriter<Text, LongWritable>

{

private DataOutputStream out;

public CustomRecordWriter(DataOutputStream stream) {

out = stream;

try {

out.writeBytes("<results>\n");

}

catch (Exception ex) {

}

}

public void close(TaskAttemptContext arg0) throws IOException, InterruptedException

{

//close our file

try {

out.writeBytes("</results>\n");

} finally {

// even if writeBytes() fails, make sure we close the stream

out.close();

}

}

public void write(Text arg0, LongWritable arg1) throws IOException, InterruptedException {

//write out our key

out.writeBytes("<result>");

out.writeBytes("\r\n<key>");

out.writeBytes(arg0.toString());

out.writeBytes("</key>");

out.writeBytes("\r\n<value>");

out.writeBytes(arg1.toString());

out.writeBytes("</value>\r\n");

out.writeBytes("</result>");

out.writeBytes("\r\n");

}

}

**XMLOutputFormat.java**:

import java.io.\*;

import org.apache.hadoop.mapreduce.\*;

import org.apache.hadoop.io.\*;

import org.apache.hadoop.mapreduce.lib.input.\*;

import org.apache.hadoop.mapreduce.lib.output.\*;

import org.apache.hadoop.fs.\*;

public class XmlOutputFormat extends FileOutputFormat<Text, LongWritable>

{

public RecordWriter<Text, LongWritable> getRecordWriter(TaskAttemptContext arg0)

throws IOException, InterruptedException

{

//get the current path

Path path = FileOutputFormat.getOutputPath(arg0);

//create the full path with the output directory plus our filename

Path fullPath = new Path(path, "result.xml");

//create the file in the file system

FileSystem fs = path.getFileSystem(arg0.getConfiguration());

FSDataOutputStream fileOut = fs.create(fullPath, arg0);

//create our record writer with the new file

return new CustomRecordWriter(fileOut);

}

}

**Matrix Example**

**Sample Input**:

A,0,0,0.0

B,0,0,0.0

A,0,1,1.0

A,0,2,2.0

A,0,3,3.0

A,0,4,4.0

B,3,1,10.0

B,3,2,11.0

…..

**Sample Output**:

(0,0) - 90.0

(0,1) - 100.0

(0,2) - 110.0

(1,0) - 240.0

(1,1) - 275.0

(1,2) - 310.0

**MatrixMultiplication.java - Driver Code**:

import java.io.IOException;

import java.util.\*;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.conf.\*;

import org.apache.hadoop.io.\*;

import org.apache.hadoop.mapreduce.\*;

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.conf.Configuration;

import org.apache.hadoop.conf.Configured;

import org.apache.hadoop.util.Tool;

import org.apache.hadoop.util.ToolRunner;

public class MatrixMultiplication extends Configured implements Tool

{

public static void main(String[] args) throws Exception {

ToolRunner.run(new Configuration(), new MatrixMultiplication(), args);

}

@Override

public int run(String[] args) throws Exception {

if (args.length < 3) {

System.err.println("Usage: <input\_path> <output\_path> <num\_reduce\_tasks>");

System.exit(-1);

}

/\* input parameters \*/

String inputPath = args[0];

String outputPath = args[1];

int numReduce = Integer.parseInt(args[2]);

Configuration conf = getConf();

//Adding configuration values

// A is an m-by-n matrix; B is an n-by-p matrix.

conf.set("m", "2");

conf.set("n", "5");

conf.set("p", "3");

Job job = new Job(conf, "MatrixMatrixMultiplicationOneStep");

job.setJarByClass(MatrixMultiplication.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

// define mapper's output key-value

job.setMapOutputKeyClass(CompositeKey.class);

job.setMapOutputValueClass(CompositeValue.class);

job.setMapperClass(MatrixMap.class);

job.setReducerClass(MatrixReduce.class);

job.setInputFormatClass(TextInputFormat.class);

job.setOutputFormatClass(TextOutputFormat.class);

FileInputFormat.addInputPath(job, new Path(inputPath));

FileOutputFormat.setOutputPath(job, new Path(outputPath));

job.setNumReduceTasks(numReduce);

job.waitForCompletion(true);

return 0;

}

}

**CompositeKey.java**:

import java.io.DataInput;

import java.io.DataOutput;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.io.WritableComparable;

public class CompositeKey implements WritableComparable<CompositeKey>

{

private IntWritable row = new IntWritable();

private IntWritable col = new IntWritable();

public CompositeKey() {

this.row = new IntWritable();

this.col = new IntWritable();

}

public void set (int row,int col)

{

this.row.set(row);

this.col.set(col);

}

@Override

public void readFields(DataInput in) throws IOException {

row.readFields(in);

col.readFields(in);

}

@Override

public void write(DataOutput out) throws IOException {

row.write(out);

col.write(out);

}

@Override

public int compareTo(CompositeKey o) {

if (row.compareTo(o.row)==0){

return (col.compareTo(o.col));

}

else return (row.compareTo(o.row));

}

public int hashCode()

{

return row.hashCode();

}

public IntWritable getCol() {

return col;

}

public IntWritable getRow() {

return row;

}

}

**CompositeValue.java**:

import java.io.DataInput;

import java.io.DataOutput;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.FloatWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.io.Writable;

public class CompositeValue implements Writable

{

private Text matrixName = new Text();

private IntWritable commonIndex = new IntWritable();

private FloatWritable fieldValue = new FloatWritable();

public CompositeValue() {

this.commonIndex = new IntWritable();

this.matrixName = new Text();

this.fieldValue = new FloatWritable();

}

public void set ( String matrixName,int commonIndex, float fieldValue)

{

this.commonIndex.set(commonIndex);

this.matrixName.set(matrixName);

this.fieldValue.set(fieldValue);

}

@Override

public void readFields(DataInput in) throws IOException {

commonIndex.readFields(in);

matrixName.readFields(in);

fieldValue.readFields(in);

}

@Override

public void write(DataOutput out) throws IOException {

commonIndex.write(out);

matrixName.write(out);

fieldValue.write(out);

}

public IntWritable getCommonIndex() {

return commonIndex;

}

public Text getMatrixName() {

return matrixName;

}

public FloatWritable getFieldValue() {

return fieldValue;

}

}

**MatrixMap.java**:

import java.io.IOException;

import java.util.\*;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.conf.\*;

import org.apache.hadoop.io.\*;

import org.apache.hadoop.mapreduce.\*;

public class MatrixMap extends Mapper<LongWritable, Text, CompositeKey, CompositeValue>

{

private final CompositeKey reducerKey = new CompositeKey();

private final CompositeValue reducerValue = new CompositeValue();

public void map(LongWritable key, Text value, Context context)

throws IOException, InterruptedException

{

Configuration conf = context.getConfiguration();

//reading configuration values

int m = Integer.parseInt(conf.get("m"));

int p = Integer.parseInt(conf.get("p"));

String line = value.toString();

String[] indicesAndValue = line.split(",");

//indicesAndValue[0] -- matrix name

//indicesAndValue[1] -- row number

//indicesAndValue[2] -- column number

//indicesAndValue[3] -- position value

if (indicesAndValue[0].equals("A")) {

for (int k = 0; k < p; k++) {

//key is result(row,k)

reducerKey.set(Integer.parseInt(indicesAndValue[1].toString()), k);

//value is (matrixname,column,positionvalue)

reducerValue.set("A",Integer.parseInt(indicesAndValue[2]),Float.parseFloat(indicesAndValue[3]));

context.write(reducerKey, reducerValue);

}

} else {

for (int i = 0; i < m; i++) {

//key is result(i,col)

reducerKey.set(i, Integer.parseInt(indicesAndValue[2]));

//value is (matrixname,row,positionvalue)

reducerValue.set("B",Integer.parseInt(indicesAndValue[1]),Float.parseFloat(indicesAndValue[3]));

context.write(reducerKey, reducerValue);

}

}

}

}

**MatrixReduce.java**:

import java.io.IOException;

import java.util.\*;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.conf.\*;

import org.apache.hadoop.io.\*;

import org.apache.hadoop.mapreduce.\*;

public class MatrixReduce extends Reducer<CompositeKey, CompositeValue, Text, Text>

{

public void reduce(CompositeKey key, Iterable<CompositeValue> values, Context context)

throws IOException, InterruptedException

{

HashMap<Integer, Float> hashA = new HashMap<Integer, Float>();

HashMap<Integer, Float> hashB = new HashMap<Integer, Float>();

for (CompositeValue val : values)

{

if ((val.getMatrixName().toString()).equals("A")) {

hashA.put(val.getCommonIndex().get(), val.getFieldValue().get());

} else {

hashB.put(val.getCommonIndex().get(), val.getFieldValue().get());

}

}

int n = Integer.parseInt(context.getConfiguration().get("n"));

float result = 0.0f;

float a\_ij;

float b\_jk;

for (int j = 0; j < n; j++) {

a\_ij = hashA.containsKey(j) ? hashA.get(j) : 0.0f;

b\_jk = hashB.containsKey(j) ? hashB.get(j) : 0.0f;

result += a\_ij \* b\_jk;

}

if (result != 0.0f) {

context.write(null, new Text("("+key.getRow().get()+","+ key.getCol()+")"+ " - " + Float.toString(result)));

}

}

}