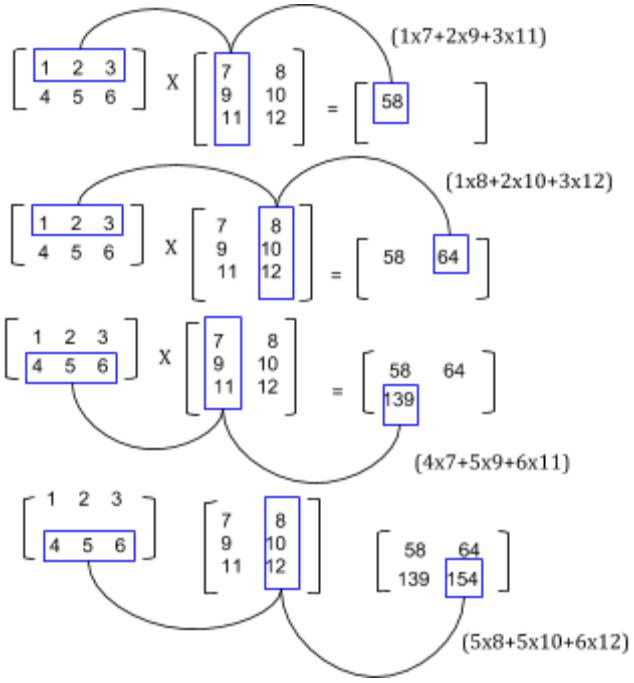
If you like my post - Do follow me on this blog - Click here to Follow my blog

Matrix Multiplication

Using MapReduce Programming

In **mathematics**, **matrix multiplication** or the **matrix product** is a binary operation that produces a matrix from two matrices. The definition is motivated by linear equations and linear transformations on vectors, which have numerous applications in applied mathematics, physics, and engineering. In more detail, if **A** is an $n \times m$ matrix and **B** is an $m \times p$ matrix, their matrix product **AB** is an $n \times p$ matrix, in which the m entries across a row of **A** are multiplied with the m entries down a column of **B** and summed to produce an entry of **AB**. When two linear transformations are represented by matrices, then the matrix product represents the composition of the two transformations.



Algorithm for Map Function.

a. for each element m_{ij} of M do
produce (key,value) pairs as ((i,k), (M,j,m_{ij}), for k=1,2,3,..
upto the number of columns of N

b. for each element njk of N do
produce (key,value) pairs as ((i,k),(N,j,N_{jk}), for i = 1,2,3,..
Upto the number of rows of M.

c. return Set of (key,value) pairs that each key (i,k), has list with values (M,j,m_{ij}) and (N, j,n_{jk}) for all possible values of j.

Algorithm for Reduce Function.

for each key (i,k) do

sort values begin with M by j in $list_M$
sort values begin with N by j in $list_N$
multiply m_{ij} and n_{jk} for jth value of each list
sum up m_{ij} x n_{jk} return (i,k), $\Sigma_{j=1}$ m_{ij} x n_{jk}

Step 1. Download the hadoop jar files with these links.

Download Hadoop Common Jar files: https://goo.gl/G4MyHp

```
$ wget https://goo.gl/G4MyHp -O hadoop-common-2.2.0.jar
```

Download Hadoop Mapreduce Jar File: https://goo.gl/KT8yfB

```
$ wget https://goo.gl/KT8yfB -O
hadoop-mapreduce-client-core-2.7.1.jar
```

Step 2. Creating Mapper file for Matrix Multiplication.

```
package www.ehadoopinfo.com;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

public class Map
  extends org.apache.hadoop.mapreduce.Mapper<LongWritable, Text, Text, Text> {
        @Override
        public void map(LongWritable key, Text value, Context context)
                        throws IOException, InterruptedException {
                Configuration conf = context.getConfiguration();
                int m = Integer.parseInt(conf.get("m"));
                int p = Integer.parseInt(conf.get("p"));
                String line = value.toString();
                // (M, i, j, Mij);
                String[] indicesAndValue = line.split(",");
                Text outputKey = new Text();
                Text outputValue = new Text();
                if (indicesAndValue[0].equals("M")) {
                        for (int k = 0; k < p; k++) {
                                outputKey.set(indicesAndValue[1] + "," + k);
                                // outputKey.set(i,k);
                                outputValue.set(indicesAndValue[0] + "," +
indicesAndValue[2]
                                                + "," + indicesAndValue[3]);
                                // outputValue.set(M,j,Mij);
                                context.write(outputKey, outputValue);
                        }
                } else {
                        // (N, j, k, Njk);
                        for (int i = 0; i < m; i++) {
                                outputKey.set(i + "," + indicesAndValue[2]);
```

```
                                outputValue.set("N," + indicesAndValue[1] + ","
                                                + indicesAndValue[3]);
                                context.write(outputKey, outputValue);
                        }
                }
        }
}
```

Step 3. Creating Reducer.java file for Matrix Multiplication.

```
package www.ehadoopinfo.com;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;
import java.util.HashMap;

public class Reduce
  extends org.apache.hadoop.mapreduce.Reducer<Text, Text, Text, Text> {
        @Override
        public void reduce(Text key, Iterable<Text> values, Context context)
                        throws IOException, InterruptedException {
                String[] value;
                //key=(i,k),
                //Values = [(M/N,j,V/W),..]
                HashMap<Integer, Float> hashA = new HashMap<Integer, Float>();
                HashMap<Integer, Float> hashB = new HashMap<Integer, Float>();
                for (Text val : values) {
                        value = val.toString().split(",");
                        if (value[0].equals("M")) {
                                hashA.put(Integer.parseInt(value[1]),
Float.parseFloat(value[2]));
                        } else {
                                hashB.put(Integer.parseInt(value[1]),
Float.parseFloat(value[2]));
                        }
                }
                int n = Integer.parseInt(context.getConfiguration().get("n"));
                float result = 0.0f;
                float m_ij;
                float n_jk;
                for (int j = 0; j < n; j++) {
                        m_ij = hashA.containsKey(j) ? hashA.get(j) : 0.0f;
                        n_jk = hashB.containsKey(j) ? hashB.get(j) : 0.0f;
                        result += m_ij * n_jk;
                }
                if (result != 0.0f) {
                        context.write(null,
                                        new Text(key.toString() + "," +
Float.toString(result)));
                }
```

```
        }
}
```

Step 4. Creating MatrixMultiply.java file for

```
package www.ehadoopinfo.com;

import org.apache.hadoop.conf.*;
import org.apache.hadoop.fs.Path;
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;

public class MatrixMultiply {

    public static void main(String[] args) throws Exception {
        if (args.length != 2) {
            System.err.println("Usage: MatrixMultiply <in_dir> <out_dir>");
            System.exit(2);
        }
        Configuration conf = new Configuration();
        // M is an m-by-n matrix; N is an n-by-p matrix.
        conf.set("m", "2");
        conf.set("n", "2");
        conf.set("p", "2");
        @SuppressWarnings("deprecation")
                Job job = new Job(conf, "MatrixMultiply");
        job.setJarByClass(MatrixMultiply.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.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);
    }
}
```

Step 5. Compiling the program in particular folder named as operation/

```
$ javac -cp
hadoop-common-2.2.0.jar:hadoop-mapreduce-client-core-2.7.1.jar:operation/:. -d
operation/ Map.java
```

```
$ javac -cp
hadoop-common-2.2.0.jar:hadoop-mapreduce-client-core-2.7.1.jar:operation/:. -d
operation/ Reduce.java
$ javac -cp
hadoop-common-2.2.0.jar:hadoop-mapreduce-client-core-2.7.1.jar:operation/:. -d
operation/ MatrixMultiply.java
```

Step 6. Let's retrieve the directory after compilation.

```
$ ls -R operation/
operation/:
www
operation/www:
ehadoopinfo
operation/www/ehadoopinfo:
com
operation/www/ehadoopinfo/com:
Map.class  MatrixMultiply.class  Reduce.class
```

Step 7. Creating Jar file for the Matrix Multiplication.

```
$ jar -cvf MatrixMultiply.jar -C operation/ .
added manifest
adding: www/(in = 0) (out= 0)(stored 0%)
adding: www/ehadoopinfo/(in = 0) (out= 0)(stored 0%)
adding: www/ehadoopinfo/com/(in = 0) (out= 0)(stored 0%)
adding: www/ehadoopinfo/com/Reduce.class(in = 2919) (out= 1271)(deflated 56%)
adding: www/ehadoopinfo/com/MatrixMultiply.class(in = 1815) (out= 932)(deflated 48%)
adding: www/ehadoopinfo/com/Map.class(in = 2353) (out= 993)(deflated 57%)
```

Step 8. Uploading the M, N file which contains the matrix multiplication data to HDFS.

```
$ cat M
M,0,0,1
M,0,1,2
M,1,0,3
M,1,1,4
$ cat N
N,0,0,5
N,0,1,6
N,1,0,7
N,1,1,8
$ hadoop fs -mkdir Matrix/
$ hadoop fs -copyFromLocal M Matrix/
$ hadoop fs -copyFromLocal N Matrix/
```

Step 9. Executing the jar file using hadoop command and thus how fetching record from HDFS and storing output in HDFS.

```
$ hadoop jar MatrixMultiply.jar www.ehadoopinfo.com.MatrixMultiply Matrix/* result/
WARNING: Use "yarn jar" to launch YARN applications.
```

```
17/10/09 14:31:22 INFO impl.TimelineClientImpl: Timeline service address:
http://sandbox.hortonworks.com:8188/ws/v1/timeline/
17/10/09 14:31:23 INFO client.RMProxy: Connecting to ResourceManager at
sandbox.hortonworks.com/10.0.2.15:8050
17/10/09 14:31:23 WARN mapreduce.JobResourceUploader: Hadoop command-line option
parsing not performed. Implement the Tool interface and execute your application with
ToolRunner to remedy this.
17/10/09 14:31:24 INFO input.FileInputFormat: Total input paths to process : 2
17/10/09 14:31:24 INFO mapreduce.JobSubmitter: number of splits:2
17/10/09 14:31:24 INFO mapreduce.JobSubmitter: Submitting tokens for job:
job_1507555978175_0006
17/10/09 14:31:25 INFO impl.YarnClientImpl: Submitted application
application_1507555978175_0006
17/10/09 14:31:25 INFO mapreduce.Job: The url to track the job:
http://sandbox.hortonworks.com:8088/proxy/application_1507555978175_0006/
17/10/09 14:31:25 INFO mapreduce.Job: Running job: job_1507555978175_0006
17/10/09 14:31:35 INFO mapreduce.Job: Job job_1507555978175_0006 running in uber mode
: false
17/10/09 14:31:35 INFO mapreduce.Job:  map 0% reduce 0%
17/10/09 14:31:45 INFO mapreduce.Job:  map 100% reduce 0%
17/10/09 14:31:53 INFO mapreduce.Job:  map 100% reduce 100%
17/10/09 14:31:54 INFO mapreduce.Job: Job job_1507555978175_0006 completed
successfully
17/10/09 14:31:55 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=198
                FILE: Number of bytes written=386063
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=302
                HDFS: Number of bytes written=36
                HDFS: Number of read operations=9
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=2
                Launched reduce tasks=1
                Data-local map tasks=2
                Total time spent by all maps in occupied slots (ms)=15088
                Total time spent by all reduces in occupied slots (ms)=6188
                Total time spent by all map tasks (ms)=15088
                Total time spent by all reduce tasks (ms)=6188
                Total vcore-seconds taken by all map tasks=15088
                Total vcore-seconds taken by all reduce tasks=6188
                Total megabyte-seconds taken by all map tasks=3772000
                Total megabyte-seconds taken by all reduce tasks=1547000
        Map-Reduce Framework
                Map input records=8
                Map output records=16
```

```
                Map output bytes=160
                Map output materialized bytes=204
                Input split bytes=238
                Combine input records=0
                Combine output records=0
                Reduce input groups=4
                Reduce shuffle bytes=204
                Reduce input records=16
                Reduce output records=4
                Spilled Records=32
                Shuffled Maps =2
                Failed Shuffles=0
                Merged Map outputs=2
                GC time elapsed (ms)=196
                CPU time spent (ms)=2720
                Physical memory (bytes) snapshot=536309760
                Virtual memory (bytes) snapshot=2506076160
                Total committed heap usage (bytes)=360185856
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=64
        File Output Format Counters
                Bytes Written=36
```

Step 10. Getting Output from part-r-00000 that was generated after the execution of the hadoop command.

```
$ hadoop fs -cat result/part-r-00000
0,0,19.0
0,1,22.0
1,0,43.0
1,1,50.0
```

In some contents I have reduced the font size to give a better code readability of this post.
Do comment me what you think about this post. If you find any errors or any suggestions or if like my post I would be glad to hear from you.
Reference:

"Matrix Multiplication with MapReduce." Lendapp, 17 May 2016, lendap.wordpress.com/2015/02/16/matrix-multiplication-with-mapreduce/.

The link to this article is as follows:

1. Google Drive: Blog MR5. Matrix Multiplication using MapReduce Programming in Java.
2. Blog Post: Post - MR5. Matrix Multiplication using MapReduce Programming in Java.
3. Follow My Blog: Follow Me Here.

4. QR Code: