**[lendapp](https://lendap.wordpress.com/)**

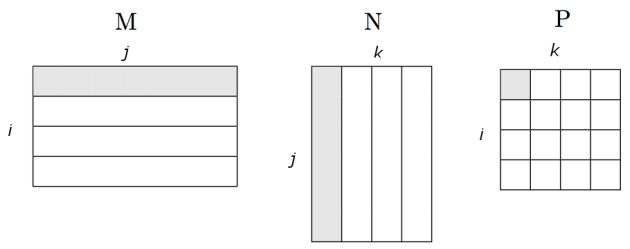
"knowledge grows when shared" – Bhartrihari

**Matrix Multiplication with MapReduce**

[33 Comments](https://lendap.wordpress.com/2015/02/16/matrix-multiplication-with-mapreduce/#respond)Posted by [Maruf Aytekin](https://lendap.wordpress.com/author/marufaytekin/" \o "Posts by Maruf Aytekin) on February 16, 2015

Matrix-vector and matrix-matrix calculations fit nicely into the MapReduce style of computing. In this post I will only examine matrix-matrix calculation as described in [1, ch.2].

Suppose we have a pxq matrix M, whose element in row *i* and column *j*will be denoted m_{ij} and a qxr matrix N whose element in row j and column k is donated by n_{jk} then the product P = MN will be pxr matrix P whose element in row i and column k will be donated by p_{ik}, where P(i,k) = m_{ij} * n_{jk}.



**Matrix Data Model for MapReduce**

We represent matrix M as a relation M(I,J,V), with tuples (i,j,m_{ij}), and matrix N as a relation N(J,K,W), with tuples (j,k,n_{jk}). Most matrices are sparse so large amount of cells have value zero. When we represent matrices in this form, we do not need to keep entries for the cells that have values of zero to save large amount of disk space. As input data files, we store matrix M and N on HDFS in following format:

M,i,j,m_{ij}

M,0,0,10.0

M,0,2,9.0

M,0,3,9.0

M,1,0,1.0

M,1,1,3.0

M,1,2,18.0

M,1,3,25.2

....

N,j,k,n_{jk}

N,0,0,1.0

N,0,2,3.0

N,0,4,2.0

N,1,0,2.0

N,3,2,-1.0

N,3,6,4.0

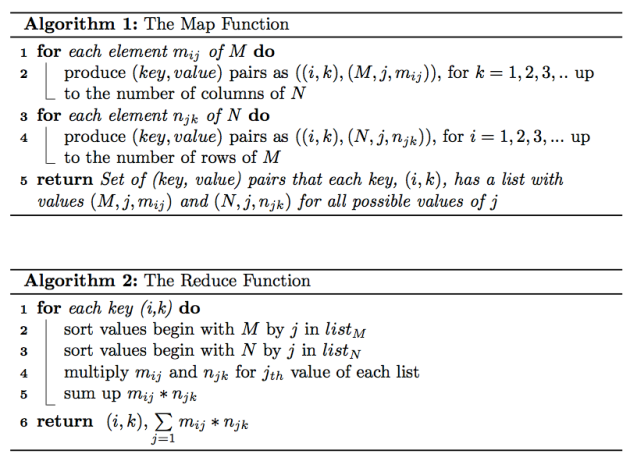
N,4,6,5.0

N,4,0,-1.0

....

**MapReduce**

We will write Map and Reduce functions to process input files. Map function will produce key,value pairs from the input data as it is described in Algorithm 1. Reduce function uses the output of the Map function and performs the calculations and produces key,value pairs as described in Algorithm 2. All outputs are written to HDFS.

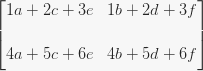


The value in row *i* and column *k* of product matrix *P* will be:

P_{(i,k)} = \sum\limits_{j=1} {m_{ij} * n_{jk}}.

Let me examine the algorithms on an example to explain the algorithms better. Suppose we have two matrices, *M*, 2×3 matrix, and *N*, 3×2 matrix as follows:

The product *P* of *MN* will be as follows:



**The Map Task**

For matrix *M,* map task (Algorithm 1) will produce key,value pairs as follows:

For matrix *N,* map task (Algorithm 2) will produce key,value pairs as follows:

After combine operation the map task will return key,value pairs as follows:

Note that the entries for the same key are grouped in the same list, which is performed by the framework. This  
output will be stored in HDFS and feed the reduce task as input.

**The Reduce Task**

Reduce task takes the $key,value$ pairs as the input and process one key at a time. For each key it divides the  
values in two separate lists for *M* and *N*. For key (1,1), the value is the list of [(M,1,1),(M,2,2),(M,3,3),(N,1,a,(N,2,c),(N,3,e)].

Reduce task sorts values begin with *M* in one list and values begin with *N* in another list as follows:

list_M = [(M,1,1),(M,2,2),(M,3,3)] \\  list_N = [(N,1,a),(N,2,c),(N,3,e)],

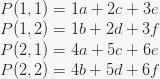
then sums up the multiplication of m_{ij} and n_{jk} for each j as follows:

P(1,1) = 1a+2c+3e

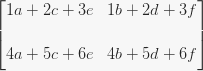
The same computation applied to all input entries of reduce task.

P_{(i,k)} = \sum\limits_{j=1} {m_{ij} * n_{jk}}

for all *i* and *k* is then calculated as follows:



The product matrix *P* of *MN* is then generated as:



**Experiments**

**Data**

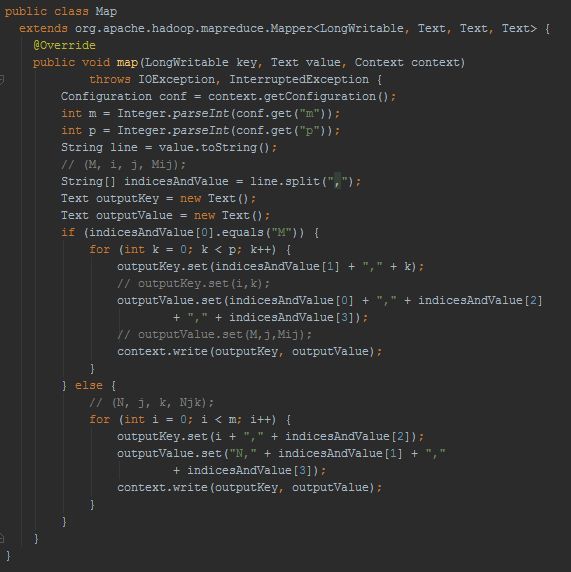
I have setup a single node Hadoop installation with HDFS and run the matrix  
calculation experiment on this installation. We used 1000 x 100 matrix *M* and  
100 x 1000 matrix *N* with sparsity level of 0.3. This means each matrix has about  
30K entries. The matrix les *M* and *N* stored in input directory on HDFS and  
the output of the computation is stored in output directory on HDFS.

**Source Code**

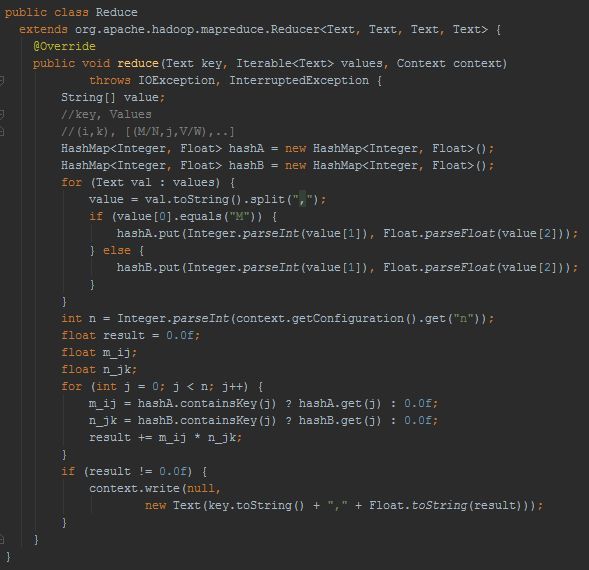
I developed mapper and reducer classes as Map.java and Reduce.java as well as the main application class called MatrixMultiply.java. As it seen in MatrixMultiply.java code below, in main method the configuration parameters are being set as well as the input/output directories of MapReduce job [2].

[](https://lendap.files.wordpress.com/2015/02/matrixmultiply-java.png)

Mapper class extends org.apache.hadoop.mapreduce.Mapper class and implements the map task described in Algorithm 1 and creates the key,value pairs from the input files as it shown in the code below:



Reducer class, extends org.apache.hadoop.mapreduce.Reducer class and implements the reduce task described in Algorithm 2 and creates the key,value pairs for the product matrix then writes its output on HDFS as it shown in the code below:

  
  
Complete source code  is [here.](https://github.com/marufaytekin/MatrixMultiply.git)

**References**

[1] Anand Rajaraman and Jerey David Ullman. Mining of Massive Datasets.  
Cambridge University Press, New York, NY, USA, 2011.

[2] One-step matrix multiplication with hadoop, [[Online](http://importantfish.com/one-step-matrix-multiplication-with-hadoop/)], 2014