Matrix Multiplication

Aaron Palumbo

9/12/2015

Contents

Matrix Multiplication

We are going to use Map Reduce to multiply together two matrices, both of size $n \times n$. For reference we assume the following terms $M \cdot N = P$.

Generate the Matrices

We will write the matrices to a local file, then transfer that file to hdfs. We do this to simulate working with matrices that are too big to fit into memory.

The file will be in csv format with the following information: –matrix, row, column, value

Here matrix will be either M or N

```
##################
# Run Parameters #
##################
# verify algorithm is working
# - this will attempt to load the matrices into
# - memory and multiply using conventional means

verify  <- TRUE
# easier to check if things are working if FALSE
randVals <- FALSE

##############
# File Setup #
##############
# We will use this method to simulate working with a matrix
# that is too big for memory
fn <- "matrix.csv"
if (file.exists(fn)) file.remove(fn)
```

```
## [1] TRUE
```

```
##########################
# Matrix Characteristics #
##########################

## size of matrix (n x n square)
# -- both M and N are the same size
n <- 6

## row column groups
# -- we will be breaking down the matrix into groups
# -- of rows and columns. How many per group?
g <- 2

## density of matrix (0-1)
# this value corresponds to the probability of
# entry i,j not being 0
p <- 1

## max value - maximum value for entry in matrix (integer)
maxVal <- 10

#####################
# Generate Matrices #
#####################

create_paste_function <- function(fileName){
  f <- function(matName, i, j, value){
    cat(paste0(matName, ", ", i, ", ", j, ", ", value),
        file=fileName,
        sep="\n",
        append=TRUE)
  }
  return(f)
}

append_to_file <- create_paste_function(fn)

return_val <- function(i, j, randVals=TRUE){
  if (randVals){
        return(as.integer(runif(1, 1, maxVal)))
      } else {
        return(i + j)
      }
}

# Stream data to file
for (i in 1:n){
  for (j in 1:n){
    if (rbinom(1, 1, p) == 1){
      append_to_file("M", i, j, return_val(i, j, randVals))
    }
    if (rbinom(1, 1, p) == 1){
```

```r
      append_to_file("N", i, j, return_val(i, j, randVals))
    }
  }
}

###########################
# Read Matrices from file #
###########################
# This is for debugging

if (verify){
  # read M and N into memory
  df <- read.csv(fn, header=FALSE)
  names(df) <- c("name", "i", "j", "val")

  df <- df[order(df$name, df$j, df$i), ]

  matList <- list(M=matrix(0, nrow=n, ncol=n),
                  N=matrix(0, nrow=n, ncol=n))

  for (r in 1:nrow(df)){
    matList[[df$name[r]]][df$i[r], df$j[r]] <- df$val[r]
  }
}

if (exists("matList")){
  matList$M
  matList$N
}
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    2    3    4    5    6    7
## [2,]    3    4    5    6    7    8
## [3,]    4    5    6    7    8    9
## [4,]    5    6    7    8    9   10
## [5,]    6    7    8    9   10   11
## [6,]    7    8    9   10   11   12
```

Push matrices into hdfs

```r
library(rhdfs)
```

```
## Loading required package: rJava
##
## HADOOP_CMD=/home/apalumbo/workspace/cuny_msda_is622/hadoop-2.7.1/bin/hadoop
##
## Be sure to run hdfs.init()
```

```r
hdfs.init()
```

```r
hdfs_dir <- "/user/apalumbo/mat_mul/"

hdfs.put(fn, hdfs_dir)  # this will overwrite
```

```
## [1] TRUE
```

Matrix Multiplication with Map Reduce

Group Hash

We will be breaking down the matrix into a grid for processing. Each cell in the grid will be g rows by g cols. Here we set up a hash function to sort row or column numbers into the correct grid.

```r
groupHash <- function(rcNum, groups){
  hash <- as.integer((rcNum + groups - 1) / groups)
}

maxg <- groupHash(n, g)
```

First Iteration

input line of the form: c(matrix, row, column, value) map (m, r, c, v) to key: (keyHash(r), keyHash(c), 1:(ceiling(n/g))) value: (m, r, c, v) map (n, r, c, v) to key: (1:ceiling(n/g), keyHash(r), keyHash(c)) value: (n, r, c, v)

First Reduce: key: (i, j, k) will have a value list like, for example, this: m, 1, 1, m11 m, 1, 2, m12 m, 2, 1, m21 m, 2, 2, m22 n, 1, 1, n11 n, 1, 2, n12 n, 2, 1, n21 n, 2, 2, n22

output: key: (i, k) values: for each key in m: –r = r(m) –j = c(m) –for each key in n where r(n) == j: —-c = c(n) —-# there will only be one pair that fits this —-create key value{key: (i, k) value: (r, c, mrj * njc)}

combiner: now locally combine keys:

(first reduce is also second map)

```r
library(rmr2)
```

```
## Warning: S3 methods 'gorder.default', 'gorder.factor', 'gorder.data.frame',
## 'gorder.matrix', 'gorder.raw' were declared in NAMESPACE but not found

## Please review your hadoop settings. See help(hadoop.settings)
```

```r
mapper <- function(null, val){
  mat <- val[1]
  r <- as.numeric(val[[2]])
  c <- as.numeric(val[[3]])
  entry <- as.numeric(val[[4]])
  if (mat == 'M'){
    df.key <- data.frame(
      i = groupHash(r, g),
      j = groupHash(c, g),
      k = 1:maxg
      )
  }
  if (mat == 'N'){
    df.key <- data.frame(
      i = 1:maxg,
      j = groupHash(r, g),
      k = groupHash(c, g)
      )
  }
  df.val <- data.frame(n=mat,
                       r=r,
                       c=c,
                       v=entry,
                       rep=1:maxg)[ ,1:4]
  return( keyval(df.key, df.val) )
}

reducer <- function(key, val){
  df.key <- data.frame(
    i=key[[1]],
    k=key[[3]]
    )
  v <- length(val)
  return(keyval(key, val))
}

path <- paste0(hdfs_dir, fn)
hdfsFile <- hdfs.file(path, "r")

mr <- mapreduce(
  input = paste0(hdfs_dir, fn),
  input.format = make.input.format(format="csv", sep=","),
  map = mapper,
  reduce = reducer
  )

if (exists('result')){
  rm(result)
}

result <- from.dfs(mr)

result
```

```
## $key
##      i j k
## 1    1 1 1
## 1.1  1 1 1
## 1.2  1 1 1
## 1.3  1 1 1
## 2    1 1 2
## 2.1  1 1 2
## 3    1 1 3
## 3.1  1 1 3
## 1.4  1 2 1
## 1.5  1 2 1
## 2.2  1 2 2
## 2.3  1 2 2
## 2.4  1 2 2
## 2.5  1 2 2
## 3.2  1 2 3
## 3.3  1 2 3
## 1.6  1 3 1
## 1.7  1 3 1
## 2.6  1 3 2
## 2.7  1 3 2
## 3.4  1 3 3
## 3.5  1 3 3
## 3.6  1 3 3
## 3.7  1 3 3
## 1.8  2 1 1
## 1.9  2 1 1
## 1.10 2 1 1
## 1.11 2 1 1
## 2.8  2 1 2
## 2.9  2 1 2
## 3.8  2 1 3
## 3.9  2 1 3
## 1.12 2 2 1
## 1.13 2 2 1
## 2.10 2 2 2
## 2.11 2 2 2
## 2.12 2 2 2
## 2.13 2 2 2
## 3.10 2 2 3
## 3.11 2 2 3
## 1.14 2 3 1
## 1.15 2 3 1
## 2.14 2 3 2
## 2.15 2 3 2
## 3.12 2 3 3
## 3.13 2 3 3
## 3.14 2 3 3
## 3.15 2 3 3
## 1.16 3 1 1
## 1.17 3 1 1
## 1.18 3 1 1
## 1.19 3 1 1
## 2.16 3 1 2
## 2.17 3 1 2
## 3.16 3 1 3
## 3.17 3 1 3
## 1.20 3 2 1
## 1.21 3 2 1
## 2.18 3 2 2
## 2.19 3 2 2
## 2.20 3 2 2
## 2.21 3 2 2
## 3.18 3 2 3
## 3.19 3 2 3
## 1.22 3 3 1
## 1.23 3 3 1
## 2.22 3 3 2
## 2.23 3 3 2
## 3.20 3 3 3
## 3.21 3 3 3
## 3.22 3 3 3
## 3.23 3 3 3
##
## $val
##    V1 r c  v
## 1   M 1 1  2
## 4   N 1 2  3
## 13  M 2 1  3
## 16  N 2 2  4
## 2   N 1 1  2
## 14  N 2 1  3
## 3   M 1 2  3
## 15  M 2 2  4
## 7   M 1 4  5
## 19  M 2 4  6
## 5   M 1 3  4
## 8   N 1 4  5
## 17  M 2 3  5
## 20  N 2 4  6
## 6   N 1 3  4
## 18  N 2 3  5
## 10  N 1 5  6
## 22  N 2 5  7
## 11  M 1 6  7
## 23  M 2 6  8
## 9   M 1 5  6
## 12  N 1 6  7
## 21  M 2 5  7
## 24  N 2 6  8
## 25  M 3 1  4
## 28  N 3 2  5
## 37  M 4 1  5
## 40  N 4 2  6
## 26  N 3 1  4
## 38  N 4 1  5
## 27  M 3 2  5
## 39  M 4 2  6
## 31  M 3 4  7
## 43  M 4 4  8
## 29  M 3 3  6
## 32  N 3 4  7
## 41  M 4 3  7
## 44  N 4 4  8
## 30  N 3 3  6
## 42  N 4 3  7
## 34  N 3 5  8
## 46  N 4 5  9
## 35  M 3 6  9
## 47  M 4 6 10
## 33  M 3 5  8
## 36  N 3 6  9
## 45  M 4 5  9
## 48  N 4 6 10
## 49  M 5 1  6
## 52  N 5 2  7
## 61  M 6 1  7
## 64  N 6 2  8
## 50  N 5 1  6
## 62  N 6 1  7
## 51  M 5 2  7
## 63  M 6 2  8
## 55  M 5 4  9
## 67  M 6 4 10
## 53  M 5 3  8
## 56  N 5 4  9
## 65  M 6 3  9
## 68  N 6 4 10
## 54  N 5 3  8
## 66  N 6 3  9
## 58  N 5 5 10
## 70  N 6 5 11
## 59  M 5 6 11
## 71  M 6 6 12
## 57  M 5 5 10
## 60  N 5 6 11
## 69  M 6 5 11
## 72  N 6 6 12
```

final map:

sum values

```
###############
# Simple Test #
###############

mapper <- function(null, val){
  m <- val[[1]]
  r <- val[[2]]
  c <- val[[3]]
  s <- val[[4]]
  v <- val[[5]]

  if (m == 'm'){
    df.key <- data.frame(r=c(r, r),
                         c=c(1, 2))
  }
  if (m == 'n'){
    df.key <- data.frame(r=c(1, 2),
                         c=c(c, c))
  }
  return(keyval(df.key, v))
}

reducer <- function(key, val.list){
  return(keyval(key, val.list))
}

small.df <- data.frame(m=c(rep('m', 4), rep('n', 4)),
                       r=c(1, 1, 2, 2, 1, 1, 2, 2),
                       c=c(1, 2, 1, 2, 1, 2, 1, 2),
                       s=letters[1:8],
                       v=1:8)

mat <- to.dfs(small.df)
mr <- mapreduce(input=mat, map=mapper, reduce=reducer)
if (exists('result')){
  rm(result)
}

result <- from.dfs(mr)

print(cbind(result$key, result$val))

lst <- list()
for (i in 1:nrow(small.df)){
  print(i)
  r <- mapper(0, small.df[i,])
  lst[[i]] <- cbind(r$key, r$val)
}
do.call(rbind, lst)
```

```
###################
# Delete this one #
###################

library(rmr2)

mapper <- function(null, val){
  mat <- val[1]
  r <- val[2]
  c <- val[3]
  entry <- as.numeric(val[[4]])

  k <- data.frame(i=r, j=c, k=1:maxg)
  v <- data.frame(m=mat, i=r, j=c, v=entry)
  return(keyval(k, v))
}

reducer <- function(key, val){
  return(keyval(key, val))
}
```

```
path <- paste0(hdfs_dir, fn)
hdfsFile <- hdfs.file(path, "r")

mr <- mapreduce(
  input = paste0(hdfs_dir, fn),
  input.format = make.input.format(format="csv", sep=","),
  map = mapper,
  reduce = reducer
  )

result <- from.dfs(mr)

result
```

```
## $key
##      V2 V3 k
## 1     1  1 1
## 2     1  1 2
## 1.1   1  2 1
## 3     1  2 3
## 2.1   1  3 2
## 3.1   1  3 3
## 1.2   1  4 1
## 2.2   1  4 2
## 1.3   1  5 1
## 3.2   1  5 3
## 2.3   1  6 2
## 3.3   1  6 3
## 1.4   2  1 1
## 2.4   2  1 2
## 1.5   2  2 1
## 3.4   2  2 3
## 2.5   2  3 2
## 3.5   2  3 3
## 1.6   2  4 1
## 2.6   2  4 2
## 1.7   2  5 1
## 3.6   2  5 3
## 2.7   2  6 2
## 3.7   2  6 3
## 1.8   3  1 1
## 2.8   3  1 2
## 1.9   3  2 1
## 3.8   3  2 3
## 2.9   3  3 2
## 3.9   3  3 3
## 1.10  3  4 1
## 2.10  3  4 2
## 1.11  3  5 1
## 3.10  3  5 3
## 2.11  3  6 2
## 3.11  3  6 3
## 1.12  4  1 1
## 2.12  4  1 2
## 1.13  4  2 1
## 3.12  4  2 3
## 2.13  4  3 2
## 3.13  4  3 3
## 1.14  4  4 1
## 2.14  4  4 2
## 1.15  4  5 1
## 3.14  4  5 3
## 2.15  4  6 2
## 3.15  4  6 3
## 1.16  5  1 1
## 2.16  5  1 2
## 1.17  5  2 1
## 3.16  5  2 3
## 2.17  5  3 2
## 3.17  5  3 3
## 1.18  5  4 1
## 2.18  5  4 2
## 1.19  5  5 1
## 3.18  5  5 3
## 2.19  5  6 2
## 3.19  5  6 3
## 1.20  6  1 1
## 2.20  6  1 2
## 1.21  6  2 1
## 3.20  6  2 3
## 2.21  6  3 2
## 3.21  6  3 3
## 1.22  6  4 1
## 2.22  6  4 2
## 1.23  6  5 1
## 3.22  6  5 3
## 2.23  6  6 2
## 3.23  6  6 3
##
## $val
##    V1 V2 V3  v
## 1   M  1  1  2
## 2   N  1  1  2
## 4   N  1  2  3
## 3   M  1  2  3
## 5   M  1  3  4
## 6   N  1  3  4
## 7   M  1  4  5
## 8   N  1  4  5
## 10  N  1  5  6
## 9   M  1  5  6
## 11  M  1  6  7
## 12  N  1  6  7
## 13  M  2  1  3
## 14  N  2  1  3
## 16  N  2  2  4
## 15  M  2  2  4
## 17  M  2  3  5
## 18  N  2  3  5
## 19  M  2  4  6
## 20  N  2  4  6
## 22  N  2  5  7
## 21  M  2  5  7
## 23  M  2  6  8
## 24  N  2  6  8
## 25  M  3  1  4
## 26  N  3  1  4
## 28  N  3  2  5
## 27  M  3  2  5
## 29  M  3  3  6
## 30  N  3  3  6
## 31  M  3  4  7
## 32  N  3  4  7
## 34  N  3  5  8
## 33  M  3  5  8
## 35  M  3  6  9
## 36  N  3  6  9
## 37  M  4  1  5
## 38  N  4  1  5
## 40  N  4  2  6
## 39  M  4  2  6
## 41  M  4  3  7
## 42  N  4  3  7
## 43  M  4  4  8
## 44  N  4  4  8
## 46  N  4  5  9
## 45  M  4  5  9
## 47  M  4  6 10
## 48  N  4  6 10
## 49  M  5  1  6
## 50  N  5  1  6
## 52  N  5  2  7
## 51  M  5  2  7
## 53  M  5  3  8
## 54  N  5  3  8
## 55  M  5  4  9
## 56  N  5  4  9
## 58  N  5  5 10
## 57  M  5  5 10
## 59  M  5  6 11
## 60  N  5  6 11
## 61  M  6  1  7
## 62  N  6  1  7
## 64  N  6  2  8
## 63  M  6  2  8
## 65  M  6  3  9
## 66  N  6  3  9
## 67  M  6  4 10
## 68  N  6  4 10
## 70  N  6  5 11
## 69  M  6  5 11
## 71  M  6  6 12
## 72  N  6  6 12
```