Final Project - Hadoop

Aaron Palumbo

12/19/2015

Setup / Dependencies

```
hadoopHome =
  "/home/apalumbo/workspace/cuny_msda_is622/hadoop-2.7.1"

hadoopCmd =
  file.path(hadoopHome, "/bin/hadoop")

hadoopStreaming =
  file.path(hadoopHome,
    "/share/hadoop/tools/lib/hadoop-streaming-2.7.1.jar")

Sys.setenv(HADOOP_HOME       = hadoopHome     )
Sys.setenv(HADOOP_CMD        = hadoopCmd      )
Sys.setenv(HADOOOP_STREAMING = hadoopStreaming)

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)
```

```
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()
```

```
hdfs.init()
```

Global Variables

```
filePath = "/user/apalumbo/final/train_triplets.txt"
# debug version
filePath = "/user/apalumbo/final/train_triplets_100.txt"
colNames <- c("user", "song", "playCount")
```

First we will attempt to determine some basic information about the data.

Number of Rows

```
fileOut  = "/user/apalumbo/final/hadoop_rows.out"

# need to make sure output file does not exist
if ( hdfs.exists(fileOut) ) {
  hdfs.rm(fileOut)
}
```

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

```
begin <- Sys.time()

out <- mapreduce(
  input = filePath,
  input.format = "text",
  output = fileOut,

  map = function(k, v)
     keyval( "line", length(v) ),

  reduce = function(k, v)
    keyval(k, sum(v))
)

finish <- Sys.time()

# Results
(numLines <- from.dfs(out))
```

```
## $key
## [1] "line"
##
## $val
## [1] 100
```

```
(finish - begin)
```

```
## Time difference of 6.804883 secs
```

We see there are 100 records and it took use 6.8048832 seconds to determine that.

Unique Users

Now let's determine how many unique users we have.

```
# set output file and make sure it doesn't exist
fileOut  = "/user/apalumbo/final/hadoop_users.out"

if ( hdfs.exists(fileOut) ) {
  hdfs.rm(fileOut)
}

splitLines <- function(line) {
  # each line comes in as a string - split on tab
  vec <- unlist(strsplit(line, '\t'))
  # transform to data frame
  df <- as.data.frame(matrix(vec, nrow=1))
  colnames(df) <- c("user", "song", "playCount")
  rownames(df) <- NULL
  # transform playCount from string to int
  df$playCount <- as.integer(as.character(df$playCount))
  return(df)
}

mapper <- function(k, v) {
  # stack lines as data frame
  df <- as.data.frame(do.call(rbind, lapply(v, splitLines)))
  return(keyval(df$user, 1))
}

begin <- Sys.time()

# We need two reductions for this:
out <- mapreduce(
  input =  mapreduce(
    input = filePath,
    input.format = "text",
    map = mapper,
    reduce = function(k, v)
      keyval(k, 1),
    ## Use the reducer as a local combiner
    combine=TRUE
  ),
  output = fileOut,
  reduce = function(k, v)
    keyval("unique_users", sum(v)),
  ## Use the reducer as a local combiner
  combine=TRUE
)

finish <- Sys.time()

# Results
numUsers <- from.dfs(out)
(finish - begin)
```

```
## Time difference of 17.52907 secs
```

As we see from above, it takes us 17.5290711 seconds to determine there are 100 unique users in the database.

Unique Songs

```
# set output file and make sure it doesn't exist
fileOut  = "/user/apalumbo/final/hadoop_songs.out"

if ( hdfs.exists(fileOut) ) {
  hdfs.rm(fileOut)
}

# We can use the splitLines function from above

mapper <- function(k, v) {
  # stack lines as data frame
  df <- as.data.frame(do.call(rbind, lapply(v, splitLines)))
  return(keyval(df$song, 1))
}

begin <- Sys.time()

# We need two reductions for this:
out <- mapreduce(
  input =  mapreduce(
    input = filePath,
    input.format = "text",
    map = mapper,
    reduce = function(k, v)
      keyval(k, 1),
    ## Use the reducer as a local combiner
    combine=TRUE
  ),
  output = fileOut,
  reduce = function(k, v)
    keyval("unique_users", sum(v)),
  ## Use the reducer as a local combiner
  combine=TRUE
)

finish <- Sys.time()

# Results
numSongs <- from.dfs(out)
(finish - begin)
```

```
## Time difference of 24.77922 secs
```

Now we see that there are 99 unique songs in the database and that it took us 24.7792192 seconds to determine that with hadoop.

Most Popular Songs

A simple recommendation system might be based on what's popular. Let's see if we can determine that using hadoop.

There are several approaches you could take to this using the map reduce paradigm. The approach we will take here is to first group the songs by number of plays. This should return a very reduced set of cadidates. The susequent mapreduce job can be treated more like a filter with a quick in memory sort at the end. This type of approach would also work better for repeated queries (e.g. top 10, top 15, top 20).

```
# set output file and make sure it doesn't exist
fileOut = "/user/apalumbo/final/hadoop_playcount.out"

if ( hdfs.exists(fileOut) ) {
  hdfs.rm(fileOut)
}

# We can use the splitLines function from above

begin <- Sys.time()

# .pc - play count
mapper.pc <- function(k, v) {
  # stack lines as data frame
  df <- as.data.frame(do.call(rbind, lapply(v, splitLines)))
  return(keyval(df$song, df$playCount))
}

out <- mapreduce(
  input = filePath,
  input.format = "text",
  output = fileOut,
  map = mapper.pc,
  # for each song sum playcounts
  reduce = function(k, v)
    keyval(k, sum(v)),
  ## Use the reducer as a local combiner
  combine=TRUE
)

sumPlayCount <- out

# At this point we have unique songs with total play count
# Now we group by playCount and return that

# set output file and make sure it doesn't exist
fileOut = "/user/apalumbo/final/hadoop_playcount-groups.out"

if ( hdfs.exists(fileOut) ) {
  hdfs.rm(fileOut)
}

out <- mapreduce(
  input = sumPlayCount,
  output = fileOut,
  map = function(k, v)
    keyval(v, 1),
  reduce = function(k, v)
    keyval(k, sum(v)),
```

```
  combine=TRUE
)

playcountGroups <- out

# set output file and make sure it doesn't exist
fileOut = "/user/apalumbo/final/hadoop_popular.out"

if ( hdfs.exists(fileOut) ) {
  hdfs.rm(fileOut)
}
# We will start by returning the top 10
top <- 10
pcGroups <- as.data.frame(from.dfs(playcountGroups))
print(nrow(pcGroups))
```

```
## [1] 11
```

```
pcGroups <- pcGroups[order(pcGroups$key, decreasing=TRUE), ]
pcGroups['cumsum'] <- cumsum(pcGroups$val)
minPlayCount <- pcGroups$key[sum(pcGroups$cumsum < top) + 1]

mapper <- function(k, v) {
  filtr <- v >= minPlayCount
  return(keyval(k[filtr], v[filtr]))
}

out <- mapreduce(
  input = sumPlayCount,
  output = fileOut,
  map=mapper
)

finish <- Sys.time()

# Results
topSongs <- as.data.frame(from.dfs(out))
(topSongs <- topSongs[order(topSongs$val, decreasing=TRUE), ])
```

```
##                   key val
## 7  SORBIYI12A6D4F7496  23
## 3  SODVHDW12A8C134043  16
## 11 SOVBAPB12A8AE48D82  12
## 6  SOPLXDP12A8C138364  10
## 12 SOVCFFC12A58A7B4EC   8
## 2  SOBRROM12A8AE4828A   6
## 4  SOFGJCW12AF72A812D   6
## 10 SOUSMXX12AB0185C24   6
## 1  SOBMRKB12A8AE481D0   5
## 5  SOKTJKI12A8C143254   5
## 8  SOTRNNH12A6D4FA411   5
## 9  SOUCHPA12AB0184B1A   5
## 13 SOVRZIX12AAF3B2A32   5
```

```
(finish - begin)
```

```
## Time difference of 37.68857 secs
```