Implementing MapReduce programs: RHadoop

Antonino Virgillito
RHadoop

- Collection of packages that allows integration of R with HDFS and MapReduce
- Hadoop provides the storage while R brings the processing
- Just a library
  - Not a special run-time, Not a different language, Not a special purpose language
- Incrementally port your code and use all packages
- Requires R installed and configured on all nodes in the cluster
Prerequisites

- Installation of Hadoop cluster
- Installation of R
- Installation of RHadoop packages
- Environment variables
  - HADOOP_CMD
  - HADOOP_STREAMING

As configured in the Sandbox:

```bash
export HADOOP_STREAMING=/usr/lib/hadoop-mapreduce/hadoop-streaming-2.2.0.2.0.6.0-102.jar
export HADOOP_CMD=/usr/bin/hadoop
```
RHadoop Packages

- rhdfs
  - Interface for reading and writing files from/to a HDFS cluster
- rmr2
  - Interface to MapReduce through R
- rhbase
  - Interface to HBase
rhdfs

- As Hadoop MapReduce programs use HDFS for taking their input and writing their output, it is necessary to access them from R console
- The R programmer can easily perform read and write operations on distributed data files.
- Basically, rhdfs package calls the HDFS API in backend to operate data sources stored on HDFS.
rhdfs Functions

- **File Manipulations**
  - `hdfs.copy`, `hdfs.move`, `hdfs.rename`, `hdfs.delete`, `hdfs.rm`, `hdfs.del`, `hdfs.chown`, `hdfs.put`, `hdfs.get`

- **File Read/Write**
  - `hdfs.file`, `hdfs.write`, `hdfs.close`, `hdfs.flush`, `hdfs.read`, `hdfs.seek`, `hdfs.tell`, `hdfs.line.reader`, `hdfs.read.text.file`

- **Directory**
  - `hdfs.dircreate`, `hdfs.mkdir`

- **Utility**
  - `hdfs.ls`, `hdfs.list.files`, `hdfs.file.info`, `hdfs.exists`

- **Initialization**
  - `hdfs.init`, `hdfs.defaults`
rmr2

- rmr2 is an R interface for providing Hadoop MapReduce facility inside the R environment.
- So, the R programmer needs to just divide their application logic into the map and reduce phases and submit it with the rmr2 methods.
- After that, rmr2 calls the Hadoop streaming MapReduce API with several job parameters as input directory, output directory, mapper, reducer, and so on, to perform the R MapReduce job over Hadoop cluster.
rhbase

- R interface for operating the Hadoop HBase data source stored at the distributed network via a Thrift server.
- The rhbase package is designed with several methods for initialization and read/write and table manipulation operations.
Our First mapreduce Job

• Compute the first thousand squares

Regular R implementation

```r
small.ints = 1:1000
sapply(small.ints, function(x) x^2)
```

mapreduce equivalent

```r
library('rhdfs')
library('rmr2')
hdfs.init()

small.ints = to.dfs(1:1000)
mapreduce(
  input = small.ints,
  map = function(k, v) cbind(v, v^2))
```
to.dfs

• It is not possible to write out big data with to.dfs, not in a scalable way.
  • useful for writing test cases, learning and debugging
• to.dfs can put the data in a file of your own choosing, but if you don't specify one it will create temp files and clean them up when done.
• The return value is something we call a big data object.
• You can assign it to variables, pass it to other rmr functions, mapreduce jobs or read it back in.
• It is a stub, that is the data is not in memory, only some information that helps finding and managing the data.
• This way you can refer to very large data sets whose size exceeds memory limits.
mapreduce

• The mapreduce function takes as input a set of named parameters
  • **input**: input path or variable
  • **input.format**: specification of input format
  • **output**: output path or variable
  • **map**: map function
  • **reduce**: reduce function

  • map and reduce function present the usual interface

  • A call to keyval(k,v) inside the map and reduce function is used to emit respectively intermediate and output key-value pairs
from.dfs

- from.dfs is complementary to to.dfs and returns a key-value pair collection that can be passed to mapreduce jobs or read into memory
  - watch out, it will fail for big data!
- from.dfs is useful in defining map reduce algorithms whenever a mapreduce job produces something of reasonable size, like a summary, that can fit in memory and needs to be inspected to decide on the next steps, or to visualize it.
- It is much more important than to.dfs in production work.
Our Second mapreduce Job

Creates a sample from the binomial distribution and counts how many times each outcome occurred

```r
library('rhdfs')
library('rmr2')
hdfs.init()

groups = rbinom(32, n = 50, prob = 0.4)
groups = to.dfs(groups)
from.dfs(
  mapreduce(
    input = groups,
    map = function(., v) keyval(v, 1),
    reduce =
      function(k, vv)
        keyval(k, length(vv))
  ))
```
WordCount in R

```r
wordcount =
  function(
    input,
    output = NULL,
    pattern = " "){

wc.map =
  function(., lines) {
    keyval(
      unlist(
        strsplit(
          x = lines,
          split = pattern)),
      1)
  }

wc.reduce =
  function(word, counts ) {
    keyval(word, sum(counts))}

mapreduce(
  input = input ,
  output = output,
  input.format = "text",
  map = wc.map,
  reduce = wc.reduce,
  combine = T)
```
Reading delimited data

tsv.reader = function(con, nrecs){
lines = readLines(con, 1)
if(length(lines) == 0)
  NULL
else {
  delim = strsplit(lines, split = "\t")
  keyval(
    sapply(delim, function(x) x[1]),
    sapply(delim, function(x) x[-1])))}

freq.counts = mapreduce(
  input = tsv.data,
  input.format = tsv.format,
  map = function(k, v) keyval(v[1,], 1),
  reduce = function(k, vv) keyval(k, sum(vv)))
Reading named columns

tsv.reader = function(con, nrecs){
  lines = readLines(con, 1)
  if(length(lines) == 0)
    NULL
  else {
    delim = strsplit(lines, split = "\t")
    keyval(sapply(delim, function(x) x[1]),
           data.frame(
                     location = sapply(delim, function(x) x[2]),
                     name = sapply(delim, function(x) x[3]),
                     value = sapply(delim, function(x) x[4]))))}

freq.counts = mapreduce(
  input = tsv.data,
  input.format = tsv.format,
  map = function(k, v) {
    filter = (v$name == "blarg")
    keyval(k[filter], log(as.numeric(v$value[filter])))},
  reduce = function(k, vv) keyval(k, mean(vv)))