Getting Started with R & Hadoop

From Local VM to the Cloud

TDWI World Boston

Pre-Conference Workshop

Hynes Convention Center
Boston, MA

Saturday, September 15, 2012

by **Jeffrey Breen**

email: jeffrey@jeffreybreen.com
http://jeffreybreen.wordpress.com
Twitter: @JeffreyBreen

http://bit.ly/tdwibos

Part 2: Introduction to R & Hadoop

Code & more on github:

http://bit.ly/tdwibos
(https://github.com/jeffreybreen/tutorial-201209-TDWI-big-data)

Outline

- Why MapReduce? Why R?
- R + Hadoop options
- RHadoop overview
- Step-by-step examples
- Advanced RHadoop features

Outline

- Why MapReduce? Why R?
- R + Hadoop options
- RHadoop overview
- Step-by-step examples
- Advanced RHadoop features

Why MapReduce? Why R?

- MapReduce is a programming pattern to aid in the parallel analysis of data
  - Popularized, but not invented, by Google
  - Named from its two primary steps: a "map" phase which picks out the identifying and subject data ("key" and "value") and a "reduce" phase where the values (grouped by key value) are analyzed
  - Generally, the programmer/analyst need only write the mapper and reducer while the system handles the rest
- R is an open source environment for statistical programming and analysis
  - Open source and wide platform support makes it easy to try out at work or "nights and weekends"
  - Benefits from an active, growing community
  - Offers a (too) large library of add-on packages (see http://cran.revolutionanalytics.com/)
  - Commercial support, extensions, training is available

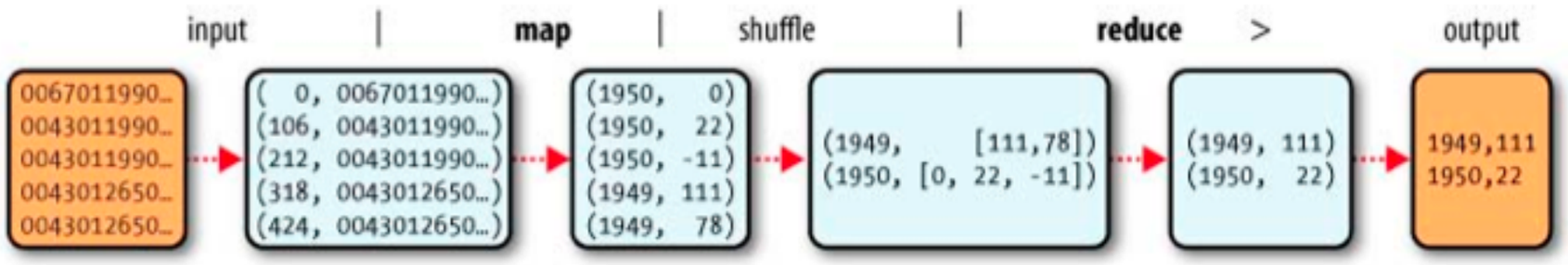
Before we go on...

I have two confessions

I was wrong about MapReduce

- When the Google paper was published in 2004, I was running a typical enterprise IT department
- Big hardware (Sun, EMC) + big applications (Siebel, Peoplesoft) + big databases (Oracle, SQL Server)

  = big licensing/maintenance bills
- Loved the scalability, COTS components, and price, but missed the fact that keys (and values) could be compound & complex

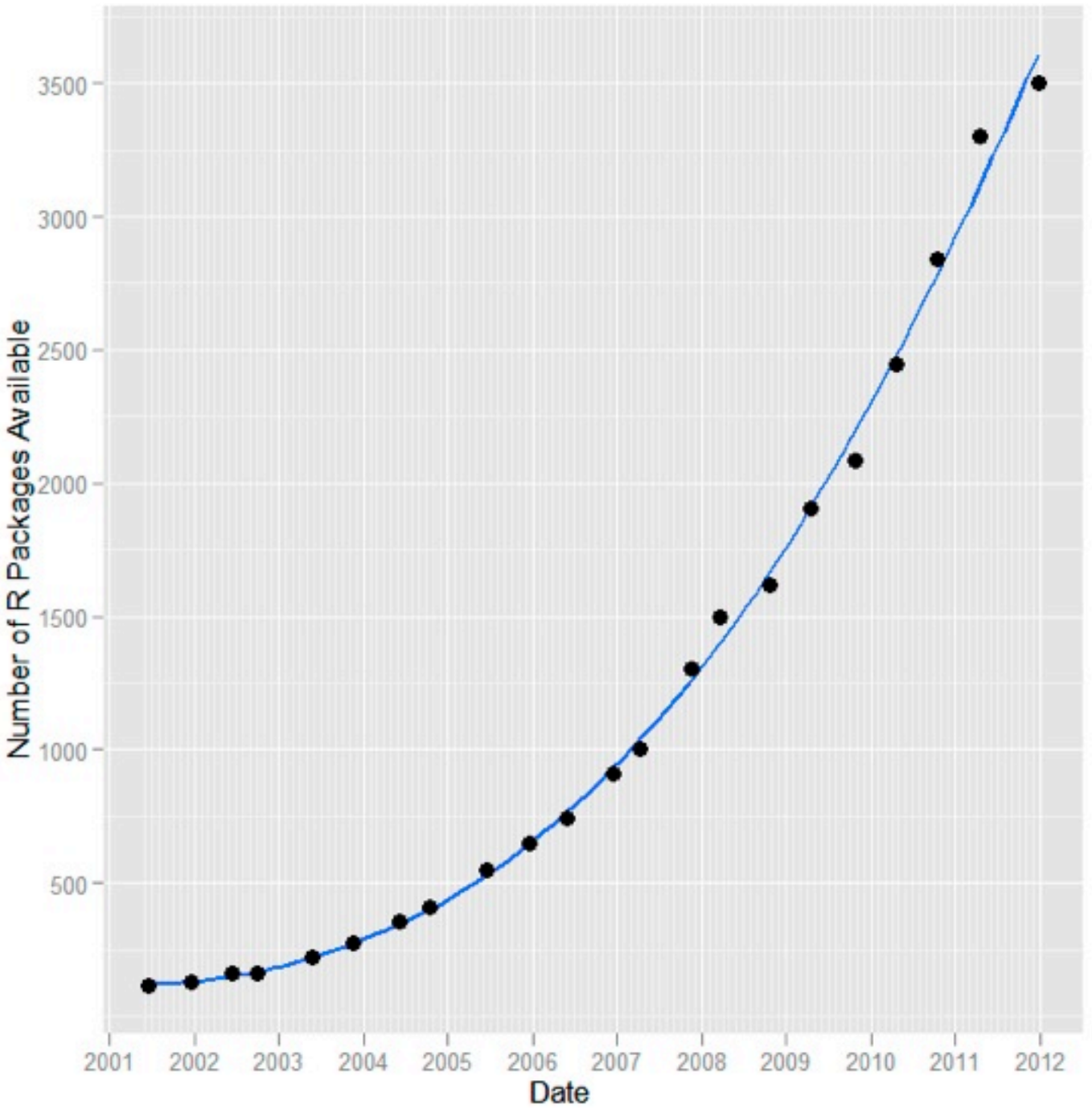


Source: *Hadoop: The Definitive Guide, Second Edition*, p. 20

And I was wrong about R

- In 1990, my boss (an astronomer) encouraged me to learn S or S+
- But I knew C, so I resisted, just as I had successfully fended off the FORTRAN-pushing physicists
- 20 years later, it's my go-to tool for anything data-related
    - I rediscovered it when we were looking for a way to automate the analysis and delivery of our consumer survey data at Yankee Group

Number of R Packages Available



How many R Packages are there now?

At the command line enter:

```
> dim(available.packages())
```

Slide courtesy of John Versotek, organizer of the Boston Predictive Analytics Meetup

Outline

- Why MapReduce? Why R?
- R + Hadoop options
- RHadoop overview
- Step-by-step examples
- Advanced RHadoop features

R + Hadoop options

- Hadoop streaming enables the creation of mappers, reducers, combiners, etc. in languages other than Java
  - Any language which can handle standard, text-based input & output will do
  - R is designed at its heart to deal with data and statistics making it a natural match for Big Data-driven analytics
  - As a result, there are a number of R packages to work with Hadoop

There's never just one R package to do anything...

Package	Latest Release *(as of 2012-07-09)*	Comments
hive	v0.1-15: 2012-06-22	misleading name: stands for "Hadoop interactIVE" & has nothing to do with Hadoop hive. On CRAN.
HadoopStreaming	v0.2: 2010-04-22	focused on utility functions: I/O parsing, data conversions, etc. Available on CRAN.
RHIPE	v0.69: "11 days ago"	comprehensive: code & submit jobs, access HDFS, etc. Unfortunately, most links to it are broken. Look on github instead: https://github.com/saptarshiguha/RHIPE/
segue	v0.04: 2012-06-05	JD Long's very clever way to use Amazon EMR with small or no data. http://code.google.com/p/segue/
RHadoop (rmr, rhdfs, rhbase)	rmr 1.2.2: "3 months ago" rhdfs 1.0.3 "2 months ago" rhbase 1.0.4 "3 months ago"	Divided into separate packages by purpose: • rmr - all MapReduce-related functions • rhdfs - management of Hadoop's HDFS file system • rhbase - access to HBase database Sponsored by Revolution Analytics & on github: https://github.com/RevolutionAnalytics/RHadoop

Any more?

- Yeah, probably. My apologies to the authors of any relevant packages I may have overlooked.
- R is nothing if it's not flexible when it comes to consuming data from other systems
  - You could just use R to analyze the output of existing MapReduce jobs and workflows
  - R can connect via ODBC and/or JDBC, so you could connect to Hive as if it were just another database
- So... how to pick?

Photo credit: http://en.wikipedia.org/wiki/File:Darts_in_a_dartboard.jpg

Thanks, Jonathan Seidman

- While Big Data big wig at Orbitz, Jonathan (now at Cloudera) published sample code to perform the same analysis of the airline on-time data set using Hadoop streaming, RHIPE, hive, and RHadoop's rmr

  https://github.com/jseidman/hadoop-R

- To be honest, I only had to glance at each sample to make my decision, but let's take a look at the code he wrote for each package

About the data & Jonathan's analysis

- Each month, the US DOT publishes details of the on-time performance (or lack thereof) for every domestic flight in the country
- The ASA's 2009 Data Expo poster session was based on a cleaned version spanning 1987-2008, and thus was born the famous "airline" data set:

```
Year,Month,DayofMonth,DayOfWeek,DepTime,CRSDepTime,ArrTime,CRSArrTime,UniqueCarrier,
FlightNum,TailNum,ActualElapsedTime,CRSElapsedTime,AirTime,ArrDelay,DepDelay,Origin,
Dest,Distance,TaxiIn,TaxiOut,Cancelled,CancellationCode,Diverted,CarrierDelay,
WeatherDelay,NASDelay,SecurityDelay,LateAircraftDelay

2004,1,12,1,623,630,901,915,UA,462,N805UA,98,105,80,-14,-7,ORD,CLT,599,7,11,0,,0,0,0,0,0,0
2004,1,13,2,621,630,911,915,UA,462,N851UA,110,105,78,-4,-9,ORD,CLT,599,16,16,0,,0,0,0,0,0,0
2004,1,14,3,633,630,920,915,UA,462,N436UA,107,105,88,5,3,ORD,CLT,599,4,15,0,,0,0,0,0,0,0
2004,1,15,4,627,630,859,915,UA,462,N828UA,92,105,78,-16,-3,ORD,CLT,599,4,10,0,,0,0,0,0,0,0
2004,1,16,5,635,630,918,915,UA,462,N831UA,103,105,87,3,5,ORD,CLT,599,3,13,0,,0,0,0,0,0,0
[...]
```

http://stat-computing.org/dataexpo/2009/the-data.html

- Jonathan's analysis determines the mean departure delay ("DepDelay") for each airline for each month

“naked” streaming

hadoop-R/airline/src/deptdelay_by_month/R/streaming/map.R

```
#! /usr/bin/env Rscript

# For each record in airline dataset, output a new record consisting of
# "CARRIER|YEAR|MONTH \t DEPARTURE_DELAY"

con <- file("stdin", open = "r")
while (length(line <- readLines(con, n = 1, warn = FALSE)) > 0) {
  fields <- unlist(strsplit(line, "\\,"))
  # Skip header lines and bad records:
  if (!(identical(fields[[1]], "Year")) & length(fields) == 29) {
    deptDelay <- fields[[16]]
    # Skip records where departure dalay is "NA":
    if (!(identical(deptDelay, "NA"))) {
      # field[9] is carrier, field[1] is year, field[2] is month:
      cat(paste(fields[[9]], "|", fields[[1]], "|", fields[[2]], sep=""),
"\t",
          deptDelay, "\n")
    }
  }
}
close(con)
```

“naked” streaming 2/2

hadoop-R/airline/src/deptdelay_by_month/R/streaming/reduce.R

```
#!/usr/bin/env Rscript

# For each input key, output a record composed of
# YEAR \t MONTH \t RECORD_COUNT \t AIRLINE \t AVG_DEPT_DELAY

con <- file("stdin", open = "r")
delays <- numeric(0) # vector of departure delays
lastKey <- ""
while (length(line <- readLines(con, n = 1, warn = FALSE)) > 0) {
  split <- unlist(strsplit(line, "\t"))
  key <- split[[1]]
  deptDelay <- as.numeric(split[[2]])

  # Start of a new key, so output results for previous key:
  if (!(identical(lastKey, "")) & (!(identical(lastKey, key)))) {
    keySplit <- unlist(strsplit(lastKey, "\\|"))
    cat(keySplit[[2]], "\t", keySplit[[3]], "\t", length(delays), "\t", keySplit[[1]], "\t", (mean
(delays)), "\n")
    lastKey <- key
    delays <- c(deptDelay)
  } else { # Still working on same key so append dept delay value to vector:
      lastKey <- key
      delays <- c(delays, deptDelay)
  }
}

# We're done, output last record:
keySplit <- unlist(strsplit(lastKey, "\\|"))
cat(keySplit[[2]], "\t", keySplit[[3]], "\t", length(delays), "\t", keySplit[[1]], "\t", (mean
(delays)), "\n")
```

hive

hadoop-R/airline/src/deptdelay_by_month/R/hive/hive.R

```
#! /usr/bin/env Rscript

mapper <- function() {
  # For each record in airline dataset, output a new record consisting of
  # "CARRIER|YEAR|MONTH \t DEPARTURE_DELAY"

  con <- file("stdin", open = "r")
  while (length(line <- readLines(con, n = 1, warn = FALSE)) > 0) {
    fields <- unlist(strsplit(line, "\\,"))
    # Skip header lines and bad records:
    if (!(identical(fields[[1]], "Year")) & length(fields) == 29) {
      deptDelay <- fields[[16]]
      # Skip records where departure dalay is "NA":
      if (!(identical(deptDelay, "NA"))) {
        # field[9] is carrier, field[1] is year, field[2] is month:
        cat(paste(fields[[9]], "|", fields[[1]], "|", fields[[2]], sep=""), "\t",
            deptDelay, "\n")
      }
    }
  }
  close(con)
}
```

```
reducer <- function() {
  con <- file("stdin", open = "r")
  delays <- numeric(0) # vector of departure delays
  lastKey <- ""
  while (length(line <- readLines(con, n = 1, warn = FALSE)) > 0) {
    split <- unlist(strsplit(line, "\t"))
    key <- split[[1]]
    deptDelay <- as.numeric(split[[2]])

    # Start of a new key, so output results for previous key:
    if (!(identical(lastKey, "")) & (!(identical(lastKey, key)))) {
      keySplit <- unlist(strsplit(lastKey, "\\|"))
      cat(keySplit[[2]], "\t", keySplit[[3]], "\t", length(delays), "\t", keySplit[[1]], "\t", (mean(delays)), "\n")
      lastKey <- key
      delays <- c(deptDelay)
    } else { # Still working on same key so append dept delay value to vector:
        lastKey <- key
        delays <- c(delays, deptDelay)
    }
  }

  # We're done, output last record:
  keySplit <- unlist(strsplit(lastKey, "\\|"))
  cat(keySplit[[2]], "\t", keySplit[[3]], "\t", length(delays), "\t", keySplit[[1]], "\t", (mean(delays)), "\n")
}

library(hive)
DFS_dir_remove("/dept-delay-month", recursive = TRUE, henv = hive())
hive_stream(mapper = mapper, reducer = reducer,
            input="/data/airline/", output="/dept-delay-month")
results <- DFS_read_lines("/dept-delay-month/part-r-00000", henv = hive())
```

RHIPE

hadoop-R/airline/src/deptdelay_by_month/R/rhipe/rhipe.R

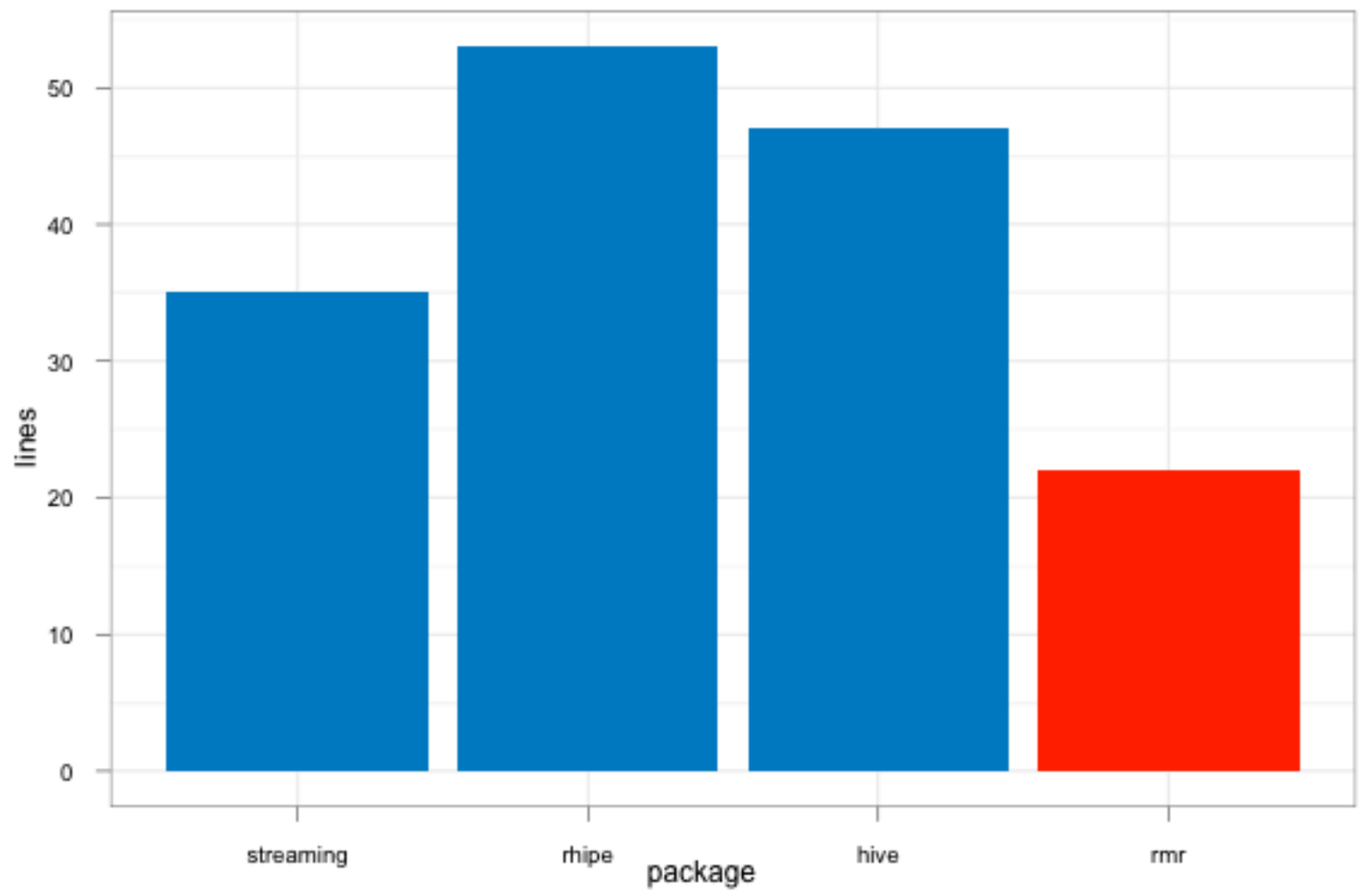
```
#! /usr/bin/env Rscript

# Calculate average departure delays by year and month for each airline in the
# airline data set (http://stat-computing.org/dataexpo/2009/the-data.html)

library(Rhipe)
rhinit(TRUE, TRUE)

# Output from map is:
# "CARRIER|YEAR|MONTH \t DEPARTURE_DELAY"
map <- expression({
  # For each input record, parse out required fields and output new record:
  extractDeptDelays = function(line) {
    fields <- unlist(strsplit(line, "\\,"))
    # Skip header lines and bad records:
    if (!(identical(fields[[1]], "Year")) & length(fields) == 29) {
      deptDelay <- fields[[16]]
     # Skip records where departure dalay is "NA":
      if (!(identical(deptDelay, "NA"))) {
        # field[9] is carrier, field[1] is year, field[2] is month:
        rhcollect(paste(fields[[9]], "|", fields[[1]], "|", fields[[2]], sep=""),
                  deptDelay)
      }
    }
  }
  # Process each record in map input:
  lapply(map.values, extractDeptDelays)
})

# Output from reduce is:
# YEAR \t MONTH \t RECORD_COUNT \t AIRLINE \t AVG_DEPT_DELAY
reduce <- expression(
  pre = {
    delays <- numeric(0)
  },
  reduce = {
    # Depending on size of input, reduce will get called multiple times
    # for each key, so accumulate intermediate values in delays vector:
    delays <- c(delays, as.numeric(reduce.values))
  },
  post = {
    # Process all the intermediate values for key:
    keySplit <- unlist(strsplit(reduce.key, "\\|"))
    count <- length(delays)
    avg <- mean(delays)
    rhcollect(keySplit[[2]],
              paste(keySplit[[3]], count, keySplit[[1]], avg, sep="\t"))
  }
)

inputPath <- "/data/airline/"
outputPath <- "/dept-delay-month"

# Create job object:
z <- rhmr(map=map, reduce=reduce,
          ifolder=inputPath, ofolder=outputPath,
          inout=c('text', 'text'), jobname='Avg Departure Delay By Month',
          mapred=list(mapred.reduce.tasks=2))
# Run it:
rhex(z)
```

rmr (1.1)

hadoop-R/airline/src/deptdelay_by_month/R/rmr/deptdelay-rmr.R

```
#!/usr/bin/env Rscript

# Calculate average departure delays by year and month for each airline in the
# airline data set (http://stat-computing.org/dataexpo/2009/the-data.html).
# Requires rmr package (https://github.com/RevolutionAnalytics/RHadoop/wiki).

library(rmr)

csvtextinputformat = function(line) keyval(NULL, unlist(strsplit(line, "\\,")))

deptdelay = function (input, output) {
  mapreduce(input = input,
            output = output,
            textinputformat = csvtextinputformat,
            map = function(k, fields) {
              # Skip header lines and bad records:
              if (!(identical(fields[[1]], "Year")) & length(fields) == 29) {
                deptDelay <- fields[[16]]
                # Skip records where departure dalay is "NA":
                if (!(identical(deptDelay, "NA"))) {
                  # field[9] is carrier, field[1] is year, field[2] is month:
                  keyval(c(fields[[9]], fields[[1]], fields[[2]]), deptDelay)
                }
              }
            },
            reduce = function(keySplit, vv) {
              keyval(keySplit[[2]], c(keySplit[[3]], length(vv), keySplit[[1]], mean(as.numeric(vv))))
            })
}

from.dfs(deptdelay("/data/airline/1987.csv", "/dept-delay-month"))
```

shorter is better



Other rmr advantages

- Well designed API
  - Your code only needs to deal with R objects: strings, lists, vectors & data.frames
- Very flexible I/O subsystem (new in rmr 1.2, faster in 1.3)
  - Handles common formats like CSV
  - Allows you to control the input parsing line-by-line without having to interact with stdin/stdout directly (or even loop)
- The result of the primary mapreduce() function is simply the HDFS path of the job's output
  - Since one job's output can be the next job's input, mapreduce calls can be daisy-chained to build complex workflows

Outline

- Why MapReduce? Why R?
- R + Hadoop options
- RHadoop overview
- Step-by-step examples
- Advanced RHadoop features

RHadoop overview

- Modular
  - Packages group similar functions
  - Only load (and learn!) what you need
  - Minimizes prerequisites and dependencies
- Open Source
  - Cost: Low (no) barrier to start using
  - Transparency: Development, issue tracker, Wiki, etc. hosted on github: https://github.com/RevolutionAnalytics/RHadoop/
- Supported
  - Sponsored by Revolution Analytics
  - Training & professional services available

RHadoop packages

- rhbase - access to HBase database
- rhdfs - interaction with Hadoop's HDFS file system
- rmr - all MapReduce-related functions

RHadoop prerequisites

- General
  - R 2.13.0+, Revolution R 4.3, 5.0
  - Cloudera CDH3 Hadoop distribution
    - Detailed answer: https://github.com/RevolutionAnalytics/RHadoop/wiki/Which-Hadoop-for-rmr
  - Environment variables
    - `HADOOP_HOME=/usr/lib/hadoop`
    - `HADOOP_CONF=/etc/hadoop/conf`
    - `HADOOP_CMD=/usr/bin/hadoop`
    - `HADOOP_STREAMING=/usr/lib/hadoop/contrib/streaming/hadoop-streaming-<version>.jar`
- rhdfs
  - R package: rJava
- rmr
  - R packages: RJSONIO (0.95-0 or later), itertools, digest
- rhbase
  - Running Thrift server (and its prerequisites)
    - see https://github.com/RevolutionAnalytics/RHadoop/wiki/rhbase

Downloading RHadoop

- Stable and development branches are available on github
  - **https://github.com/RevolutionAnalytics/RHadoop/**
- Releases available as packaged "tarballs"
  - **https://github.com/RevolutionAnalytics/RHadoop/downloads**
- Most current as of August 2012
  - **https://github.com/downloads/RevolutionAnalytics/RHadoop/rmr_1.3.1.tar.gz**
  - **https://github.com/downloads/RevolutionAnalytics/RHadoop/rhdfs_1.0.5.tar.gz**
  - **https://github.com/downloads/RevolutionAnalytics/RHadoop/rhbase_1.0.4.tar.gz**
- Or pull your own from the master branch
  - **https://github.com/RevolutionAnalytics/RHadoop/tarball/master**

Primary rmr functions

- Convenience
  - keyval() - creates a key-value pair from any two R objects. Used to generate output from input formatters, mappers, reducers, etc.
- Input/output
  - from.dfs(), to.dfs() - read/write data from/to the HDFS
  - make.input.format() - provides common file parsing (text, CSV) or will wrap a user-supplied function
- Job execution
  - mapreduce() - submit job and return an HDFS path to the results if successful

First, an easy example

Let's harness the power of our Hadoop cluster... to square some numbers

```
library(rmr)

small.ints = 1:1000

small.int.path = to.dfs(1:1000)

out = mapreduce(input = small.int.path,
      map = function(k,v) keyval(v, v^2)
      )
df = as.data.frame( from.dfs( out,
      structured=T ) )
```

see https://github.com/RevolutionAnalytics/RHadoop/wiki/Tutorial

Example output (abridged edition)

```
> out = mapreduce(input = small.int.path, map = function(k,v) keyval(v, v^2))
12/05/08 10:31:17 INFO mapred.FileInputFormat: Total input paths to process : 1
12/05/08 10:31:18 INFO streaming.StreamJob: getLocalDirs(): [/tmp/hadoop-cloudera/
mapred/local]
12/05/08 10:31:18 INFO streaming.StreamJob: Running job: job_201205061032_0107
12/05/08 10:31:18 INFO streaming.StreamJob: To kill this job, run:
12/05/08 10:31:18 INFO streaming.StreamJob: /usr/lib/hadoop-0.20/bin/hadoop job  -
Dmapred.job.tracker=ec2-23-22-84-153.compute-1.amazonaws.com:8021 -kill
job_201205061032_0107
12/05/08 10:31:18 INFO streaming.StreamJob: Tracking URL: http://
ec2-23-22-84-153.compute-1.amazonaws.com:50030/jobdetails.jsp?
jobid=job_201205061032_0107
12/05/08 10:31:20 INFO streaming.StreamJob:  map 0%  reduce 0%
12/05/08 10:31:24 INFO streaming.StreamJob:  map 50%  reduce 0%
12/05/08 10:31:25 INFO streaming.StreamJob:  map 100%  reduce 0%
12/05/08 10:31:32 INFO streaming.StreamJob:  map 100%  reduce 33%
12/05/08 10:31:34 INFO streaming.StreamJob:  map 100%  reduce 100%
12/05/08 10:31:35 INFO streaming.StreamJob: Job complete: job_201205061032_0107
12/05/08 10:31:35 INFO streaming.StreamJob: Output: /tmp/Rtmpu9IW4I/file744a2b01dd31
> df = as.data.frame( from.dfs( out, structured=T ) )
> colnames(df) = c('n', 'n2')
> str(df)
'data.frame': 1000 obs. of  2 variables:
 $ n : int  1 2 3 4 5 6 7 8 9 10 ...
 $ n2: num  1 4 9 16 25 36 49 64 81 100 ...
```

see https://github.com/RevolutionAnalytics/RHadoop/wiki/Tutorial

Components of basic rmr jobs

- Process raw input with formatters
  - see make.input.format()
- Write mapper function in R to extract relevant key-value pairs
- Perform calculations and analysis in reducer function written in R
- Submit the job for execution with mapreduce()
- Fetch the results from HDFS with from.dfs()

Outline

- Why MapReduce? Why R?
- R + Hadoop options
- RHadoop overview
- Step-by-step examples
- Advanced RHadoop features

Preparing to run examples

- data/ directory on github contains sample data sets
  - data/hadoop/ contains small extracts for the VM-based one-node Hadoop cluster
    - bin/populate.hdfs.sh contains script to copy data to HDFS
- data/local/ contains even smaller extracts for use with rmr's local backend (emulates Hadoop cluster for development and debugging)
  - examples run with local backend will create output in 'out/' directory on local file system

bin/populate.hdfs.sh

```
#!/bin/sh

/usr/bin/hadoop fs -mkdir /user/cloudera

/usr/bin/hadoop fs -mkdir /user/cloudera/wordcount
/usr/bin/hadoop fs -mkdir /user/cloudera/wordcount/
data
/usr/bin/hadoop fs -put data/hadoop/wordcount/*
        /user/cloudera/wordcount/data

/usr/bin/hadoop fs -mkdir /user/cloudera/airline
/usr/bin/hadoop fs -mkdir /user/cloudera/airline/data
/usr/bin/hadoop fs -put data/hadoop/airline/*
        /user/cloudera/airline/data
```

Example 1: wordcount

The "hello world" of Hadoop

- Overview
  - Perhaps the most common example of MapReduce, this analysis simply counts the occurrence of the words which appear in a text
- Objective
  - Provide a simple example which demonstrates the basics of using rmr: writing a mapper & reducer, submitting the job, and fetching the results
- Data
  - Any text will do!
  - Sample data (Shakespeare, Bible, etc.) available from Cloudera

https://github.com/cloudera/cloudera-training/tree/master/data

wordcount: code

```
map = function(k,v) {
    lapply(
        strsplit(x = v, split = '\\W')[[1]],
                function(w) keyval(w,1)
        )
}

reduce = function(k,vv) {
    keyval(k, sum(unlist(vv)))
}

wordcount = function (input, output = NULL) {
    mapreduce(input = input ,
              output = output,
              input.format = "text",
              map = map,
              reduce = reduce,
              combine = T)}
```

see R/1-wordcount.R and https://github.com/RevolutionAnalytics/RHadoop/wiki/Tutorial for code & discussion

wordcount: mapper

```
map = function(k,v) {
     lapply( strsplit(x = v, split = '\\s')[[1]],
                    function(w) keyval(w,1) )
}
```

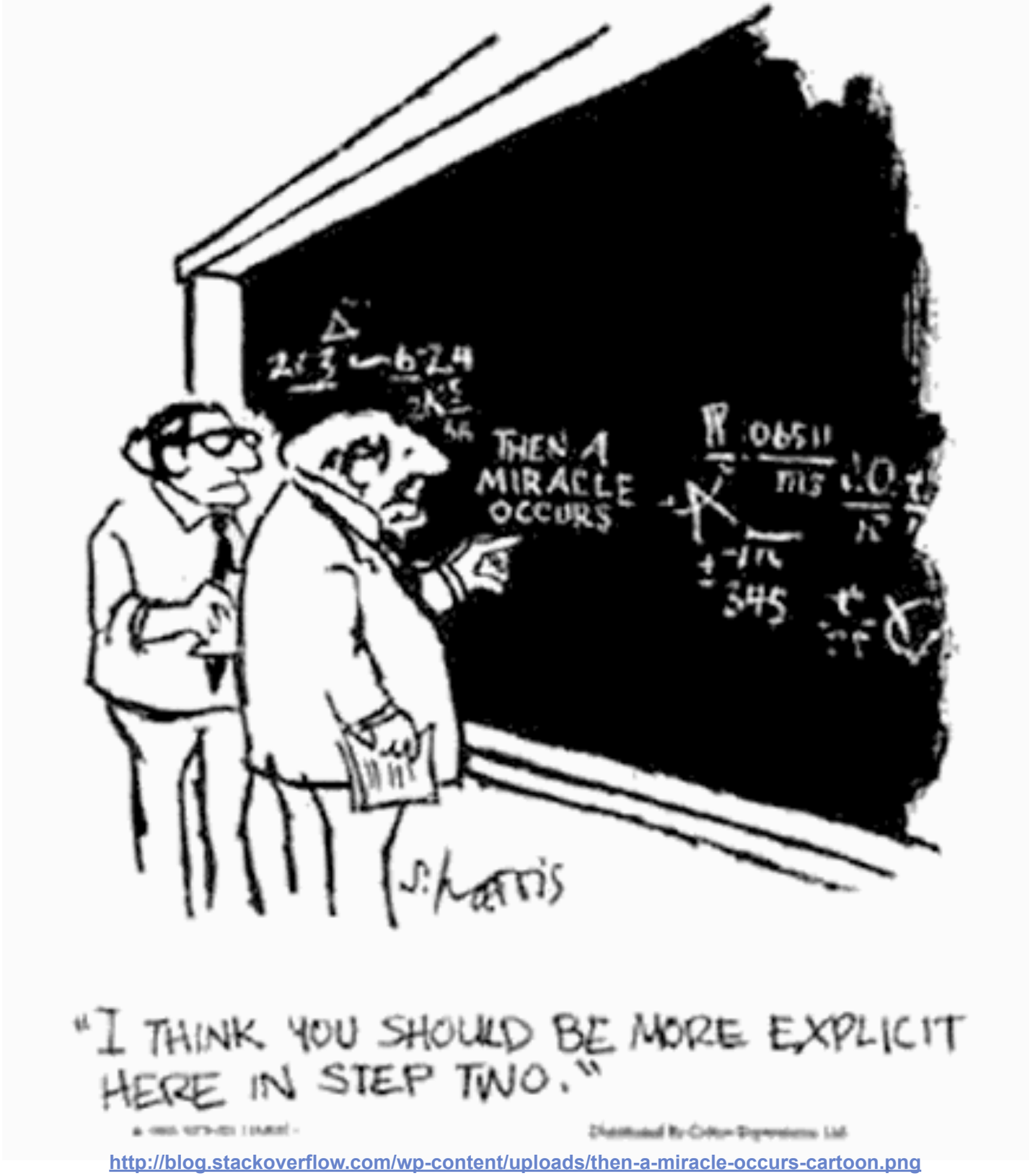
Input is simply a line of text from the "text" input formatter (key is NULL):

```
k = NULL
v = structure("Second Carrier\tPeas and beans are as dank here as a dog, and
that", rmr.input = "debug/wordcount/data/all-shakespeare.100")
```

Output is a list of keyvals: key = word, value = 1 (its occurrence count):

```
> str(emit[[1]])
List of 2
 $ key: chr "Second"
 $ val: num 1
 - attr(*, "rmr.keyval")= logi TRUE
> str(emit[[2]])
List of 2
 $ key: chr "Carrier"
 $ val: num 1
 - attr(*, "rmr.keyval")= logi TRUE
```

Hadoop then collects mapper output by key



http://blog.stackoverflow.com/wp-content/uploads/then-a-miracle-occurs-cartoon.png

wordcount: reducer

Code

```
reduce = function(k,vv) {
   keyval(k, sum(unlist(vv)))
}
```

Input is key, value with key = word, value = list of counts

```
k = "and"
vv = list(1, 1, 1, 1, 1, 1, 1, 1, 1)
```

Output is keyval with key=word, value=sum of counts

```
$ key: chr "and"
 $ val: num 9
 - attr(*, "rmr.keyval")= logi TRUE
```

see R/1-wordcount.R and https://github.com/RevolutionAnalytics/RHadoop/wiki/Tutorial for code & discussion

wordcount: combiner

What's a combiner?

- A function used to consolidate mapper output locally before the big shuffle/sort happens across nodes

When to use?

- Analyses which deal with aggregates (e.g., lengths, sums), extremes (e.g., min, max), etc. can often benefit
- But if you need details of the output (e.g., median, mean, quantile) or cross-node results, combiners may not be for you
- In this case, since we're just counting, we can use a combiner

How to specify

- Specify your function via the "combine" parameter to mapreduce()
- "combine=T" tells it to use the same function as for reducing...

wordcount: combiner

...and that's why our reducer code uses `sum()` and not `length()`:

```
reduce = function(k,vv) {
   keyval(k, sum(unlist(vv)))
}
```

see R/1-wordcount.R and https://github.com/RevolutionAnalytics/RHadoop/wiki/Tutorial for code & discussion

wordcount: submit job

Submit job

```
> out = wordcount(hdfs.data, hdfs.out)
```

Fetch results from HDFS and sort:

```
> df = as.data.frame( from.dfs(out, structured=T) )
```

```
> head(df)
          V1 V1.1
1        Ten    1
2 Attendants    1
3     chaste    1
4       hell    1
5   sleeping    1
6     minion    1
```

Example 2: airline enroute time

- Since Hadoop keys and values needn't be single-valued, let's pull out a few fields from the data: scheduled and actual gate-to-gate times and actual time in the air keyed on year and airport pair
- To review, here's what the data for a given day (3/25/2004) and airport pair (BOS & MIA) might look like:

```
2004,3,25,4,1445,1437,1820,1812,AA,399,N275AA,215,215,197,8,8,BOS,MIA,1258,6,12,0,,0,0,0,0,0,0
2004,3,25,4,728,730,1043,1037,AA,596,N066AA,195,187,170,6,-2,MIA,BOS,1258,7,18,0,,0,0,0,0,0,0
2004,3,25,4,1333,1335,1651,1653,AA,680,N075AA,198,198,168,-2,-2,MIA,BOS,1258,9,21,0,,0,0,0,0,0,0
2004,3,25,4,1051,1055,1410,1414,AA,836,N494AA,199,199,165,-4,-4,MIA,BOS,1258,4,30,0,,0,0,0,0,0,0
2004,3,25,4,558,600,900,924,AA,989,N073AA,182,204,157,-24,-2,BOS,MIA,1258,11,14,0,,0,0,0,0,0,0
2004,3,25,4,1514,1505,1901,1844,AA,1359,N538AA,227,219,176,17,9,BOS,MIA,1258,15,36,0,,0,0,0,15,0,2
2004,3,25,4,1754,1755,2052,2121,AA,1367,N075AA,178,206,158,-29,-1,BOS,MIA,1258,5,15,0,,0,0,0,0,0,0
2004,3,25,4,810,815,1132,1151,AA,1381,N216AA,202,216,180,-19,-5,BOS,MIA,1258,7,15,0,,0,0,0,0,0,0
2004,3,25,4,1708,1710,2031,2033,AA,1636,N523AA,203,203,173,-2,-2,MIA,BOS,1258,4,26,0,,0,0,0,0,0,0
2004,3,25,4,1150,1157,1445,1524,AA,1901,N066AA,175,207,161,-39,-7,BOS,MIA,1258,4,10,0,,0,0,0,0,0,0
2004,3,25,4,2011,1950,2324,2257,AA,1908,N071AA,193,187,163,27,21,MIA,BOS,1258,4,26,0,,0,0,21,6,0,0
2004,3,25,4,1600,1605,1941,1919,AA,2010,N549AA,221,194,196,22,-5,MIA,BOS,1258,10,15,0,,0,0,0,22,0,0
```

Why Hadoop for structured data?

- Hadoop and other MapReduce systems are often associated with unstructured data
- The mapping phase provides the opportunity to determine what data to use and how
- But this flexibility can be very useful for structured data too.
  - Remember all those meetings trying to getting your schema "just right"?
  - How often have you wanted to run a query against a field for which you forgot to build an index?
  - Ever have data feeds start arriving in a (slightly) different format?

rmr input formatter

- The input formatter is called to parse each input line for mapping
    - in v1.3, speed can be improved by processing batches of lines, but the idea's the same
    - While rmr can parse CSV files out of the box, we're going to write our own parser
    - Use `make.input.format()` to wrap your custom function
        - We start with Jonathan's code, but we're going to get fancy and name the fields of the resulting vector

https://raw.github.com/jeffreybreen/tutorial-201209-TDWI-big-data/master/R/2-airline.R

code: input formatter

```
asa.csvtextinputformat = make.input.format( format =
function(con, nrecs) {

    line = readLines(con, nrecs)

    values = unlist( strsplit(line, "\\,") )

    if (!is.null(values)) {

        names(values) = c('Year','Month','DayofMonth','DayOfWeek',
        'DepTime','CRSDepTime','ArrTime','CRSArrTime',
        'UniqueCarrier','FlightNum','TailNum','ActualElapsedTime',
        'CRSElapsedTime','AirTime','ArrDelay','DepDelay',
        'Origin','Dest','Distance','TaxiIn','TaxiOut',
        'Cancelled','CancellationCode','Diverted',
        'CarrierDelay','WeatherDelay','NASDelay',
        'SecurityDelay','LateAircraftDelay')

        return( keyval(NULL, values) )
    }
}, mode='text' )
```

data view: input formatter

Sample input (string):

```
2004,3,25,4,1445,1437,1820,1812,AA,399,N275AA,215,215,197,8,8,BOS,MIA,
1258,6,12,0,,0,0,0,0,0,0
```

Sample output (key-value pair):

```
structure(list(key = NULL, val = c("2004", "3", "25", "4", "1445",
     "1437", "1820", "1812", "AA", "399", "N275AA", "215", "215",
     "197", "8", "8", "BOS", "MIA", "1258", "6", "12", "0", "", "0",
     "0", "0", "0", "0")), .Names = c("key", "val"),
      rmr.keyval = TRUE)
```

(For clarity, column names have been omitted on these slides)

mapper

Note the improved readability due to named fields and the compound key-value output:

```
#
# the mapper gets a key and a value vector generated by the formatter
# in our case, the key is NULL and all the field values come in as a vector
#
mapper.year.market.enroute_time = function(key, val) {

      # Skip header lines, cancellations, and diversions:
      if ( !identical(as.character(val['Year']), 'Year')
           & identical(as.numeric(val['Cancelled']), 0)
           & identical(as.numeric(val['Diverted']), 0) ) {

            # We don't care about direction of travel, so construct 'market'
            # with airports ordered alphabetically
            # (e.g, LAX to JFK becomes 'JFK-LAX'
            if (val['Origin'] < val['Dest'])
                market = paste(val['Origin'], val['Dest'], sep='-')
            else
                market = paste(val['Dest'], val['Origin'], sep='-')

            # key consists of year, market
            output.key = c(val['Year'], market)

            # output gate-to-gate elapsed times (CRS and actual) + time in air
            output.val = c(val['CRSElapsedTime'], val['ActualElapsedTime'], val['AirTime'])

            return( keyval(output.key, output.val) )
      }
}
```

https://raw.github.com/jeffreybreen/tutorial-201209-TDWI-big-data/master/R/2-airline.R

data view: mapper

Sample input (key-value pair):

```
structure(list(key = NULL, val = c("2004", "3", "25", "4", "1445",
     "1437", "1820", "1812", "AA", "399", "N275AA", "215", "215",
     "197", "8", "8", "BOS", "MIA", "1258", "6", "12", "0", "", "0",
     "0", "0", "0", "0")), .Names = c("key", "val"),
     rmr.keyval = TRUE)
```

Sample output (key-value pair):

```
structure(list(key = c("2004", "BOS-MIA"),
               val = c("215", "215", "197")),
     .Names = c("key", "val"), rmr.keyval = TRUE)
```

reducer

For each key, our reducer is called with a list containing all of its values:

```
#
# the reducer gets all the values for a given key
# the values (which may be multi-valued as here) come in the form of a list()
#
reducer.year.market.enroute_time = function(key, val.list) {

      # val.list is a list of row vectors
      # a data.frame is a list of column vectors
      # plyr's ldply() is the easiest way to convert IMHO
      if ( require(plyr) )
          val.df = ldply(val.list, as.numeric)
      else { # this is as close as my deficient *apply skills can come w/o plyr
          val.list = lapply(val.list, as.numeric)
          val.df = data.frame( do.call(rbind, val.list) )
      }
      colnames(val.df) = c('crs','actual','air')

      output.key = key
      output.val = c( nrow(val.df), mean(val.df$crs, na.rm=T),
                                    mean(val.df$actual, na.rm=T),
                                    mean(val.df$air, na.rm=T) )

      return( keyval(output.key, output.val) )
  }
```

https://raw.github.com/jeffreybreen/tutorial-201209-TDWI-big-data/master/R/2-airline.R

data view: reducer

Sample input (key + list of vectors):

```
key:
  c("2004", "BOS-MIA")
value.list:
  list(c("215", "215", "197"), c("187", "195", "170"),
     c("198", "198", "168"), c("199", "199", "165"),
     c("204", "182", "157"), c("219", "227", "176"),
     c("206", "178", "158"), c("216", "202", "180"),
     c("203", "203", "173"), c("207", "175", "161"),
     c("187", "193", "163"), c("194", "221", "196") )
```

Sample output (key-value pair):

```
$key
[1] "2004"    "BOS-MIA"
$val
[1]  12.0000 202.9167 199.0000 172.0000
```

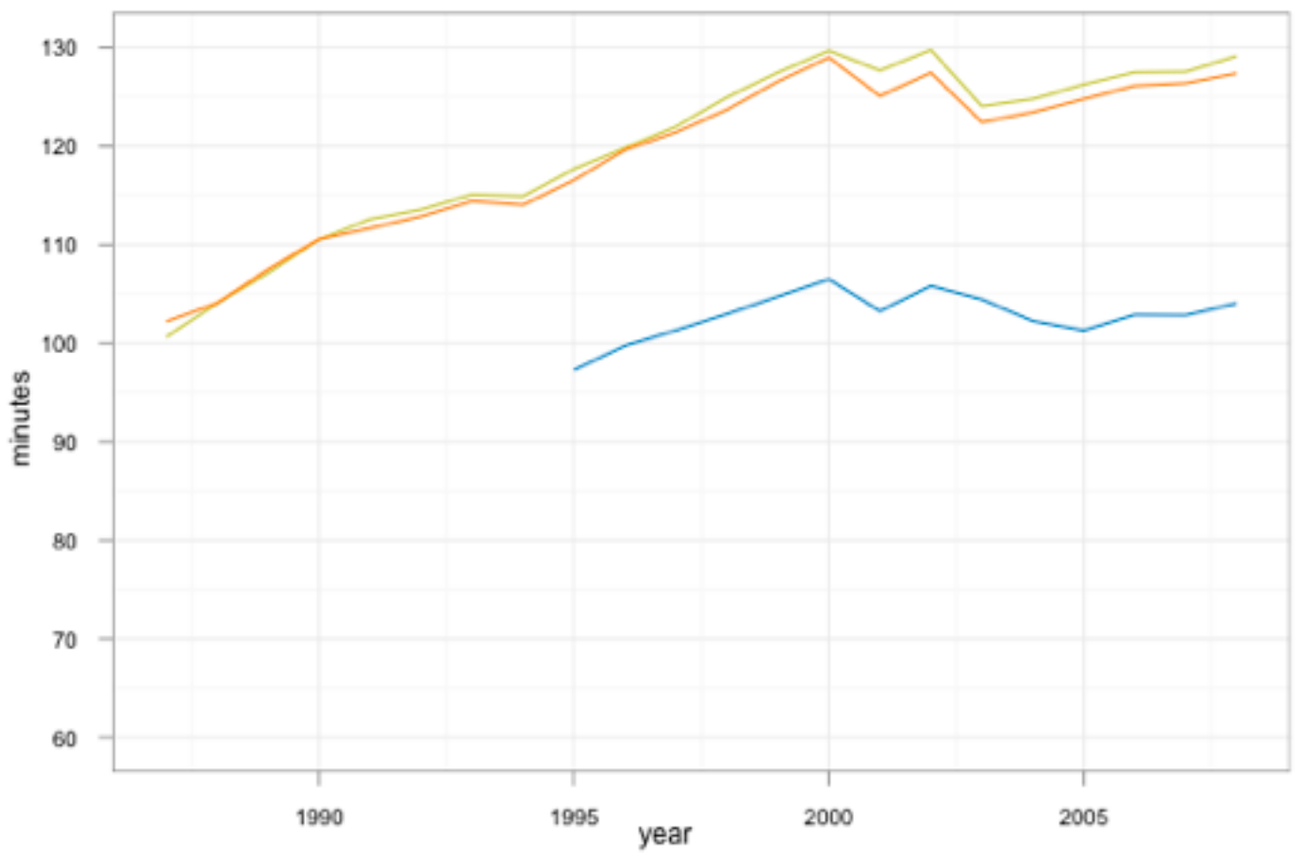
submit the job and get the results

```
mr.year.market.enroute_time = function (input, output) {
    mapreduce(input = input,
              output = output,
              input.format = asa.csvtextinputformat,
              map = mapper.year.market.enroute_time,
              reduce = reducer.year.market.enroute_time,
              backend.parameters = list(
                        hadoop = list(D = "mapred.reduce.tasks=10")
                        ),
              verbose=T)
}

hdfs.output.path = file.path(hdfs.output.root, 'enroute-time')
results = mr.year.market.enroute_time(hdfs.input.path, hdfs.output.path)

results.df = as.data.frame( from.dfs(results, structured=T) )
colnames(results.df) = c('year', 'market', 'flights', 'scheduled',
'actual', 'in.air')

save(results.df, file="out/enroute.time.RData")
```

Big Data In, Small Results Out

Note the Julia Child-like sleight of hand
(*i.e.*, I used all 21 years of data)

```
> nrow(results.df)
[1] 42612
> yearly.mean = ddply(results.df, c('year'), summarise,
                      scheduled = weighted.mean(scheduled, flights),
                      actual = weighted.mean(actual, flights),
                      in.air = weighted.mean(in.air, flights))
> ggplot(yearly.mean) +
    geom_line(aes(x=year, y=scheduled), color='#CCCC33') +
    geom_line(aes(x=year, y=actual), color='#FF9900') +
    geom_line(aes(x=year, y=in.air), color='#4689cc') + theme_bw() +
    ylim(c(60, 130)) + ylab('minutes')
```



Outline

- Why MapReduce? Why R?
- R + Hadoop options
- RHadoop overview
- Step-by-step example
- Advanced RHadoop features

rmr's local backend

- rmr can simulate a Hadoop cluster on your local machine
- Just set the 'backend' option:

```
rmr.options.set(backend='local')
```

- Very handy for development and testing
- You can try installing rmr completely Hadoop-free, but your mileage may vary

did you notice...

- ...that we simply declared functions in our local R environment, and they were magically available on all the nodes?
- When generating the Hadoop streaming job, rmr's mapreduce() function bundles up everything in your global and local environments
- This can make writing map- or reduce-side joins very simple
  - Just load your reference tables and use them when you need them!

Other RHadoop packages

- rhbase - access to HBase database
- rhdfs - interaction with Hadoop's HDFS file system
- rmr - all MapReduce-related functions

rhbase function overview

- Initialization
  - `hb.init()`
- Create and manage tables
  - `hb.list.tables(), hb.describe.table()`
  - `hb.new.table(), hb.delete.table()`
- Read and write data
  - `hb.insert(), hb.insert.data.frame()`
  - `hb.get(), hb.get.data.frame(), hb.scan()`
  - `hb.delete()`
- Administrative, etc.
  - `hb.defaults(), hb.set.table.mode()`
  - `hb.regions.table(), hb.compact.table()`

rhdfs function overview

- File & directory manipulation
  - `hdfs.ls(), hdfslist.files()`
  - `hdfs.delete(), hdfs.del(), hdfs.rm()`
  - `hdfs.dircreate(), hdfs.mkdir()`
  - `hdfs.chmod(), hdfs.chown(), hdfs.file.info()`
  - `hdfs.exists()`
- Copying, moving & renaming files to/from/within HDFS
  - `hdfs.copy(), hdfs.move(), hdfs.rename()`
  - `hdfs.put(), hdfs.get()`
- Reading files directly from HDFS
  - `hdfs.file(), hdfs.read(), hdfs.write(), hdfs.flush()`
  - `hdfs.seek(), hdfs.tell(con), hdfs.close()`
  - `hdfs.line.reader(), hdfs.read.text.file()`
- Misc.
  - `hdfs.init(), hdfs.defaults()`

Next up:

Taking it to the cloud