Getting Started with R and Hadoop, Parts I and II

Jeffrey Breen
Principal, Think Big Academy
jeffrey.breen@thinkbiganalytics.com
http://www.thinkbigacademy.com/

Big Data TechCon
Cambridge, MA

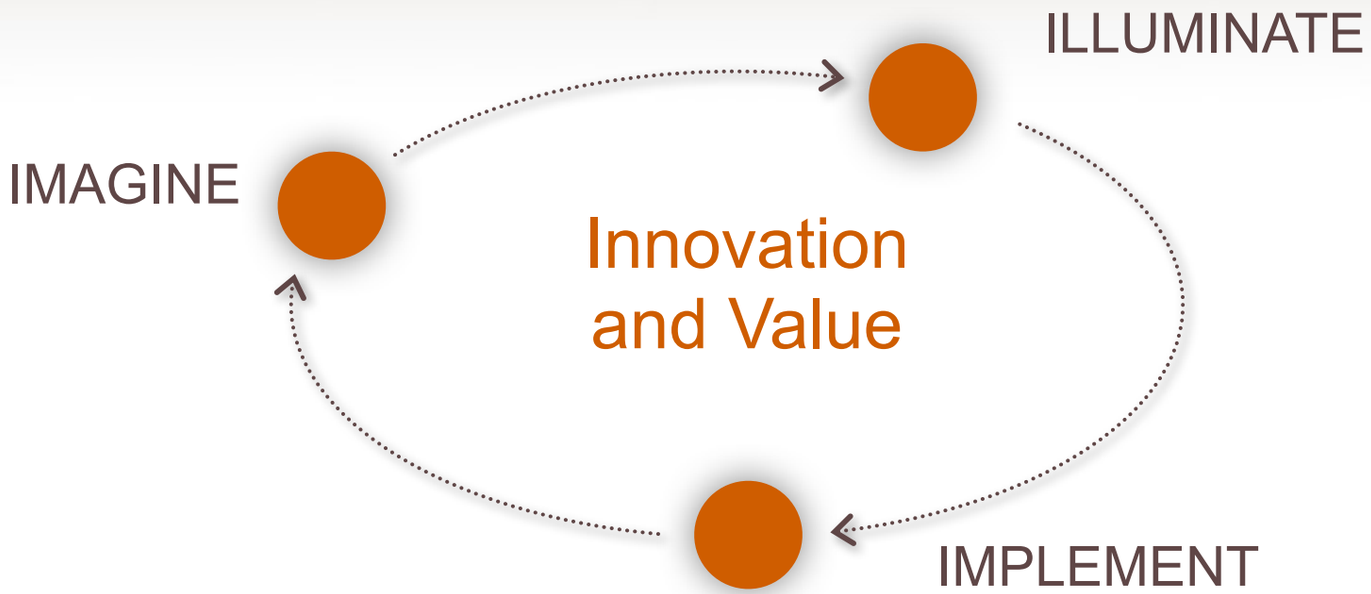
April 2014

Driving New Value from Big Data Investments

Slides and code at
http://bit.ly/rmr2airline

THINK BIG Analytics Methodology

Experiment-Driven Short Projects with Nimble Test Solution Cycles



We Accelerate Your Time to Value

- Breaking Down Business and IT Barriers
- Discrete Projects with Beginning and End
- Early Releases to Validate ROI and Ensure Long Term Success

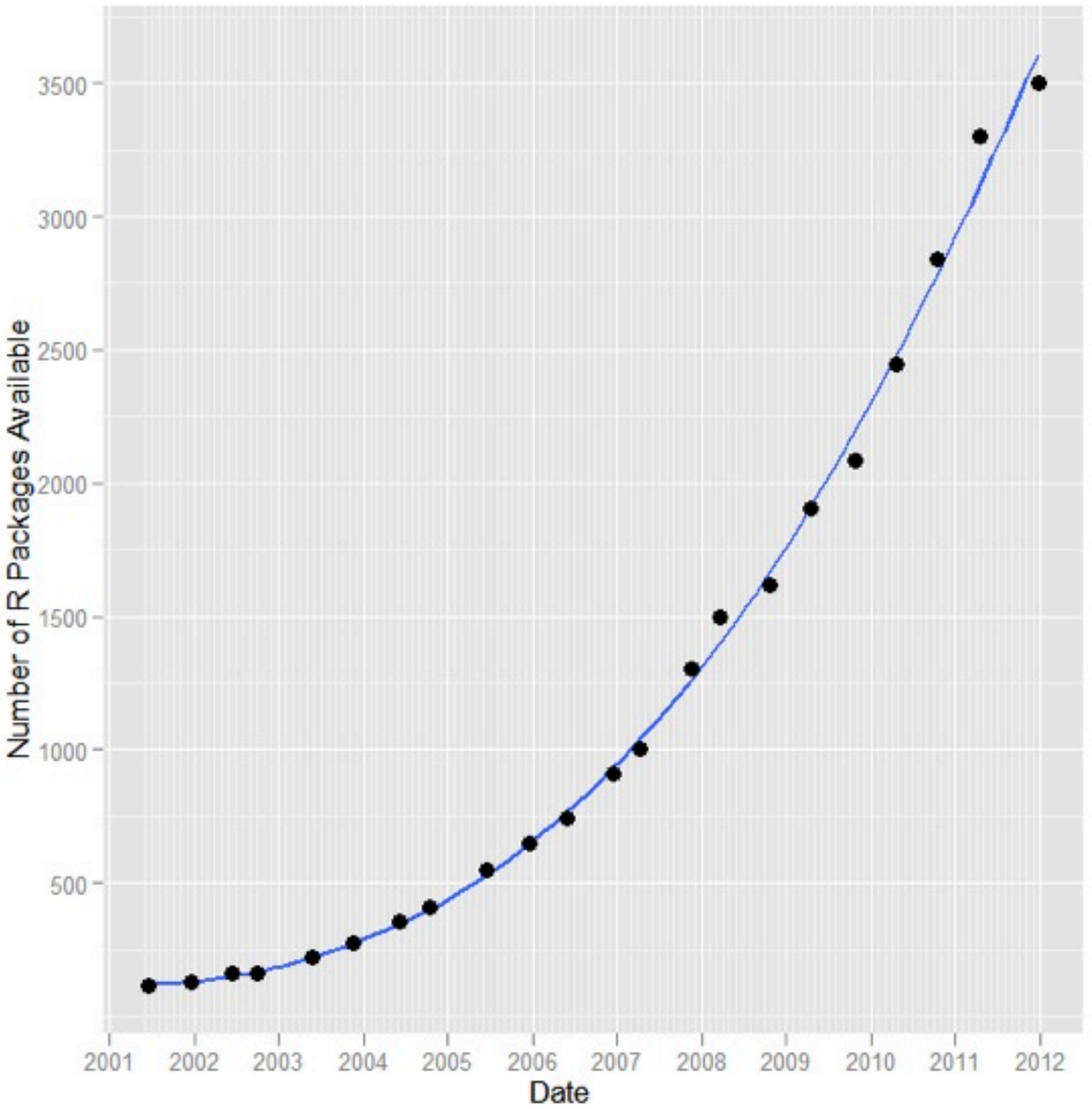
Agenda

- Part I: Basics and Infrastructure
  - Why R? Why Hadoop?
  - How to use R and Hadoop together
  - About RHadoop
  - Learning Hadoop without Hadoop
  - Counting words with rmr2
  - Running Hadoop in the cloud
  - Q&A
- Part II: Airline data examples
  - Structured data with Hadoop?
  - Writing a custom input formatter
  - Analyzing flight delays
  - Advanced techniques: map-side joins
  - Q&A

http://thebalancedguy.blogspot.com/2010/09/with-3-boys-and-having-been-cub-scout.html

http://www.wengerna.com/giant-knife-16999

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

R + Hadoop options

- Originally, Hadoop was a Java-only ecosystem
- Hadoop's Streaming API 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

(Some of) R's Hadoop-related packages

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: 2012-10-29	focused on utility functions: I/O parsing, data conversions, etc. Available on CRAN.
RHIPE	v0.71: 2012-11-18	comprehensive: code & submit jobs, access HDFS, etc. Unfortunately, most links to it are broken. Look on github instead: https://github.com/saptarshiguha/RHIPE/
segue	v0.05: 2012-07-09	Very clever way to use Amazon EMR with small or no data. http://code.google.com/p/segue/
RHadoop (rmr2, rhdfs, rhbase)	rmr2 2.1.0: 2013-02-25 rhdfs 1.0.5: 2012-08-02 rhbase 1.1.1: 2013-03-07	Divided into separate packages by purpose: • rmr2 - 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 one of Orbitz's Big Data whizzes, 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

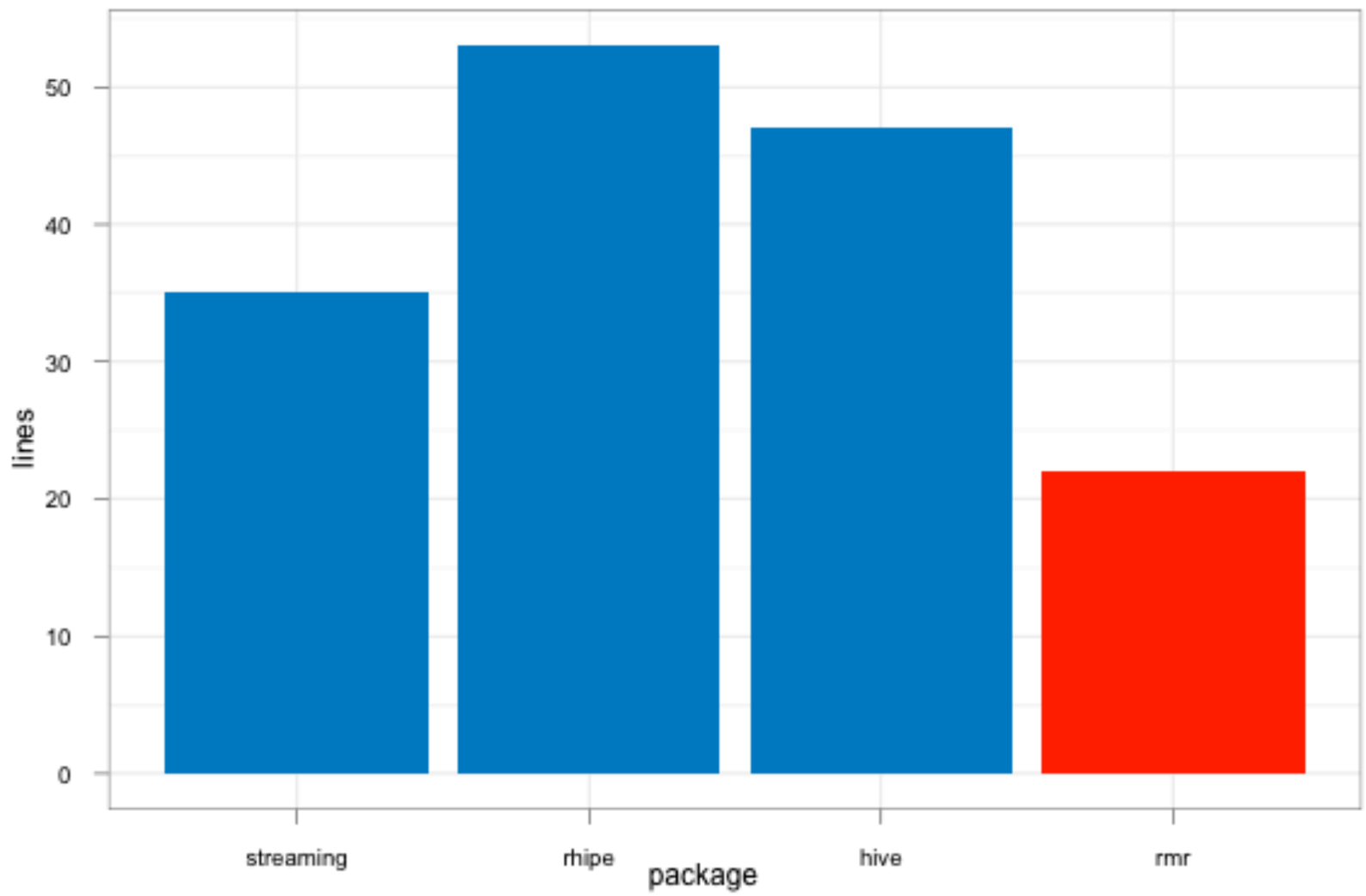
```
#! /usr/bin/env Rscript

# 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



Enter RHadoop

- RHadoop is an open source project sponsored by Revolution Analytics
- Package Overview
  - rmr2 - all MapReduce-related functions
  - rhdfs - interaction with Hadoop's HDFS file system
  - rhbase - access to the NoSQL HBase database
- rmr2 uses Hadoop's Streaming API to allow R users to write MapReduce jobs in R
  - handles all of the I/O and job submission for you (no `while(<stdin>)`-like loops!)

RHadoop Advantages

- 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
  - Support available with Revolution R Enterprise subscriptions

RHadoop Advantages: rmr2

- Well designed API
  - Your code only needs to deal with R objects: strings, lists, vectors & data.frames
- Very flexible I/O subsystem
  - Handles common formats like CSV
  - Allows you to control the input parsing without having to interact with stdin/stdout directly (or 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

rmr2's Commonly Used 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

Install RHadoop's **rmr2** package

The **rmr2** package contains all the mapreduce-related functions, including generating Hadoop streaming jobs and basic data exchange with HDFS

Note: **rmr2** needs to be installed on all cluster nodes

First install prerequisite packages (run R as root to install system-wide)

```
$ sudo R
> install.packages( c('RJSONIO', 'itertools', 'digest', 'Rcpp', 'functional', 'plyr', 'reshape2'),
repos='http://cran.revolutionanalytics.com')
```

Download the latest stable release (2.2.2) from github

```
$ wget --no-check-certificate https://github.com/RevolutionAnalytics/rmr2/archive/2.2.2.tar.gz
```

Unpack the package from the tar file and install

```
$ tar zxf 2.2.2.tar.gz ; cd rmr2-2.2.2
$ sudo R CMD INSTALL pkg
```

Test that it loads

```
$ R
> library(rmr2)
Loading required package: Rcpp
Loading required package: RJSONIO
Loading required package: bitops
Loading required package: digest
Loading required package: functional
Loading required package: stringr
Loading required package: plyr
Loading required package: reshape2
```

Installation test

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

```
library(rmr2)
small.ints = 1:1000
small.int.path = to.dfs(small.ints)
out = mapreduce(input = small.int.path,
        map = function(k,v) keyval(v, v^2))
results = from.dfs( out )
results.df = as.data.frame(results,
stringsAsFactors=F )
```

Example output (abridged edition)

```
> out = mapreduce(input = small.int.path, map = function(k,v) keyval(v, v^2))

packageJobJar: [/tmp/Rtmp7TIZqD/rmr-local-env14976d243ee6, /tmp/Rtmp7TIZqD/rmr-global-env14973c98058b, /tmp/
Rtmp7TIZqD/rmr-streaming-map14976f65b318, /var/lib/hadoop-0.20/cache/cloudera/hadoop-
unjar7068598134102001273/] [] /tmp/streamjob5503003800760025237.jar tmpDir=null

12/12/09 00:05:20 INFO mapred.FileInputFormat: Total input paths to process : 1

12/12/09 00:05:20 INFO streaming.StreamJob: getLocalDirs(): [/var/lib/hadoop-0.20/cache/cloudera/mapred/local]

12/12/09 00:05:20 INFO streaming.StreamJob: Running job: job_201212082354_0002

12/12/09 00:05:20 INFO streaming.StreamJob: To kill this job, run:

12/12/09 00:05:20 INFO streaming.StreamJob: /usr/lib/hadoop-0.20/bin/hadoop job  -
Dmapred.job.tracker=0.0.0.0:8021 -kill job_201212082354_0002

12/12/09 00:05:20 INFO streaming.StreamJob: Tracking URL: http://0.0.0.0:50030/jobdetails.jsp?
jobid=job_201212082354_0002

12/12/09 00:05:21 INFO streaming.StreamJob:  map 0%  reduce 0%

12/12/09 00:05:32 INFO streaming.StreamJob:  map 100%  reduce 0%

12/12/09 00:05:34 INFO streaming.StreamJob:  map 100%  reduce 100%

12/12/09 00:05:34 INFO streaming.StreamJob: Job complete: job_201212082354_0002

12/12/09 00:05:34 INFO streaming.StreamJob: Output: /tmp/Rtmp7TIZqD/file14973d4894f2

> results = from.dfs( out )

> results.df = as.data.frame(results, stringsAsFactors=F )

> str(results.df)

'data.frame':   1000 obs. of  2 variables:

 $ key: int  1 2 3 4 5 6 7 8 9 10 ...

 $ val: num  1 4 9 16 25 36 49 64 81 100 ...
```

wordcount: mapper

Code

```
map = function(k,lines) {
  words.list = strsplit(lines, '\\s')
  words = unlist(words.list)
  return( keyval(words, 1) )
}
```

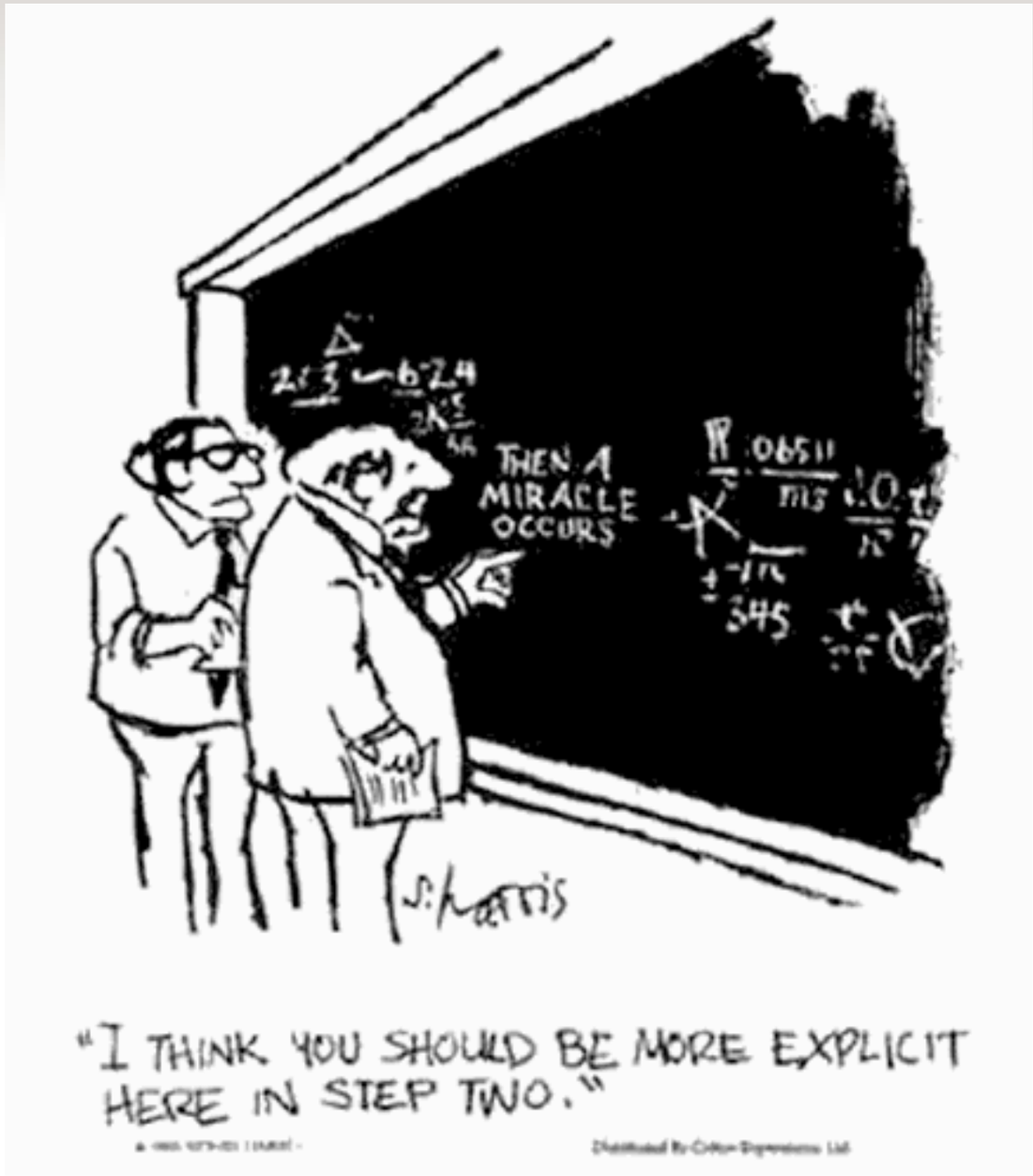
Input is simply a vector of lines of text from the “text” input formatter (key is NULL)

```
k = NULL
v = c("KING HENRY IV So shaken as we are, so wan with
care,", "\tFind we a time for frighted peace to pant,",
"\tAnd breathe short-winded accents of new
broils", ... )
```

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

```
List of 2
 $ key: chr [1:6450] "KING" "HENRY" "IV" "So"
"shaken"...
 $ val: num 1
```

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(word, counts) {
  return( keyval(word, sum(counts)) )
}
```

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

```
key = "And"
val = c(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1)
```

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

```
List of 2
 $ key: chr "And"
 $ val: num 50
```

wordcount: code

```
library(rmr2)

map = function(k,lines) {

  words.list = strsplit(lines, '\\s')
  words = unlist(words.list)

  return( keyval(words, 1) )
}

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

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

from Revolution Analytics' *Getting Started with RHadoop* course

wordcount: submit job and fetch results

Submit job

```
> hdfs.root = 'wordcount'
> hdfs.data = file.path(hdfs.root, 'data')
> hdfs.out = file.path(hdfs.root, 'out')
> out = wordcount(hdfs.data, hdfs.out)
```

Fetch results from HDFS

```
> results = from.dfs( out )
> results.df = as.data.frame(results, stringsAsFactors=F )
> colnames(results.df) = c('word', 'count')
> head(results.df)
      word count
1 greatness     2
2    damned     3
3       tis     5
4      jade     1
5  magician     1
```

from Revolution Analytics' *Getting Started with RHadoop* course

Code notes

- Scalable
  - Hadoop and MapReduce abstract away system details
  - Code runs on 1 node or 1,000 nodes without modification
- Portable
  - You write normal R code, interacting with normal R objects
  - RHadoop's rmr2 library abstracts away Hadoop details
  - All the functionality you expect is there—including Enterprise R's
- Flexible
  - Only the mapper deals with the data directly
  - All components communicate via key-value pairs
  - Key-value "schema" chosen for ***each analysis*** rather than as a prerequisite to loading data into the system

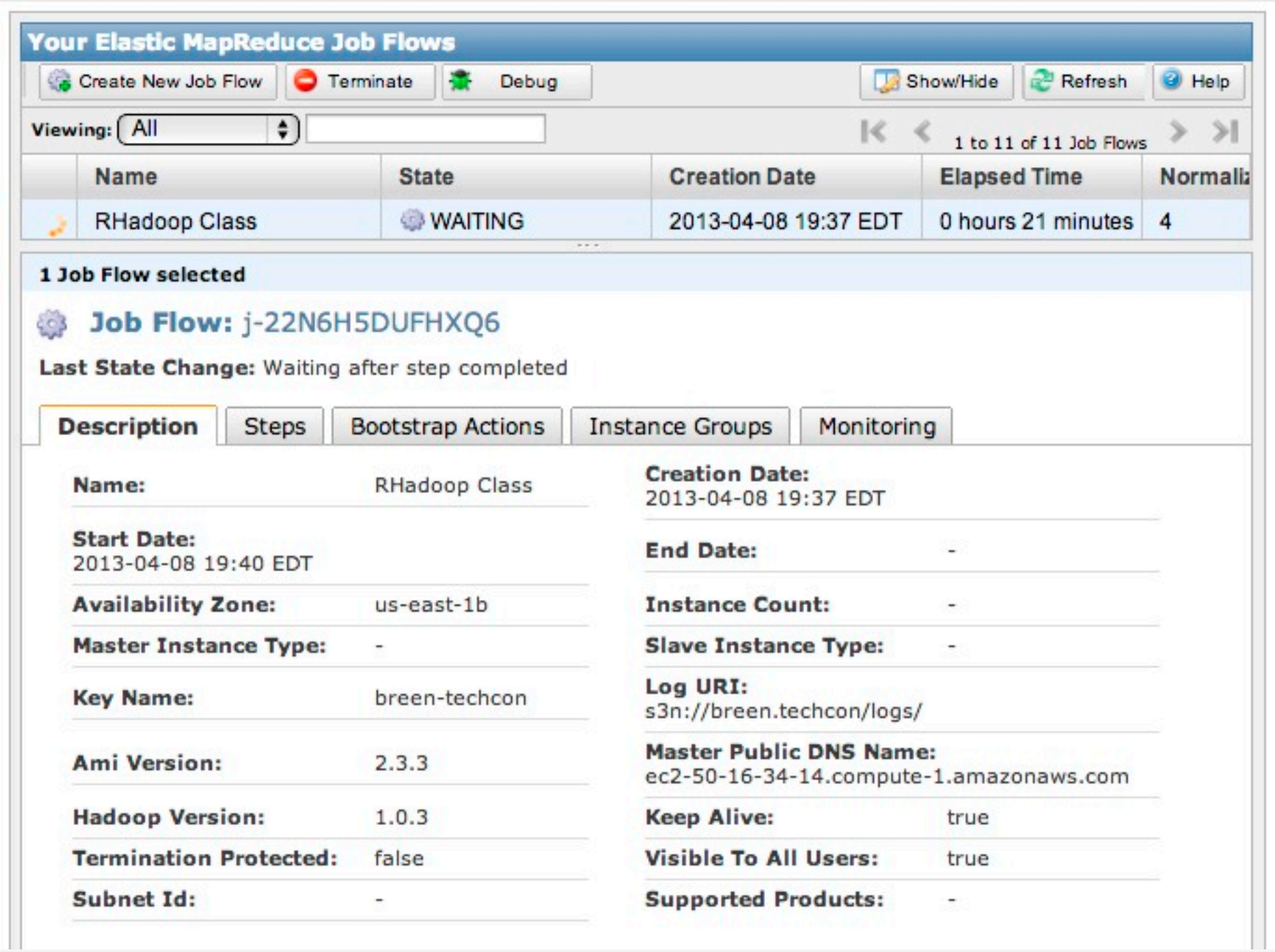
Running Hadoop in the Cloud

- Amazon's Elastic MapReduce provides Hadoop in the cloud, on demand
  - Offers Apache distribution or MapR's M3 and M5
- Leverages AWS cloud infrastructure
  - EC2 virtual servers, S3 storage, etc.
- Many developer and admin tools available
  - Amazon Elastic MapReduce Ruby Client
    - Create, destroy, modify clusters from the command line
    - Supports spot instances, IAM, bootstrap actions, etc.
    - Download @ http://aws.amazon.com/developertools/2264

Using the elastic-mapreduce CLI

```
$ elastic-mapreduce --create --alive --name 'RHadoop tutorial'
--hive-interactive --instance-group master --instance-type
m1.large --instance-count 1 --debug --bootstrap-action s3://
thinkbig-rhadoop/bootstrap/bootstrap-r-rmr2.sh --bootstrap-
action s3://thinkbig-rhadoop/bootstrap/bootstrap-rstudio.sh
Created job flow j-22N6H5DUFHXQ6
$ elastic-mapreduce --list --active
j-22N6H5DUFHXQ6     STARTING                  RHadoop Class
   PENDING        Setup Hive
[...]
$ elastic-mapreduce --list --active
j-22N6H5DUFHXQ6     WAITING
ec2-50-16-34-14.compute-1.amazonaws.com   RHadoop Class
   COMPLETED      Setup Hive
```

“WAITING” -- for you!



ssh tunnel

```
$ ssh -i breen-techcon.pem hadoop@ec2-50-16-34-14.compute-1.amazonaws.com \
-L 8787:localhost:8787 -L 9100:localhost:9100
The authenticity of host 'ec2-50-16-34-14.compute-1.amazonaws.com (50.16.34.14)' can't be established.
RSA key fingerprint is 3e:b3:a1:5f:82:ee:9a:74:55:ea:f7:9a:d8:3b:d4:a3.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'ec2-50-16-34-14.compute-1.amazonaws.com,50.16.34.14' (RSA) to the list of known hosts.
Linux (none) 3.2.30-49.59.amzn1.x86_64 #1 SMP Wed Oct 3 19:54:33 UTC 2012 x86_64
-----------------------------------------------------------------------------

Welcome to Amazon Elastic MapReduce running Hadoop and Debian/Squeeze.

Hadoop is installed in /home/hadoop. Log files are in /mnt/var/log/hadoop. Check
/mnt/var/log/hadoop/steps for diagnosing step failures.

The Hadoop UI can be accessed via the following commands:

  JobTracker    lynx http://localhost:9100/
  NameNode      lynx http://localhost:9101/

-----------------------------------------------------------------------------
hadoop@ip-10-145-222-36:~$
```

Part 2...

- Part I: Basics and Infrastructure
  - Why R? Why Hadoop?
  - How to use R and Hadoop together
  - About RHadoop
  - Learning Hadoop without Hadoop
  - Counting words with rmr2
  - Running Hadoop in the cloud
  - Q&A
- Part II: Airline data examples
  - Structured data with Hadoop?
  - Writing a custom input formatter
  - Analyzing flight delays
  - Advanced techniques: map-side joins
  - Q&A

Getting Started with R and Hadoop, Part II

Jeffrey Breen
Principal, Think Big Academy
jeffrey.breen@thinkbiganalytics.com
http://www.thinkbigacademy.com/

Big Data TechCon
Cambridge, MA

April 2014

Driving New Value from Big Data Investments

Slides and code at
http://bit.ly/rmr2airline

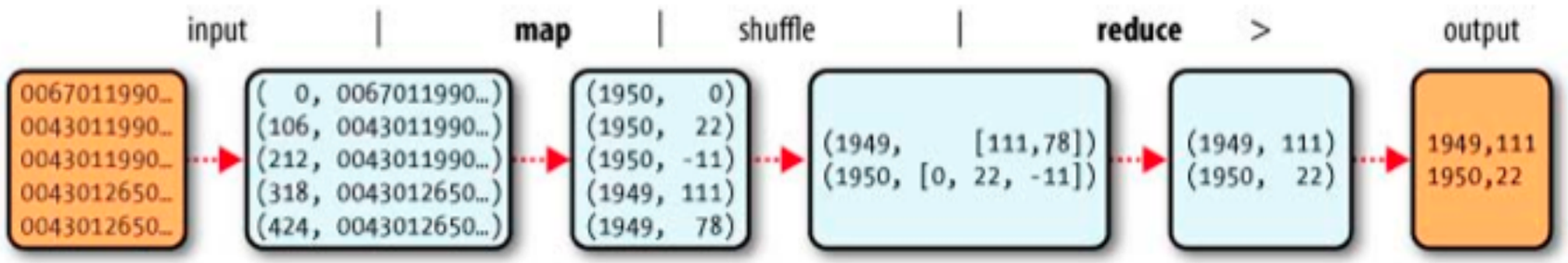
Agenda

- Part I: Basics and Infrastructure
  - Why R? Why Hadoop?
  - How to use R and Hadoop together
  - About RHadoop
  - Learning Hadoop without Hadoop
  - Counting words with rmr2
  - Running Hadoop in the cloud
  - Q&A
- Part II: Airline data examples
  - Structured data with Hadoop?
  - Writing a custom input formatter
  - Analyzing flight delays
  - Advanced techniques: map-side joins
  - Q&A

True confession: 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 & support costs
- Loved the scalability, COTS components, and price, but missed the fact that keys (and values) could be compound & complex
- ... and examples like Wordcount didn't help!



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

Structured data from the airline industry

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

Selecting meaningful keys and values

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

For a given day (3/25/2004) and airport pair (BOS & MIA), here's what the data 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
```

Writing an input formatter

- The input formatter is called by mapreduce() to parse the input
- rmr2 can parse text, CSV, JSON files out of the box, or you can use `make.input.format()` to pass in your own parameters to `read.table()` or to wrap your own function
- For this example, we will pass our own parameters to split by commas and label each column:

```
asa.csv.input.format = make.input.format(format='csv', mode='text', streaming.format =
NULL,
sep=',', col.names = 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'),
stringsAsFactors=F)
```

Custom parameters to **`read.table()`**

see R/2-airline.R

data flow: input formatter

Sample input (lines of text):

```
Year,Month,DayofMonth,DayOfWeek,DepTime,CRSDepTime,ArrTime,CRSArrTime,UniqueCarrier,FlightNum,Ta
ilNum,ActualElapsedTime,CRSElapsedTime,AirTime,ArrDelay,DepDelay,Origin,Dest,Distance,TaxiIn,Tax
iOut,Cancelled,CancellationCode,Diverted,CarrierDelay,WeatherDelay,NASDelay,SecurityDelay,LateAi
rcraftDelay
2004,3,25,4,1815,1820,2128,2136,UA,839,N613UA,373,376,337,-8,-5,JFK,LAX,2475,3,33,0,,0,0,0,0,0,0
2004,3,25,4,1304,1305,2107,2115,UA,840,N643UA,303,310,277,-8,-1,LAX,JFK,2475,6,20,0,,0,0,0,0,0,0
2004,3,25,4,1119,1120,1931,1927,UA,904,N657UA,312,307,282,4,-1,LAX,JFK,2475,10,20,0,,0,0,0,0,0,0
```

Sample output (key=NULL, val=data.frame):

```
'data.frame':     45 obs. of  29 variables:
 $ Year              : chr  "Year" "2004" "2004" "2004" ...
 $ Month             : chr  "Month" "3" "3" "3" ...
 $ DayofMonth        : chr  "DayofMonth" "25" "25" "25" ...
 $ DayOfWeek         : chr  "DayOfWeek" "4" "4" "4" ...
 $ DepTime           : chr  "DepTime" "1815" "1304" "1119" ...
 $ CRSDepTime        : chr  "CRSDepTime" "1820" "1305" "1120" ...
 $ ArrTime           : chr  "ArrTime" "2128" "2107" "1931" ...
[...]
```

Note that all lines of the split are included, including headers!

code: mapper

Note the improved readability due to our labeled input data:

```
mapper.year.market.enroute_time = function(key, val.df) {

    # Remove header lines, cancellations, and diversions:
    val.df = subset(val.df, Year != 'Year' & Cancelled == 0 & Diverted == 0)

    # We don't care about direction of travel, so construct a new 'market' vector
    # with airports ordered alphabetically (e.g, LAX to JFK becomes 'JFK-LAX')
    market = with( val.df, ifelse(Origin < Dest,
                                  paste(Origin, Dest, sep='-'),
                                  paste(Dest, Origin, sep='-')) )

    # key consists of year, market
    output.key = data.frame(year=as.numeric(val.df$Year), market=market, stringsAsFactors=F)

    # emit data.frame of gate-to-gate elapsed times (CRS and actual) + time in air
    output.val = val.df[,c('CRSElapsedTime', 'ActualElapsedTime', 'AirTime')]
    colnames(output.val) = c('scheduled', 'actual', 'inflight')

    # and finally, make sure they're numeric while we're at it
    output.val = transform(output.val,
                                    scheduled = as.numeric(scheduled),
                                    actual = as.numeric(actual),
                                    inflight = as.numeric(inflight)
    )

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

see R/2-airline.R

data flow: mapper

Input is the keyval output from the input formatter:

```
'data.frame':   45 obs. of  29 variables:
 $ Year             : chr  "Year" "2004" "2004" "2004" ...
 $ Month            : chr  "Month" "3" "3" "3" ...
 $ DayofMonth       : chr  "DayofMonth" "25" "25" "25" ...
[...]
 $ Origin           : chr  "Origin" "JFK" "LAX" "LAX" ...
 $ Dest             : chr  "Dest" "LAX" "JFK" "JFK" ...
 $ Cancelled        : chr  "Cancelled" "0" "0" "0" ...
 $ CancellationCode : chr  "CancellationCode" "" "" "" ...
 $ Diverted         : chr  "Diverted" "0" "0" "0" ...
 $ UniqueCarrier    : chr  "UniqueCarrier" "UA" "UA" "UA" ...
 $ FlightNum        : chr  "FlightNum" "839" "840" "904" ...
 $ TailNum          : chr  "TailNum" "N613UA" "N643UA" "N657UA" ...
 $ ActualElapsedTime: chr  "ActualElapsedTime" "373" "303" "312" ...
 $ CRSElapsedTime   : chr  "CRSElapsedTime" "376" "310" "307" ...
 $ AirTime          : chr  "AirTime" "337" "277" "282" ...
[...]
 - attr(*, "rmr.input")= chr "local/airline/data/20040325-jfk-lax.csv"
```

Output is a keyval with key=year & market, val=elapsed times:

```
> str(key)
'data.frame':   44 obs. of  2 variables:
 $ year  : num  2004 2004 2004 2004 2004 ...
 $ market: chr  "JFK-LAX" "JFK-LAX" "JFK-LAX" "JFK-LAX" ...
> str(val)
'data.frame':   44 obs. of  3 variables:
 $ scheduled: num  376 310 307 303 308 302 303 365 362 362 ...
 $ actual   : num  373 303 312 289 319 297 305 358 364 359 ...
 $ inflight : num  337 277 282 270 281 274 279 331 342 337 ...
```

code: 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 data.frame
#
reducer.year.market.enroute_time = function(key, val.df) {
   output.key = key
   output.val = data.frame(flights = nrow(val.df),
                           scheduled = mean(val.df$scheduled, na.rm=T),
                           actual = mean(val.df$actual, na.rm=T),
                           inflight = mean(val.df$inflight, na.rm=T) )

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

see R/2-airline.R

data flow: reducer

Hadoop has collected all the mapped values for a given key (year & market):

```
> str(key)
'data.frame':     1 obs. of  2 variables:
 $ year  : num 2004
 $ market: chr "JFK-LAX"
> str(val)
'data.frame':     44 obs. of  3 variables:
 $ scheduled: num  376 310 307 303 308 302 303 365 362 362 ...
 $ actual   : num  373 303 312 289 319 297 305 358 364 359 ...
 $ inflight : num  337 277 282 270 281 274 279 331 342 337 ...
```

Our reducer computes and emits the results:

```
> str(key)
'data.frame':     1 obs. of  2 variables:
 $ year  : num 2004
 $ market: chr "JFK-LAX"
> str(val)
'data.frame':     1 obs. of  4 variables:
 $ flights  : int 44
 $ scheduled: num 338
 $ actual   : num 338
 $ inflight : num 308
```

submit job and fetch results

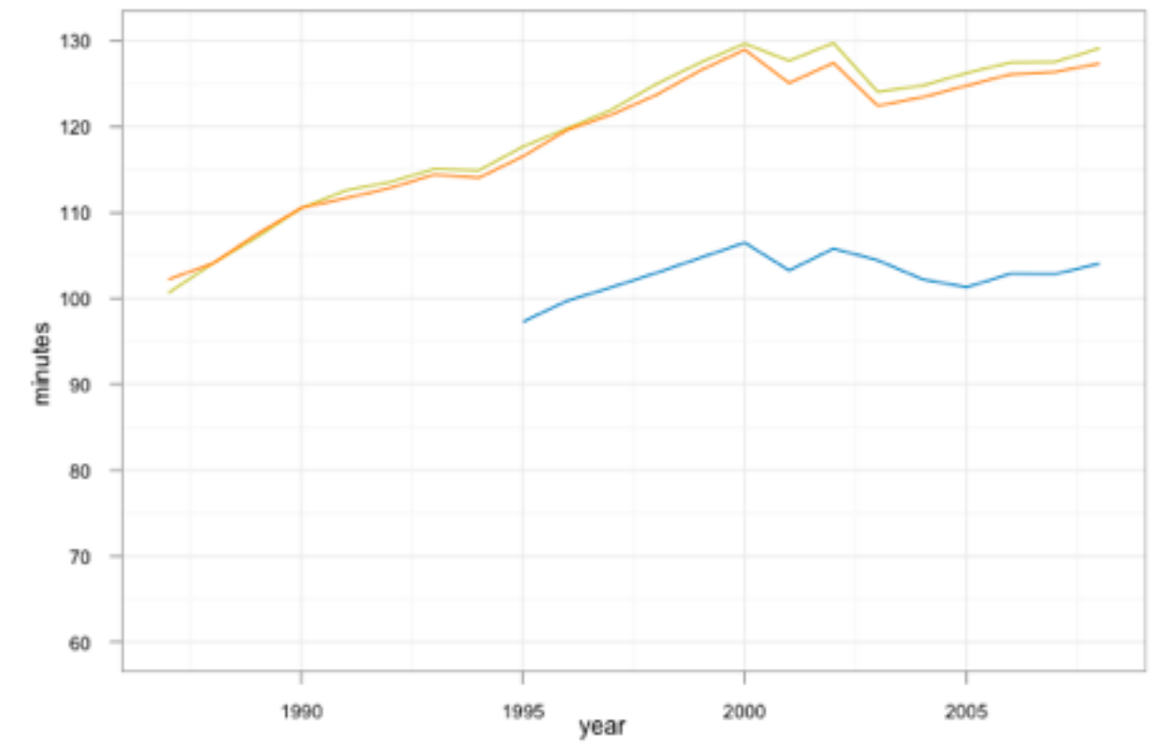
```
mr.year.market.enroute_time = function (input, output) {
     mapreduce(input = input,
                    output = output,
                    input.format = asa.csv.input.format,
                    map = mapper.year.market.enroute_time,
                    reduce = reducer.year.market.enroute_time,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

out = mr.year.market.enroute_time(hdfs.data, hdfs.out)
[...]
results = from.dfs( out )
results.df = as.data.frame(results, stringsAsFactors=F )
colnames(results.df) = c('year', 'market', 'flights', 'scheduled', 'actual',
'inflight')

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

see R/2-airline.R

with all the crunching out of the way, R can handle the rest itself

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),
                      inflight = weighted.mean(inflight, 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=inflight), color='#4689cc') + theme_bw() +
ylim(c(60, 130)) + ylab('minutes')
```



mapreduce() behind the scenes

- How did our input formatter, mapper, and reducer code get to all the nodes?
  - We simply defined them locally, then called mapreduce():

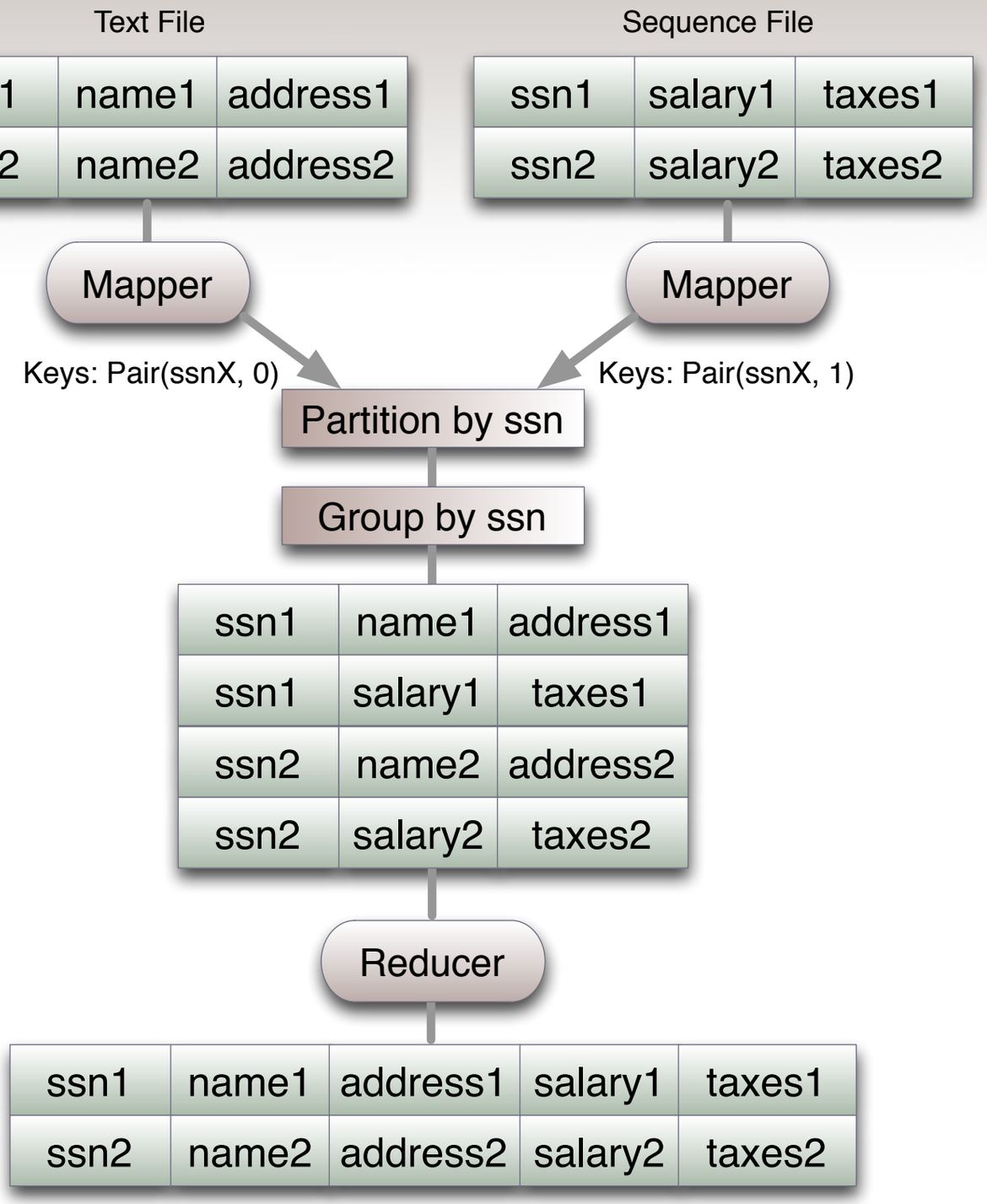
```
mapreduce(input = input,
          output = output,
          input.format = asa.csv.input.format,
          map = mapper.year.market.enroute_time,
          reduce = reducer.year.market.enroute_time,
          verbose=T)
```

- mapreduce() serializes R's Global and local environment, including all objects (functions and data), and distributes them via Hadoop's distributed cache to all the task trackers
- Global and local environments are restored on all nodes automagically

Side-loading data with Hadoop's distributed cache

- If "Big Data" comes in through the front door (stdin in the case of Streaming)...
- ..."small data" can come in through the side door.
- Reference data, lookups, and small data sets of all kinds
- In general, joins occur in the reducer, so all the intermediate data must flow through the (expensive) sort-shuffle phase
- But with side-loaded data, joins can occur in the mapper
  - Such "map-side joins" are an important optimization in MapReduce

Reduce-side joins in MapReduce



Map-Side joins

```
mapper.year.mfr.enroute_time = function(key, val.df) {

  # Remove header lines, cancellations, and diversions:
  val.df = subset(val.df, Year != 'Year' & Cancelled == 0 & Diverted == 0)

  # merge in manufacturer data from global "lookup.df":
  val.df = merge(val.df, lookup.df, by.x='TailNum', by.y='n.number')

  # key consists of year, manufacturer
  output.key = data.frame(year=as.numeric(val.df$Year), mfr=val.df$mfr, stringsAsFactors=F)

  # emit data.frame of gate-to-gate elapsed times (CRS and actual) + time in air
  output.val = val.df[,c('CRSElapsedTime', 'ActualElapsedTime', 'AirTime')]
  colnames(output.val) = c('scheduled', 'actual', 'inflight')

  # and finally, make sure they're numeric while we're at it
  output.val = transform(output.val,
                         scheduled = as.numeric(scheduled),
                         actual = as.numeric(actual),
                         inflight = as.numeric(inflight)
  )

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

see R/3-join.R

Getting Started with R and Hadoop, Part II

Jeffrey Breen
Principal, Think Big Academy
jeffrey.breen@thinkbiganalytics.com
http://www.thinkbigacademy.com/

Big Data TechCon
Cambridge, MA

April 2014

Thank you! Questions?

Slides and code at
http://bit.ly/rmr2airline