Big Data with R and Hadoop

Jamie F Olson

June 11, 2015
R and Hadoop

Review various tools for leveraging Hadoop from R.

- MapReduce
- Spark
- Hive/Impala
- Revolution R
Scaling R to Big Data

R has scalability issues:

- Performance
- Memory
Scaling R to Big Data

R has scalability issues:

- Performance?
- Memory?
R Performance Limits

R performance bottlenecks are largely gone:

- Memory model tweaks
- Just-in-time compiler
- Highly performant data manipulation tools (e.g. dplyr, data.table)
R Memory Limits

Two choices for dealing with memory issues:

- Native R solutions: `ff`, `bigmemory`
- Leverage external tools: e.g. Hadoop, RDBMS
Value of Leaving R

- Purpose-built
- Highly engineered
- Better scalability
Cost of Context-Switching

External tools rarely share R’s core concepts and features:

- Vectorization
- Functional programming
Choosing External Tools

Does the value of the tool justify the increased development/conceptual cost?
Outline

1. MapReduce
2. Spark
3. Hadoop Databases
4. Revolution R ScaleR
5. Concluding
The Original Hadoop

- Map
- Reduce
Map

**Figure:** Apply the same computation to all data.
Reduce

Figure: Group and Reduce data
Why MapReduce?

- Data localization
- Simple to understand
- But extremely flexible
- Extreme scalability
Why not MapReduce?

- Large overhead
- Limited support for complex workflows
Really, Why MapReduce?

rmr2
Seemlessly Integrated with R

- First-class support for R types
  - Atomic vectors (including factor and NA)
  - Does what you want with data.frame, matrix, array
  - Works with any R values.

- Recreates your local session in Hadoop
  - Local and global variables
  - Packages
Hello rmr

x <- to.dfs(1:100)
y <- values(from.dfs(mapreduce(x,
    map=function(.x)keyval(x,x*x))))
head(y)

## [1]  1  4  9 16 25 36
Fancy rmr

```r
mpg_model <- lm(mpg ~ wt, data=mtcars)
new_weights <- to.dfs(seq(1,5,by=.01))
new_mpg <- values(from.dfs(mapreduce(x, map=function(.,wt)
   keyval(wt,predict(mpg_model, newdata=data.frame(wt=wt)))))
head(new_mpg)

##    1    2    3    4    5    6
## 31.94 26.59 21.25 15.91 10.56  5.22
```

R-Friendly Data Import

- Parse text with `read.table`
- Read JSON with `RJSONIO`
- Load Avro record data into `data.frame`
Directly Control Job Configuration

mapreduce(..., backend.parameters = list(...))

- Reduce tasks
- Memory/Cpu resources
- JVM parameters
Write Results to HDFS

MapReduce/rmr read *from HDFS* and write *to HDFS* making it easy to integrate scripts with the rest of your Hadoop workflows.
Great documentation on the wiki with tutorial and topics on performance and data formats.

Highly optimized typedbytes serialization written in C.

Installation only requires defining environmental variables.
Caveats

- Everything is batch.
- Data issues can be difficult to track down.

The API is great, but MapReduce can be limiting.
Try It Out

```r
install_github("RevolutionAnalytics/rmr2", subdir = "pkg")
rmr.options(backend = "local")
```
Hadoop 2.0

- Standalone compute engine ported to YARN
- Hybrid memory model keeps more data in RAM
- “Lazy” evaluation allows efficient and complex workflows
- API with more than just Map and Reduce
Why Spark?

- It runs faster (on the same workflow)
- You can develop faster
- Iterative algorithms are feasible
Why not SparkR?

Version: 0.1
Writing for Spark not for R

Spark uses key-value tuples, so does SparkR

tuple <- list(list("key1", "value1"), list("key2", "value2"))

This is an awkward value in R.
Wordcount: rmr

```r
mapreduce(input_txt, map = function(.x, txt) {
  words <- unlist(strsplit(txt, " "))
  keyval(words, 1)
}, reduce = function(word, ones) {
  keyval(word, sum(ones))
})
```
words <- `flatMap`(lines, function(line) {
  `strsplit`(line, " ")[[1]]
})
wordCount <- `map`(words, function(word) `list`(word, 1L))

counts <- `reduceByKey`(wordCount, "+", 2L)
SparkR Tuples

This is awkward and slow to do in R.

```r
wordCount <- map(words, function(word) list(word, 1L))
```

Compared with a vectorized version implemented through an API like keyval.

```r
## NOT VALID SPARKR
wordCount <- map(words, function(wordsVec) keyval(wordsVec, 1L))
```
Limited API for Data Formats

All data starts is a text file.

You receive it as a character vector.
Installation Troubles

- Requires compilation specific to your specific:
  - Hadoop version
  - Spark version
  - YARN vs no-YARN

- Additional build tools are required
  - Scala
  - Maven
Return Results to R

- Enables exploratory data analysis and ad-hoc analytics
- Cannot return output to HDFS for integration with other tools
Why SparkR, Again?

It’s the future.

The API just needs some work.
Try It Out

install_github("amplab-extras/SparkR-pkg", subdir = "pkg")
sc <- sparkR.init(master = "local")
Outline

1. MapReduce
2. Spark
3. Hadoop Databases
4. Revolution R ScaleR
5. Concluding
Can you (S/H)QL?

- Integrate with existing data lake
- Leverage existing SQL skills
Connecting from R

Connection options

- (R)ODBC*
- (R)JDBC

*Available from Hadoop distributors.
Caveats

- Apache Sentry
- Kerberos can be tricky
- rJava Java version must match Hadoop’s
- Many driver changes in the past few years
Thoughts

Danger Zone:

- Dynamic SQL in R is not pretty
- Hive QL has a strong “flavor”

Recommend:

- Data reshaping for inputs
- ETL to recombine R outputs
RJDBC Examples

```r
input_df <- dbGetQuery(
    paste0("SELECT * FROM ",
    command_line_arg))

# Do Something
hdfs.put(output_df,output_hdfs_path)

dbSendQuery(paste0(
    "CREATE EXTERNAL TABLE r_output",
    "(...)",
    "LOCATION ",output_hdfs_path)
```

Outline

1. MapReduce
2. Spark
3. Hadoop Databases
4. Revolution R ScaleR
5. Concluding
Write Once Deploy Anywhere

- Efficient linear-scaling algorithms for big data
- Cross-platform support for distributed computing
  - Hadoop
  - In-Database
  - Platform LSF
Modeling not Distributed Computing

- Focus on modeling
  - Generalized Linear Models
  - Tree-based models
  - Clustering

- And data transformation
ScaleR on Hadoop

- “Inside” architecture with R in the cluster
- Maximum scalability
- Currently uses MapReduce
- “Beside” architecture with R on the edge node
- Efficient binary format optimizes IO
- Medium-large data(< 1 TB) can be faster than in-Hadoop
- Connect to Hive/Impala via ODBC(with unixODBC)
Outline

1. MapReduce
2. Spark
3. Hadoop Databases
4. Revolution R ScaleR
5. Concluding
It Depends

- rmr2 is mature and integrated
- Spark is better, but SparkR is immature
- There’s always a role for SQL
Questions?
Thank you

Revolution Analytics is the leading commercial provider of software and support for the popular open source R statistics language.

www.revolutionanalytics.com
1.855.GET.REVO
Twitter: @RevolutionR