REVOLUTION ANALYTICS

Getting started with RHadoop

RevolutionAnalytics

December 3, 2013

Who are You?

- R
- Statistics
- Machine learning
- Hadoop
- Data
- Other

Course Objectives

We will

- Understand Map/Reduce concepts
- Be aware of the Hadoop system
- Reason about MapReduce algorithms and performance
- Convert standard R code to MapReduce code
- Know the basics of the RHadoop package, especially `rmr`

We will not cover

- RHadoop installation and configuration
- Hadoop configuration
- Advanced features of `rhdfs`, `rhbase`

Course Outline

- Background: Thinking Computationally
- The MapReduce Framework
- The Hadoop Infrastructure
- Concrete Examples
- Designing Map and Reduce functions
- MapReduce Limitations
- MapReduce Performance Considerations
- More Examples
- Concluding Thoughts

Outline

Mathematical vs Computational Thinking

*Why do **I** need to care about computer hardware/infrastructure?*

- It impacts model performance
- It impacts model feasibility
- **With large datasets, you must consider the computational aspects of your model**

What is a computer?

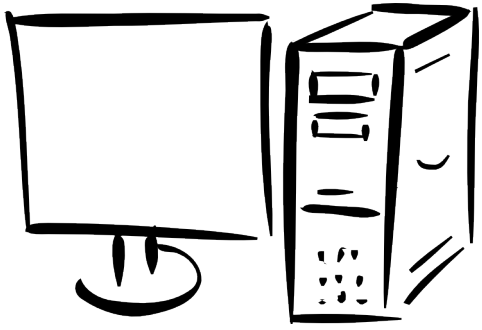


Figure : What is your mental model often computation?

Computers

Computers

Photograph © Andrew Dunn, 9 November 2005

Computers

Photograph © Andrew Dunn, 9 November 2005

Computers

Photograph © Andrew Dunn, 9 November 2005

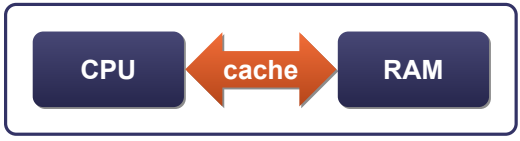
A lot of other things

Computing Mental Model

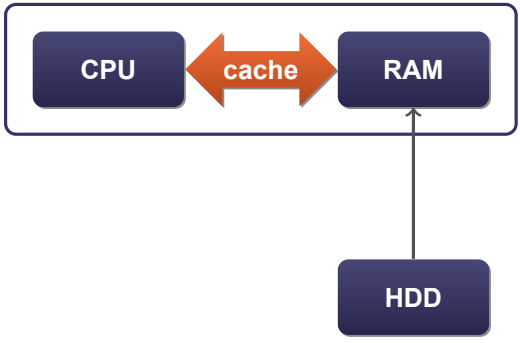
Computing Mental Model

Computing Mental Model

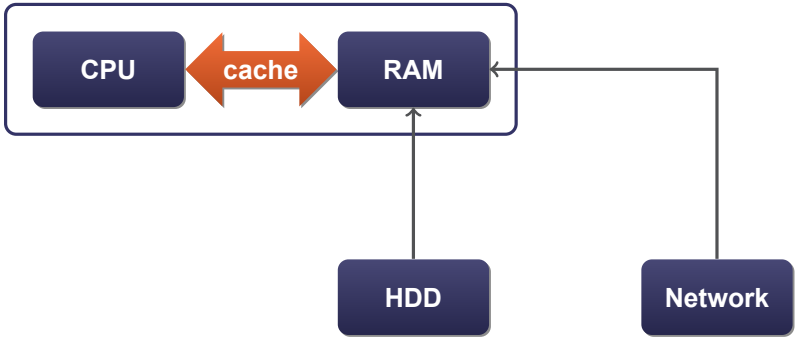
Computing Mental Model



Computing Mental Model



Computing Mental Model



From Math to Computation

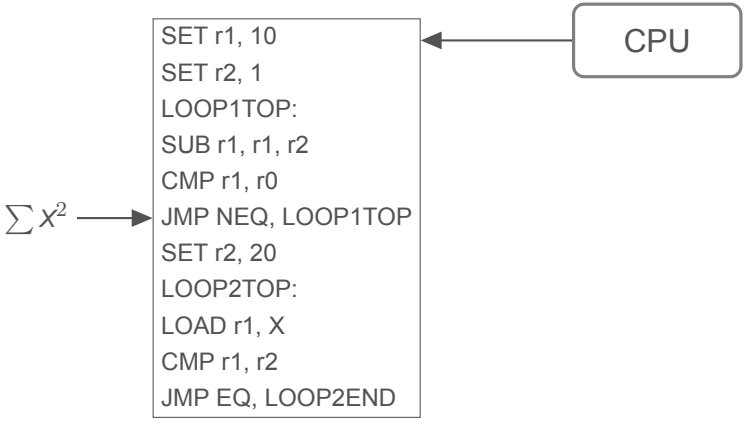
$$\sum x^2$$

From Math to Computation

$\sum X^2$ →

```
SET r1, 10
SET r2, 1
LOOP1TOP:
SUB r1, r1, r2
CMP r1, r0
JMP NEQ, LOOP1TOP
SET r2, 20
LOOP2TOP:
LOAD r1, X
CMP r1, r2
JMP EQ, LOOP2END
```

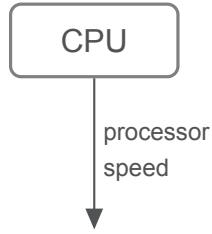
From Math to Computation



From Math to Computation

$\sum X^2$ →

```
SET r1, 10
SET r2, 1
LOOP1TOP:
SUB r1, r1, r2
CMP r1, r0
JMP NEQ, LOOP1TOP
SET r2, 20
LOOP2TOP:
LOAD r1, X
CMP r1, r2
JMP EQ, LOOP2END
```

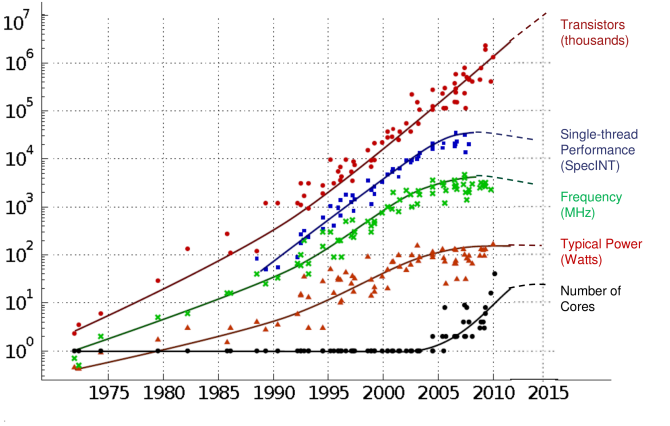


Single Core Computing

Evaluate a sequence of operations in order

- Simple model for programming
- Predictable performance
- Depends on single core “speed”

Single Core Speed



Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten
Dotted line extrapolations by C. Moore

Figure : Image from these slides

Single Core Speed

Despite the ever-shrinking transistor, gains in single-processor performance have slowed over the last decade.

- Excess heat
- Energy consumption
- Physical limitations

How do we go faster?

How do we go faster?

Parallelism

Doing multiple things at the same time.

A Metaphor

Photo by Matthew Simoneau

A Metaphor

Photo by Matthew Simoneau

Photo by [Dale Gillard](http://www.flickr.com/photos/matthewsim/4837794129/in/photolist-8nuXDB/)

A Metaphor, Tortured

What to do?

- Keep going

A Metaphor, Tortured

What to do?

- Keep going
- Carry more buckets

A Metaphor, Tortured

What to do?

- Keep going
- Carry more buckets
- Buy a dump truck and front-end loader

A Metaphor, Tortured

What to do?

- Keep going
- Carry more buckets
- Buy a dump truck and front-end loader
- Pay a circus to train some monkeys to do it for you

Parallel Computing Options

Our options are functionally the same

- Wait longer

Parallel Computing Options

Our options are functionally the same

- Wait longer
- Multi-core

Parallel Computing Options

Our options are functionally the same

- Wait longer
- Multi-core
- High-performance computing

Parallel Computing Options

Our options are functionally the same

- Wait longer
- Multi-core
- High-performance computing
- Clusters of machines

Parallel How?

In order to work together, processors must share resources.

- Connect multiple cores together and send instructions to all of them

Parallel How?

In order to work together, processors must share resources.

- Connect multiple cores together and send instructions to all of them
  - How do you do connect the cores together?

Parallel How?

In order to work together, processors must share resources.

- Connect multiple cores together and send instructions to all of them
  - How do you do connect the cores together?
  - What do they share?

Parallel How?

In order to work together, processors must share resources.

- Connect multiple cores together and send instructions to all of them
  - How do you do connect the cores together?
  - What do they share?
  - **The shared resource/communication will be the bottleneck**

Parallelism: What is shared/connected?

- Everything: multi-core cpu (but only a few cores)

Parallelism: What is shared/connected?

- Everything: multi-core cpu (but only a few cores)
- Almost everything: high-performance-computing (really expensive and special-purpose)

Parallelism: What is shared/connected?

- Everything: multi-core cpu (but only a few cores)
- Almost everything: high-performance-computing (really expensive and special-purpose)
- (Almost) Nothing: clusters of computers (e.g. Hadoop)

Why does this matter?

Resource	Delay
L1 cache reference	0.5 ns
L2 cache reference	7 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Read 1 MB sequentially from disk	20,000,000 ns

Table: *Latency Comparison Numbers* source

> *A 3 Ghz processor can execute 3 instructions in 1 ns: You'd better be sending a lot of work at once*

Outline

What is MapReduce?

- Programming framework for distributed computing
- Provides flexibility to programmer while allowing efficient implementation
- Two concepts
  - `Map`
  - `Reduce`

Map

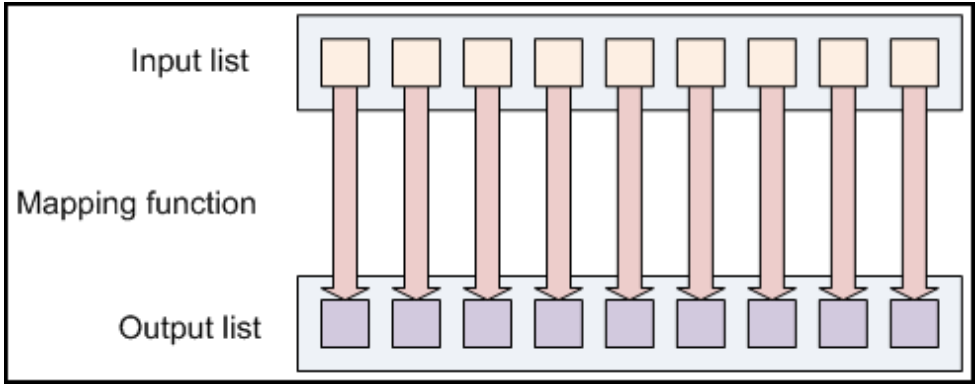


Figure : Apply the same computation to all data

- One-to-one function
- Your output will be **the same size as your input**

Map Examples

- X^2
- $\sqrt{X}$
- $\cos X$
- $\log X$
- …

Reduce

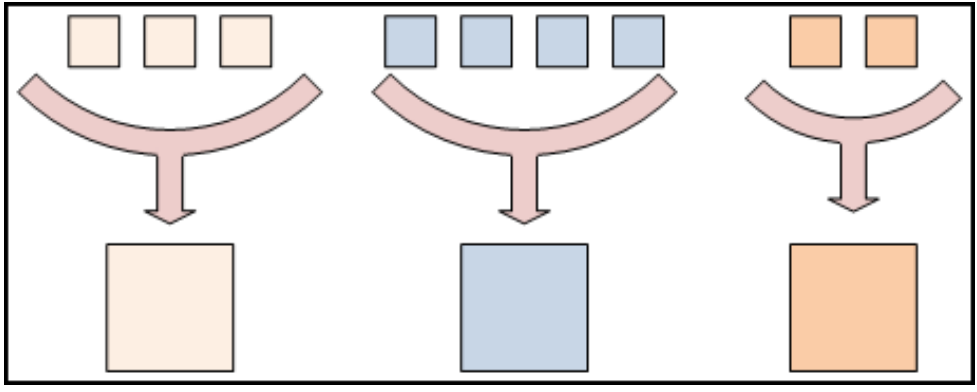


Figure : Group and Reduce data

- Many-to-one function
- Your output will be **smaller than your input**

Reduce Examples

- $Var[X]$
- $\sum X$
- ...

Map and Reduce in R?

What functions in R are like Map/Reduce?

- Rewrite `x^2` as a map-like

Map and Reduce in R?

What functions in R are like Map/Reduce?

- Rewrite `x^2` as a map-like
- `sapply(x,function(xi) xi^2)`

Map and Reduce in R?

What functions in R are like Map/Reduce?

- Rewrite `x^2` as a map-like
- `sapply(x,function(xi) xi^2)`
- If `Map` is like the `applys`, what's like `Reduce`

Map and Reduce in R?

What functions in R are like Map/Reduce?

- Rewrite `x^2` as a map-like
- `sapply(x,function(xi) xi^2)`
- If `Map` is like the `applys`, what's like `Reduce`
- `aggregate`, `ddply` + `summarize`

Map and Reduce in R?

What functions in R are like Map/Reduce?

- Rewrite `x^2` as a map-like
- `sapply(x,function(xi) xi^2)`
- If `Map` is like the `applys`, what's like `Reduce`
- `aggregate`, `ddply` + `summarize`
- `lapply(split(kv$values,kv$keys),reduce)`

Communication in MapReduce

No coordination or communication happens until the `Reduce` phase

Coordination is Still Expensive

Single-core:

```
system.time(result=lapply(data,function(x) x^2))
```

Multi-core:

```
system.time(result=mclapply(data,function(x) x^2))
```

Hadoop:

```
system.time(result=mapreduce(data,map=function(x) x^2))
```

Outline

Why Hadoop?

Hadoop is an *open source implementation* of Google's MapReduce processing framework.

- Many components: e.g. distributed file system, job scheduling and management system.
- MapReduce is a *programming pattern* distributed computing
- Two primary steps:
  - "map" phase which picks out identifying and subject data ("key" and "value")
  - "reduce" phase aggregates values (grouped by key value)
- **Programmer/analyst need only write the mapper and reducer** while the system handles the rest(But you need to understand performance implications.)

Why R?

R is an open source environment for statistical programming and analysis.

- Active, growing community
- Large (massive, actually) library of add-on packages
- Commercial support, extensions, training
- Multi-paradigm language: functional, procedural, object-oriented
- Low-level interface with many languages (e.g. C, C++)
- Near a mathematical level of programming

Outline

rmr: Example 0

Let's harness the power of our Hadoop cluster… to compute some sums of squares:

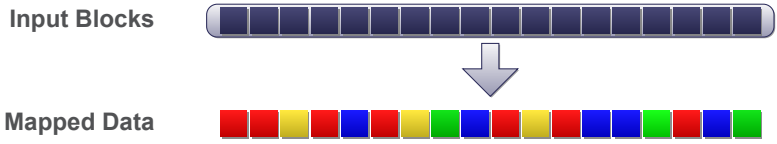
```
library(rmr2)
n <- 1000
small.ints = data.frame(
    color=sample(c("Blue","Green","Yellow","Red"),n,replace=TRUE),
    number=1:1000)
small.int.path = to.dfs(small.ints)
out = mapreduce(input = small.int.path,
    map = function(k,v) keyval(v$color, v$number^2),
    reduce = function(k,vv)keyval(k,sum(vv)) )
results = from.dfs( out )
results.df = as.data.frame(results, stringsAsFactors=F )
```

Example 0: Data Flow

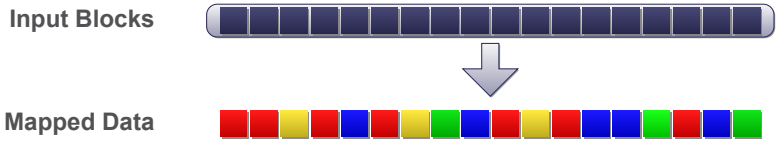
Example 0: Data Flow

Example 0: Data Flow

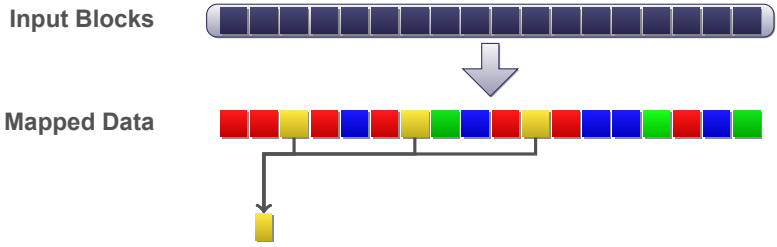
Example 0: Data Flow



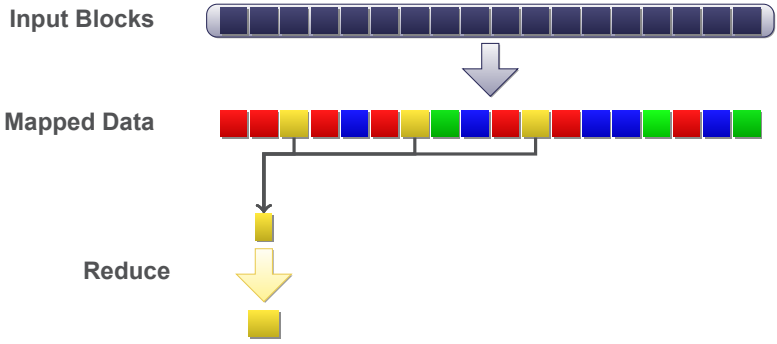
Example 0: Data Flow



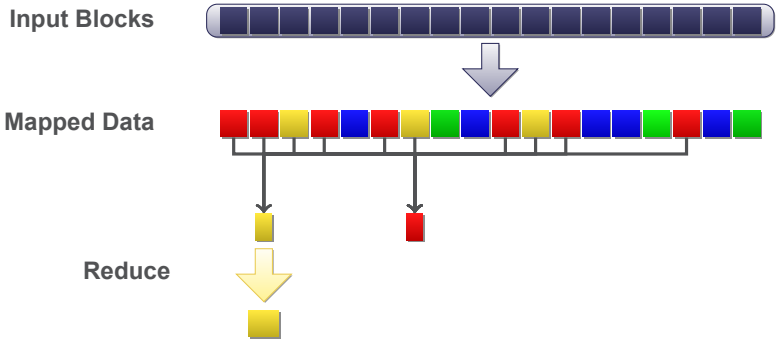
Example 0: Data Flow



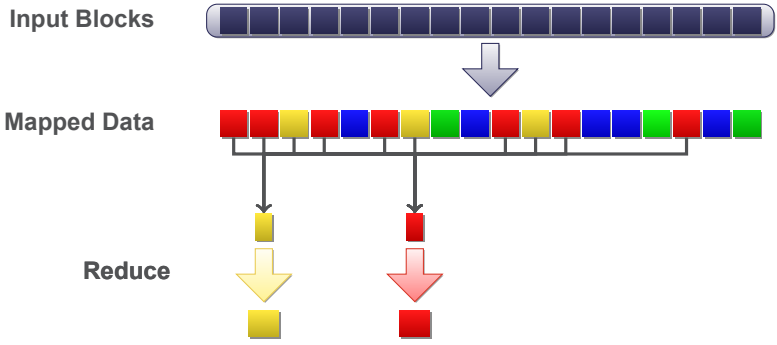
Example 0: Data Flow



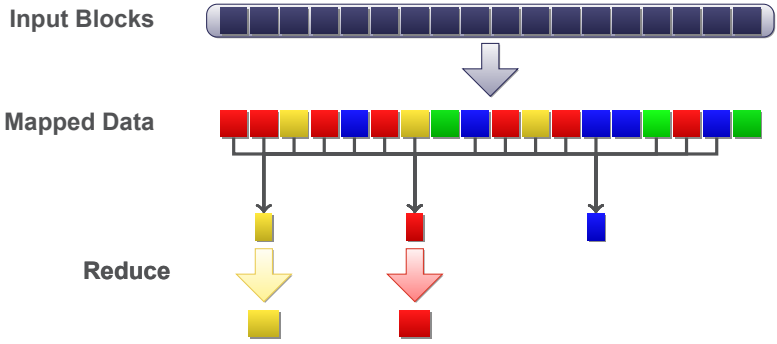
Example 0: Data Flow



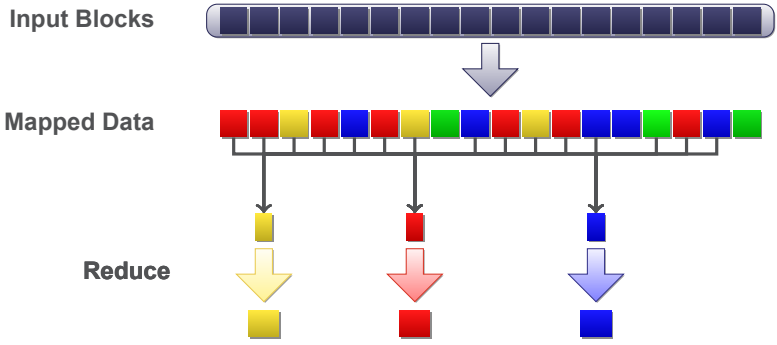
Example 0: Data Flow



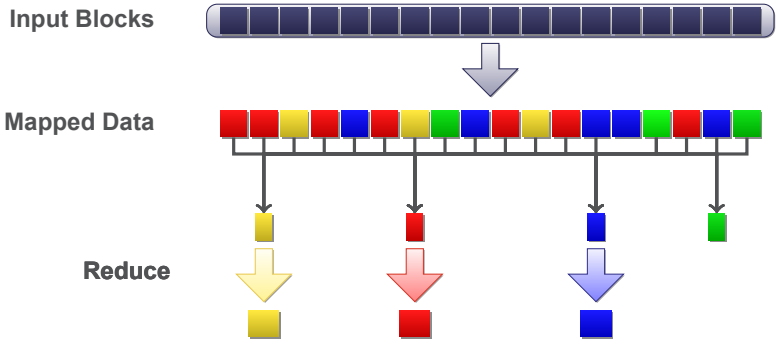
Example 0: Data Flow



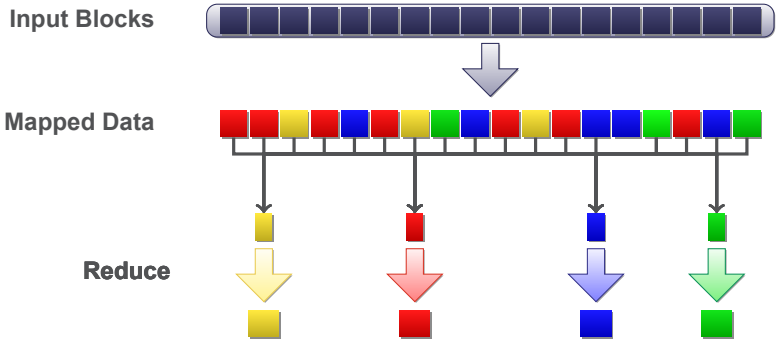
Example 0: Data Flow



Example 0: Data Flow



Example 0: Data Flow

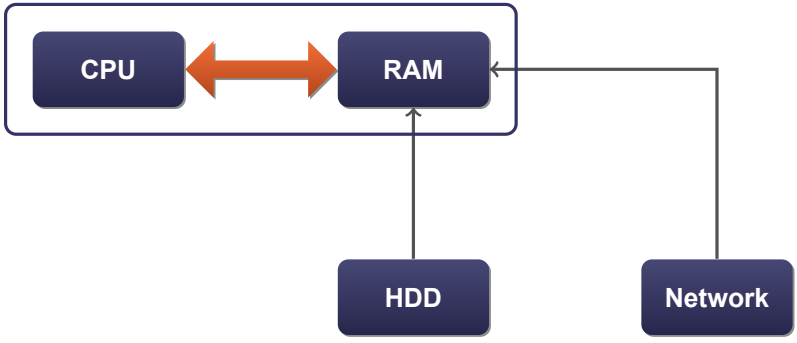


Outline

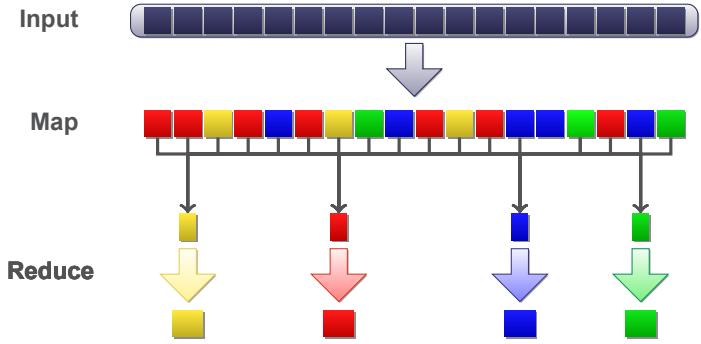
The Hadoop System

Many components for two core services: - MapReduce - HDFS

Hadoop Mental Model



MapReduce Data Flow



Hadoop: The Metaphor

You've decided to hire the monkeys, what could possibly go wrong?

- Certain parts of the pile are heavier than others
- They see something shiny
- They get hungry

Hadoop: Potential Problems

What problems can happen when trying to run a job?

- Certain parts of the data take longer to process
- Hardware problems: machine dies, network traffic
- Software problems: run out of memory

Hadoop: Solutions

Hadoop can handle all of these things and more, using:

- Data redundancy
- Load-balancing
- Job monitoring and resubmitting

Hadoop: Architecture

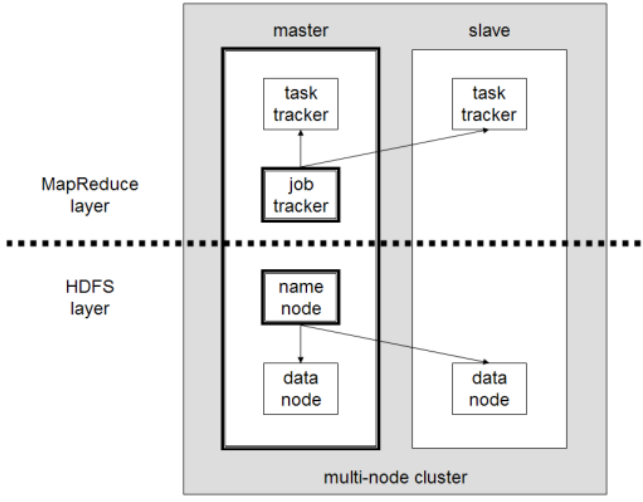


Figure : MapReduce and HDFS on Hadoop

Outline

rmr2 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

Components of basic rmr2 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

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 rmr2: writing a mapper & reducer, submitting the job, and fetching the results

Data Any text will do

Sample data (Shakespeare, Bible, etc.) available from Cloudera

wordcount: Code

```
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: Code

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

wordcount: Map

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

wordcount: Map Input and Output

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

```
k = NULL
v = c("KING HENRY IV\tSo 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" "\tSo" "shaken"...
 $ val: num 1
```

wordcount: Reducer

```
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, ...)
```

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

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

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
6     hands     2
```

Your turn: get rid of the punctuation

The results in the previous slide may not be exactly what we're looking for:

```
9995  good; 29
9996  good? 15
9997  goods 13
9998  goose 10
```

Your mission: Modify the example so the punctuation is not included in the words. Use rmr2's local backend (set `LOCAL=T`) and the supplied data extract to try it out.

Remember: there's always more than one way to do it… but you might want to look at the regular expression in the mapper

Outline

Normalizing

Normalize `mpg` within each `cyl` group for the mtcars dataset.

```
mtcars_hdfs <- to.dfs(mtcars)
mt_norm <- as.data.frame(from.dfs(mapreduce(input=mtcars_hdfs,
      map=function(k,v){
      ...
      },
      reduce=function(k,vv){
      ...
      })))
```

Scoring a Model

Given a linear model for predicting `mpg` using `hp` and `drat`, predict `mpg` for new cars.

```
mpg_model <- lm(mpg ~ hp+drat,data=mtcars)
new_data <- to.dfs(data.frame(hp=runif(1000,50,350),drat=rnorm(1000,4.0)
predicted <- as.data.frame(from.dfs(mapreduce(new_data,...)))
```

Hint: check out the `predict.lm` function