Introduction

SCALABLE DATA PROCESSING IN R

Simon Urbanek

Member of R-Core, Lead Inventive Scientist, AT&T Labs Research

bigmemory

- All data must be stored on a single disk
- Data must be represented as a matrix

iotools

- Data can multiple types - i.e., data frames
- Stored across multiple machines
- Processes data in "chunks"

Process one chunk at a time sequentially

- Limits resource usage by controlling chunk size
- Allows results to be carried over

Process each chunk independently

- Corresponds to split-compute-combine
- No information can be shared between chunks
- Allows parallel and distributed processing

Mapping and Reducing for More Complex Operations

```r
# Create a random vector
x <- rnorm(100)
# Find the mean
mean(x)
```

```
-0.01996644
```

```r
# Take the sum of chunks of
# the vector
sl <- Map(function(v) {
         c(sum(v), length(v))},
  list(x[1:25], x[26:100]))

# Add the sums and lengths
slr <- Reduce(`+`, sl)
# Find the mean
slr[1]/slr[2]
```

```
-0.01996644
```

Not all things fit into Split-Apply-Combine

Operations that require all the data at once, can't be computed using the Split-Apply-Combine approach.

Example: Median

However ..

> Many regression routines can be written in terms of split-apply-combine

Let's practice!

SCALABLE DATA PROCESSING IN R

A first look at iotools: Importing data

SCALABLE DATA PROCESSING IN R

Simon Urbanek

Member of R-Core, Lead Inventive Scientist, AT&T Labs Research

Chunk-wise processing

1. Load pieces of data
2. Convert them into native objects
3. Perform computation and store the results

Repeat 1 to 3 until all data is processed

Importing data

- Often loading data takes more time than processing, and it happens in 2 steps
    - Retrieving data from disk is a relatively slow operation
    - Converting raw data into native R objects

Importing data using iotools

> In the iotools package, the physical loading of data and parsing of input into R objects are separated for better flexibility and performance.

iotools: Importing data

- `readAsRaw()` reads the entire data into a raw vector
- `read.chunk()` reads the data in chunks into a raw vector

iotools: Parsing data

- `mstrsplit()` converts raw data into a matrix
- `dstrsplit()` converts raw data into a data frame

iotools: Loading and parsing data

`read.delim.raw()` = `readAsRaw()` + `dstrsplit()`

Chunk-wise processing

- Not necessary to import all the data
- Read a "chunk" of rows at a time from the data source
- No intermediate structure

File connections

```
# Open a file connection
fc <- file("data-file.csv", "rb")
# Read the firt line if the data has a header
readLines(fc, n = 1)
....
# Code to import and parse the data
....
# Close the file connection
close(fc)
```

Let's practice!

SCALABLE DATA PROCESSING IN R

chunk.apply

SCALABLE DATA PROCESSING IN R

Simon Urbanek

Member of R-Core, Lead Inventive Scientist, AT&T Labs Research

chunk.apply()

- Abstracts the looping process
- Enables Parallel execution
- `iotools` is the basis of `hmr`, which allows you to process data on the Apache Hadoop infrastructure

mstrsplit() reads chunks as matrices

```r
# Use chunk.apply to get chunks of rows from foo.csv
chunk_col_sums <- chunk.apply("foo.csv",
 # A function to process each of the chunk
 function(chunk) {
   # Turn the chunk into a matrix
   m <- mstrsplit(chunk, type = "numeric", sep = ",")
   # Return the column sums
   colSums(m)
 },
 # Maximum chunk size in bytes
 CH.MAX.SIZE = 1e5)
# Get the total sum
colSums(chunk_col_sums)
```

dstrsplit() reads chunks as data frames

```
# Use chunk.apply to get chunks of rows from foo.csv
chunk_col_sums <- chunk.apply("foo.csv",

 # A function to process each of the chunk
 function(chunk) {
   # Turn the chunk into a data frame
   d <- dstrsplit(chunk, col_types = rep("numeric", 3), sep = ",")
   # Return the column sums
   colSums(d)
 },
 # Maximum chunk size in bytes
 CH.MAX.SIZE = 1e5)

# Get the total sum
colSums(chunk_col_sums)
```

Parallelizing chunk.apply()

```r
# Use chunk.apply to get chunks of rows from foo.csv
chunk_col_sums <- chunk.apply("foo.csv",

 # A function to process each of the chunk
 function(chunk) {

   # Turn the chunk into a data frame
   d <- dstrsplit(chunk, col_types = rep("numeric", 3), sep = ",")
   colSums(d)
 },
 # 2 processors read and process data
 parallel = 2)

# Get the total sum
colSums(chunk_col_sums)
```

Note about parallelization

- Increasing the number of processors won't always speed up your code
- There are usually diminishing returns when you add additional processors on a single machine

Let's practice!

SCALABLE DATA PROCESSING IN R