Big data processing

Fundamentals of Computing and Data Display

Christoph Kern Ruben Bach

(c.kern, r.bach)@uni-mannheim.de

Outline

Introduction

Approaches to programming with big data

- Utilize local machine better (multiple cores)
  - Various R packages for parallelization available
- Use better remote machine
  - e.g. AWS
- Use multiple remote machines (cluster)
  - Hadoop
  - Spark

Things to care about/ bottlenecks

- Processing capacity
- Disk, memory storage
- Transfer speed

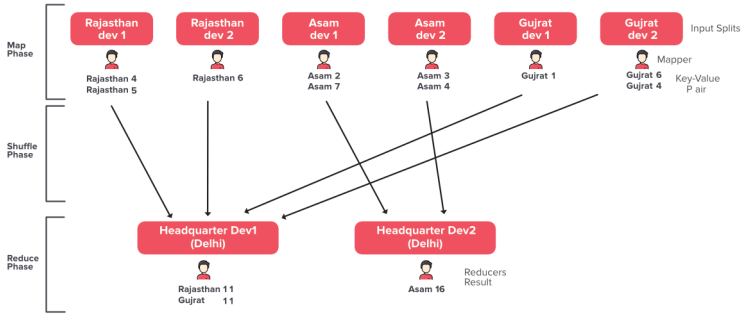
MapReduce

A programming paradigm for parallel computing

1. Map
   - Input: A list of key-value pairs
   - Apply a (user-specified) map function (utilize multiple mappers)
2. Shuffle
   - Sorts and groups map output by keys
3. Reduce
   - Apply a reduce function on the grouped data
   - Output: List of values

MapReduce

Figure: MapReduce example[1]



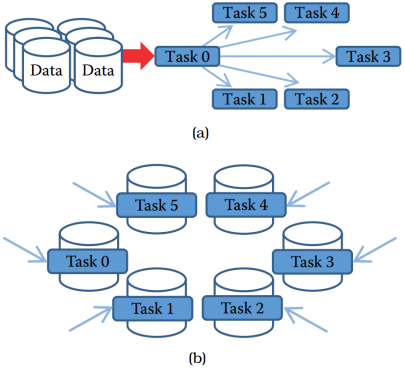
[1] https://www.geeksforgeeks.org/mapreduce-understanding-with-real-life-example/

Hadoop

A MapReduce implementation

- Hadoop Distributed File System (HDFS)
  - Input is split into blocks and stored on different nodes
  - Blocks are also replicated across nodes
- Bringing compute to the data
  - Combining computing and storage at each node
  - Enables fast data access (data locality)

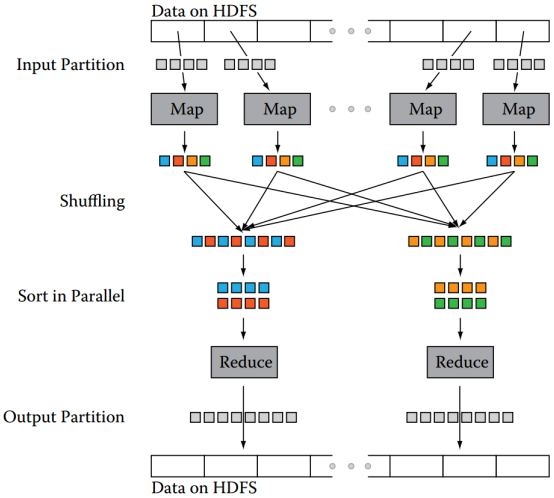
Figure: Traditional (a) and Hadoop's (b) computing approach[3]



[3] Foster et al. 2017

Hadoop

Figure: MapReduce process in Hadoop[4]



[4] Foster et al. 2017

Spark

A beyond-MapReduce cluster-computing framework

- Iterative processes in MapReduce typically require writing data to storage system
  - Slow data sharing, many read-write operations
- Spark builds on resilient distributed datasets (RDD)
  - Distributed collections of objects that may be computed on different nodes
  - RDDs can be kept loaded in memory for fast access

AWS

AWS

Amazon Web Services (AWS)

- Amazon Elastic Compute Cloud (EC2)
- Amazon Simple Storage Service (Amazon S3)
- ...many more

EC2 instances

- `https://aws.amazon.com/ec2/instance-types/`
- Optimized for different tasks
  - General purpose, compute, memory, storage, ...
- 0.5 GB up to 3904 GB of memory
- 1 core up to 96 cores
- Free tier `t2.micro` instance

AWS

Amazon Machine Image (AMI)

- Pre-packed system for usage on own server
- Operating system + (potentially) software
- Pre-configured **RStudio Server AMIs** available!
  - `http://www.louisaslett.com/RStudio_AMI/`

AWS

Setup process

1. Create AWS account and sign in
2. Choose an (RStudio Server) AMI
3. Choose an instance type
4. Edit the security group settings and open ports
   - 22 for SSH
   - 80 for HTTP
   - 443 for HTTPS
   - 8787 for RStudio Server
5. Create (or choose) a key pair and download file
6. Launch Instance
7. Open the public DNS in your favorite browser
8. Login to Rstudio Server
   - Your Username: rstudio
   - Your Password: <Your Instance ID>

Tools for R

Tools for R

Selected tools for big(ger) data processing in R

- Import and transform data with enhanced data frames
  - `data.table`
- Speed up processing by parallelizing code
  - `parallel`, `doparallel`
- Utilize Spark cluster
  - `sparklyr`

data.table

Table: Data import in R

	data.table `data.table`	**Base R** `data.frame`	**readr**[5] `tibble`	**foreign** `data.frame`	**haven** `tibble`
Generic / Text Files	`fread()`	`read.table()`	`read_delim()`		
Comma-Separated	`fread()`	`read.csv()` `read.csv2()`	`read_csv()` `read_csv2()`		
Tab-Separated	`fread()`	`read.delim()` `read.delim2()`	`read_tsv()` `read_tsv2()`		
SPSS files				`read.spss()`	`read_sav()`
Stata files				`read.dta()`	`read_dta()`
SAS files				`read.ssd()`	`read_sas()`

[5] Up to 10x faster than Base R, 1.2-2x slower than data.table (`https://readr.tidyverse.org/`).

data.table

What is a `data.table`?

- Enhanced data frame
- Recommended for large datasets
- Comes with its own (SQL-inspired) syntax
  - `DT[i, j, by]`
  - WHERE (i), SELECT or UPDATE (j), GROUP BY (by)
- (Or) can be used as a `dplyr` backend
  - `https://cran.r-project.org/web/packages/dtplyr/`

data.table

Table: Working with `data.table`

(a) Selecting rows

Action	`data.table`
Select specific rows	`DT[c(1,2,5:10)]`
Select last row	`DT[.N]`
Filter by condition	`DT[a==15]`
Select first k rows	`head(DT, k), DT[1:k]`
Select last k rows	`tail(DT, k)`

(b) Selecting columns

Action	`data.table`
Select by col. num.	`DT[ , c(1,3:5)]`
Select by name	`DT[ , .(A,B,C)]`
Modify column	`DT[ , .(A,round(B))]`
Create new column	`DT[ , .(A, M = B+C)]`

data.table

Table: Working with `data.table`

(a) Adding and Updating Columns

Action	`data.table`
Update / Create new var.	`DT[, X := k]`
Update / Create multiple vars	`DT[, c("M","N") := list(A,B/2)]`
Remove a column	`DT[, A := NULL]`
Replacing missing val.	`DT[is.na(A), A := k]`

(b) Aggregating

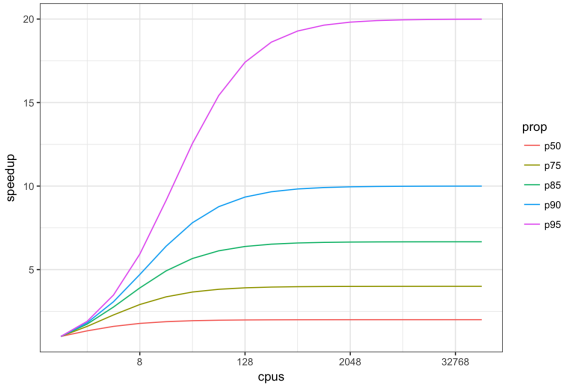
Action	`data.table`
By one group	`DT[,.(agg = f(X)), by=A]`
By multiple groups	`DT[,.(agg = f(X)), by=.(A,B)]`
Multiple vars, same function	`DT[,.(agg1 = f(X), agg2=f(Y)), by=.(A,B)]`
Multiple vars, different functions	`DT[,.(agg1 = f(X), agg2=g(Y,Z),by=.(A,B,N=f(C))]`

(do)parallel

When to parallelize?

- Perfectly ("embarrassingly") parallel vs. inherently serial problems
  - Is function 2 independent or dependent on the output of function 1?
- Overhead
  - Copying data/ code when initializing parallelization reduces theoretical gains

Figure: Amdahl's law[6]



[7] https://nceas.github.io/oss-lessons/parallel-computing-in-r/parallel-computing-in-r.html

(do)parallel

Two approaches to parallel processing in R

- `apply`-like iterations: `parallel`
  - `multicore`
    - Fork/ copy process to multiple processors on one machine (excludes Windows)
  - `snow`
    - Spawn new process on local or remote machine (cluster of computers)
  - `vignette("parallel")`
- For loops: `doparallel`
  - `parallel`
  - `foreach`
  - `vignette("gettingstartedParallel")`

sparklyr

An R interface to Spark

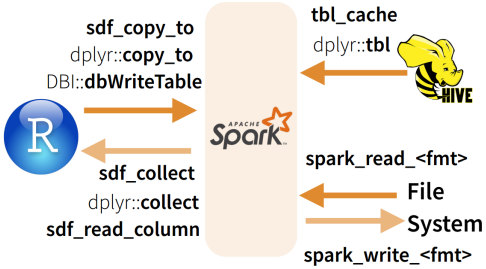
- Provides a `dplyr` backend to communicate with data in a Spark cluster
- Integrated into RStudio IDE
- Connect to a local Spark cluster from within R
  1. Install Spark locally: `spark_install()`
  2. Connect: `spark_connect(master = "local")`
  3. Experiment with Spark
  4. Disconnect: `spark_disconnect()`
- Connect to a remote Spark cluster

sparklyr

Example workflow

1. Connect to Spark
2. Copy data from R, file system, database to Spark
3. Communicate with data in Spark
   - Using `dplyr` language
   - Using SQL with `DBI` and `dbGetQuery()`
4. Run analysis within Spark (`MLlib`)
5. Extract results, create plots in R

Figure: Moving data from/to Spark



Resources

- Spark for Social Science
  - https://urbaninstitute.github.io/spark-social-science-manual/
- R Task View
  - https://cran.r-project.org/web/views/HighPerformanceComputing.html
- R Tools
  - https://github.com/Rdatatable/data.table/wiki
  - http://spark.rstudio.com/
- Cheatsheets
  - https://github.com/rstudio/cheatsheets/raw/master/datatable.pdf
  - https://github.com/rstudio/cheatsheets/raw/master/sparklyr.pdf

References

Foster, I., Ghani, R., Jarmin, R. S., Kreuter, F., and Lane, J. (Eds.). (2017). *Big Data and Social Science: A Practical Guide to Methods and Tools*. Boca Raton, FL: CRC Press Taylor & Francis Group.

White, T. (2015). Hadoop: The Definitive Guide. Sebastopol, CA: O'Reilly Media.