Large Data Processing in R

Naeem Khoshnevis

Research Software Engineer

Ph.D. Geophysics

M.Sc. Computer Science

HUIT Research Computing

Harvard University

April 7, 2023

Overview

- ✓ R
- ✓ Computing resources
- ✓ Definition of big data
- ✓ Different types of computational bounds
- ✓ Working with data that does not fit into the memory
- ✓ Processing a single instruction multiple data problem on shared and distributed memory systems

Slides and the course materials are available at the following repository:

https://github.com/fasrc/User_Codes/tree/master/Large_Data_Processing_R

R

- R is a language and environment for statistical computing and graphics.
- Created by statisticians Ross Ihaka and Robert Gentleman.
- Was released in 1995 as an opensource software.

- **R is single-threaded**
  - By default, R runs only on a single thread on the CPU.
- **R Processes data in-memory**
  - In general, the size of memory limits the size of data.

- **CRAN**
- **RStudio**
- **tidyverse**

https://www.r-project.org/about.html
https://en.wikipedia.org/wiki/R_(programming_language)

Computing Systems

Cloud Computing

Google Cloud

aws

Azure

Grid Computing

World Community Grid

7.5 M devices are connected.

Distribubted Memory

Cluster Computing

Shared Memory

Intel® Xeon Platinum 9282
Total Cores: 56
Total Threads: 112

Multiple Core

Intel® Core™ i7-8557U Processor
Total Cores: 4
Total Threads: 8

Single Core

Intel® Celeron® Processor G440
Total Cores: 1
Total Threads: 1

Big Data Categories by Data Sizes

Small	Medium	Big
Easy to use on PC 16 GB RAM 4 GB Data	Easy to use on a server 500 GB RAM 125 GB Data	Needs to be spread across multiple nodes 500+ GB Data

Data is big if it does not fit into the system's memory.

- **90%** can be reduced to "small data" problems through sampling, sub-setting, or summarizing.
- **9%** Can be reduced to multiple small data problems through chunking and iterating.
- **1%** Irreducibly big.

More available options

More Development time

Sources:

- R for Big Data in the Cloud by Mark Vaisman, Pipelines for data analysis in R by Hadley Wickham.

Computational Bounds

- **I/O Bound**
  - The progress of the processes are limited by the speed of the I/O.
  - For example, reading lines of a file on disk and running some analyses.

✓ Use an efficient data format
✓ Only read data that you need
✓ Use asynchronous programming

- **CPU Bound**
  - The progress of the processes are limited by the speed of CPU.
  - For example, working with small matrices that fits in the memory.

✓ Improve the algorithm
✓ Use parallel processing

- **Memory Bound**
  - The progress of the processes are limited by the amount of memory.
  - For example, working with large matrices.

✓ Improve the algorithm
✓ Distribute work among nodes
✓ Only read data that you need
✓ Spill to disk (or use swap memory)

Importance of the Data Format

Example of NYC Taxi Data

(Working with data that does not fit in Memory)

Data Format

- Highest parsing overhead
- Inefficient storage
- No well-defined schema

- Least parsing overhead
- Most efficient storage
- Well defined schema

Source: The Parquet Format and Performance Optimization Opportunities Boudewijn Braams (Databricks) (https://www.youtube.com/watch?v=1j8SdS7s_NY)

Large Data with Appropriate Data Format

How many were Lyft rides taken in New York City during 2020?

Taxi & Limousine Commission

hvfhs_license_num	dispatching_base_num	pickup_datetime	dropoff_datetime	PULocationID	DOLocationID	SR_Flag
HV0003	B02682	2020-01-31 23:14:53	2020-01-31 23:38:04	170	145	NA
HV0005	B02510	2020-01-31 23:32:57	2020-01-31 23:39:09	132	10	1
HV0003	B02764	2020-01-31 23:29:14	2020-01-31 23:57:17	48	148	NA
HV0003	B02395	2020-01-31 23:01:04	2020-01-31 23:15:42	152	159	NA
HV0003	B02395	2020-01-31 23:17:42	2020-01-31 23:51:42	159	191	NA
HV0003	B02395	2020-01-31 23:53:46	2020-02-01 00:06:56	191	10	NA

9.06 GB

fhvhv_tripdata_2020-01.csv

fhvhv_tripdata_2020-02.csv

fhvhv_tripdata_2020-03.csv

...

fhvhv_tripdata_2020-11.csv

fhvhv_tripdata_2020-12.csv

```
library(tidyverse)
fhvhv_data <- map(fhvhv_csv_files, read_csv) %>% bind_rows(show_col_types=FALSE)
```

```
## Error in eval(expr, envir, enclos): cannot allocate vector of size xx Mb.
```

Large Data with Appropriate Data Format (Cont.)

Solution 0

- Read all csv files.
- Combine them into one.
- Select the “hvfhs_license_num” column.
- Count the occurrences of HV0005.

```
library(tidyverse)
fhvhv_data <- map(fhvhv_csv_files, read_csv) %>% bind_rows(show_col_types=FALSE)
```

However, this will raise an error (if data is large enough and system’s RAM is small enough).

```
## Error in eval(expr, envir, enclos): cannot allocate vector of size xx Mb.
```

Large Data with Appropriate Data Format (Cont.)

Solution 1

- Read each CSV file sequentially.
- Load only the "hvfhs_license_num" column.
- Count the number of occurrences of "HV0005" for each file.
- Sum up the counts from all files.

Large Data with Appropriate Data Format (Cont.)

Solution 2

- Read each CSV file in parallel.
- Load only the "hvfhs_license_num" column.
- Count the number of occurrences of "HV0005" for each file.
- Sum up the counts from all files.

+

Parallel Package

Large Data with Appropriate Data Format (Cont.)

Solution 3

- Convert .csv file to *parquet* using *arrow*
- Only read in data you need

Parquet is created to make the advantages of compressed, efficient columnar data representation available to any project in the Hadoop ecosystem.

Apache Arrow is a software development platform for building high performance applications that process and transport large data sets.

```
## this doesn't yet read the data in, it only creates a connection
csv_ds <- open_dataset("data/original_csv",
                       format = "csv",
                       partitioning = c("year", "month"))

## this reads each csv file in the csv_ds dataset and converts it to a .parquet file
write_dataset(csv_ds,
              "data/converted_parquet",
              format = "parquet",
              partitioning = c("year", "month"))
```

Large Data with Appropriate Data Format (Cont.)

csv format

```
Size on the disk: 9.06 GB
```

Reading speed

```
##    user  system elapsed
##  16.952   1.030  20.000
```

parquet format

```
Size on the disk: 1.44 GB
(84% less disk space)
```

```
##    user  system elapsed
##   3.322   1.640   1.315
```

(15 times faster)

Large Data with Appropriate Data Format (Cont.)

```
fhvhv_ds <- open_dataset("data/converted_parquet",
                         schema = schema(hvfhs_license_num=string(),
                                         dispatching_base_num=string(),
                                         pickup_datetime=string(),
                                         dropoff_datetime=string(),
                                         PULocationID=int64(),
                                         DOLocationID=int64(),
                                         SR_Flag=int64(),
                                         year=int32(),
                                         month=int32()))
```

```
library(dplyr, warn.conflicts = FALSE)

fhvhv_ds %>%
  filter(hvfhs_license_num == "HV0005") %>%
  select(hvfhs_license_num) %>%
  collect() %>%
  summarize(total_Lyft_trips = n())
```

```
## # A tibble: 1 × 1
##   total_Lyft_trips
##              <int>
## 1         37250101
```

Arrow will collect rows with hvfhs_license_num == “HV0005”. But still, we need to load the entire collection into memory and count them.

Large Data with Appropriate Data Format (Cont.)

Solution 4

- Convert .csv file to *parquet* using *arrow*
- Summarize data in streaming fashion using duckdb

DuckDB is designed to support analytical query workloads, also known as Online analytical processing (OLAP).

```
library(dplyr)

con <- DBI::dbConnect(duckdb::duckdb())
fhvhv_tbl <- to_duckdb(fhvhv_ds, con, "fhvhv")
```

```
## number of Lyft trips, tidyverse style
fhvhv_tbl %>%
  filter(hvfhs_license_num == "HV0005") %>%
  select(hvfhs_license_num) %>%
  count()
```

```
## # Source:   lazy query [?? x 1]
## # Database: duckdb_connection
##           n
##       <dbl>
## 1 37250101
```

Parallel Computing

(Example of Monte Carlo Approach in Computing PI)

An Embarrassingly Parallel Problem

```r
mc_pi <- function(sample_size){

  set.seed(as.integer(proc.time()[[3]]*1000))
  x <- runif(sample_size)
  y <- runif(sample_size)
  z <- sqrt(x^2+y^2)
  pi <- (length(which(z<=1))*4)/length(z)

  return(pi)
}
```

```r
PI <- 3.14159265358979323846
```

$$A_{circle} = (1/4) \times \pi \times r^2$$

$$r = 1$$

$$\pi = 4 \times A_{circle}$$

$$A_{circle} = \frac{\#Random\ samples\ in\ the\ circle}{\#All\ random\ samples}$$

$$pi = 4 \times (25/32) = 3.125$$

Computational Resources

Sequential	Parallel Shared Memory Laptop / PC	Parallel Shared Memory Server node	Parallel Distributed Memory Cluster
	`library(parallel)`	`library(parallel)`	`library(Rmpi)`
`lapply`	`parallel::parLapply`	`parallel::parLapply`	`Rmpi::mpi.parLapply`

```
list_of_results <- lapply(list_of_inputs, function)
length(list_of_results) = length(list_of_inputs)
```

Sequential Approach

```
source("0_helper_functions.R")
n <- 1000  # number of samples in each trial
m <- 1000000 # number of trials.

trial_vec <- (numeric(m)+1)*n

t1 <- proc.time()
pi_list_tmp <- lapply(trial_vec, mc_pi)
t2 <- proc.time()

print(paste("Processing time: ",t2[[3]] - t1[[3]], " s."))

pi_list <- c(do.call(rbind, pi_list_tmp))
pi <- mean(pi_list)

options(digits=20)
print(paste("PI value: ", PI))
print(paste("Est.  pi: ", pi))
print(paste("Number of matched chars: ", match_chars(pi, PI)))
```

Unlist the results

Parallel Approach (Shared Memory)

Load library

Create a cluster

Stop the cluster

```r
library(parallel)

n <- 1000  # number of samples in each trial
m <- 100000 # number of trials.

trial_vec <- (numeric(m)+1)*n

# create cluster of workers
nthread <- 12
cl <- parallel::makeCluster(nthread, type="PSOCK")

t1 <- proc.time()
pi_list_tmp <- parallel::parLapply(cl, trial_vec, mc_pi)
t2 <- proc.time()

parallel::stopCluster(cl)
```

The first input is the cluster

Use parLapply instead of lapply

Parallel Approach (Shared Memory) – On the Server Node

https://vdi.rc.fas.harvard.edu/pun/sys/dashboard/batch_connect/sessions

Parallel Approach (Shared Memory) – On the Server Node

Click on FAS-RC Remote Desktop

FAS RC FAS Research Computing Harvard University Faculty of Arts & Sciences

Clusters Files Jobs Interactive Apps

Home / My Interactive Sessions / FAS-RC Remote Desktop

Interactive Apps

Desktops

FAS-RC Remote Visualization

FAS-RC Remote Desktop

Containerized FAS-RC Remote Desktop

FAS CGA

OmniSci

Postgresql db

FAS Informatics

Jupyter Lab (scipy-notebook)

RStudio Server (Bioconductor + tidyverse)

FAS-RC Remote Desktop version: 82b4294

This form will launch an interactive desktop session on a compute node in the remote_desktop partition.

You can select the amount of RAM (in GB), number of cores, and Timelimit (in hrs).

Upon request you can receive email notification when the job starts. You must specify your email address.

You can optionally select to display on the desktop some of your preferred GUI applications.

Resolution

width 1024 px height 768 px

Reset Resolution

Memory Allocation in GB

184

Number of cores

48

Parallel Approach (Shared Memory) – On the Server Node

Input requested memory

Input requested number of cores

FAS CGA

OmniSci

Postgresql db

FAS Informatics

Jupyter Lab (scipy-notebook)

RStudio Server (Bioconductor + tidyverse)

GUIs

Desktop Environment for Totalview

Matlab

Resolution

width 1024 px height 768 px

Reset Resolution

Memory Allocation in GB

184

Number of cores

48

Number of Cpus to allocate

Allocated Time (expressed in MM , or HH:MM:SS , or DD-HH:MM). Maximum for this partition is 28 days

00-04:00:00

ive an email when the session starts

Only relevant if you have multiple slurm accounts. It can be left blank otherwise

Launch

* The FAS-RC Remote Desktop session data for this session can be accessed under the data root directory.

Input requested allocated time

Launch the request!

Parallel Approach (Shared Memory) – On the Server Node

Session was successfully created. ×

Home / My Interactive Sessions

Interactive Apps

Desktops

- FAS-RC Remote Visualization
- FAS-RC Remote Desktop
- Containerized FAS-RC Remote Desktop

FAS CGA

FAS-RC Remote Desktop (3012001) 1 node | 48 cores | Starting

Delete

Created at: 2022-04-05 14:14:19 EDT

Time Remaining: 3 hours and 59 minutes

Session ID: 069c7132-a2e0-44bc-a0da-fe0b7bbf6e86

Your session is currently starting... Please be patient as this process can take a few minutes.

Parallel Approach (Shared Memory) – On the Server Node

Home / My Interactive Sessions

Interactive Apps

Desktops

- FAS-RC Remote Visualization
- FAS-RC Remote Desktop
- Containerized FAS-RC Remote Desktop

FAS CGA

- OmniSci
- Postgresql db

FAS Informatics

- Jupyter Lab (scipy-notebook)

FAS-RC Remote Desktop (3012001) 1 node | 48 cores | Running

Host: >_holy2a03306.rc.fas.harvard.edu

Delete

Created at: 2022-04-05 14:14:19 EDT

Time Remaining: 3 hours and 59 minutes

Session ID: 069c7132-a2e0-44bc-a0da-fe0b7bbf6e86

noVNC Connection | VNC Desktop Client

Compression

0 (low) to 9 (high)

Image Quality

0 (low) to 9 (high)

Launch FAS-RC Remote Desktop

View Only (Share-able Link)

Click!

Parallel Approach (Shared Memory) – On the Server Node

Open Terminal

Parallel Approach (Shared Memory) – On the Server Node

- ✓ After opening the terminal, you need to load a module of R.
- ✓ Different builds can be found in the following website.

https://portal.rc.fas.harvard.edu/p3/build-reports/

R

Select from the available application types

HeLmod CentOS 7 | x86_64 binary | Singularity 3 | Easy Build | Bioconda | Java | Open OnDemand

R

This module loads R_core (the base R package), and R_packages (a large set of the most popular packages from CRAN).

HeLmod CentOS 7

R/4.1.0-fasrc01

To activate this build:

```
module load R/4.1.0-fasrc01
```

Copy and Paste to Terminal

Parallel Approach (Shared Memory) – On the Server Node

- ✓ Transfer files and run the R code.
- ✓ You can monitor the cores activity by the ***htop*** command. Type *htop* in the terminal.

Parallel Approach (Distributed Memory)

✓ Going beyond one node requires batch submission.

```
ssh your_user_name@login.rc.fas.harvard.edu
```

parLapply_mpi.R

```
n <- 1000000  # number of samples in each trial
m <- 100000000 # number of trials.
val <- (numeric(m)+1)*n

time_a <- proc.time()
pi_list_tmp <- mpi.parLapply(val, mc_pi)
time_b <- proc.time()

print(paste("Processing time: ",time_b[[3]] - time_a[[3]], " s."))

pi_list <- c(do.call(rbind, pi_list_tmp))
pi_hat <- (mean(pi_list))

options(digits=20)
print(paste("PI value: ", PI))
print(paste("Est.  pi: ", pi_hat))
print(paste("Number of matched chars: ", match_digit(pi_hat, PI)))

# Tell all slaves to close down, and exit the program
mpi.close.Rslaves(dellog = FALSE)
mpi.quit()
```

Use mpi.parLapply

Close mpi communication.

Parallel Approach (Distributed Memory)

✓ Installing Rmpi package

https://github.com/fasrc/User_Codes/tree/master/Parallel_Computing/R

run.sh

```
#!/bin/bash
#SBATCH -J mpi
#SBATCH -o %j_job.out
#SBATCH -e %j_job.err
#SBATCH -p shared
#SBATCH -n 100
#SBATCH -t 50
#SBATCH --mem-per-cpu=4000

# Load required software modules
module load R/3.5.1-fasrc01
module load gcc/10.2.0-fasrc01 openmpi/4.1.1-fasrc01

# Set up Rmpi package
export R_LIBS_USER=$HOME/apps/R/3.5.1:$R_LIBS_USER
export R_PROFILE=$HOME/apps/R/3.5.1/Rmpi/Rprofile

# Run program
export OMPI_MCA_mpi_warn_on_fork=0
srun -n 100 --mpi=pmix R CMD BATCH --no-save --no-restore parLapply_mpi.R
```

```
sbatch run.sh
```

Parallel Approach (Distributed Memory)

#cores (ncore)	Wall Clock Time in s (wc)
4	30215.443
8	12509.134
16	5331.597
32	2613.654
64	1312.239
128	636.674
256	356.642
512	179.275

~ 8:23 Hours

~ 3 minutes

Based on the linear regression, WC of one core should be about 30 Hours.

How to get help?

FAS-RC Documentation

https://docs.rc.fas.harvard.edu/

FAS-RC Trainings

https://docs.rc.fas.harvard.edu/kb/training-links/

Office hours

https://www.rc.fas.harvard.edu/training/office-hours/

Ticket Submission

https://portal.rc.fas.harvard.edu/rcrt/submit_ticket

Large Data Processing in R

Naeem Khoshnevis

Research Software Engineer

Ph.D. Geophysics

M.Sc. Computer Science

HUIT Research Computing

Harvard University

April 7, 2023