Lab05 name: Yixuan Li; UPI:Yil845

05/09/2021

The purpose of this lab is to gain experience with basic parallel computing (in R).

The data source for this lab is a set of 31 CSV files containing 15-minute vehicle counts from January 2013 to September 2020.

Import

1, This code generates a list of file names including the file directory /course/NZTA, named “files”.

```
files<- list.files(path = "/course/NZTA", full.names = TRUE, pattern = "\\Hour.csv$")
files
```

```
##  [1] "/course/NZTA/20130101_20130331_TMSTrafficQuarterHour.csv"
##  [2] "/course/NZTA/20130401_20130630_TMSTrafficQuarterHour.csv"
##  [3] "/course/NZTA/20130701_20130930_TMSTrafficQuarterHour.csv"
##  [4] "/course/NZTA/20131001_20131231_TMSTrafficQuarterHour.csv"
##  [5] "/course/NZTA/20140101_20140331_TMSTrafficQuarterHour.csv"
##  [6] "/course/NZTA/20140401_20140630_TMSTrafficQuarterHour.csv"
##  [7] "/course/NZTA/20140701_20140930_TMSTrafficQuarterHour.csv"
##  [8] "/course/NZTA/20141001_20141231_TMSTrafficQuarterHour.csv"
##  [9] "/course/NZTA/20150101_20150331_TMSTrafficQuarterHour.csv"
## [10] "/course/NZTA/20150401_20150630_TMSTrafficQuarterHour.csv"
## [11] "/course/NZTA/20150701_20150930_TMSTrafficQuarterHour.csv"
## [12] "/course/NZTA/20151001_20151231_TMSTrafficQuarterHour.csv"
## [13] "/course/NZTA/20160101_20160331_TMSTrafficQuarterHour.csv"
## [14] "/course/NZTA/20160401_20160630_TMSTrafficQuarterHour.csv"
## [15] "/course/NZTA/20160701_20160930_TMSTrafficQuarterHour.csv"
## [16] "/course/NZTA/20161001_20161231_TMSTrafficQuarterHour.csv"
## [17] "/course/NZTA/20170101_20170331_TMSTrafficQuarterHour.csv"
## [18] "/course/NZTA/20170401_20170630_TMSTrafficQuarterHour.csv"
## [19] "/course/NZTA/20170701_20170930_TMSTrafficQuarterHour.csv"
## [20] "/course/NZTA/20171001_20171231_TMSTrafficQuarterHour.csv"
## [21] "/course/NZTA/20180101_20180331_TMSTrafficQuarterHour.csv"
## [22] "/course/NZTA/20180401_20180630_TMSTrafficQuarterHour.csv"
## [23] "/course/NZTA/20180701_20180930_TMSTrafficQuarterHour.csv"
## [24] "/course/NZTA/20181001_20181231_TMSTrafficQuarterHour.csv"
## [25] "/course/NZTA/20190101_20190331_TMSTrafficQuarterHour.csv"
## [26] "/course/NZTA/20190401_20190630_TMSTrafficQuarterHour.csv"
## [27] "/course/NZTA/20190701_20190930_TMSTrafficQuarterHour.csv"
## [28] "/course/NZTA/20191001_20191231_TMSTrafficQuarterHour.csv"
## [29] "/course/NZTA/20200101_20200331_TMSTrafficQuarterHour.csv"
## [30] "/course/NZTA/20200401_20200630_TMSTrafficQuarterHour.csv"
## [31] "/course/NZTA/20200701_20200930_TMSTrafficQuarterHour.csv"
```

2, The function readFile() provided below uses the ‘data.table’ package to read a file into R and collapse from 15-minute counts to daily counts. It produces a ‘data.table’.

```
library(data.table)
readFile <- function(filename) {
    countsDT <- fread(filename)
    countsDT[, day := substr(startDatetime, 1, 11)][, .(count = sum(count)), .(day, siteRef, cla
ss)]
}
```

this code uses readFile() function to read five of the CSV files into R and combines them into a single data table called trafficSeries. The total timespan is ~33 seconds.

```
system.time(trafficSeries<- do.call(rbind,lapply(1:5,function(i) readFile(files[[i]]))))
```

```
##    user  system elapsed 
## 159.677   2.735  32.492
```

```
dim(trafficSeries)
```

```
## [1] 365635      4
```

```
head(trafficSeries)
```

```
##            day  siteRef class   count
## 1: 21-FEB-2013 01S00376     H  1926.0
## 2: 23-FEB-2013 01S00376     H   901.5
## 3: 24-FEB-2013 01S00376     L 12360.5
## 4: 25-FEB-2013 01S00376     L  9858.5
## 5: 25-FEB-2013 01S00376     H  1729.5
## 6: 26-FEB-2013 01S00376     L  9399.0
```

3, Repeat the previous task, but using mclapply to create “forked” CPUs (workers) to read the files in parallel. The total time spent is ~ 7 seconds, about 4 times faster than one CPU processing time.

```
library(parallel)
system.time(trafficSeries1<- do.call(rbind,mclapply(1:5,function(i) readFile(files[[i]]), mc.cor
es = 5)))
```

```
##    user  system elapsed 
##  21.084   1.796   7.267
```

4, this code checks that the results from the previous two tasks are the same.

```
identical(trafficSeries,trafficSeries1)
```

```
## [1] TRUE
```

5, Repeat the reading-five-files task, but this time using a “cluster” of workers to read the files in parallel. Here I used makeCluster function to create 5 clustered workers which does not share the same RAM as the master. I used clusterExport function to export

function readFile and data files to clustered workers, as well as clusterEvalQ to allocate data.table library to clusters.ParLapply is further used to assign tasks to the clusters. In the end, stopCluster is used to release the clustered CPU from duty.The total time used is ~20 seconds. Comparing to Q3, this approach takes more time because of the clustering and environmental settings.

```
system.time({
cl<- makeCluster(5)
clusterEvalQ(cl, c(library(data.table)))
clusterExport(cl, c("readFile", "files"),envir=environment())
trafficSeries2<- do.call(rbind,parLapply(cl, 1:5, function(i) readFile(files[i])))
stopCluster(cl)
})
```

```
##    user  system elapsed
##   0.156   0.024  18.122
```

```
## Check if results obtained are the same.
identical(trafficSeries,trafficSeries2)
```

```
## [1] TRUE
```

```
identical(trafficSeries1,trafficSeries2)
```

```
## [1] TRUE
```

6, The underlying codes read all 31 files into a single R data table using mclapply approach (the fastest method as determined in the previous tasks) and create a single data table called traffic. I used gc() to check memory usage before and after the code was running, we found the peak memory usage is ~87Mb, the total memory usage is ~300Mb (gc() last column - gc(reset=T) last column).

```
gc(reset = TRUE)
```

```
##            used  (Mb) gc trigger  (Mb) max used  (Mb)
## Ncells   579239  31.0     993654  53.1   579239  31.0
## Vcells 34768367 265.3  110563644 843.6 34768367 265.3
```

```
traffic<- do.call(rbind,mclapply(1:31,function(i) readFile(files[[i]]), mc.cores = 5))
## Total memory used so far
gc()
```

```
##            used  (Mb) gc trigger  (Mb) max used  (Mb)
## Ncells   581806  31.1     993654  53.1   599003  32.0
## Vcells 45621628 348.1  110563644 843.6 74117071 565.5
```

```
## peak memory usage
object.size(traffic)
```

```
## 87060696 bytes
```

the following code creates a new scount column and to remove missing values from that column. A few lines of object “traffic” are shown.

```
traffic <- traffic[, .(day, siteRef, class, scount = sqrt(count))]
traffic <- traffic[!is.na(scount), ]
head(traffic)
```

```
##            day  siteRef class    scount
## 1: 21-FEB-2013 01S00376     H  43.88622
## 2: 23-FEB-2013 01S00376     H  30.02499
## 3: 24-FEB-2013 01S00376     L 111.17779
## 4: 25-FEB-2013 01S00376     L  99.28998
## 5: 25-FEB-2013 01S00376     H  41.58726
## 6: 26-FEB-2013 01S00376     L  96.94844
```

```
dim(traffic)
```

```
## [1] 2710949       4
```

Model

7, According to lab03 report, the estimated memory usage is 15MbX31 = 465MB for scount vs. class fitting (15Mb per file); as well as 1.4X31 = 44Gb memory for scount vs. class + siteRef fitting (1.4Gb per file). The following code proves that the actual memory usage is ~403Mb for scount vs. class fitting, and ~38Gb for scount vs. class+siteRef fitting.

```
library(profmem)
mem.cl<-profmem(model.matrix(scount ~ class, traffic))
total(mem.cl)
```

```
## [1] 403780792
```

```
mem.ref<-profmem(model.matrix(scount ~ class + siteRef, traffic))
total(mem.ref)
```

```
## [1] 37561345480
```

```
rm(mem.ref)
rm(mem.cl)
```

8,The following code generates index values (to split the data into ten equal-size chunks) and defines a fitLM() function that splits the data into test and training sets, fits the two models, scount ~ class and scount ~ class + siteRef (using glm4() from 'MatrixModels', to take advantage of sparse model matrices), and returns the RMSE for both models. The fitLM() function takes a single integer argument that specifies which chunk of the data to use as the test set.

Total time elapsed is ~80sec.

```
library(MatrixModels)
index <- sample(rep(1:10, length.out=nrow(traffic)))
sites <- levels(factor(traffic$siteRef))
RMSE <- function(obs, pred) {
  sqrt(apply((obs - pred)^2,2,mean, na.rm=TRUE))
}
fitLM <- function(i) {
    train <- traffic[index != i, ]
    train[, siteRef := factor(siteRef, levels=sites)]
    test <- traffic[index == i, ]
    test[, siteRef := factor(siteRef, levels=sites)]
    fit1 <- glm4(scount ~ class, data=train, sparse=TRUE)
    coef1 <- coef(fit1)
    rm(fit1)
    fit2 <- glm4(scount ~ class + siteRef, data=train, sparse=TRUE)
    coef2 <- coef(fit2)
    rm(fit2)
    pred1 <- model.Matrix(scount ~ class,
                          data=test, sparse=TRUE) %*%
                 coef1
    pred2 <- model.Matrix(scount ~ class + siteRef,
                          data=test, sparse=TRUE) %*%
                 coef2
    obs <- test$scount
    c(RMSE(obs, pred1), RMSE(obs, pred2))
}

system.time(TotalRMSE<-lapply(1:10, function(i) fitLM(i)))
```

```
##    user  system elapsed
##  73.561   5.251  74.904
```

```
## RMSE for both models (first model - scount vs. class, second model - scount vs. class+siteRe
f)
TotalRMSE
```

```
## [[1]]
## [1] 43.93421 22.21179
##
## [[2]]
## [1] 43.91217 22.23085
##
## [[3]]
## [1] 43.85947 22.13215
##
## [[4]]
## [1] 44.02656 22.14216
##
## [[5]]
## [1] 44.03096 22.17145
##
## [[6]]
## [1] 44.02651 22.20799
##
## [[7]]
## [1] 43.97938 22.14569
##
## [[8]]
## [1] 44.01208 22.22876
##
## [[9]]
## [1] 44.10779 22.19854
##
## [[10]]
## [1] 44.08887 22.23408
```

```
colMeans(do.call(rbind,TotalRMSE))
```

```
## [1] 43.99780 22.19035
```

9, Repeat the previous task, I used mclapply to do linear fitting on the same task. The total time (~20sec) for 5 CPUs working simutaneously is ~ 4 times faster than a single CPU timespan. Reseted gc() is used as reference and to check peak and total memory usage as shown underlysing. Here we can see total memory usage and peak memory usage.

```
gc(reset = TRUE)
```

```
##            used  (Mb) gc trigger    (Mb) max used  (Mb)
## Ncells  1856624  99.2    8158422   435.8  1856624  99.2
## Vcells 50161460 382.8 1839421116 14033.7 50161460 382.8
```

```
system.time(mclapply(1:10, function(i) fitLM(i), mc.cores = 5))
```

```
##    user  system elapsed
##  55.656  16.487  20.083
```

```
## total memory usage
gc()
```

```
##            used  (Mb) gc trigger   (Mb) max used  (Mb)
## Ncells  1856581  99.2    5221391  278.9  1866754  99.7
## Vcells 50136501 382.6 1177229515 8981.6 50171866 382.8
```

```
## peak memory overall
object.size(mclapply(1:10, function(i) fitLM(i), mc.cores = 5))
```

```
## 816 bytes
```

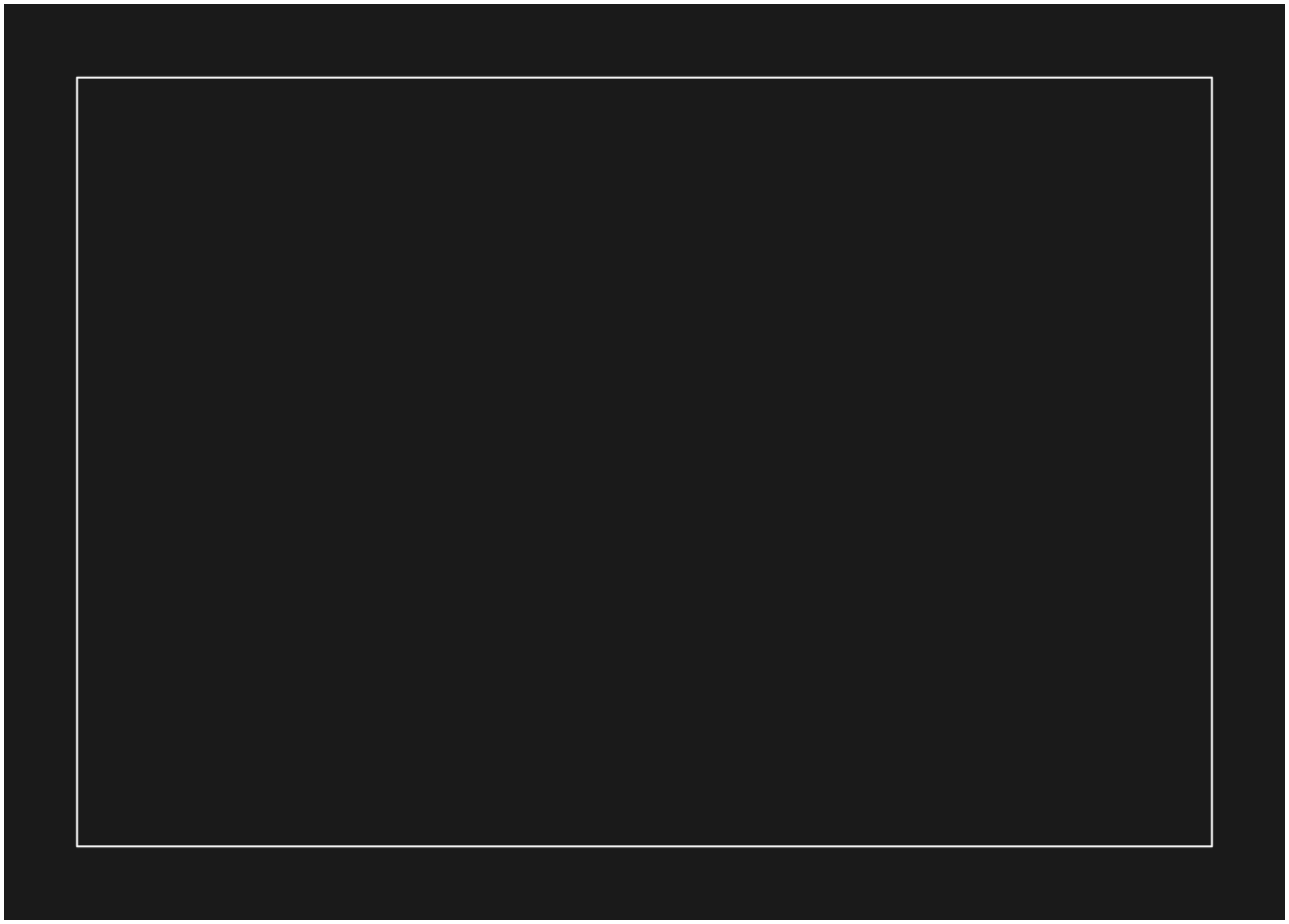
Visualize

10, The following code produces a plot of traffic tshowing the range of scount values at each site for both H (yellow) and L (green) vehicles along with the value predicted by the second model (red dots)

```
alllmFit<- glm4(scount ~ class + siteRef, data=traffic, sparse=TRUE)

traffic$pred <- fitted.values(alllmFit)
traffic$site <- reorder(as.numeric(factor(traffic$siteRef)), traffic$scount)
trafficClass <- split(traffic, traffic$class)

plotClass <- function(x, col) {
   sites <- split(x, x$site, drop=TRUE)
   siteMin <- sapply(sites, function(y) min(y$scount))
   siteMax <- sapply(sites, function(y) max(y$scount))
   pred <- sapply(sites, function(y) y$pred[1])
   s <- as.numeric(factor(sapply(sites, function(y) y$siteRef[1]),
                          levels=levels(x$site)))
   segments(s, siteMin, s, siteMax, col=col)
   points(s, pred, pch=16, cex=.2, col="red")
}
bg <- "grey10"
par(bg=bg, mar=rep(2, 4))
plot(traffic$site, traffic$scount, border=NA, ann=FALSE, axes=FALSE,
     ylim=range(traffic$scount, traffic$pred))
usr <- par("usr")
rect(usr[1], usr[3], usr[2], usr[4], col=bg)
box(col="white")
plotClass(trafficClass[[1]], adjustcolor("yellow", alpha=.5))
plotClass(trafficClass[[2]], adjustcolor("green", alpha=.5))
```



Summary:

In this lab we learned parallel computing using different approaches. Due to necessity of processing big data and limitation of CPU speed and capacity, we can use multiple CPUs to do the tasks. There are two main approaches we used in this lab, one is mclappy() which is to assign tasks to forked workers (CPUs share the same RAM as the master); as well as parLapply(), which is to allocate tasks to clustered workers (CPUs have their unique RAM and do not share with the master). The first approach is easy to use and CPUs can be released immediately after the command is executed. The second one requires mannual allocation of objects/functions to clustered CPUs and releasing CPU mannually with stopcluster() function. Although the second approach is relatively complicated to use, it has advantage in case of using remote CPUs or CPUs in clouds. We found out the timespan for parallel computing is much faster than a single CPU (~ 4 times faster in case of 5 forked workers). In addition, we used all 31 data files to do linear fitting based on 90% of training data for two models, scount vs.class and scount vs. class+siteRef. The results of both RMSE are produced. Further a plot of all data is generated in Q10, where yellow lines are data of Heavy vehicles in traffic data, green lines are light vehicles in traffic data, read spots shows predicted data based on linear model of scount vs. class+siteRef of all traffic data.