BIG DATA MANAGEMENT FOR BETTER ELECTRICITY CONSUMPTION PREDICTION

REPORT 1

PRACE SUMMER OF HPC

INDEX

1. Introduction.

The project that we can see in this document consists of the parallelization of a code provided by the course mentor (Janez Povh). The aforementioned code is made under the programming language called "R" and allows us to make a prediction using various statistical methods.

2. Developing.

In this section, we will see how the code has been parallelized using two methods: "Doparallel" and "Lapply". In the same way, we will see how we have modified both the sequential code and the parallel code to be able to measure the execution times of each section of interest of the code, as well as the total execution time.

2.1 Sequential time measurements.

For this section, the code provided by the mentor has been understood and has been modified following the steps that we can see below:

- We have added the tictoc library.
- Inside the main loop we have added the function “System.time” (it allows us to be able to measure the total execution time).
- The tictoc library has been used to measure each of the parts of interest (load files, create and save the model and generate the prediction).
- We save each calculated time in a separate Excel file.

Next, we will see the modified code section.

```
# Generate each model
system.time(
  for (varConsumerID in varConsumerIDs){
  varConsumerID_rel_ind=varConsumerID_rel_ind+1;

  tic()
  # Load file
  if(fromMemory){
    input_file=paste0(pwd,"Data/test_data_2021_06_30/hist_data_",varConsumerID,".json")
    load(file=input_file)
  } else {
    histData<-f_getHistoricData(var_custID = varConsumerID, date_from = first_day,date_to = last_day,var_location='Ljubljana')
    save(histData,file=paste0(pwd,'/Data/Data_Output/consumer_',varConsumerID,'_data.RData'))
  }
  listLoadFile <- c(listLoadFile,toc())

  tic()
  # Create and save model
  set.seed(1)
  pred_model_clust_RF <- f_create_model_clust_api_new(histData,varNOOfClusters = NofClusters,modelType = 'RF')
  pred_model_clust_NN <- f_create_model_clust_api_new(histData,varNOOfClusters = NofClusters,modelType = 'NN')
  pred_model_GLM <- f_create_model_GLM_api(histData)
  pred_model_GLM_small=pred_model_GLM[["small_model"]]

  save(pred_model_clust_RF, file = paste0(models_filepath,varConsumerID,"_model_RF_",NofClusters,"_centroids_",clustering_method,".RData"))
  save(pred_model_clust_NN, file = paste0(models_filepath,varConsumerID,"_model_NN_",NofClusters,"_centroids_",clustering_method,".RData"))
  save(pred_model_GLM_small, file = paste0(models_filepath,varConsumerID,"_model_GLM",".RData"))
  listCreateSaveModel <- c(listCreateSaveModel,toc())
```

```
  tic()
  # Generation Prediction
  for (prediction_date_ind in c(0:6)){
    pred_date=as.Date(last_forecast_day)+8+prediction_date_ind
    pred_file=paste0(pwd,"Data/test_data_2021_06_30/forecast_data_",varConsumerID_rel_ind,"_",pred_date,".json")
    load(pred_file)
    prediction_GLM <- f_prediction_with_GLM_api(forecastData,histData,pred_model_GLM_small,full_model = FALSE)
    prediction_clust_RF <- f_prediction_with_RF_api_new(forecastData,histData,pred_model_clust_RF)
    prediction_clust_NN <- f_prediction_with_NN_api_new(forecastData,histData,pred_model_clust_NN)
  }
  listGeneratePrediction <- c(listGeneratePrediction,toc())
  }
)

write.csv(x=unlist(listLoadFile), file="./TimeResults/Sequential/TimeLoadFile.csv")
write.csv(x=unlist(listCreateSaveModel), file="./TimeResults/Sequential/TimeCreateSaveModel.csv")
write.csv(x=unlist(listGeneratePrediction), file="./TimeResults/Sequential/TimeGenerationPrediction.csv")
```

2.2DoParrallel.

For this section, we have modified the code obtained from the previous section under the following steps:

- A function called "ExecuteCode" has been created that, through a parameter that we send to it, allows us to be able to carry out all the main code for a single consumer.

```
ecuteCode <- function(varConsumerID){

varConsumerID_rel_ind=varConsumerID_rel_ind+1;
tic()
if(fromMemory){
  input_file=paste0(pwd,"Data/test_data_2021_06_30/hist_data_",varConsumerID,".json")
  load(file=input_file)
} else {
  histData<-f_getHistoricData(var_custID = varConsumerID, date_from = first_day,date_to = last_day,var_location='Ljubljana')
  save(histData,file=paste0(pwd,'/Data/Data_Output/consumer_',varConsumerID,'_data.RData'))
}
listLoadFile <- c(listLoadFile,toc())

tic()
set.seed(1)
pred_model_clust_RF <- f_create_model_clust_api_new(histData,varNOOfClusters = NofClusters,modelType = 'RF')
pred_model_clust_NN <- f_create_model_clust_api_new(histData,varNOOfClusters = NofClusters,modelType = 'NN')
pred_model_GLM <- f_create_model_GLM_api(histData)
pred_model_GLM_small=pred_model_GLM[["small_model"]]

save(pred_model_clust_RF, file = paste0(models_filepath,varConsumerID,"_model_RF_",NofClusters,"_centroids_",clustering_method,".RData"))
save(pred_model_clust_NN, file = paste0(models_filepath,varConsumerID,"_model_NN_",NofClusters,"_centroids_",clustering_method,".RData"))
save(pred_model_GLM_small, file = paste0(models_filepath,varConsumerID,"_model_GLM",".RData"))
listCreateSaveModel <- c(listCreateSaveModel,toc())

tic()
# Generation Prediction
for (prediction_date_ind in c(0:6)){
  pred_date=as.Date(last_forecast_day)+8+prediction_date_ind
  pred_file=paste0(pwd,"Data/test_data_2021_06_30/forecast_data_",varConsumerID_rel_ind,"_",pred_date,".json")
  load(pred_file)
  prediction_GLM <- f_prediction_with_GLM_api(forecastData,histData,pred_model_GLM_small,full_model = FALSE)
  prediction_clust_RF <- f_prediction_with_RF_api_new(forecastData,histData,pred_model_clust_RF)
  prediction_clust_NN <- f_prediction_with_NN_api_new(forecastData,histData,pred_model_clust_NN)
}
listGeneratePrediction <- c(listGeneratePrediction,toc())

write.csv(x=unlist(listLoadFile), file=paste("./TimeResults/Parallel/", varConsumerID,"_TimeLoadFile.csv"))
write.csv(x=unlist(listCreateSaveModel), file=paste("./TimeResults/Parallel/", varConsumerID,"_TimeCreateSaveModel.csv"))
write.csv(x=unlist(listGeneratePrediction), file=paste("./TimeResults/Parallel/", varConsumerID,"_TimeGenerationPrediction.csv"))
```

- An external loop has been created that allows us to go through the list of consumers using a “Foreach” and the number of cores to be used has been registered.

```
cores <- 3
registerDoParallel(cores)

system.time(
  foreach (i=1:8) %do% {
    ExecuteCode(varConsumerIDs[i])
  }
)
```

2.3 Lapply.

For this section, the changes that we can see below have been made:

- A cluster with 3 cores has been created and added to the “parSapply” function together with the name of the function to be executed.

```
# Create a cluster
cl <- makeCluster(3)

# Export objects from the master to the workers
clusterExport (cl, varlist=c("object1", "object2"))

ptm <- proc.time()
  parSapply(cl, FUN=ExecuteCode())
  stopCluster(cl)
proc.time() - ptm
```

- The loop has been added inside the function to gain more simplicity in the code.

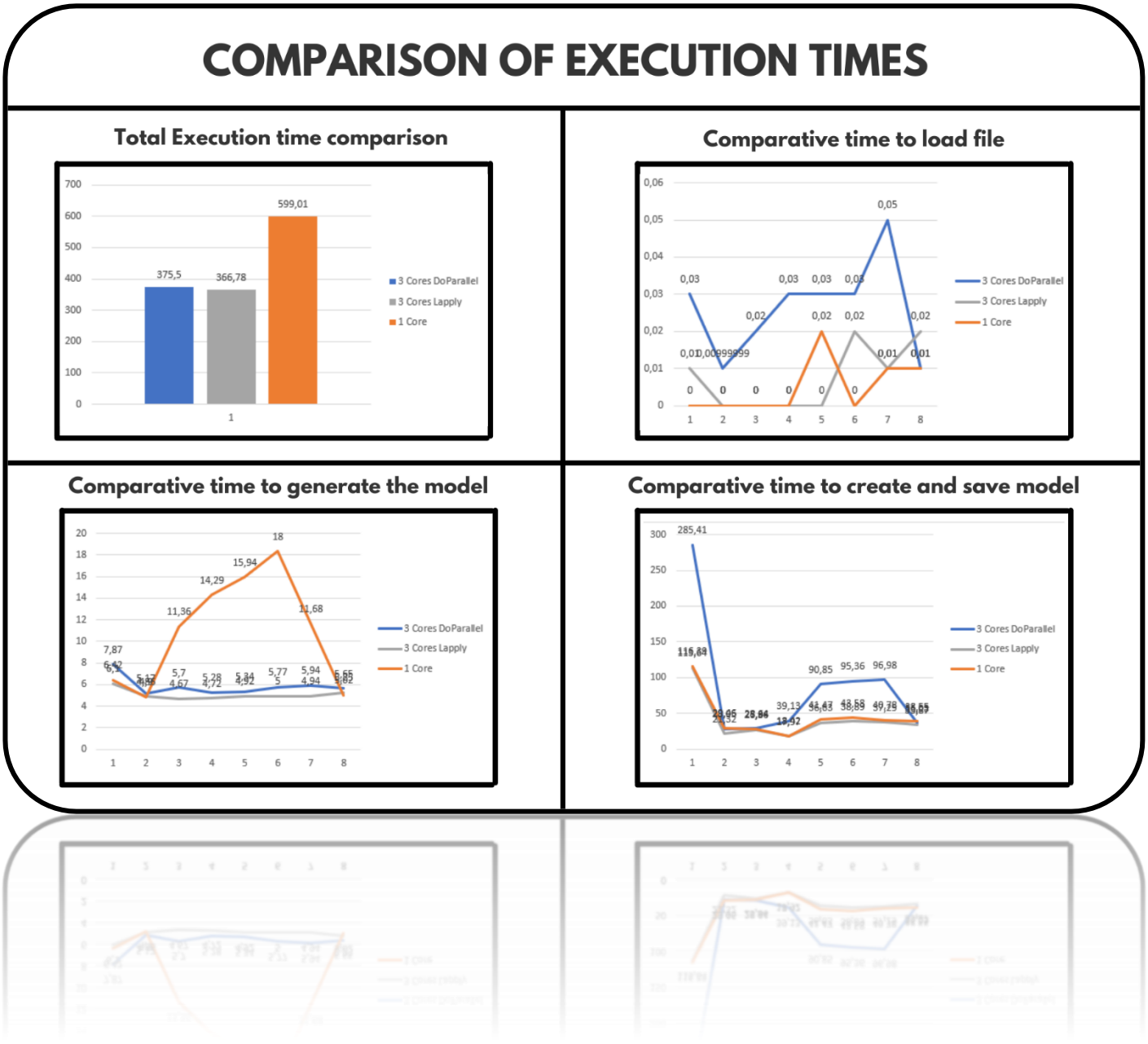
```
ExecuteCode <- function(varConsumerID){

  for (i in c(1:length(varConsumerIDs))){}

}
```

3. Comparative.

Next, we can see a comparison of the execution times obtained for each of the implementations made.



As we can see in the previous image, the highest execution time obtained is in the sequential version. Similarly, it should be noted that the times obtained in each parallel version are more or less similar, therefore, we should not opt for any at least for now.