BIG DATA MANAGEMENT FOR BETTER ELECTRICITY CONSUMPTION PREDICTION

REPORT 1

PRACE SUMMER OF HPC

INDICE

1. Introducción. 3
2. Desarrollo. 3
2.1 Medidas tiempo secuencial. 3
2.2 DoParrallel. 4
2.3 Lapply. 5
3. Comparativa. 6

1. Introducción.

El proyecto que podremos ver en este documento consiste en la paralelización de un código aportado por el mentor del curso (Janez Povh). El código mencionado, está realizado bajo el lenguaje de programación denominado “R” y nos permite poder sacar una predicción utilizando diversos métodos estadísticos.

2. Desarrollo.

En este apartado, veremos como se ha paralelizado el código utilizando dos métodos: “Doparallel” y “Lapply”. De igual forma, veremos cómo hemos modificado tanto el código secuencial, como el código paralelo para poder medir los tiempos de ejecución de cada sección de interés del código, así como, el tiempo total de ejecución.

2.1 Medidas tiempo secuencial.

Para esta sección, se ha entendido el código aportado por el mentor y se ha modificado siguiendo los pasos que podremos ver a continuación:

- Hemos agregado la librería tictoc.
- Dentro del bucle principal hemos agregado la función “System.time” (nos permite poder medir el tiempo total de ejecución).
- Se ha utilizado la librería tictoc para poder medir cada una de las partes de interés (cargar ficheros, crear y guardar el modelo y generar la predicción).
- Guardamos cada tiempo calculado en un archivo de Excel por separado.

A continuación, veremos la sección del código modificada.

```
 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")
```

2.2 DoParrallel.

Para este apartado, hemos modificado el código obtenido del apartado anterior bajo los siguientes pasos:

- Se ha creado una función denominada “ExecuteCode” que a través de un parámetro que le mandamos, nos permite poder realizar todo el código principal para un único consumidor.

```
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"))
```

- Se ha creado un bucle externo que nos permite poder recorrer la lista de consumidores utilizando un “Foreach” y se ha registrado el número de cores a utilizar.

```
cores <- 3
registerDoParallel(cores)

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

2.3 Lapply.

Para este apartado, se han realizado los cambios que podremos ver a continuación:

- Se ha creado un clúster con 3 cores y se ha agregado a la función “parSapply” junto con el nombre de la función a ejecutar.

```
# 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
```

- Se ha agregado el bucle dentro de la función para ganar mayor simplicidad en el código.

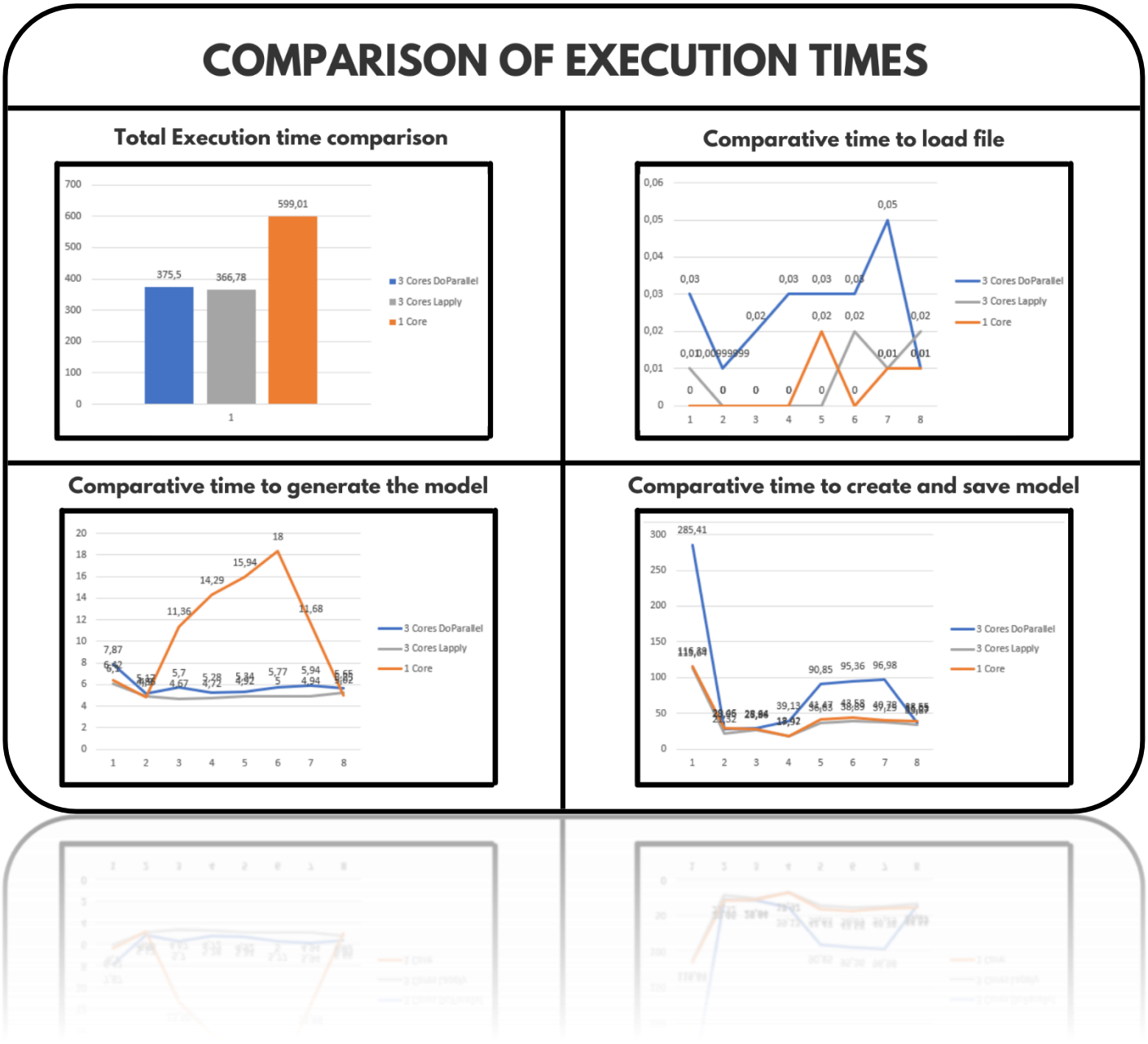
```
ExecuteCode <- function(varConsumerID){

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

}
```

3. Comparativa.

A continuación, podremos ver una comparativa de los tiempos de ejecución obtenidos para cada una de las implementaciones realizadas.



Como podremos ver en la imagen anterior, el mayor tiempo de ejecución obtenido es en la versión secuencial (como es lógico). De igual forma, cabe resaltar que los tiempos obtenidos en cada versión paralela son mas o menos similares, por tanto, no nos deberíamos de decantar por ninguno por lo menos de momento.