	Finolex Academy of Management and Technology, Ratnagiri			
	Department of Information Technology			
Subject name: Big Data Lab				Subject Code: ITC801
Class	BE IT	Semester – (CBSGS) VIII		Academic year: 2019-20
Name of Student	**Kazi Jawwad A Rahim**		**QUIZ Score : NA**	
Roll No	**28**	Assignment/Experiment No.		10
Title: **Case Study – GPU and its role in Big Data Analytics**				

1. Course objectives applicable: **COB1**.Get familiar with Big Data Analytics concepts and Hadoop framework

2. Course outcomes applicable:

CO1. Understand the key issues in big data management and its associated applications in intelligent business and scientific computing.

CO2 -Acquire fundamental enabling techniques and scalable algorithms like Hadoop, Map Reduce and NO SQL in big data analytics.

CO3 -Interpret business models and scientific computing paradigms, and apply software tools for big data analytics.

3. Learning Objectives:

1. To understand Learning for Big Data
2. To understand Deep Learning and its application
3. To understand how deep learning using GPU
4. To list and comment on deep learning services using Cloud based GPUs

4. Practical applications of the assignment/experiment: Analytics of Big Data by Deep Learning

5. Prerequisites:

1. Understanding of Machine Learning

6. Hardware Requirements:

1. PC with 4GB RAM, 500GB HDD,

7. Software Requirements:

1. Internet access

8. Quiz Questions (if any): NA

1.

9. Experiment/Assignment Evaluation:

Sr. No.	Parameters	Marks obtained	Out of
1	Technical Understanding (Assessment may be done based on Q & A **or** any other relevant method.) Teacher should mention the other method used – **Case Study Discussion**		6
2	Neatness/presentation		2
3	Punctuality		2

Date of performance (DOP)		**Total marks obtained**		**10**
Date of checking (DOC)		**Signature of teacher**		

Case Study on GPU and its role in Big Data Analytics:

Introduction:

GPU acceleration is nowadays becoming more and more important. The main two drivers for this shift are:

1. The world's amount of data is doubling every year.
2. Moore's law is now coming to an end because of limitations imposed by the quantum realm.

As a demonstration for this shift, an increasing number of online data science platforms is now adding GPU enabled solutions. Some examples are: Kaggle, Google Colaboratory, Microsoft Azure and Amazon Web Services (AWS).

RAPIDS

Many solutions have been proposed in the last few years in order to work with large amounts of data. Some examples are MapReduce, Hadoop and Spark. RAPIDS is now designed to be the next evolutionary step in data processing. Thanks to its Apache Arrow in-memory format, RAPIDS can lead to up to around 50x speed improvement compared to Spark in-memory processing (Figure 1). Additionally, it is also able to scale from one to multi-GPUs.

All RAPIDS libraries are based on Python and are designed to have Pandas and Sklearn like interfaces to facilitate adoption.

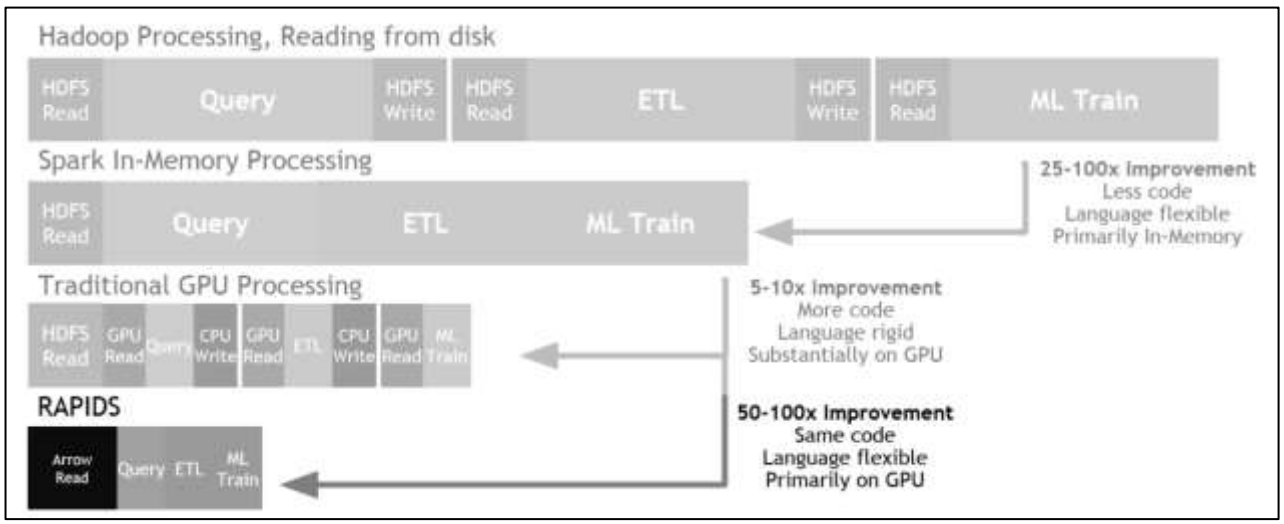

Figure 1: Data Processing Evaluation

All RAPIDS packages are now free to be used on Anaconda, Docker and cloud-based solutions such as Google Colaboratory. RAPIDS structure is based on different libraries in order to accelerate data science from end to end (Figure 2). Its main components are:

1. cuDF = used to perform data processing tasks (Pandas like).
2. cuML = used to create Machine Learning models (Sklearn like).
3. cuGraph = used to perform graphing tasks (Graph Theory).

RAPIDS has additionally integration with: PyTorch & Chainer for Deep Learning, Kepler GL for visualization, and Dask for distributed computation.

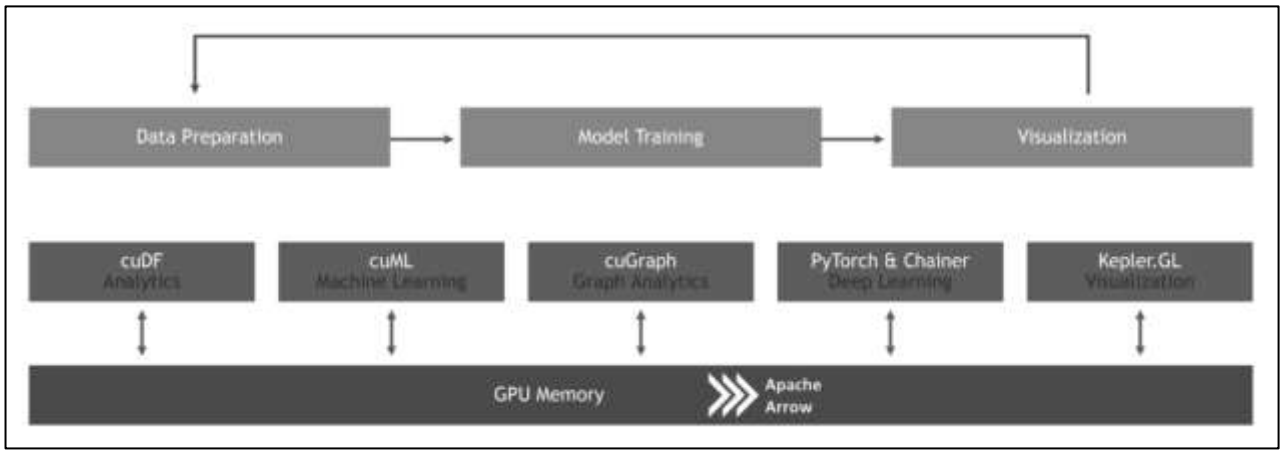

Figure 2: RAPIDS Architecture

Demonstration

I will now demonstrate to you how using RAPIDS can lead to a faster Data Analysis compared to using Pandas and Sklearn. All the code I will be using is available on Google Colaboratory, so feel free to test it out yourself! In order to use RAPIDS, we need first of all to enable our Google Colaboratory notebook to be used in GPU mode with a Tesla T4 GPU and then install the required dependencies (guidance is available on my Google Colabortory notebook).

Preprocessing

Once is everything set up, we can then import all the necessary libraries.

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from xgboost import XGBClassifier
import cudf
import xgboost as xgb
from sklearn.metrics import accuracy_score
```

In this example, I will show you how RAPIDS can speed up your Machine Learning workflow compared to just using Sklearn. In this case, I decided to use Pandas for preprocessing both the RAPIDS and Sklearn analysis. On my Google Colaboratory notebook is available also another example in which I use instead cuDF for preprocessing. Using cuDF instead of Pandas, can lead to faster preprocessing especially if working with a large amount of data.

For this example, I decided to fabricate a simple dataset using Gaussian Distributions consisting of three features and two labels (0/1).

```
dataset_len = 8000000
dlen = int(dataset_len/2)
X_11 = pd.Series(np.random.normal(2,2,dlen))
X_12 = pd.Series(np.random.normal(9,2,dlen))
X_1 = pd.concat([X_11, X_12]).reset_index(drop=True)
X_21 = pd.Series(np.random.normal(1,3,dlen))
X_22 = pd.Series(np.random.normal(7,3,dlen))
X_2 = pd.concat([X_21, X_22]).reset_index(drop=True)
X_31 = pd.Series(np.random.normal(3,1,dlen))
X_32 = pd.Series(np.random.normal(3,4,dlen))
X_3 = pd.concat([X_31, X_32]).reset_index(drop=True)
```

```
Y = pd.Series(np.repeat([0,1],dlen))
df = pd.concat([X_1, X_2, X_3, Y], axis=1)
df.columns = ['X1', 'X2', 'X3', 'Y']
df.head()
```

The values of the means and standard deviations of the distributions have been chosen so that to make this classification problem fairly easy (linearly separable data).

	X1	X2	X3	Y
0	3.061037	-1.572610	1.795625	0
1	-2.079158	3.987116	3.999877	0
2	0.183407	2.534023	2.596007	0
3	1.780752	-3.260380	2.010033	0
4	1.738812	-5.279497	2.857876	0

Figure 3: Sample Data Set

Once created the dataset, I divided it features and labels and then defined a function to preprocess it.

```
X = df.drop(['Y'], axis = 1).values
y = df['Y']
def preproces(df, X, y, train_size = 0.80):
  # label_encoder object knows how to understand word labels.
  label_encoder = preprocessing.LabelEncoder()
  # Encode labels
  y = label_encoder.fit_transform(y)
  # identify shape and indices
  num_rows, num_columns = df.shape
  delim_index = int(num_rows * train_size)
  # Splitting the dataset in training and test sets
  X_train, y_train = X[:delim_index, :], y[:delim_index]
  X_test, y_test = X[delim_index:, :], y[delim_index:]
  # Checking sets dimensions
  print('X_train dimensions: ', X_train.shape, 'y_train: ', y_train.shape)
  print('X_test dimensions:', X_test.shape, 'y_validation: ', y_test.shape)
  # Checking dimensions in percentages
  total = X_train.shape[0] + X_test.shape[0]
```

```
  print('X_train Percentage:', (X_train.shape[0]/total)*100, '%')
  print('X_test Percentage:', (X_test.shape[0]/total)*100, '%')
  return X_train, y_train, X_test, y_test
X_train, y_train, X_test, y_test = preproces(df, X, y)
```

Now that we got our Training/Test sets we are finally ready to get started with our Machine Learning. In this example, I will be using XGBoost (Extreme Gradient Boosting) as classifier.

RAPIDS

In order to use XGBoost with RAPIDS we need first to convert our Training/Tests inputs in matrix form.

```
dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)
```

Successively, we can start training our model.

```
%%time
# Initial xgb parameters
params = {}
clf = xgb.train(params, dtrain)
```

The output of the above cell is shown below. Using the XGBoost library provided by RAPIDS took just under two minutes to train our model.

```
CPU times: user 1min 54s, sys: 307 ms, total: 1min 54s
Wall time: 1min 54s
```

Additionally, RAPIDS XGBoost library provides also a really handy function to rank and plot the importance of each feature in our dataset (Figure 4).

```
# Feature Importance plot!
xgb.plot_importance(clf)
```

This can be really useful in order to reduce the dimensionality of our data. By selecting just, the most important features and training our model on it, we would, in fact, reduce the risk to overfit our data and we would also speed up training times.

0

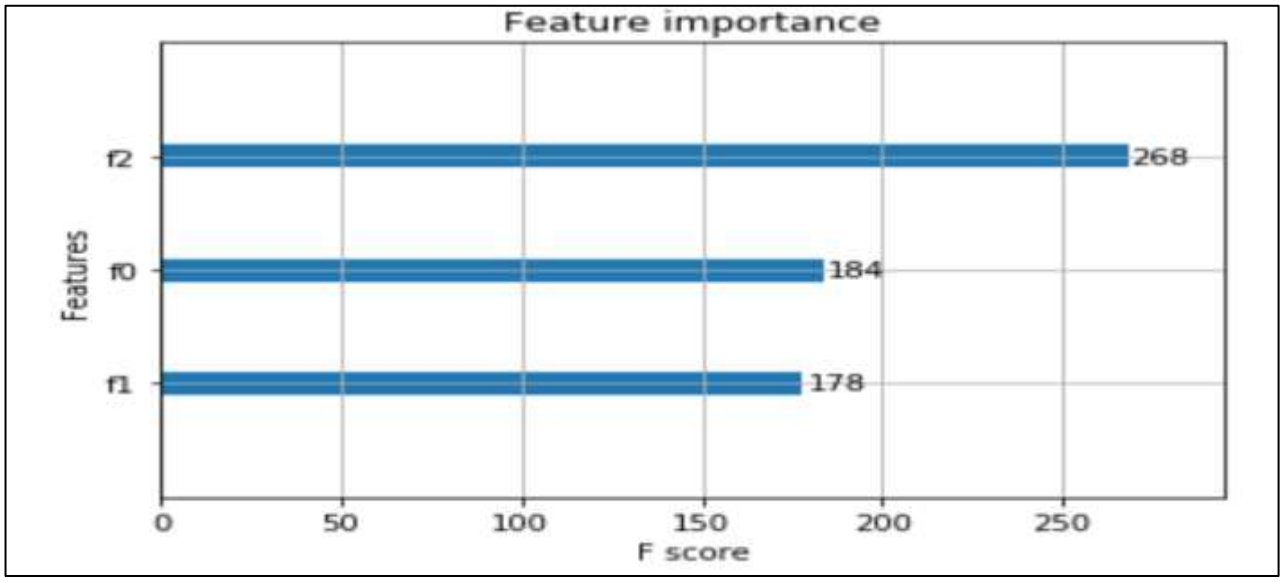

Figure 4: XGBoost Feature Importance

Finally, we can now calculate the accuracy of our classifier.

```
rapids_pred = clf.predict(dtest)
rapids_pred = np.round(rapids_pred)
rapids_acc = round(accuracy_score(y_test, rapids_pred), 2)
print("XGB accuracy using RAPIDS:", rapids_acc*100, '%')
```

The overall accuracy of our model using RAPIDS was equal to 98%.

XGB accuracy using RAPIDS: 98.0 %

Sklearn

I will now repeat the same analysis using plain Sklearn.

```
%%time
model = XGBClassifier()
model.fit(X_train, y_train)
```

In this occasion, it took just above 11 minutes to train our model. That means using Sklearn for this problem size was 5.8 times slower than using RAPIDS (662s/114s). By using cuDF instead of Pandas in the preprocessing stage we can reduce the execution time even more for the overall workflow of this example.

CPU times: user 11min 2s, sys: 594 ms, total: 11min 3s
Wall time: 11min 2s

Finally, calculated the overall accuracy of the model using Sklearn.

```
sk_pred = model.predict(X_test)
sk_pred = np.round(sk_pred)
sk_acc = round(accuracy_score(y_test, sk_pred), 2)
print("XGB accuracy using Sklearn:", sk_acc*100, '%')
```

Also, in this case, the overall accuracy was equal to 98%. That means that using RAPIDS can lead to faster results without compromising at all our model accuracy.

XGB accuracy using Sklearn: 98.0 %

Conclusion

As we can see from this example, using RAPIDS lead to a consistent decrease in execution time. This can be greatly important when working with large amounts of data since RAPIDS would be able to reduce execution time from days to hours and from hours to minutes.