Computadores de Altas Prestaciones

Practice 3

Daniel Varela Sánchez & Guillermo Martín-Coello Juárez

Part I: Spark

Exercise I

Read the material provided. Read the documentation of all functions used and classified them among the three aforementioned categories.

Creation if RDDs: parallelize

Transformations: map, flatmap,filter, distinct, sample, union.

Actions: count, reduce, sum, takeOrdered, collect, take

For each transformation, evaluate the size of the output RDD depending on the input RDD. Give particular code examples of the transformation achieving minimum/maximum sizes.

map: RDD size =*N*

Size will always be N. Example:

```
charsPerLine = quijote.map(lambda s: len(s))
```

flatMap:(RDD size =*K*≤*N*)

Minimum size will be 0. Example:

```
charsPerLine = quijote.flatmap(lambda s: [])
```

Maximum size could reach N. Example:

```
charsPerLine = quijote.flatmap(lambda s: len(s))
```

distinct: (RDD size =*K*≤*N*)

Minimum size will be 0. Example if not distinct words in allWords:

```
allWordsUnique = allWords.map(lambda s: s.lower()).distinct()
```

Maximum size could reach N. Example if all distinct words in allWords:

```
allWordsUnique = allWords.map(lambda s: s.lower()).distinct()
```

sample: (RDD size =*N***fraction*)

Max size will be N. Size will be determined by fraction. Example:

```
sampleWords = allWords.sample(withReplacement=True, fraction=0.2, seed=666)
```

union: (RDD size =*N*1+*N*2)

Size will always be N1 + N2. Example:

```
weirdSampling = sampleWords.union(allWordsNoArticles.sample(False, fraction=0.3))
```

From the previous classification, highlight the ones that only apply to K-V RDDs. Provide a code example for all of them.

We haven't used any Transformation specific for K-V RDDs, but we have used an Action specific for it: reduceByKey. Example:

```
frequencies = words.reduceByKey(lambda a,b: a+b)
```

Describe the different forms of persistence of RDDs. How do you save an RDD to disk? Which formats are supported?

There are two forms of persistence of RDDs:
Save cache (memory) with cache. Example:

```
allWords.cache()
```

Save to disk with persist:

```
allWords2.persist(StorageLevel.MEMORY_AND_DISK)
```

Or with saveAsTextFile:

```
allWords3.saveAsTextFile("palabras_parte2")
```

Exercise II

Is the current implementation of the word cloud using Spark to speed up the computations? Why?/Why not? Implement an improved version. Hint: WordCloud(...).generate_from_frequencies. Draw a wordcloud of your choice with data from Reddit.

No, the current implementation is not using spark to speed up the computations because it is not parallelizing. In order to speed up the computations the code would be the following:

```
allWords  =  text.flatMap(lambda  w:  re.sub(""";|:|\.|,|-|-|"|'|\s""","  ",  w.lower()).split("
")).filter(lambda a: len(a)>4)
freq = allWords.map(lambda a: (a, 1))
ffreq = freq.reduceByKey(lambda a,b: a+b)
```

First we generate a flatmap that contains all the words of the posts gathered with the reddit api (with more than 4 characters to avoid articles and avoiding punctuation signs). After that we make a key value map and finally we reduce by key and add the frequency values.
After all this when we call the Wordcloud function with generate_from _frequencies we pass the ffreq argument only now with the collectAsMap function:

```
wordcloud   =   WordCloud(max_words=10000,   mask=~mask[:,:,   0],   background_color="black",
mode="RGBA").generate_from_frequencies(ffreq.collectAsMap())
```

We obtain the following word cloud image:

Most popular words in Real Betis subreddit's posts

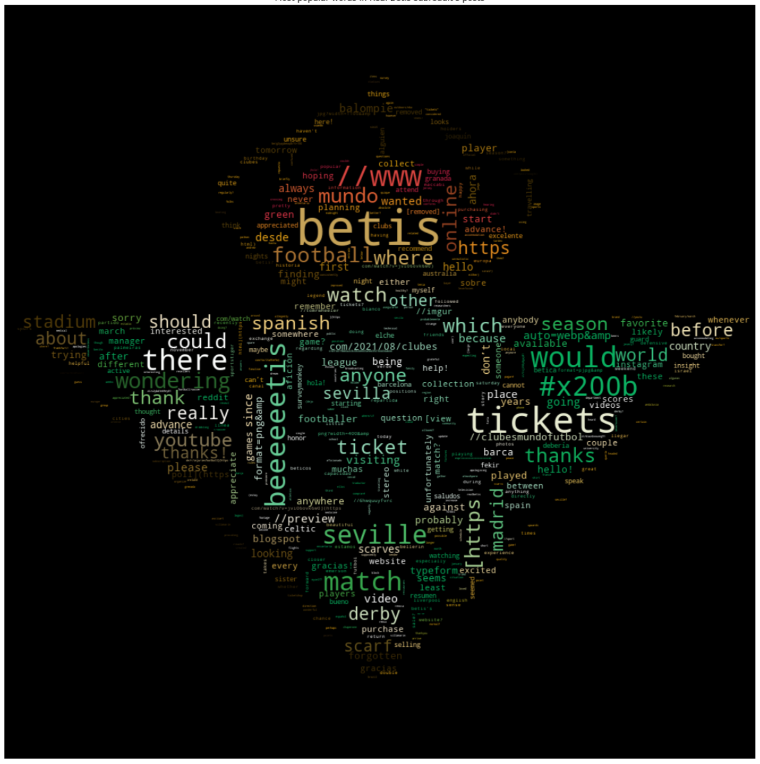

Image I: Cloud Image parallelized

Part II: Kubernetes

Exercise II

The first step in our task is defining what our Pods should run, this is, build images for the roles of our Spark cluster. For that purpose, three Dockerfiles are recommended.

Three Dockerfiles were created. The first one was the base dockerfile, used to create a docker container able to download and install all the necessary programs for the execution, (in this case spark and hadoop). The second dockerfile was the master dockerfile, used to create a container to run the master program, that will be talking to the workers program. The final dockerfile was the worker dockerfile, ready to create a container of a worker program.

Exercise III

Once the images are built, we are going to use them in two k8s Deployments, one for the master and another one for the workers. Bear in mind that master should be able to talk to any worker using port 7077. Also, in order to use local images, you may need to specify an image pull policy in the container spec section (i.e. imagePullPolicy: IfNotPresent).

Two deployments were created; master-deployment and worker-deployment. These were configured to assure the creation of master and worker pods. To let the master talk to the workers the workers pod was configured to be 7077. An image pull policy was set to IfNotPresent for both deployments so it will stop pulling an image if it already exists.

To expose the cluster to the host, a Service should also be deployed. Create a Service of the proper type (NodePort, LoadBalancer, ...) to expose the cluster to port 30077 of the host.

To expose the cluster to the host we decided to use NodePort. NodePort lets the cluster be accessible by the host using a specific port (the node port); this port was configured to be 30077 as demanded.

As a last part, a Python script that performs a basic test on Apache Spark should be provided. It is important to pay attention to the creation of the SparkSession.

A python script was created to perform a basic test on Apache Spark.

Exercise IV

1. Scale up and down the number of workers. Are the changes automatically detected by the Spark cluster? To check it, you may need to expose port 8080 of the master.

Changes are automatically detected by the Spark cluster, as it uses the selected number of workers automatically when assigned.

2. Delete the Apache Spark (without deleting minikube).

In order to delete the spark application without deleting minikube we need to use the kubectl delete command and delete the application's yaml file.

```
kubectl delete -f /<path-to-spark-application-yaml-name>
```

3. Deploy two separate Spark clusters on the same k8s infrastructure. They must be totally independent. What changes should be done to the YAML files?

A configuration file describes clusters, users, and contexts.
To add cluster details to a configuration file we use the following commands:

```
kubectl config --kubeconfig=config-demo set-cluster development --server=https://1.2.3.4
--certificate-authority=fake-ca-file
kubectl config --kubeconfig=config-demo set-cluster scratch --server=https://5.6.7.8
--insecure-skip-tls-verify
```

Where config-demo is our newly created config file. This yaml file will contain all the information of the different clusters. The configuration file will also need to have user details and context details, that can also be added via commands.
After the configuration file is done, different contexts will be used to access the different clusters.
To change between clusters, a kubectl command can be used:

```
kubectl config --kubeconfig=config-demo use-context context-name
```

Where the cluster corresponding to the context (in this case context-name) will be used.