NOM BOUKHELOUA

Prénom Rémi

Promo 2020

Date 16.04.

20150257: BOUKHELOUA Rémi
M1:
ST2FDP-DE (16/04/2019)
Amphi orange

MATIÈRE Functional Data Programming (1/2)

Q1:

a. Immutability protects ~~the~~ variables from side effect. This is when a variable is changed somewhere it shouldn't ~~or~~ and when it gets a value it shouldn't have. Immutability prevent from modifying its value so from side effect. ✓

b. A high order function is a function that takes another function as a parameter or/and return a function. It is possible in Scala as we use the paradigm "everything as a function". ✓ ✓

c. Map(f) ~~create~~ return a new list containing the image ~~of~~ by f of initial element. ✓

d. FlatMap is a fusion of map and flatten where flatten means transforming List of List into a simple List. For example, List(List("a","b"), List("c")).flatten(.) returns List("a","b","c"). ✓

Q2: There is no Q2 in subject.

Q3:

a. A RDD is a Resilient ~~Fast~~ Distributed Dataset. Resilient means fault tolerant, distributed mean divided into partitions and dataset means partitioned data. It supports two types of manipulation: actions (from RDD to not RDD) and transformations (from RDD to RDD). ✓
It can be created from various sources such as file system (HDFS for ex), distributed source (KAFKA, Cassandra), data from driver or another RDD.

b. A DataFrame is a distributed collection with columns structure (unlike RDD)
It's ~~imposed~~ by similar to MySQL tables and called table in SPARQSQL
It can be created from various sources such as file system, ~~of~~ RDDs, external DB and Hive tables (a little bit less than RDD sources). Finally it's at ~~first~~ a higher level than RDD and has great performances.

c. Given "lines : RDD[String]"

```
lines.map( P => P.split(" ") )      space
    .flatMap( w => (w, 1) )
    .reduceByKey( _ + _ )
```

It will divide the line into list of words it's composed of
Flat and create tupples (word, 1)
Sum tupples of same word

d. A Spark computation is ~~trigger~~ by any evaluation that requires a result.

Q4:

a. 1. A broker is in charge of receiving message from producers and persist them. It can handle several millions of message each second.

a. 2. Zookeeper is the centralisation tool. Intern in Kafka, it is in charge of ~~the~~ managing tasks performed in Kafka.

b. A ~~The~~ partition is where ~~pb~~ topics are stored depending on their key. It works as a append only and guarantee the order. Partitions are dispatched on a cluster. We could resume it as following:

2/6+

It is important because if we have a consumer group formed by 3 consumers, Kafka will provide a third of the partitions to it. for example: 3 consumers A, B, C and 6 partitions 1, 2, 3, 4, 5, 6.

A will get 1; 2, B will get 3; 4 and C will get 5 and 6.

It allows distributed work.

But from different groups will have every partitions!

c. As one broker is in charge of partitions, if it fails, partitions will be lost. That's why they introduced replications in order to store a partition on different brokers. (entirely) So if partition #1 is stored in broker A and replicated on B, and if A stops, the copy on B will become the partition 1

It's even more important that Kafka is a distributed system that runs over cheap materials (Big Data horizontal scaling) so those material are way more furquently crashing. So we need replication to make sure not to loose data!

d. The offset is an Integer that corresponds to the position of last element read by a given consumer group in a partition.

It is incremented after the consumer got a message and comitted.

Q6: a. def fact(nb: Int, acc: Int = 1): Int = nb match {

case 0 => return acc

case _ => return fact(nb - 1, acc * nb)

}

println(fact(5))

b. def boys(students: RDD[Student]): RDD[Student] = {

students.filter(_.isMale)

}

c. def boysAge(students: RDD[Student]): RDD[Int] = {

boys(students).map(s => age(s.birthday))

}

d. def youngest(students: RDD[Student]): Int = {

students.foldLeft(age(students(0).birthday))((acc, e) => {if(acc < e) acc

else e})

}

e. def hasHomonymous(aStudent: Student, students: RDD[Student]): Boolean = {

val c = students.filter(s => s.name == aStudent.name)

.count()

c >= 1

}

NOM Boukhelova

Prénom Rémi

Promo 2022

Date 16.04

MATIÈRE Functionnal Data Programming (2/2)

Q5: Regarding the customer page, I think we should record when a customer click on a model to have more information, when he add or delete an item from his shopping cart and when he confirm his payment. With following we would get infos about model viewed, hesitation / annulations and payment. These three message would be composed of:

Click msg:	Add/Delete msg:	Payment msg:
login	login	login
time	time	time
model	model	cart: [., .]
inCart (is it already in cart or not?)	action ("Add" or "delete")	totalPrice

As this is going to be used for statistical purpose, I think that we want to take it as a DF on SPARK in order to handle it easily in Spark Notebook (with SparkSQL). But also, we want consumer to store it into some kind of HDFS with xls files. For those two reasons, we want a consumer to handle only one format of message. That's why I create a Topic Customer shared over three partitions (at least one for each).

I want to create a Topic for Customer's Action where every message will fetch with a key depending on its format ("click", "modify", "pay"). So consumer will be able to get one partition.

Regarding the Administration page, we have to handle three different actions : add a new sheet, set stocks and change description
These actions would produce following messages :

Add msg :
- model
- size : [ ]
- stock [ ]
- price
- description
- time.

Set msg :
- model
- size : [ ]
- stock : [ ]
- time.

Description msg :
- description
- model
- time.

As for Customer interface, I think it would be interesting to have a topic reserved to Administration partitioned regarding event type. We won't need real time view but to show or modify DB it would be easier to have a different consumer in each partitions.

Those consumers would be linked to HDFS / real time view (for Customer) and to a Available and Partition Tolerant DB such as Cassandra or CouchDB to keep a track of stocks on the website. (for Administrator). We don't mind of consistency because we prefer the system to be available at any time and fault tolerant to keep any informations.

6/4

~~It is~~ The Architecture would look like: