Nom :
Prénom :

BDLE – Seconde Partie

Exemen réparti du 6 Novembre 2015
Durée : 2 Heures – CORRIGÉ
Documents autorisés

Le but des trois premières questions est d'exprimer des instructions Scala pour Spark. Pour vous guider, le résultat escompté et son type Scala est fourni.

Question 1 (3 points)

On considère qu'on a exécuté l'instruction suivante :

```
scala> val r = sc.parallelize(List("a1|b1","a2|b2|c2","a3|b3|c3|d3"))
r: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[33] ...
```

Compléter les instructions suivantes.

Réponse :

```
scala> val R = r. . . .

. . .

R : org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[4] at map at <console> :23
scala> R.collect
res1 : Array[Array[String]] = Array(Array(a1, b1), Array(a2, b2, c2), Array(a3, b3, c3, d3))
```

Réponse :

```
scala> val even = R. . . .

. . .

even : org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[6] at filter at <console> :25
scala> even.collect
res2 : Array[Array[String]] = Array(Array(a1, b1), Array(a3, b3, c3, d3))
```

Rappel scala> even.collect
res2 : Array[Array[String]] = Array(Array(a1, b1), Array(a3, b3, c3, d3))

Réponse :
scala> val even_pair = even. . . .

. . .

even_pair : org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[7] ...
scala> even_pair.collect
res4 : Array[(String, Int)] = Array((a1,2), (a3,4))

Remarque Utiliser impérativement la méthode *split*

Réponse :
scala> val even_pair_letter = even_pair. . . .

. . .

even_pair_letter : org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[11] ...
scala> even_pair_letter.collect
res7 : Array[(String, Int)] = Array((a,2), (a,4))

Remarque Utiliser impérativement la méthode *split*

Réponse :
scala> val even_pair_index = even_pair. . . .

. . .

even_pair_index : org.apache.spark.rdd.RDD[(Int, Int)] = MapPartitionsRDD[10] ...
scala> even_pair_index.collect
res8 : Array[(Int, Int)] = Array((1,2), (3,4))

Solution:

```
q1[1/2 pt] : r.map(x=>x.split("\\|"))
q2 [1pt] : filter(x=>x.length%2==0)
q3 [1/2pt] : map(x=>(x(0),x.length))
q4 [1/2pt] : map(x=>(x._1.split("")(0),x._2))
q5 [1/2 pt] : map(x=>(x._1.split("")(1).toInt,x._2))
```

Question 2 (3 points)

On considère qu'on a exécuté l'instruction suivante :

```
scala> val data = sc.parallelize(List((1, "31oct2015", 20) ,
(1, "30oct2015", 22), (2, "30oct2015", 24) ,
(3, "29oct2015", 19), (3, "29oct2015", 17),
(4, "30oct2015", 25), (4, "29oct2015", 23) ))
temp: org.apache.spark.rdd.RDD[(Int, String, Int)] = ...

scala> val code = sc.parallelize(List(("DC",1),("NY",2),("NH",3),
("MI",4)))
code: org.apache.spark.rdd.RDD[(String, Int)] = ...
```

Compléter **lisiblement** les instructions ci-dessous **en écrivant une instruction (Map, Reduce, ..) par ligne**.

Réponse :

scala> val avg_per_state = data. . .

. . .

. . .

. . .

avg_per_state : org.apache.spark.rdd.RDD[(Int, Int)] = MapPartitionsRDD[34] ...
scala> avg_per_state.collect
res23 : Array[(Int, Int)] = Array((1,21), (2,24), (3,18), (4,24))

Réponse :

scala> val res = . . .

. . .

. . .

res : org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[31] at map at <console> :30
scala> res.collect res21 : Array[(String, Int)] = Array((DC,21), (NY,24), (NH,18), (MI,24))

Solution:

```
[2pt] map{case(code, date, tval) => (code, (tval,1))}.
reduceByKey((v,w)=>(v._1+w._1, v._2+w._2)).
map{case(code, (sumt,nbt))=>(code,sumt/nbt)}

[1pt] code.map{case(state,cod)=>(cod,state)}.
join(avg_per_state).map{case(cod,(state,avgt))=>(state,avgt)}
```

Question 3 (4 points)

On considère les triplets de l'encadré ci-dessous. Ces triplets, de la forme *s,p,o*, décrivent pour des personnes (luke, liz, etc) l'université où ils ont suivi leurs études (ex. *luke,studiedAt,CMU*), la personne qui les a encadré (ex. *monica,supervisedBy,Yang*), éventuellement l'université de leur encadrant (*Horrow,studiedAt,MIT*) ainsi que l'état où se situe leur université (*CMU,locatedAt,PA*).

luke,hasDegree,eng	Yang,studiedAt,UCSD	MIT,locatedAt,MA
luke,studiedAt,CMU	rick,supervisedBy,Horrow	luke,livesIn,PA
liz,studiedAt,CMU	Horrow,studiedAt,MIT	liz,livesIn,PA
rick,studiedAt,MIT	CMU,locatedAt,PA	rick,livesIn,MA
suzan,studiedAt,UCSD	UCSD,locatedAt,CA	monica,livesIn,CA
monica,supervisedBy,Yang	UCSB,locatedAt,CA	

Afin de faciliter la réponse aux questions, on suppose que les instructions suivantes ont été exécutées.

```
scala> val triples = sc.textFile("...").map(x=>x.split(",")).map(x=>(x(0),x(1),x(2)))

scala> val studiedAt = triples.filter{case (s,p,o) => p.contains("studiedAt")}
scala> val supervisedBy = triples.filter{case (s,p,o) => p.contains("supervisedBy")}
scala> val locatedAt = triples.filter{case (s,p,o) => p.contains("locatedAt")}
scala> val livesIn = triples.filter{case (s,p,o) => p.contains("livesIn")}
```

Exprimer les requêtes suivantes.

(q1) Les personnes *p* qui étudient dans une université *u* qui se situe dans un endroit *l*, ces personnes doivent avoir des encadrants *s*. La requête doit retourner *p, u* et *l*. La requête équivalente en Sparql est comme suit

```
select ?p ?u ?l
where {?p studiedAt ?u. ?u locatedAt ?l. ?p supervisedBy ?s}
```

Le résultat de cette requête sur l'extrait *triples* retourne la paire (rick,MIT,MA).

Remarque : une instruction par ligne

Réponse :

```
scala> val q1 = ...

...

...

...

...

...

...


scala> q1.collect res4 : Array[(String, String)] = Array((rick,MIT,MA))
```

Solution:

```
 [2pt] ?p studiedAt ?u. ?p supervisedBy ?s. ?u locatedAt ?l

studiedAt.map{case(p,sat,u)=>(p,u)}.
join(supervisedBy.map{case(p,sby,s)=>(p,s)}).
map{case(p,(u,s))=>(u,p)}.
join(locatedAt.map{case(uu,lat,ll)=>(uu,ll)}).
map{case(uu, (p,ll))=>(p,uu,ll)}
```

(q2) Les personnes *p* qui étudient dans la meme université *u* que leur encadrants *s*. La requête retourne *p* et *s*. Voici la requête équivalente en Sparql :

```
select ?p ?s
where {?p studiedAt ?u. ?p supervisedBy ?s. ?s studiedAt ?u}
```

Le résultat de cette requête sur l'extrait *triples* retourne le triplet (rick,Horrow).

Remarque : une instruction par ligne

Réponse :

scala> val q2 = . . .

. . .

. . .

. . .

. . .

. . .

. . .

. . .

scala> q2.collect res5 : Array[(String, String, String)] = Array((rick,Horrow))

Solution:

```
%[3pt] ?p studiedAt ?u. ?p supervisedBy ?s. ?s studiedAt ?u
val q2 = studiedAt.
map{case(p,sat,u)=>(p,u)}.
join(supervisedBy.map{case(p,sby,s)=>(p,s)}).
map{case(p,(u,s))=>(s,(p,u))}.
join(studiedAt.map{case(ps,sat,us)=>(ps,us)}).
map{case(s,((p,u),us))=>(p,u,s,us)}.
filter{case(p,u,s,us)=>u==us}.map{case(p,u,s,us)=>(p,s)}
```

Question 4 (2 points)

Soit la relation $\mathrm{Table(A, B, C)}$ dont les nuplets sont notés par $<a_1, b_1, c_1>, <a_2, b_2, c_2>, ...$

Soit le modèle de réponse ci-dessous.

Requête SQL
select A, **sum**(B) **from Table** **group by** A
Pseudo-code MR
Entrée : les tuples de Table **Map** pour chaque tuple $<a, b, c>$, émettre la paire (a, b) *Fonction(s) de hachage* une seule fonction notée h et ayant pour domaine $Dom(A)$ *Shuffle* envoyer chaque paire (a, b) vers le reducer $h(a)$ **Reduce** pour chaque paire $(a, [b_1, ..b_n])$ retourner $(a, \Sigma_i b_i)$ Résultat : une séquence de tuples de la forme $< a, N >$ où N est un entier.

En considérant le schéma suivant R(A,B), S(B,C), T(C,D) Remplir le tableau suivant en utilisant la traduction des jointures MR vue en cours.

Requête SQL

```
select A, B from R, S, T
where R.B=S.B and S.C=T.C and R.A=20 and T.D=30;
```

Pseudo-code MR

Entrée :

Map

Fonction(s) de hachage

Shuffle

Reduce

Résultat :