UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES

DataScale-NoSQL
2015-16 L. YEH

Nom:
Prénom:

Durée 45 min. Tous documents autorisés. Justifier vos réponses lorsque c'est nécessaire.

1 QCM

Question 1:

Les SGBD NoSQL implémentent la réplication pour (entourer la ou les bonnes réponses) :

1. Assurer la consistance
2. Permettre le passage à l'échelle (scalabilité).

Pourquoi ?

Réponse (2). Pour passer à l'échelle, une solution est de multiplier le nombre de noeuds participants à la résolution d'une requête. L'idée est d'exécuter les opérateurs de la requête sur chaque noeud où il y a la donnée, évitant ainsi les transferts réseaux qui peuvent être pénalisant pour du big data. Pour un noeud A qui possède beaucoup de données de D1 à Dk, le noeud peut être un goulot d'étranglement si il doit tout traiter. Par contre, si d'autres noeuds possèdent des réplicats de D1 à Dk, le SGBD peut mieux distribuer l'execution. Les autres noeuds peuvent executer en parallèle et ainsi soulager le noeud A. Il est donc important d'avoir des replicats.

Question 2: Que pensez vous des affirmations suivantes :

1. Les SGBDs Hive et Pig offrent du partitionnement vertical (stockage colonne) car sans cela il ne peuvent paralléliser (vrai/faux) Pourquoi ?

   Faux : La parallélisation requière de partitionner horizontalement les enregistrements. Un partitionnement vertical conduit à distribuer sur différents noeuds les enregistrements. Ceci conduit inévitablement à des transferts réseaux pour reconstruire les enregistrements. Donc ne permet pas un passage à l'échelle. Le partitionnement vertical est possible dans certains structures de stockage comme Parquet car il permet de réduire le nombre E/S disque lorsqu'on a besoin que de certains attributs des enregistrements. Il n'est pas concu pour la parallélisation.

2. Les Mises à Jours ne sont pas possibles à cause des défaillances de nœuds dans un cluster (vrai/faux) Pourquoi ?

   Faux. La defaillance des noeuds n'empeche pas les MAJ. Le problème est que des mises à jours requières de la synchronisation qu'on evite dans les systèmes NoSQL afin d'être scalable. Les systèmes NoSQL n'ont pas vocation a être transactionnel. La raison technique est qu'obtenir la scalabilité rend difficile d'obtenir la cohérence (ceci nécessite de synchroniser tous les nœuds !)

3. La raison pour laquelle, les systèmes NoSQL évitent d'utiliser un index BTree est dûe au volume trop important à gérer (vrai/faux) Pourquoi ?

   Faux, Le volume n'est pas un problème.

   Explication 1 : A cause du mode de fonctionnement distribué, on ne sait utiliser un index Btree distribuée pouvant être intégré au coeur d'un système NoSQL.

   Explication 2 : Un Btree est une structure de graphe (arbre). Pour exploiter un Btree, il faut soit qu'un noeud possède l'ensemble du graphe soit qu'il y qu'un Btree soit distribué. Dans le premier cas, l'arbre risque de ne pas tenir dans la mémoire d'un noeud (big data). Dans le second cas, ca

suppose de lier des morceaux de sous arbre entre eux et donc entre les noeuds d'un cluster. Ceci est en contradition avec le le "share nothing" qui est à la base du paradigme Map Reduce.

2 PIG

On re utilise la base des "*appels*" présentée en TD et qui a pour schéma :

`calls(de,vers,duree)`
`users(Prenom,Nom,Tel, Dept,Ville)`

Question 3: Après une séquence de commandes, on obtient un résultat intermédiaire de schéma :

```
grunt> describe gjf1;
gjf1: {group: chararray,jf1: {(u44::nom: chararray,calls::vers: int)}}
```

Nous avons, pour chaque nom de personne qui habite dans le département 44, les numéros appelés.

Donner la séquence de commandes PIG qui aboutit au schéma `gjf1`.

```
calls=load ...
users=load ...
v1 = FILTER calls BY dep=44;
jo = join users by phone, v1 by vers;
jf1= foreach jo generate nom, vers;
gjf1 = group jf1 by nom;
```

Puis compléter la requête pour donner les noms des personnes du 44 qui ont passé plus de 50 appels.
Le résultat doit suivre le schéma : `r: {nom: chararray,nb_appels: long}`

```
r = foreach gjf1 generate group as nom, COUNT(gjf1);
```

3 XML

On reprend le schéma XML du TD ci dessus. Une école gère sa formation dans un document XML suivant la structure :

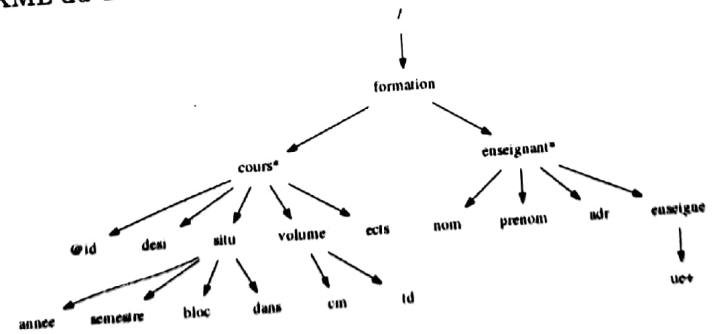

FIGURE 1 Tree Guide du Document

L'élément *ue* signifie "unité d'enseignement". Il contient le code du cours correspondant à `/formation/cours/@id`

Pour simplifier, *adr* ne contient que la région où réside l'enseignant.

Question 4: Écrire une requête XQuery qui donne le ou les noms des enseignants qui enseignent un cours intitulé BD mais qui n'a jamais enseigné un cours de "Gestion d'entreprise". On donne pour ces enseignants également tous ses UE enseignés. Le résultat doit suivre la schéma :

```
<res> (<candidat><nom/><enseigne>(<ue_enseigné/>)*</enseigne></candidat>)*</res>
```

Une solution possible :

```
<res>
{
let $ueBD:=collection("ecole")/formation/cours[contains(desi,"BD")]/@id
let $ueGE:=collection("ecole")/formation/cours[contains(desi,"Gestion entreprise")]/@id
for $i in collection("ecole")/formation/enseignant[enseigne/ue=$ueBD]
where every $p in $i/enseigne/ue satisfies ($p!= $ueGE)
return
 <candidat>{$i/nom}
<enseigne>
 { for $uee in $i/enseigne/ue
   return <ue_enseigné> {$uee} </ue_enseigné>
 }
</enseigne>
 </candidat>
 }
 </res/
```