I. Manolescu, M. Mohanty
Institut Polytechnique de Paris

3 hours

January 15, 2024

Architectures for Big Data (TPT-DATAIA-921)

You are allowed to use: **paper copies of all course and lab material, as well as personal notes**. You are not allowed to use any electronic equipment (computer, tablet, or phone).

1. (1p) We assume the relation R is sharded in 1000 fragments, numbered R_i, $i = 1, \ldots, 1000$. Write the relationship between R and all the R_i shards in relational algebra. You can use the relational operators you need, among: select (σ), project (π), join ($\bowtie$), union ($\cup$), difference ($\setminus$).

2. (1p) Together with the relation R above, we consider a second relation S of 500 tuples, and we want to compute $R \bowtie S$, assuming that the shards of R are uniformly distributed across $N = 100$ machines.

- Briefly describe a simple algorithm to join R and S.
- What is the relational algebraic equivalence law that guarantees your solution is correct?

In all exercices below, <u>underlined</u> attributes are primary keys for the respective relations.

3. (2p) We consider a relation R(<u>id</u>, category, description) where id is an integer. R has 512 million (512M) tuples, and we assume there are 800 different values of the category attribute. Further, we consider a second relation C(category, comment) of 200M tuples, where the attribute category comes from the same domain as R.category.

We have at our disposal a set of $N = 1000$ machines connected in a Distributed Hash Table (DHT) of logical size $NH = 1024$.

1. (1p) Propose a way to distribute R in the DHT so that queries of the form

   ```
   select * from R where id=PARAM
   ```

   where PARAM is a user-supplied constant, run as efficiently as possible. Briefly describe the cost of processing such a query and explain your choice.

2. (1p) Propose a way to distribute R and C in the DHT so that queries of the form

   ```
   select R.category, R.details, C.description
   from R, C
   where R.category=C.category and substring(c.description, KEYWORD)
   ```

   where KEYWORD is a user-supplied constant, run as efficiently as possible. Briefly describe the reasoning behind your choice, and the cost of processing such a query on the data distributed as you propose.

4. **(6p)** Suppose that we are given the following three data sources:

cheese(<u>cheeseName</u>, cheesePrice), e.g., ('camembert', 5.50)
wine(<u>brand</u>, wineFamily, alcohol, price), e.g., ('Pomerol', 'Bordeaux', 13, 20)
goesWith(cheeseName, wineFamily), e.g., ('camembert', 'Bordeaux')

You are given the global schema :

foodProduct(prodCategory, prodName, prodDetailName1, prodDetailValue1, prodDetailName2, prodDetailValue2, prodPrice)
purchaseSuggestion(ifBoughtProdName, buyAlsoProdName)

In this exercise, all the mappings should be SQL queries.

1. (1p) Write the Global-As-View mapping that can populate the relation **foodProduct** based on the three data sources.
2. (2p) Write the Global-As-View mapping that can populate the relation **purchaseSuggestion** so that when a client buys a cheese that goes with a certain family of wine, we suggest they also buy every wine of that family; and symetrically, when a client buys a brand of wine, we suggest that they buy every cheese that goes with the wine family.
   Hint: in SQL, cast($R.a$ as t) allows casting a value of attribute $R.a$ to the type t.
3. (1p) Is it possible to write a mapping that defines the relation **cheese**, and another that defines the relation **wine**, as views over the global schema, in Local-As-View style? In each case, if yes, provide each mapping; if not, briefly explain why.
4. (2p) Is it possible to write a mapping that defines the relation **goesWith** as views over the global schema, in Local-As-View style? If yes, provide the mapping; if not, briefly explain why.

5. (10p) We consider a social network database organized in relations named **User**, **Friend**, and **Post**:

User(uID, name, age, city, country)
Friend(uID1, uID2)
Post(postID, uID, date, title, content, repliesTo)

where: **uID** denotes an user identifier, **postID** identifies each message (or *post*), **repliesTo** is the identifier of a message to which this message replies, or *null* if this message is the first in a conversation. A few sample tuples appear below:

User

uID	name	age	city	country
u1	Anne	25	Orsay	France
u2	Ben	26	Gif	France

Friend

uID1	uID2
u1	u2

Post

postID	uID	date	title	content	repliesTo
p1	u1	1/6/14	"Programming"	http://mashable.com/2014/04/30/programming-is-hard.html	*null*
p2	u2	2/6/14	"Re: Programming"	I liked that!	p1

The three relations are evenly distributed over the nodes in a Hadoop cluster. Give the Map-Reduce programs which compute:

1. (**1p**) For each post, the number of posts which replied to it. (We count the posts *directly* in reply to the first, not replies to replies).
2. (**2p**) For each French user ID, the number of her friends.
3. (**3p**) The IDs of the 10 users having posted the largest numbers of posts.
4. (**4p**) We say user a is in the audience of user b if (i) a has replied to a post of b, and (ii) no one has authored more posts to which a replied, than b. We need to compute all the (a, b) pair such that a is in the audience of b.

Map-Reduce programs should be supplied as diagrams where each task is represented by a rectangle and dependencies between tasks as arrows, together with a short natural-language explanation of each task's role and output. For instance:

$$\boxed{M_1} \longrightarrow \boxed{R_1}$$

"M_1 groups users by their cities, it outputs (city, user) pairs;
R_1 counts the users in each city, it outputs (city, number of users) pairs"