Ioana Manolescu 3 hours November 14, 2019
Université Paris Saclay **Track: D&K** **Architectures for Big Data**
You are allowed to use : **paper copies of all course and lab material**

1. (1p) Give an example *one application* and *a CAP feature* such that the application can run well on top of a system which does not (or not fully) support that feature.

2. (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"

3. (9p) An *interval* is a range $[t_s, t_e]$ where t_s, t_e are two values (for instance, moments in time) such that $t_s \leq t_e$; they are called the start, respectively, the end of the interval.

The so-called *Allen predicates* are defined on intervals; they are illustrated in the figure below, where r_1, r_2 denote intervals.

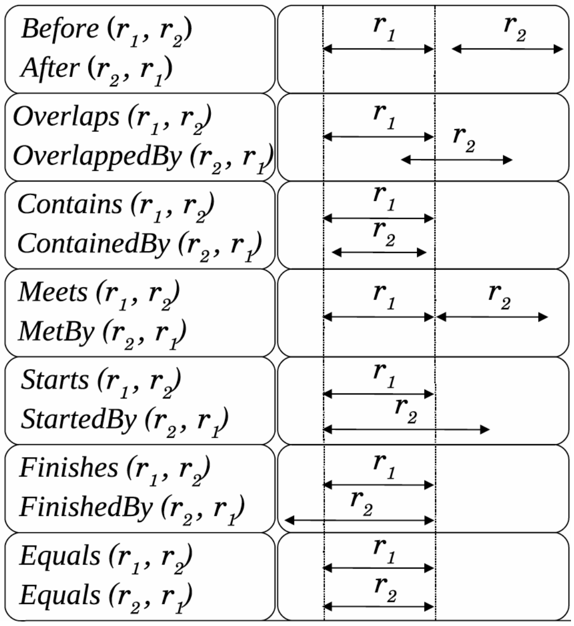

Figure 1: Allen predicates and sample r_1, r_2 interval pairs for which they hold.

For instance, *Before*(r_1, r_2) is true if and only if *After*(r_2, r_2) is true, which holds if and only if $r_1.t_e < r_2.t_s$. The semantics of the other predicates is similarly defined.

Let R_1, R_2 be two relations, each with exactly one attribute, which is of type interval. The *interval join of* R_1, R_2 *on an Allen predicate* p, noted $R_1 \bowtie_p R_2$, is defined as the set of pairs (r_1, r_2) where $r \in R_1, r_2 \in R_2$) such that $p(r_1, r_2)$ is true.

The goal of the exercise is to study *join algorithms for interval data on Map-Reduce.* To help you do that, we introduce the following *helper concepts and operations*:

- Let $[t_0, t_n)$ be the *complete (maximum) time range* in which an interval can occur.
- A *partitioning* $\mathcal{P}$ of $[t_0, t_n)$ is a sequence of contiguous intervals $([t_{i0}, t_{i1}), [t_{i1}, t_{i2}), \ldots, [t_{i(l-1)}, t_{il}))$ such that $t_{i0} = t_0$ and $t_{il} = t_n$. We may also represent a partitioning such as $\mathcal{P}$ by $\mathcal{P} = (p_1, p_2, \ldots, p_l)$ where the *partition-interval* p_j represents the interval $[t_{i(j-1)}, t_{ij})$.
- For an interval u, and partitioning $\mathcal{P}$, we define:
  - **Project**$(u, \mathcal{P}) = \{(p_i, u) | u.t_s \in p_i\}$, in other words: Project$(u, \mathcal{P})$ returns a (key, value) pair where the key is the interval p_i of the partition such that the start of u is in p_i, and the value is u.
  - **Split**$(u, \mathcal{P}) = \{(p_i, u) | u \cap p_i \neq \emptyset\}$, in other words: Split$(u, \mathcal{P})$ returns all the (key, value) pairs where the key is a partition interval that overlaps with u, and the value is u.
  - **Replicate**$(u, \mathcal{P}) = \{(p_i, u) | u \cap p_i \neq \emptyset \vee u.t_s < p_i.t_s\}$, in other words: Replicate$(u, \mathcal{P})$ returns the set of all (key, value) pairs such that: the key is a partition interval having at least one point which is greater than or equal to the start point of u, and the value is u.

- We extend Project, Split and Replicate to sets of intervals (or, equivalently, to relations having only one attribute of type interval), in the natural way: applying Project, Split or Replicate on a set of intervals yields the union of the results obtained by applying the same operator on each interval of the set. Figure 2 illustrates this.

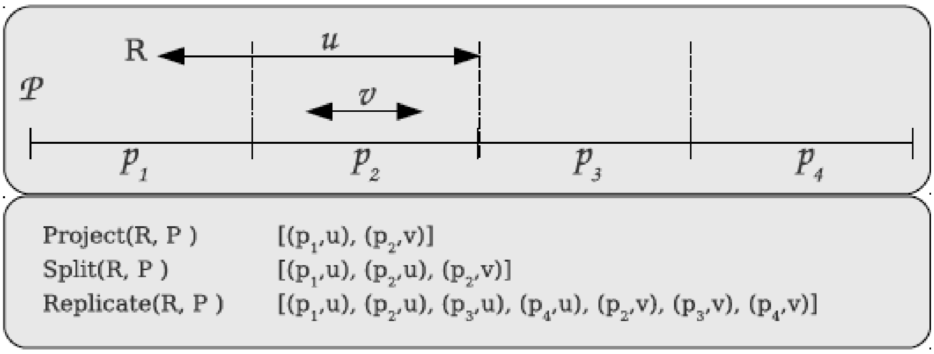

Figure 2: Illustration of Project, Split and Replicate. $R = \{u, v\}$ is a relation containing two intervals, and $\mathcal{P}$ is a partition consisting of four intervals.

We consider two relations (sets of intervals) R_1, R_2, a Map-Reduce cluster of k machines, and a partitioning $\mathcal{P}$ of k successive time intervals.

Using Project, Split, Replicate, describe a Map-Reduce implementation for the following joins:

1. **(2p)** $R_1 \bowtie_{Before} R_2$;
2. **(3p)** $R_1 \bowtie_{Overlaps} R_2$;
3. **(2p)** $R_1 \bowtie_{Contains} R_2$;
4. **(2p)** $R_1 \bowtie_{Meets} R_2$.

Explain how the data must be partitioned in the cluster, and describe the Map-Reduce program implementing each join.