Lecture #4.

Replicated Directories (Distributed Dictionary)

Fischer & Micheal

- maintain a database consisting of a dictionary, i.e., a set of elements with two update operations

  INSERT

  DELETE

  and a single query operation

  LIST.

Insert(x): adds x to the set

Delete(x): deletes x from the set.

List: returns an enumeration of the elements.

Goal: database to be highly available even when nodes or the network is not operational.

Available: any operational node should be able to perform any of the basic operations at any time, regardless of the status of the system.

Each node maintains its copy or more formally VIEW of the database. - and all operations are performed initially on the local VIEW.

Periodically — node sends its VIEW to others. A node receiving such information then updates its own VIEW.

SEND(m) + receive(m)

⇒ eventually everyone converges to the global <u>VIEW</u>.

Correctness condition:

an element $x \in View[i]$ iff i knows of its insertion but does not know of its deletion.

<u>2 Restrictions:</u>

<u>R1.</u> At most one occurrence of insert(x) for each x. (once x has been deleted it can never be reinserted).

<u>R2.</u> delete(x) is only legal at a node j if $x \in View[j]$.

<u>Discuss R1.</u>

- ensures that insert(x) → delete(x)
- if know about both ins(x) + del(x) ⇒ $x \notin View$.
- associate a timestamp/unique-id

$$\left.\begin{array}{lcl} \text{"foo"} & \rightarrow & foo \\ \text{"foo"} & \rightarrow & foo' \end{array}\right\} \text{control point-of-view}$$

- multiple deletions of the same x are permitted.

Algorithm

$OP = \{ insert(x),\ delete(x) \mid x \in D \} \cup$

$\{ List \} \cup$

$\{ send(m),\ delete(m) \}.$

an event $\underline{e}$

$op(e) \in OP.$

$node(e) \in \{1, 2, \ldots, N\}.$

E be the set of all events e.

$D(E) = \{ x \in D \mid op(e) = insert(x) \text{ for some } e \in E \}.$

E is a partial order as defined by the logical clocks.

$x \in view(e')$ iff

V1: $\exists e$ s.t. $e \rightarrow e'$ and $op(e) = insert(x)$, and

V2: $\forall e$ s.t. if $op(e) = delete(x)$, $e \not\rightarrow e'$.

A Naive Solution.

Each node i: $\underbrace{I_i}_{\text{inserts}}$ and $\underbrace{D_i}_{\text{deletes.}}$

$\Rightarrow V_i = I_i \setminus D_i$

send(m)@j: send($\langle I_j, D_j \rangle$)

receive($\langle I_m, D_m \rangle$)@i:

$$I_i = I_i \cup I_m$$
$$D_j = D_i \cup D_m.$$

Drawback:

- Unbounded size of I_i, & D_i

- List operation involves expensive set operation.

Ideally — maintain V_i at all times.

But → can't replace V_i by $V_i \cup V_m$

why?

$x \in V_i \cup V_m$ may be missing from V_k (either V_i or V_m)

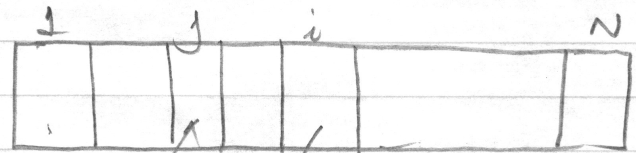
1. x used to be in V_k but it has been deleted, or
2. x was inserted so recently that node k has not yet heard about it?

<u>Case 1</u> x should not be in the resultant view

Case 2 x should be in the resultant view.

To distinguish this each node maintains:

1.



$\underline{Clock_i}$: logical clock @ i

i's view of the insertions @ j.

2. each x is tagged with $(node_x, T_x)$

id of the node where the insertion took place

the timestamps

$$del(V, T, x) \text{ iff } \{x \notin V \wedge T_x \leq T[cre_x]\}$$

Insert(x): $T_x = Clock[i]{+}{+}$
$cre_x = i$
$V_i = V_i \cup \{\langle x, cre_x, T_x \rangle\}$

Delete(x): $V_i = V_i \setminus x$;

List : return V_i

send(m): send $\langle V_i, T_i \rangle$

receive (m) :

Let m be $\langle \overline{V}, \overline{T} \rangle$.

$$V_i = \{ x \in (V_i \cup \overline{V}) \mid \sim del(V_i, T_i, x) \wedge \sim del(\overline{V}, \overline{T}, x) \}$$

$$\underline{\forall k} \quad T_i[k] = max(T_i[k], T[k])$$

<u>Initially</u>

$\forall i,j \quad T_i[j] = 0$

$\qquad V_i = \emptyset$.