Lecture #4

Replicated Directories (Distributed Dictionary)
Fischer & Michael

- 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 in the local VIEW.

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

SEND(view) + receive(m)
⇒ eventually everyone converges to the global VIEW.

Correctness condition:

an element \(x \in \text{View}[i]\) iff i knows of its insertion but does not know of its deletion.

2 Restrictions:

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

R2. delete(\(x\)) is only legal at a node \(j\) if \(x \in \text{View}[ij]\).

Discuss R1.

- ensures that insert(\(x\)) ⇒ delete(\(x\))
- if know about both insert(\(x\)) + delete(\(x\)) ⇒ 
  \(x \notin \text{View}\)
- associate a timestamp/unique-id
  "foo" ⇒ "foo", "control point" \(\notin \text{View}\)
  "foo" ⇒ "foo", "control point" \(\notin \text{View}\)
- multiple deletions of the same \(x\) are permitted.
Algorithm

\[ \text{OP} = \{ \text{insert}(x), \text{delete}(x) \mid x \in D \} \cup \{ \text{List}\} \cup \{ \text{send}(m), \text{delete}(m) \} \]

an event \( e \)

\[ o_p(e) \in \text{OP} \]

\[ \text{node}(e) \in \{1, 2, \ldots, N\} \]

\( E \) be the set of all events \( e \)

\[ D(E) = \{ x \in D \mid o_p(e) = \text{insert}(x) \text{ for some } e \in E \} \]

\( E \) is a partial order as defined by its logical clocks.
\[ x \in \text{over} (e') \iff \]

\[ \forall i: \exists x. e ightarrow e' \text{ and } op(e) = \text{insert}(x), \text{ and} \]

\[ \forall j: \forall y. op(e) = \text{delete}(x), e \not\rightarrow e'. \]

**A Naive Solution**

- Each node \( i \): \[ T_i \text{ and } D_i \]

\[ \text{insert } \quad \text{delete}. \]

\[ \Rightarrow V_i = T_i \setminus D_i \]

\[ \text{send}(m) @ j: \text{send}(<T_j, D_j>) \]

\[ \text{receive}(<I_m, D_m>) @ i: \]

\[ D_i = T_i \cup I_m \]

\[ D_j = D_i \cup D_m. \]

**Drawback:**

- Unbounded size \( T_i, D_i \)

- List operation involves expensive set operation.
Ideally - maintain $V_i$ at all times.

But I can't replace $V_i$ by $V_j U V_m$

Why?

$x \in V_i U V_m$ 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 redo $k$ has not yet been done on it.

Case 1 $x$ should not be in the resultant view.

Case 2 $x$ should be in the resultant view.
To distinguish this each node maintains:

1. $i$’s view of the insertions $@ j$.

clock: logical clock $@ i$

2. Each $x$ is tagged with $(nodelx, Tx)$

$del(V, T, x)$ if $f(x) \notin V \land T_x < T[cpx]$?

$Insert(x)$:

$\begin{align*}
    T_x &= clock[i]++ \\
    cpx &= i \\
    V_i &= V_i \cup \{(x, cpx, T_x)\}
\end{align*}$

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

$List()$: return $V_i$

$Send(m)$: Send $\langle V_i, T_i \rangle$
receive \( m \):

- let \( m = (\overline{\mathbf{V}}, \overline{\mathbf{I}}) \).

\[
\begin{align*}
V_i &= \max \left\{ x \in (V_i \cup \overline{V}) \mid \neg \text{del}(V_i, \overline{V}, x) \land \neg \text{del}(\overline{V}, \overline{\mathbf{I}}, x) \right\} \\
\forall k \quad T_i[k] &= \max(T_i[k], T[k])
\end{align*}
\]

Initially:

- \( \forall i, j \quad T_{i,j}[0] = 0 \)
- \( V_i \) is \( \emptyset \).