The Dictionary Problem by Wu & Bernstein

Replicated Log to achieve mutual consistency

The log contains: record of updates on the objects

Model:
- Same as before - N nodes.
- \((E, \rightarrow)\)

  \(E_1\) events at the same node are totally ordered

  \(E_2\) send \((e)\) and receive \((e)\): \(e_1 \rightarrow e_2\).

The Log Problem.

type Event =
  record
    op: Op
    time: Time
    node: Node

event record \(e\) of \(E\)

\(E\).op \(op(e)\) = operation

\(E\).time \(time(e)\) =

\(E\).node \(node(e)\)

\(f \rightarrow e \iff f \rightarrow \text{ Log}(e)\)
Dictionary

- Same as before.

P2. \( x \in V(e) \iff e_x \rightarrow e \not\exists x-\text{delete} \text{ event } e' \), \( s, t, e' \rightarrow e \).

\[ V(e) = \{ x \mid e_RCL(e) \land \nexists g \text{RCL } s, t, q.R.p = \text{delete}(x) \} \]

O1. The entire log is sent in each message.

O2. A new view is repeated computed.

O3. The entire log is maintained.

Each node \( N_i \): maintains:

1. Clock:

2. A two-dimensional time-table \( T_i \): \( T_i [1..N] [1..N] \).
$T_i[i,k] = t \Rightarrow N_i$ knows that $N_j$ has learned all events up to at $N_k$ up to time $t \oplus N_k$

$T_i[i,i] = N_i$'s clock.

$T_i[i,k] = N_i$'s knowledge of events at $N_k$.

3. A copy of the log.

$\text{taotecrd}(T_i,eR,k) = T_i[k,eR,\text{node}] \oplus \text{time}$

$\Rightarrow N_i$ knows that $N_k$ has learned of $e$. 
Init

\[ V_i = \emptyset \]
\[ PL_i = \emptyset \]
\[ T_i[*,*] = \emptyset \]

\[ \text{insert}(x) : \]
\[ T_i[i,i] = \text{clock}_i + \]
\[ PL_i = PL_i \cup \{<\text{ins}(x), T_i[i,i], i> \} \]
\[ V_i = V_i \cup \{x\} \]

\[ \text{delete}(x) : \]
\[ T_i[i,i] = \text{clock}_i + \]
\[ PL_i = PL_i \cup \{<\text{del}(x), T_i[i,i], i> \} \]
\[ V_i = V_i \setminus \{x\} \]

send(m) to N_k.

\[ NP = \{e \in R | e \in R \in PL \land \neg \text{hasrecord}(T_i, eR, k) \} \]

send \(\langle NP, T_i \rangle\) to N_k.
\[ \text{receive } \langle N_{P_k}, T_k \rangle \text{ from } N_k. \]
\[ NE = \{ R | R \in N_{P_k} \land \neg \text{hasRecord} \left( T_i, R, i \right) \} \]
\[ V_i = \{ x | x \in V_i \text{ or } e_x \notin \text{NE} \} \]
\[ \wedge \left\{ \exists \text{dRENE } s.t. \text{drop} = \text{del}(W) \right\} \]
\[ \forall I \text{ do } T_i[i, I] = \max \left\{ T_i[i, I], T_k[i, I] \right\} \]
\[ \forall I, J \text{ do } T_i[I, J] = \max \left\{ T_i[I, J], T_k[I, J] \right\} \]
\[ \text{PL}_i = \{ e_R | e_R \in \text{PL}_i \cup \text{UNB} \} \wedge \]
\[ \left( \exists j \text{ s.t. } \neg \text{hasRecord} \left( T_i, e_R, j \right) \right) \]
Global Snapshots in Distributed Systems.

Source: Chandy & Lamport

Motivation:
- global state of a distributed computation.
- Why:
  i) deadlock in the system?
  ii) computation has terminated?
  iii) checkpointing the system state (why?)

Why is it difficult?
- processes can record its own state and the messages it sends and receives.
- it can record nothing else.
- processes must cooperate with others.
- all processes cannot record their local states at the same time.
- no common clock, no shared memory.

- the problem is compute global snapshot:
  - local states + commun, channel states.

Finally, should not impede/interfere with the underlying computation.
System Model.

A distributed system:
- a finite set of processes, and
- a finite set of channels

Channels:
- infinite capacity
- error-free
- deliver messages in the order sent

The state of a channel:
sequence of messages sent excluding
last messages received

\[ m_1, m_2, m_3 \quad \rightarrow \quad m_1, m_2, m_3 \]

State: \[ m_2, m_3 \]
Example 2.1

System: $P \xrightarrow{c} Q$

Process state:
- $S_0$: Send token
- $S_1$: Receive token

$P$ in $S_1$; $Q$ in $S_0$.
The Algorithm:

- Use an example to motivate the steps of the algorithm.
- Assume that:
  - We can record the state of the channel in-continuously.

Ex.

```
record set of P in in-P.
↓
in-C
↓
record; state of y
state of c
state of c'
⇒ 2 tokens in-P + in-C.
```

Inconsistency:

- state of P before P sent a message along C

  +

- state of C after P sent the message.

Let m be the # of messages sent along C before P's state change.
Let m' be the # of messages sent along C before C's state change.

Global state inconsistent if

\[ m < m'. \]
Alternate Scenario:

state(C) recorded in P,

state(P), state(q), state(c') recorded in N,

⇒ No token.

Inconsistent if C receives before P sends a message along C and state of P after P sends that by

n > n'⇒ inconsistent global state\n
m = m' for consistency.

m' ≥ m' record

m ≥ m
Alg.
Marker-sending Rule for a Process p:

For each output channel C of p:

p sends a "marker" along C after recording its state and before sending any further messages.

Marker-receiving Rule for a Process q:

If q has not recorded its state then:

q records its state
q records the state of C as the empty seq.
q sends marker on all outgoing C

else

q records the state of C as the seq. of messages received along C after q's state was recorded and before q record the marker along C.