Logical Clock, Group Communication and Multicasting

CMSC 602
Advanced Operating Systems

- More on Vector
- Group Communication
- Matrix Clock
- Reading §3.4.2 to §3.4.4, §4.1.5, §9.2, §12.3.3

Ju Wang, 2003 Fall

Virginia Commonwealth University
More on Vector Clock: Causality Violation

With Lamport’s original logical clock, we still may encounter some problems. For example: causality violation:

P1
migrate O to P2
"on P2"
M1
P2
"where is O?"
M2
"I don’t know"
M3
P3
Error!
"where is O"
More on Vector Clock: Causality Violation

• Lamport’s logical clock cannot be used to detect if two events have a causal relationship.

• It can only be guaranteed: if there is a causal relation between two events, their logical clock values are reasonable.

• Vector clock is proposed to capture all the causality relationships.
Detecting Causality Violation with Vector Clock

migrate O to P2

"on P2"

(2,0,1)

(3,0,1)

(3,1,3)

(3,2,3) "I don’t know"

(3,3,3)

(1,0,0)

P1

M1

P2

(0,0,1)

"where is O?"

M2

(3,0,2)

M3

(3,0,3)

"where is O"

(3,0,3)

(3,2,4)

P3

Error!
Group Communication

All messages are multicasting message

- FIFO order: Message from a single source are delivered in the order of sent. Can be easily achieved using message sequencing.

- Causal order: Causally related messages should be delivered in causal order.

- Total order: All messages multicast to a group are delivered to all members in the same order.
  - §4.1.5 two phase total order
  - §12.2.6 Atomic Group Multicast
Causality Communication

- A communication where a processor never experiences a causality violation

- Method: delay the processing of all messages with causality violation

- Similar to TCP in-sequence delivery, here use a sequence vector (a special case of vector clock)

```
message arrival order
x1 x2 x4 x5 x3
  deliver deliver buffer buffer deliver x3 x4 and x5
```

How to achieve causality comm for non multicasting scenario? Deadlock!!!
A Causal Multicasting Protocol

- Assumes multicast in a closed group, consisting of node 1..n, with the source a member of the group.

- Each group member maintains a vector $S = (S_1..S_n)$ representing the number of messages received from each member.

- When member i sends a multicast message, it increments $S_i$ by 1 and attach the updated vector to the message, denoted as $T = (T_1...T_n)$

- When member j receives a multicast message m originated from i, it will
  - Accept m if $T_i = S_i + 1$ and $T_k \leq S_k$ for all $k \neq i$.
  - Delay m if $T_i > S_i + 1$ or there is a k such that $T_k > S_k$
  - Reject m if $T_i \leq S_i$. It is a duplicate.

- Notice that the similarity between message sequence vector and logical vector clock
Causality DAG (Directed Acyclic Graph)

- Each message is uniquely identified using a (processid, sequence) pair.

- Explicitly state the ID of all causally related message in the message body
  - Use some transitive property of causality to reduce the message size

- The receiver will update the causality DAG according the causality information contained in the message

- If it is found that some causally earlier message is missing, buffer the received message.

- Detailed protocol is discussed in §12.2
Two-phase Total Order Multicasting

First Phase:

- Message sender broadcasts messages,
- All the group member will reply ACK with logical time stamps
- Sender will wait and collect all the ACK

Second Phase:

- Sender calculate the highest ACK timestamps
- Sender broadcasts a commit message with the new logical timestamps
- Group member will decide whether a committed message should be buffered or not, based on the commit timestamp of the multicast message.
Two-phase Total Order Multicasting: Example

A member with initial logical time \( n \)

Time increase \( d=1 \) between two events

ACK message

Multicast message
Two-phase Total Order Multicasting: Delivery Decision

How to decide there is no message with smaller committeeman timestamp?

- Pending message: those message whose commitment time is not known yet
- Find the minimum ACK time of all pending messages $TACK_{pending}$
- If the commit time of a committed message is less or equal than $TACK_{pending}$, then it can be delivered.

Buffer Management in process g1

<table>
<thead>
<tr>
<th>Multicast Message</th>
<th>Acknowledge Time</th>
<th>Commit Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>M0</td>
<td>2</td>
<td>Delivered</td>
</tr>
<tr>
<td>M1</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>M2</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>M3</td>
<td>10</td>
<td>pending</td>
</tr>
</tbody>
</table>
Two-phase Total Order Multicasting: Summary

- It is a distributed algorithm
  - A centralized solution has to assume a global sequencer.
- Local timer follow Lamport’s logical time
- Not able to preserve causality relation
- Committeeman time is decided by the latest ACK message
- Only guarantee that the delivery order at all member are the same!
  - Why?
Matrix Logical Clock

- A $n \times n$ matrix clock $MC_i[k, l]$ is used at each process $P_i$

- The $j^{th}$ row of the matrix means the knowledge process $P_i$ has about the vector logical clock at process $P_j$.

- For process $P_i$, the $i^{th}$ row is actually the vector clock.

- Updating rules are similar to those for vector logical clocks.
  - For each local event at $P_i$, do $MC_i[i, i] = MC_i[i, i] + d$
  - When sending a message from $P_i$ to $P_j$, the entire matrix clock is attached
  - $P_j$ update the $j^{th}$ row of its matrix clock by $MC_j[j, l] = \max(MC_j[j, l], TS_i[j, l])$, for $l = 1...n$
  - $P_j$ update the other row of its matrix clock by $MC_j[k, l] = \max(MC_j[k, l], TS_i[k, l])$, for $k \neq j$ and $l = 1...n$
Garbage Collection with Matrix Logical Clock

- Log propagation problem (§12.3.3)

- How to prevent the log from unbounded increasing

- When processor $p$ propagate its log to processor $q$, it only need to send those log information that is not known by $q$, which is based on the matrix clock.