Distributed Algorithms

Assignment 05

Group 11	Xugang Zhou	352032
	Fangzhou Yang	352040
	Yuwen Chen	352038

December 16, 2013

1 Vector Clocks Theory

- a. The logical time stamps C(e) define a partial order on the set of events: $e_1 < e_2 \Leftrightarrow C(e_1) < C(e_2)$.
- b. For each event a and b, since $a \rightarrow b \Leftrightarrow V(a) < V(b)$, and $V(a) < V(b) \Rightarrow C(a) < C(b)$, thus $a \rightarrow b \Rightarrow C(a) < C(b)$. Therefore, the sum clock C fulfills the clock condition.
- c. Supplement to a total order by using process identities as tiebreaker: $e_1 < e_2 \Leftrightarrow C^{'}(e_1) < C^{'}(e_2) \Leftrightarrow C_1 < C_2 \vee C_1 = C_2 \wedge P_1 < P_2$.
  Since the process identities are unique, for two arbitrary events:

$$e_1 \neq e_2 \Rightarrow e_1 < e_2 \vee e_2 < e_1$$

2 Vector Clocks Practical

It is possible to check termination by looking at the vector time stamps. For a vector of length n, when the current time stamp of the Initiator has fulfilled the following condition:

$$\forall i \in [0, n) : V_i == 4$$

Then the algorithm can be stated terminated, since such a vector indicates that every node has witnessed 4 local events(including two explorer sending events and two echo receiving events).

3 Relation between Time Stamps

- Vector Clocks: $a \rightarrow b \Leftrightarrow V(a) < V(b)$, which means $a \rightarrow b \Rightarrow V(a) < V(b)$, $V(a) < V(b) \Rightarrow a \rightarrow b$ and $\neg(V(a) < V(b) \vee V(b) < V(a)) \Rightarrow a||b$, $a||b \Rightarrow \neg(V(a) < V(b) \vee V(b) < V(a))$
- Exact Real Clocks: $a \rightarrow b \Rightarrow R(a) < R(b)$, The inverse does not apply, cause $R(a) < R(b) \Rightarrow a \rightarrow b \vee a||b$
- Lamport Clocks: $a \rightarrow b \Rightarrow L(a) < L(b)$, The inverse does not apply, cause $L(a) < L(b) \Rightarrow a \rightarrow b \vee a||b$

4 Distributed Deadlock Detection

The algorithm is implemented and tested on a logical unidirectional ring topology. At the initial stage a deadlock would be formed by setting up a circular waiting (every process is waiting for its successor). The Initiator start the detection procedure by sending a probe message to its successor.
After a deadlock is detected, the initiator resolve it by releasing the resource it holds to the predecessor which is waiting.
The algorithm can be applied to arbitrary topology by making an extension. In such a scenario before sending or passing a probe message, the process should be aware of the id of the process that is currently holding the resource, without

which the third part of the probe message cannot be determined. That requires extra discovery procedure such as flooding or other neighbor discovering algorithms, which is outside the scope of deadlock detection.