CMPSCI 677 Distributed Operating Systems **Spring 2015**

Lecture 13: March 9

Lecturer: Prashant Shenoy *Scribe: Fubao Wu*

13.1 Logical Clocks

From last class we know that it is not accurate to use local time to order events in distributed system because of clock drift. One possible solution is to apply logical clocks.

13.1.1 Happened before relation

It is a relation between the result of two events.

- The first observation is that: If A and B are events in the same process and A executed before B, That is, B follows A, then $A \rightarrow B$
- The second observation is that: If A represents sending of a message and B is the receipt of this message, then $A \rightarrow B$.

  If D is happened before A, B is happened before E, we use transitive relation, $D \rightarrow E$.

  It is based on message sending and receiving, no timestamps here.
- If there is no message exchanges between two events, the relation is undefined. For example, D happened before A, H happened before B, A is sending message to B, we can not tell whether D is happened before or after H.

Now the problem is how to define an ordering of all events in a distributed system without global clock or synchronized physical local clocks.

13.1.2 Event Ordering Using HB

According to Happened before relation, we can come out an event order algorithm to use logic clock rather than physical logic.

Our goal is define a notion of time value $C(A)$ of event A, $C(B)$ is the time notion of event B,

- If $A \rightarrow B$ then $C(A) < C(B)$.
- If A and B are concurrent, then $C(A) <, = or > C(B)$.

Lamport developed an logic clock algorithm to derive the order of events and capture the causal relationships. It is also called Lamport clock.

It is assumed each process i has a logic clock which is simply a digit, a counter.LC_i

Any events you care about including each instruction, IO operation, Printing some message etc. occurs at the process in the local machine, increase the logic clock by 1, $LC_i = LC_i + 1$

When process i sends message to j with the value of local clock at process i, process j update its clock value to be $max(LC_i, LC_j) + 1$;

Because we take care of all the events before the sending in sender process A, and all the events after the receiving in receiver process B,the order is based on happened before relation, and also we set the values of the receiver the max between the two processess values.

So all the clock values of all the events after the receiving in B is distributively greater than the values of every events at and before the sending in A which meets the above goals.

13.1.2.1 An example to use Lamport Clock—Totally-Ordered Multicasting

Here is the total order which is different from partial order introduced before. The copy of the clock value of a process is sent to other processes in addition to the destination.

The example given is a big database bank which has two copies of databases, one is in eastern coast, the other is in western coast. Any transaction and update of any account have to go to both database, so they will be sent to one each other.

User 1 arrive one database replica 1 which is closer to him, faster than the other database far from him, while user 2 visit replica 2 first and replica 1 later. The operation from user 1 and user 2 should be executed on the same order of database to prevent the inconsistent state. For example, the database 1 make a deposit of 15 dollars to account, user 2 wants to draw 15 dollars before the deposit, it has only 10 dollars, the result is that it hints it doesnt have enough money. But for user 1, it can withdraw money. So user1 and user2 have different results. They have different order and is in an inconsistent state. Actually, we need all the databases to do the same thing.

To overcome this problem, each query should be sent to multiple databases not only the local database regardless of the delay. We use Lamport clock to get the total order of all the operations. Messages are considered, not the local events. Every processes receives all other processs clock value to get maximum value of the clock, so the state can keep consistent.

13.1.2.2 Limitation of Lamport clock

If $A \rightarrow B$ then $C(A) < C(B)$, but the reverse is not true. We cant say anything based on the comparison of timestamps of two events. What we can do is define another property- causality, that is,

- If $a \rightarrow b$ then a is casually related to b
- In distributed system, there are many Causal delivery: If $send(m) \rightarrow send(n) => deliver(m) \rightarrow deliver(n)$. It means one message is sent before another message, the deliver will also be ordered.

These are properties of causality to capture causal relationships between processes. Then we come up with timestamps mechanism: If $T(A) < T(B)$ then A should have causally preceded B

13.1.3 Vector Clocks

To generalize the notion of Lamport clock, we come up with Vector Clocks Vector clock is a vector of logical clocks in every process that store every clock values of its and other processes, it has the following rule to update clocks.

- All Clock values elements in all processes are initialized as 0
- When a process has an local event, it increase its own logic vector in the vector by one.
- When a process send a message, it sends its entire vector along with the message being sent.
- When a process receives a message, it increases its own logical clock in the vector by one and updates each element in its vector by taking the maximum of the value in its own vector clock and the value in the vector in the received message (for every element).

It is noteworthy that if it is comparable(for which one event A is happened before the other B), every element in that vector clock B is greater than the correspondent element value in the vector A.

13.1.3.1 Global state computation of a distributed system

There are several processes doing computing in distributed system. If one of them crashes, one way to deal with this is to restart all the processes. Another approach is to only restart the crashed process from where it crashes, not from the beginning.

Every process periodically save the state of checkpoint. Its easy to do in single process,once it crashes, it can restart the process from the most recent checkpoint. It is complicated to deal with in distributed system. One way is terminate all the processes, and restart them from their recent checkpoints.

However, if the checkpoints are not synchronized, we could solve them by getting a persistent synchronized consistent checkpoint, or other techniques to capture the recent snapshot.

Global state of a distributed system

- it is a local state of each process
- the messages captured that are in transit but not received.

Each process takes a memory snapshot to record its local memory state when memory is done. Messages which are sent from processes are needed to be captured. Because processes are independent and they do not have global clock and synchronization, We need a consistent state(distributed snapshot) to solve it.

To understand what is consistent and inconsistent state, it is important to remember that for consistent state, duplicate messages and lost messages are not allowed when restart the checkpoint recovery.