**Lecture 12 - Basic distributed protocols**

**Issues**

* no global state (no process has a complete knowledge of the current state of everyone);
* no well-defined time (infinitely accurate clock synchronization is not even theoretically possible).

**Causal relations**

A cross-section of a diagram

Description automatically generated with medium confidence

*a happens-before b* if either:

* *a* and *b* take place on the same node and *a* occurs before *b* in the program there;
* *a* is the event of sending a message on the sender node and *b* is the event of receiving the same message on the destination node;
* there is a transitive chain from *a* to *b* based on the two cases above;

Note: non causally related events are called *concurrent*.

**Consistent chronology problem**

**Problem**

We want that all nodes agree on the order some events happen.

We want to assing to each event a distinct timestamp, such that all timestamps are totally ordered and, furthermore, the order is compatible with causal relations (if event A *happens-before* event B, then the timestamp of A must be smaller than the timestamp of B).

**Lamport clocks**

A diagram of a number of lines

Description automatically generated with medium confidence

Each node keeps the current time as an integer (node: this integer has no relation with the real, physical time; it is just a number that increases as time goes by).

Each time a local event happens on a node, that node increments its clock;

Each message carries the sending timestamp;

When a message is received, the destination sets the clock forward if needed, so that the local clock is larger than the message timestamp;

If we need a strict order of the events, we use the node ID as a tie-break to distinguish events that otherwise have the same timestamp (note that such events can only be concurrent).

**Finding causal relations: vector clocks**

A diagram of a graph

Description automatically generated with medium confidence

Each node keeps a local clock, plus, for each of the other nodes, the most recent value of the clock of that node.

Each timestamp is a vector with the clocks of all processes.

A local event leads to incrementing the local clock, but keeping fixed all other clocks (on the node the event happens).

Each message carries its sending timestamp (the vector of all clock values on the source node).

When receiving a message, the local clock is incremented and then each clock of the receiving node is set to the maximum of the current value and the value from the message timestamp.

Causal relation = all components are in larger than or equal to relation.

**Consistency models (for shared memory and for broadcast of events)**

* Sequential consistency: all events appear to happen in the same order for all processes;
* Causal consistency: two events that are causally related appear in the causal order for all processes; concurrent events may appear in different orders for different processes.
* Aquire/release consistency

**Distributed transactions**

A diagram of a diagram

Description automatically generated

* 2 phases;
* Phase 1: at the end, each sends a message "ready" or "failed" to the coordinator; if the message is ready, the process must be able to commit;
* Phase 2: the coordinator sends "commit" or "abort"

A diagram of a work flow

Description automatically generated

**Distributed shared memory**

A diagram of a program

Description automatically generated