UNIVERSITÄT SIEGEN

Fakultät IV
Betriebssysteme und verteilte Systeme
Prof. Dr. rer. nat. Roland Wismüller

Excercise Sheet 10
Solution

Lecture Distributed Systems
Summer Term 2020

Exercise 1: Totally Ordered Multicast

A second approach is to send the message immediately via multicast, but to postpone delivery until the sequencer has multicast a sequence number for it. This happens after the message is received by the sequencer.

A third approach is to first request a sequence number from the sequencer and then multicast the message.

The first approach (to send the operations to the sequencer) involves sending a point-to-point message with the operation and a multicast message.

The second approach requires two multicast messages: one with the operation and one with the sequence number.

The third approach costs a point-to-point message with the sequence number followed by a multicast message containing the operation.

Exercise 2: Totally Ordered Multicast / Lamport time stamps

No, it is sufficient to multicast any other message type as long as this message has a timestamp greater than that of the received message.

Exercise 3: Transactions

Transactions are a concept closely related to mutual exclusion algorithms because, like these algorithms, they protect a shared resource against simultaneous access by multiple processes. Furthermore, transactions can allow a process to access and modify multiple data elements within a single atomic operation. If the process is interrupted during the transaction, everything is restored as it looked before the transaction started.

a) The most important characteristics of transactions are:

- Atomicity - For the outside world, the transaction will be carried out indivisibly.
- Consistency - The transaction does not violate any system invariants.
- Isolation - Concurrent transactions do not affect each other.
- Durability - After a transaction is committed, the changes remain permanently.

b) A deadlock can occur in the context of concurrency when two transactions wait for each other to release a resource. Since the goal of concurrency is to execute multiple transactions at the same time while maintaining a consistent status of all data elements, a concurrency control is necessary here.

Exercise 4: Two-Phase Commit

A participant could wait in its INIT status for a VOTE-REQUEST message from the coordinator. If this message is not received after a certain time, the participant simply decides to cancel the transaction locally and sends a VOTE-ABORT message to the coordinator.
Similarly, the coordinator can block in the WAIT state, where it waits for the other participants to vote. If not all votes can be determined within a certain time, the coordinator should also decide to cancel the transaction and send a GLOBAL-ABORT to all participants.
The third status in which it is possible to block is the READY status of the participants. Here you could block because a participant is waiting for the global voting result sent by the coordinator. If this message is not received within a certain time, the participant cannot simply decide to cancel the transaction. Instead, it must determine which message the coordinator has sent. The easiest way to solve this problem is to block each participant until the coordinator is up and running again.

Exercise 5: Consistency Models

a) Since strict consistency is not always necessary for many applications, weaker consistency models were introduced. The weaker the consistency model becomes, the easier and more efficient the implementation of memory management becomes, but the effort for the programmer increases.

Weak consistency models stem from the need to use replicas to improve performance. However, efficient replication is only possible if we can avoid global synchronizations, which in turn can be achieved by loosening consistency constraints.

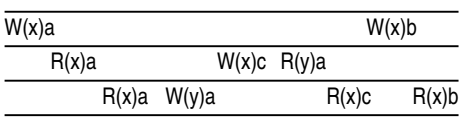
b) Left sequence

- The execution is **not** strictly consistent, because P_2 still reads an old value of x.
- Sequentially consistent means that all processes see all accesses in the same order (not necessarily sorted by time), or that the execution behavior can be obtained by an arbitrary interlocking of the accesses, in which the accesses for each process occur in program order. The sequence is **not** sequentially consistent, since there can be no such interlocking:
  - $R_3(x)b$ must always occur after $W_1(x)b$ (otherwise P_3 would read a),
  - $R_2(y)a$ must occur after $W_3(y)a$ for the same reason,
  - $W_1(x)a, W_1(x)b$ as well as $R_2(y)a, R_2(x)a$ and $R_3(x)b, W_3(y)a$ must be in this order because they take place in the same process.

  For reasons of transitivity, it follows that $R_2(y)a$ must be in the interleaving **after** $W_1(x)b$, which is a contradiction (then b would have to be read).
- Causal consistent means that all processes see causally dependent accesses in the same order. The execution is **not** causally consistent:
  - $W_3(y)a$ is causally dependent on $W_1(x)b$ (because of the communication between P_1 and P_3 via the variable x: $W_1(x)b$ and $R_3(x)b$),
  - P_2 sees these writes in reverse order: it already sees $y = a$ before it sees $x = b$ (the read access $R_2(x)a$ that P_2 executes after $R_2(y)a$ still returns the old value a).

Right sequence:

- The execution is not strictly consistent because P_3 still reads an old value of x ($R_3(x)a$).
- The execution is sequentially consistent: a possible interleaving of the accesses is, e.g., $W_1(x)a, R_2(x)a, R_3(x)a, W_3(y)a,$



- The execution is causally consistent. Causal dependencies occur between $W_1(x)a$ and $W_2(x)c$ and between $W_1(x)a$ and $W_3(y)a$. In both cases, all processes see the same order. $W_1(x)b$ and $W_2(x)c$ are causally **in**dependent, so it is no problem that P_3 first sees $x = c$ and then $x = b$.
  (The sequence must also be causally consistent because it is sequentially consistent.)

Exercise 6: Sequential Consistency

P_1 reads the value of $x = 2$ before setting the value of y to 1, but P_2 sets x to 2 only after reading $y = 1$ (y was zero before). Therefore, these two executions are incompatible and the storage behavior is not sequentially consistent.

Exercise 7: Causal Consistency

a) P_2: Reading x and writing y creates a causal dependency. I.e., $W_1(x)1$ is causally before $W_2(y)2$. But to explain the behavior at P_3, the global order would have to be $W_2(y)2$, $R_3(y)2$, $R_3(x)0$, $W_1(x)1$, which contradicts this ($W_1(x)1$ is after $W_2(y)2$).

b) The causal consistency may be sufficient. The problem is that reactions to changes in stock values should be consistent. Changes to independent stock values can be displayed in different order.