Algorithms and Distributed Systems 2019/2020 (Lab Five)

MIEI - Integrated Master in Computer Science and Informatics

Specialization block

João Leitão (jc.leitao@fct.unl.pt)

Class Structure

- Quorum Algorithm: ABD
- Test revisions

Read/Write Variable Replication

- It's a particular case of state machine replication.
- The State Machine (that is replicated) only supports two operations: read(); write(v).
  - Write returns an "ack" upon termination.
  - Read returns a value that was previously written (or V_0 *(the initial value)* is no write had been issued).

ABD Algorithm [Attiya, Bar-Noy, Dolev]

- Assumes asynchronous system.
- Assumes Reliable channels (Perfect p2plinks)
- Relies on 2f+1 replicas to tolerate up to f crash faults (assumes fault crash model)
  - Liveness is only guaranteed up to f faults.
- While the algorithm is presented for a single replicated value, it can be generalized to multiple values.
  - By adding data object (value) identifiers to its interface, and running independent instances of ABD (in parallel) for each independent data object.

ABD: State and Write Algorihtm

- State (for process i):
  - val_i → value, initially assumed to be v_0
  - tag_i → pair <sequence number, id> initially <0,0>
    - <s1,i1> > <s2,i2> iff s1 > s2 | (s1 == s2 & i1> i2)
- Algorithm used by client c to write: WRITE(v)
  - Phase 1: send message “read-tag” to all replicas
  - Wait for a quorum Q (majority) of replies
  - Let seqmax = max{sn:<sn,id>$\in$Q}
  - Phase 2: Send message “write(<seqmax+1,c>,v)”
  - Wait by a quorum of Acks (Majority).
  - Return “Ack” (Write is considered terminated)

ABD: Replica Algorithm (process i)

- Upon reception of message **read-tag**:
  - Return tag_i
- Upon reception of message **write(new-tag,new-v)**:
  - If new-tag > tag_i then
    - tag_i = new-tag
    - val = new-v
  - Return ack

- Upon reception of message **read**:
  - Return <tag_i,val_i>

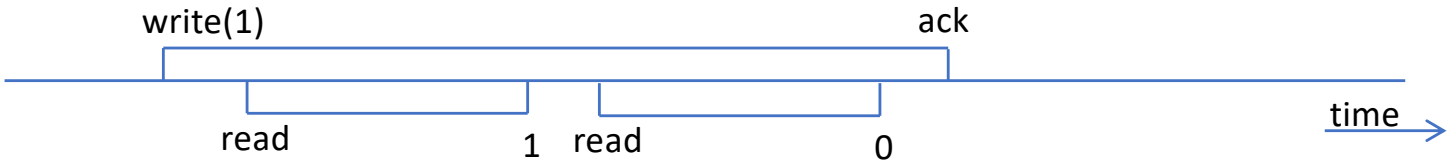
ABD: Read Algorihtm (First Draft)

- Algorithm used by a client c to perform: READ()
  - Send “read” message to all replicas
  - Wait for a quorum Q (majority) of replies
  - Let <tagmax,valmax>$\in$Q with highest tag value
  - Return valmax

- Would this ensure atomicity?

ABD: Read Algorithm (First Draft)
Does not provides atomicity.

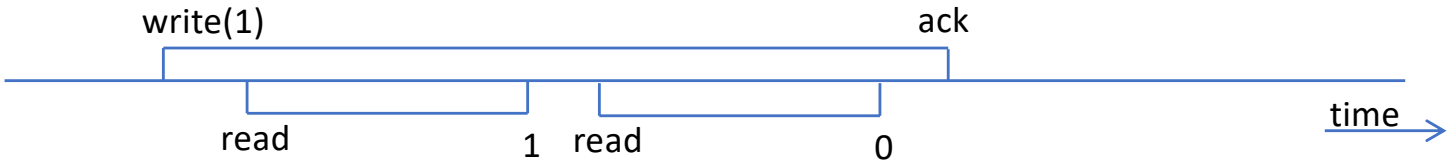
- The following execution is allowed by the algorithm, in which a client first observers the value being written and then observes the previous value.



- How can this happen?

ABD: Read Algorithm (First Draft)
Does not provides atomicity.

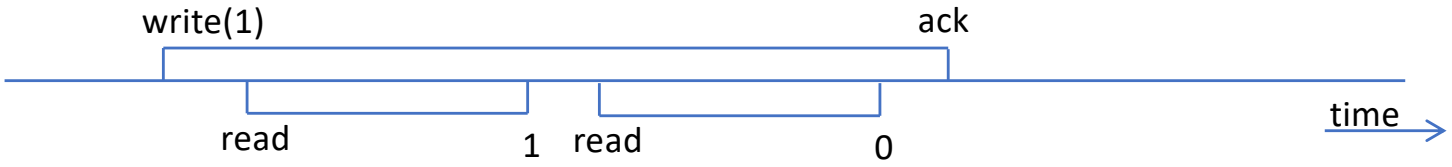
- The following execution is allowed by the algorithm, in which a client first observers the value being written and then observes the previous value.



- How can this happen?
- How can we avoid this?

ABD: Read Algorithm (First Draft)
Does not provides atomicity.

- The following execution is allowed by the algorithm, in which a client first observers the value being written and then observes the previous value.



- How can this happen?
- How can we avoid this?
- Solution: Add a write-back phase to the read protocol.

ABD: Read Algorithm (Correct)

- Algorithm for a client c to perform: READ()
  - Phase 1: Send message “read” to all replicas
  - Wait for a quorum Q (majority) of replies
  - Let <tagmax,valmax> $\in$ Q with highest tag
  - Phase 2: Send message “write(tagmax,valmax)” to all replicas
  - Wait for a quorum of ACKs.
  - Return valmax

ABD: Correction Arguments

- Termination (assuming at most f faults) and that messages are well-formed then this terminates due to properties of underlying links.
- Atomicity (proof sketch)
  - There are serialization points that are ordered by the tag, which are defined in the second phases of both the read and write operation, with the writes necessarily happening before the read.
  - The algorithm is constructed to enforce that read operations read for sure the most recent value, set by a write that completed. If the write did not complete, the read operation upon observing the value forces the write to complete.
  - Reinforcement of a previous write by the read operation will never affect values affected by a subsequent write (since there is a guard on the sequence number)
  - Seriation point for each operation (write) is defined for sure in between the start and the end of the operation (due the way that the sequence numbers are managed)

ABD: More details

Edsger W. Dijkstra Prize in Distributed Computing: 2011

Hagit Attiya, Amotz Bar-Noy, **and Danny Dolev**,
for their paper
"Sharing Memory Robustly in Message-Passing Systems" which appeared in the Journal of the ACM (JACM) 42(1):124-142 (1995).

Available at: http://www.cse.huji.ac.il/course/2004/dist/p124-attiya.pdf