Ίδρυμα Προώθησης Έρευνας

ΚΥΠΡΙΑΚΗ ΔΗΜΟΚΡΑΤΙΑ

ΕΥΡΩΠΑΪΚΗ ΕΝΩΣΗ

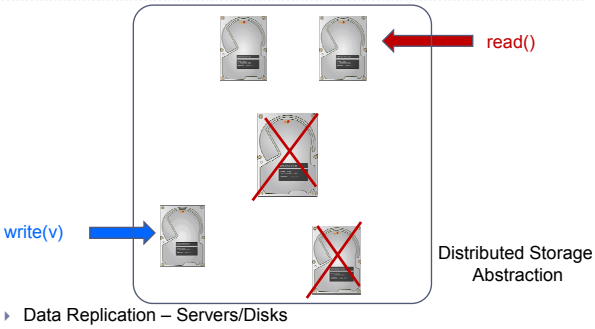
Quorum Views: Client Side Tools for Atomic Distributed Register Implementations

Nicolas Nicolaou

University of Connecticut

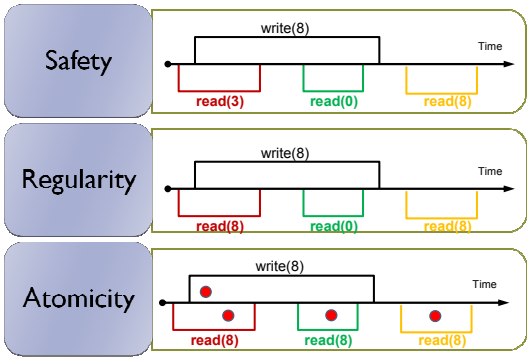
Funded by the Cyprus Research Promotion Foundation, the Republic of Cyprus and the European Regional Development Fund

What is a Distributed Storage System?



- Data Replication – Servers/Disks
  - Survivability and Availability
- Read/Write operations
- Consistency Semantics

Consistency Semantics [Lamport86]



Complexity Measure-Operation Latency

- Consistent Register Implementations
  - **Message-Passing, Asynchronous** model
  - Access **multiple replicas** per operation
  - Perform **multiple accesses** per operation

Operation Latency is measured in
Communication Rounds (round-trips)

Fastness of Atomic Implementations

Traditional SWMR
- Single round writes
- Two round reads
  - Phase 1: Obtain latest value
  - Phase 2: Propagate latest value
    - Folklore belief: "Reads must Write"

SWMR Fast
- Single round (***fast***) writes and reads
  - Bounded readers: $R<(S/f)-2$ where S servers & f failures
  - Impossible in MWMR model

SWMR Semifast
- Fast writes
- Only a single complete 2-round (***slow***) read per write
  - Unbounded readers
  - Bounded Virtual Nodes: $V<(S/f)-2$
  - Impossible in the MWMR model

Quorum Systems

- Quorum System:
  A collection of sets with non-empty pairwise intersections
- Numerous results exploit quorums for
  - MWMR and Dynamic Atomic Memory Implementations
  - "Slow" operations
    - Some achieve fast reads when a write is confirmed [DGLSW03, CGGMS05]
- Refined Quorum Systems - Guerraoui and Vukolic [PODC2007]
  - Introduce Fast Operations in Quorum-Based systems
  - Rely on operation timeout

Question

Can we obtain Fast/Semifast Implementations in a general quorum-based framework?

First the Bad News

- Quorum-Based Fast Implementations are not Fault-Tolerant
  - Possible iff there exists a common intersection among all the quorums
  - A single failure in the common intersection collapses the Quorum system.
- Quorum-Based Semifast Implementations are not Fault-Tolerant
  - One of the properties of the semifast definition is violated if no common intersection exists among all the quorums.
    - A single complete "slow" read operation is not enough for every write operation

Now the Good News

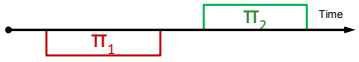
- Trade speed for efficiency and fault-tolerance
  - Allow multiple "slow" reads per write operation but maintain the fast behavior when possible
- To do so, we introduce **Quorum Views**

Model: Definitions

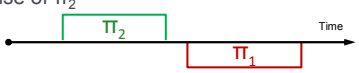
- Asynchronous, Message-Passing model
  - Process sets: writers W, readers R, servers S (replica hosts)
  - Reliable Communication Channels
  - Well Formedness
- Environments:
  - SWMR: $|W|=1, |R|\geq 1$
  - MWMR: $|W|\geq 1, |R|\geq 1$
- Failures:
  - Crash Failures
- Correctness: Atomicity (safety), Termination (liveness)
- Order write operations
  - SWMR: <timestamp, value> pairs
  - MWMR: <tag, value> pairs

Definition: Operation Relations

- Precedence Relations for two operations π_1, π_2:
  - π_1 precedes π_2 if the response of π_1 happens before the invocation of π_2

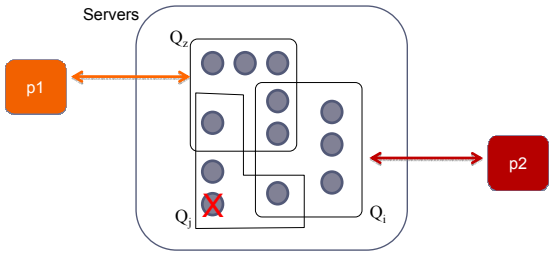


  - π_1 succeeds π_2 if the invocation of π_1 happens after the response of π_2



  - π_1 is concurrent with π_2 if π_1 neither precedes nor succeeds π_2

Definition: Quorum systems



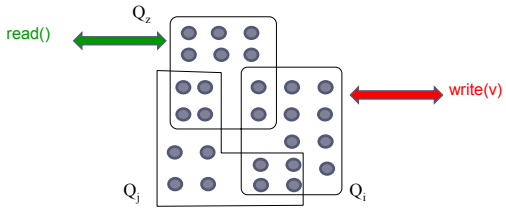
- Q_i, Q_j, Q_z are **quorums**
- **Quorum System** is the set $\{Q_i, Q_j, Q_z\}$
  - Property: every pair of quorums intersects
- Every R/W operation communicates with a single quorum
- Faulty Quorum: Contains a faulty process

Definition: Fastness

- A process p performs a communication round during an operation π if:
  - p sends a message m to a set of servers for π
  - Any server that receives m replies to p
  - Once p receives responses from a single quorum completes π or proceeds to a next communication round
- Fast Operation
  - Completes at the end of its first round
- Fast Implementation
  - All operations are fast
- Communication scheme
  - Message delivery: Servers to Clients
  - No server to server or client to client communicaiton

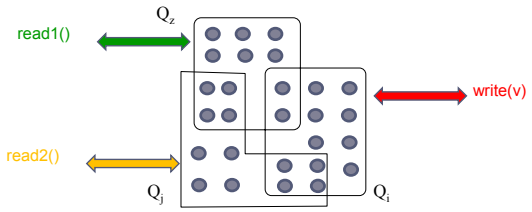
Non-Robust Fast Implementations: Proof Sketch

- Execution a:
  - Complete write(v) => Q_i
  - Complete read() => Q_z
    - read() returns v to preserve atomicity



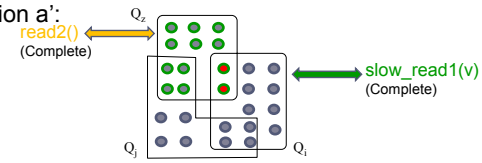
Not Robust Fast Implementations (Proof Sketch)

- Execution b:
  - Incomplete write(v) => $Q_i \cap Q_z$
  - Complete read1() => Q_z
    - read1() cannot distinguish between executions a and b, thus returns v
  - Complete read2() => Q_j
    - read2() returns an older value since does not observe v

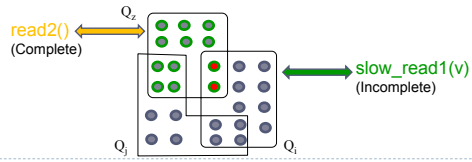


Not Robust SemiFast Implementations (Proof Sketch)

Execution a':



Execution b':



Tool: Quorum Views

Idea:

- Try to determine the state of the write operation based on the distribution of the latest value in the replied quorum.
- Write State in the First Round of Read Operation

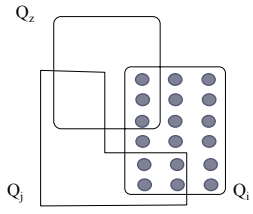
  Determinable => Read is Fast

  Undeterminable => Read is Slow

Determinable Write - Qview(1)

- All members of a quorum contain maxTs

$$[qView(1)] : \forall s \in Q_i : s.ts = \max TS$$

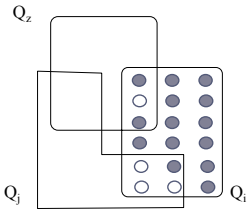


(Potentially) Write Completed

Determinable Write - Qview(2)

- Every intersection contains a member with ts<maxTs

$$[qView(2)]: \forall j \neq i, \exists A \subseteq Q_i \cap Q_j \; s.t. \; A \neq \emptyset \text{ and } \forall s \in A : s.ts < \max TS$$

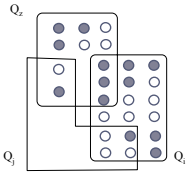


(Definitely) Write <maxTag,v> Incomplete

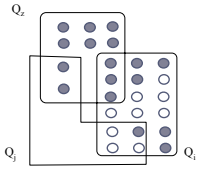
Undeterminable Write - Qview(3)

- There is intersection with all its members with ts=maxTs

$$[qView(3)]: \exists s \in Q_i \; s.t. s.ts < \max Ts \;\; and \;\; \exists j \neq i \; s.t. \forall s \in Q_i \cap Q_j : s.ts = \max TS$$



qV(3) and Incomplete Write



qV(3) and Complete Write

Undeterminable => second Com. Round

Algorithm:SLIQ

Write Protocol: one round

- P1:Writer increments ts and propagates the <ts,v> to a quorum

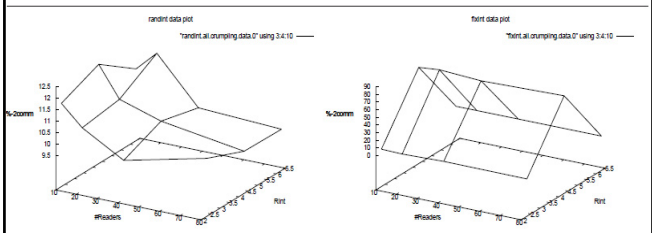
Read Protocol: one or two rounds

- P1: send read requests and wait for replies from a quorum Q
  - $QView_Q(1)$ – **Fast** and return maxTS
  - $QView_Q(2)$ – **Fast** and return maxTS-1
  - $QView_Q(3)$ – **Slow** proceed to P2 and return maxTS
- P2: propagate <maxTS,v> to a quorum and return <maxTS,v>

Server Protocol: passive role

- Receive requests, update local timestamp and return <ts,v>

SLIQ: Simulation Results



Setting a: Stochastic simulations Setting b: Fixed interval simulations

Fig. 3. Simple runs using Crumbling Walls

Follow-Up Question

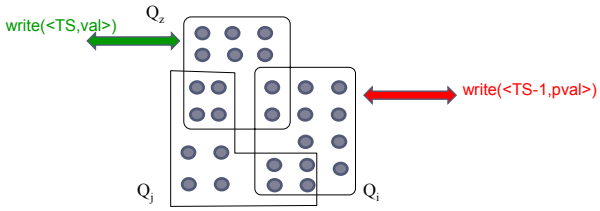
Can Quorum Views be used to obtain fast operations in the MWMR environment?

Tagging the values

- TAG : <timestamp, wid> pair
  - tag1 > tag2 if either:
    - tag1.timestamp > tag2.timestamp, or
    - tag1.timestamp = tag2.timestamp AND tag1.wid > tag2.wid
- Why wid is necessary?
  - Separate writes with the same timestamp

Why be fast in Qview(2) in SWMR?

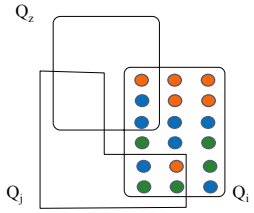
- Single writer
  - If it writes <TS, val> => it completed writing <TS-1, pval>



Any read operation will observe either QView1 or QView3 for <TS-1,pval>

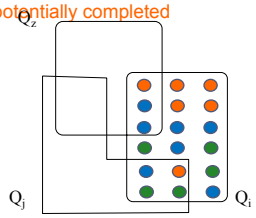
What happens in MWMR?

- MWMR environment
  - Concurrent writes
  - Multiple concurrent values
- For values <tag1,v1> , <tag2, v2>, <tag3,v3>
  - Let tag1 < tag2 < tag3



Idea: Uncover the Past

- Discover the latest potentially completed write
- For values <tag1,v1> , <tag2, v2>, <tag3,v3>:
  - <tag3,v3> not completed (servers possibly contained <tag2, v2>)
  - <tag2, v2> not completed (servers possibly contained <tag1,v1>)
  - <tag1,v1> potentially completed



Algorithm: CWFR

Traditional Write Protocol: two rounds

- P1: Query a single quorum for the latest tag
- P2: Increment the max tag, send <newtag, v> quorum

Read Protocol: one or two rounds

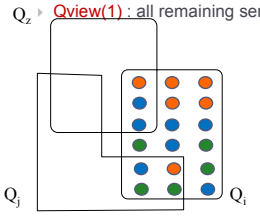
- Iterate to discover smallest completed write
- P1: receive replies from a quorum Q
  - $QView_Q(1)$ – Fast: return maxTag of current iteration
  - $QView_Q(2)$ – remove servers with maxTag and re-evaluate
  - $QView_Q(3)$ – Slow: propagate and return $maxTag_Q$

Server Protocol: passive role

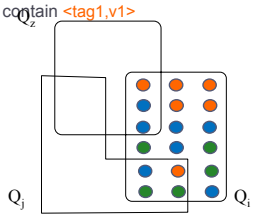
- Receive requests, update local timestamp and return <tag,v>

Read Iteration: Discard Incomplete Tags

- For values <tag1,v1> , <tag2, v2>, <tag3,v3>:
  - <tag3,v3> not completed: remove servers that contain <tag3,v3>
  - <tag2, v2> not completed: remove servers that contain <tag2, v2>
  - <tag1,v1> potentially completed in Q_i
    - Qview(1) : all remaining servers contain <tag1,v1>



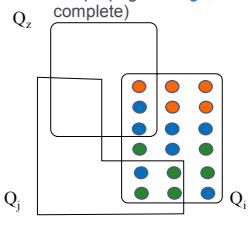
Server Removal



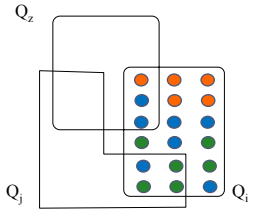
Past Prediction

Read Iteration: Discard Incomplete Tags

- For values <tag1,v1> , <tag2, v2>, <tag3,v3>:
  - <tag3,v3> not completed: remove servers that contain <tag3,v3>
  - <tag2, v2> potentially completed in Q_j
    - Qview(3) : an intersection of the remaining servers contains <tag2, v2>
    - P2: propagate <tag3,v3> to a complete quorum (help <tag3,v3> to complete)



Server Removal



Past Prediction

CWFR: Empirical Results

- In progress experimentation on Planetlab
  - Planetary scale real-time environments
- Measure the percentage of ‘‘slow’’ reads
- Get the average of operation latency
- Compare with classic 2 round aproaches

Closing Remarks