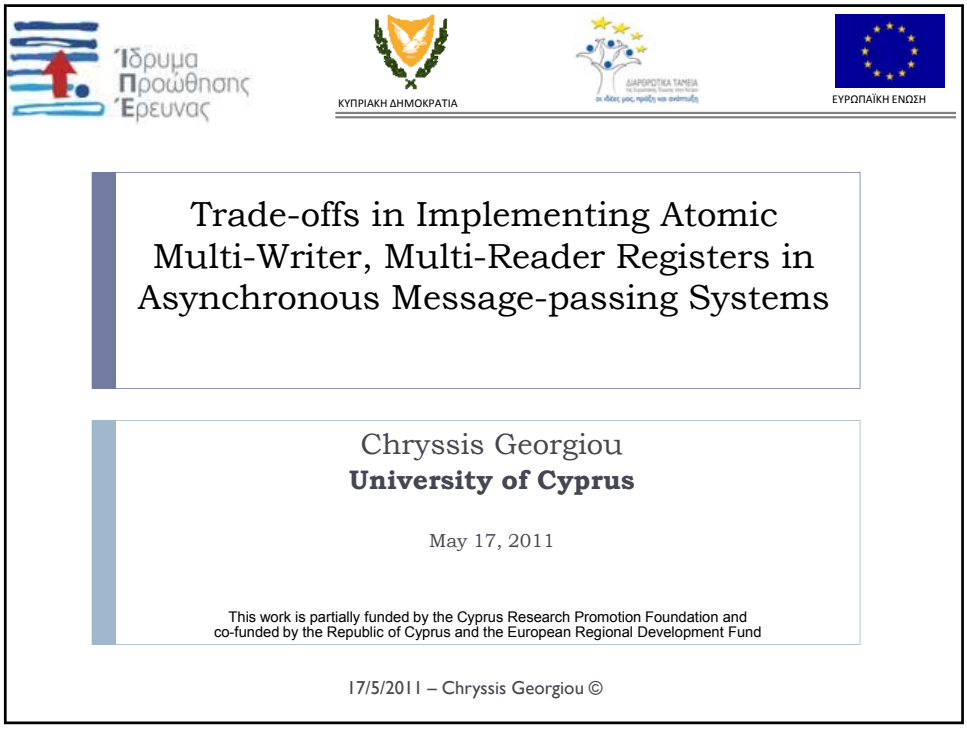

Ίδρυμα Προώθησης Έρευνας
ΚΥΠΡΙΑΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΔΙΑΡΘΡΩΤΙΚΑ ΤΑΜΕΙΑ
ΕΥΡΩΠΑΪΚΗ ΕΝΩΣΗ
Trade-offs in Implementing Atomic Multi-Writer, Multi-Reader Registers in Asynchronous Message-passing Systems
Chryssis Georgiou
University of Cyprus
May 17, 2011
This work is partially funded by the Cyprus Research Promotion Foundation and co-funded by the Republic of Cyprus and the European Regional Development Fund
17/5/2011 – Chryssis Georgiou ©


Motivation for this Research
17/5/2011 – Chryssis Georgiou ©

Survivability of Data

- Data Survivability is essential in today's systems and applications
- Popular approach – RAID [Patterson, Gibson, Katz 1988]
  - Use of a Redundant Array of Inexpensive Disks
  - Mirroring or erasure code is used to prevent loss of data upon a disk failure
  - A RAID system contains
    - a single box
    - residing at a single physical location
    - connected to clients via a single network interface

Single point of failure

[Chockler, Keidar, Guerraoui, Vukolic 09]
[Schroeder, Gibson 07]

Distributed Storage

- Uses Replication

- Fault-Tolerance, Availability and Geographical Proximity

No Free Lunch

Great Challenge:

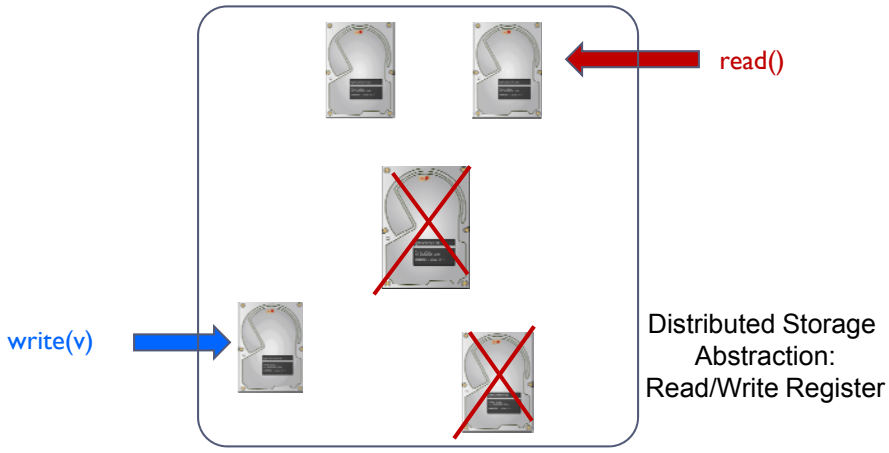
Maintain **consistency** among the replicas despite system **asynchrony** and **failures**

*And do so **efficiently***

Akamai's Top Ten Challenges
Harald Prokop, Senior Vice President of Engineering, July 2008

Problem Abstraction

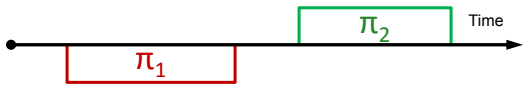
Read/Write Object (register)



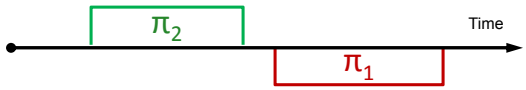
- Read/Write operations – invocation and response
- Concurrency
- Consistency Semantics

Operation Relations

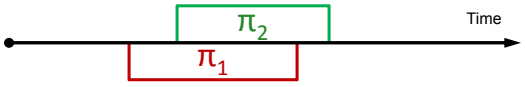
- Precedence Relations for two operations π_1 ,π_2:
  - π_1 precedes π_2 if the response of π_1 happens before the invocation of π_2



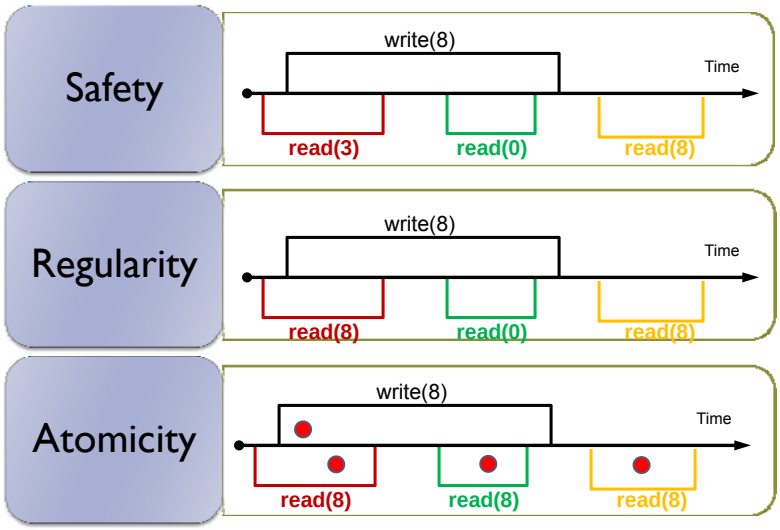
  - π_1 succeeds π_2 if the invocation of π_1 happens after the response of π_2



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



Consistency Semantics [Lamport 1986]



Atomic Read/Write Register [Lamport 1986]

- An atomic Read/Write register allows concurrent processes
  - to share information through a common variable
  - as if they were accessing this variable in a sequential manner
- It supports read and write operations
  - Read() returns the latest written value of the object
  - Write(v) sets the value to v and returns OK

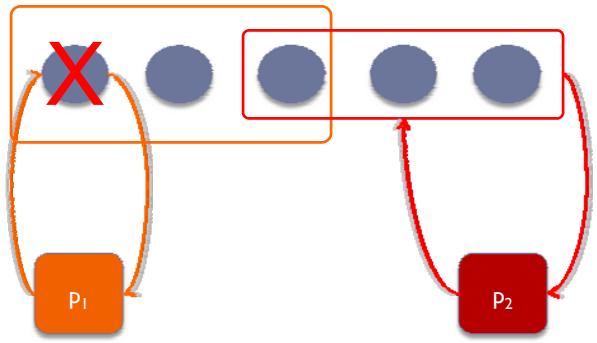
Applicability

- The atomic register is fundamental in distributed computing and is at the heart of a large number of distributed algorithms [Attiya & Welch 1998/2004]
- Atomic register implementations are used as building blocks for complex distributed storage systems [Fleet 2000, SBQ-L 2002, PASIS 2004, HP's FAB 2006, Amazon's Dynamo 2007]
- The can also be used directly to build distributed file systems [Chockler, Keidar, Guerraoui, Vukolic 09]

Challenge: Ordering of read/write operations

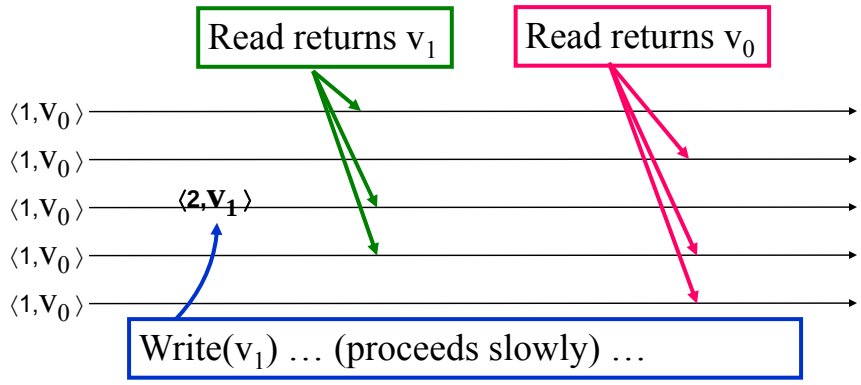
- Using the "time" at which each operation is invoked
  - Clock Synchronization
  - Access to centralize clock (e.g. , GPS)
- Associate a sequence number with each value written
  - tags=<timestamp, wid>, [Attiya, Bar-Noy, Dolev 1995]
    - Timestamp: counter (positive integer)
    - wid: writer id
  - tag1 > tag2 if either:
    - tag1.timestamp > tag2.timestamp, or
    - tag1.timestamp = tag2.timestamp AND tag1.wid > tag2.wid

Challenge: Communication Rounds



Multiple Round-Trips

- Consider the following example [Attiya, Bar-Noy, Dolev 1995]:



Complexity Measure-Operation Latency

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

A process p performs a communication round for an operation π if
- Sends a message m regarding π to a subset (potentially all) of processes
- Any process that receives m, replies to p
- Process p collects "enough" of such replies and proceeds accordingly.

Research Questions

What is the operation latency of atomic register implementations in a fail-prone, message-passing, asynchronous distributed system?

What are the trade-offs to achieve such performance?

Background and Prior Work

Model of Computation

- Asynchronous, Message-Passing model
  - Process sets: writers W, readers R, servers S (replica hosts)
  - Reliable Communication Channels
  - Well Formedness
- Environments:
  - Single Writer, Multiple Readers: $|W|=1, |R|\geq 1$
  - Multiple Writers, Multiple Readers: $|W|\geq 1, |R|\geq 1$
- Failures:
  - Crash Failures
    - A subset of the servers may crash, all writer and reader processes may crash
- Correctness: Atomicity (safety), Termination (liveness)

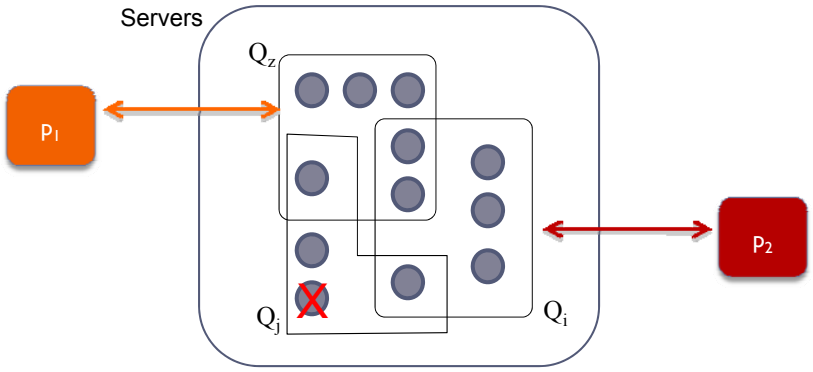
Fastness

- Fast Operation
  - Completes at the end of its first round
- Fast Implementation of an atomic register
  - *All* operations are fast
- Communication scheme
  - Message delivery: Servers ↔ Clients
  - No server to server or client to client communication

  Data-centric communication

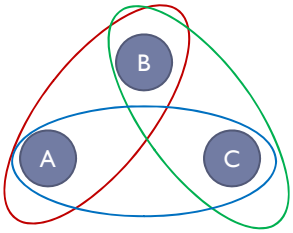
  [Martin, Alvisi, Dahlin 2002]
  [Chockler, Keidar, Guerraoui, Vukolic 2009]

Tools: Quorum systems

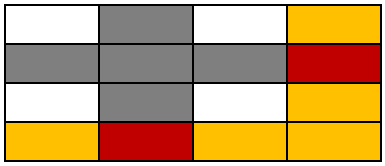


- Q_i, Q_j, Q_z are **quorums**
- **Quorum System** is the set $\{Q_i, Q_j, Q_z\}$
  - Property: every pair of quorums intersects
- Every R/W operation communicates with a single quorum
- Faulty Quorum: Contains a faulty process

Quorum Systems Examples



Majorities [Thomas79,Gifford79]



Matrix Quorums [Vitanyi, Awerbuch 1992]

Prior Work: Traditional Implementations

SWMR

- e.g., [Attiya, Bar-Noy, Dolev 95]
- Single round writes
- Two round reads
  - Phase 1: Obtain latest value (max tag)
  - Phase 2: Propagate latest value (max tag)
    - Folklore belief: "Reads must Write"

MWMR

- e.g., [Lynch Shvartsman 97, 02, Englert, Shvartsman 00]
- Two round writes
  - Phase 1: Discover latest value (max tag)
  - Phase 2: Order new value after the latest and propagate
    - Belief: "Writes must Read"
- Two round reads

Algorithm: Simple

Write Protocol: two rounds

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

Read Protocol: two rounds

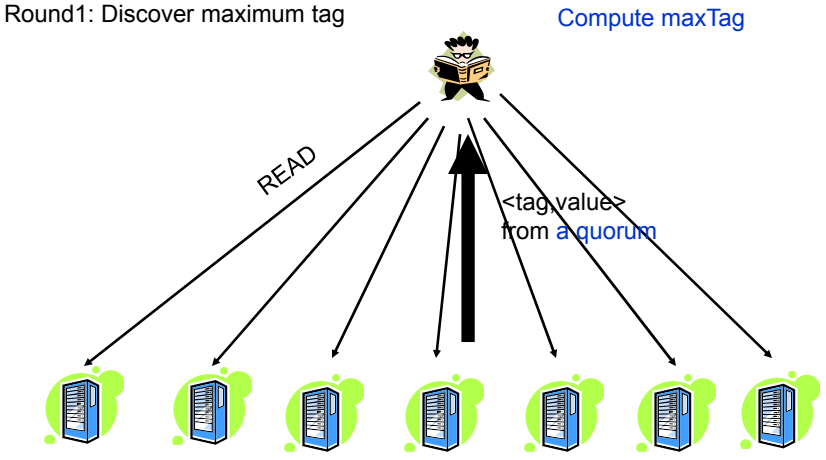
- P1: Query a single quorum for the latest tag
- P2: Propagate <maxtag,v> to a single quorum

Server Protocol: passive role

- Receive requests, update local timestamp (if msg.tag>server.tag) and reply with <server.tag,v>

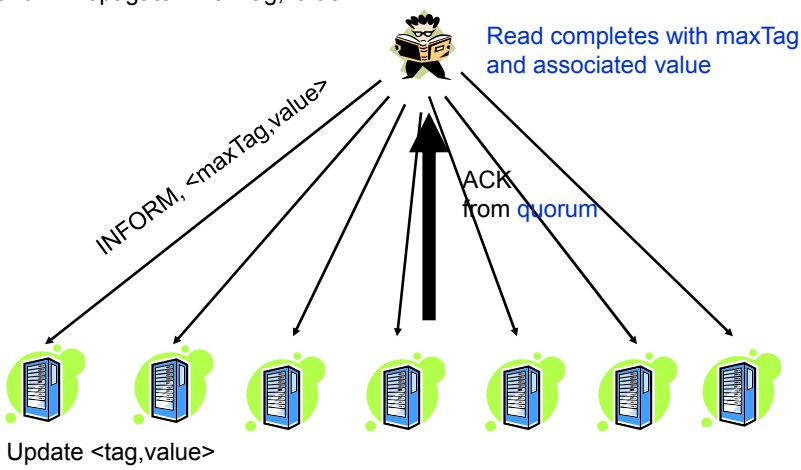
Reader

Round1: Discover maximum tag



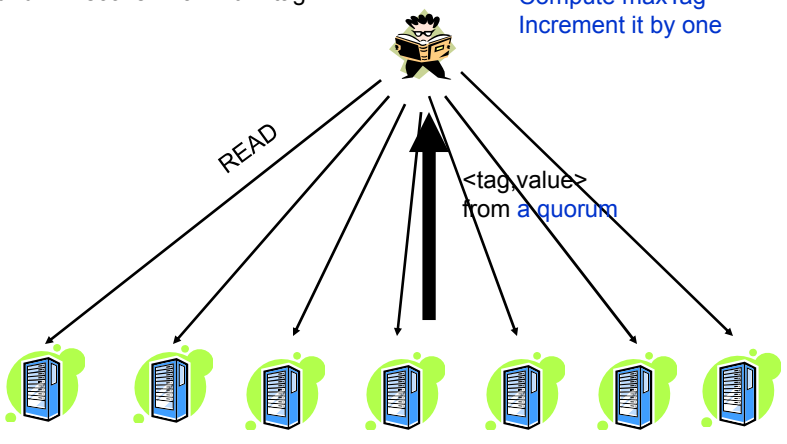
Reader

Round2: Propagate <maxTag,value>



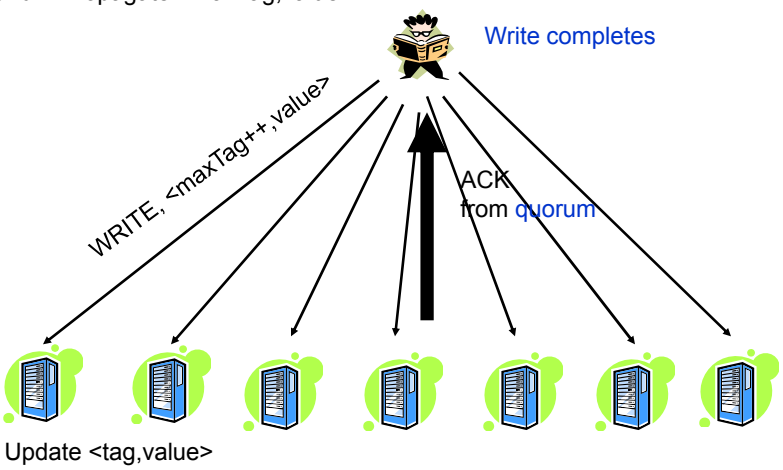
Writer

Round1: Discover maximum tag



Writer

Round2: Propagate <maxTag,value>



Prior Work: Fast Operations

SWMR Fast

- [Dutta et al. 04]
- Single round (***fast***) writes and reads
  - Bounded readers: $R<(S/f)-2$ where S servers & f failures
  - Impossible in MWMR model

SWMR Semifast

- [Georgiou, Nicolaou, Shvartsman 06/09]
- Fast writes
- At most a single complete 2-round (***slow***) read per write (experimental results: 10% slow reads)
  - Unbounded readers
  - Impossible in the MWMR model

SWMR Weak-Semifast

- [Georgiou, Nicolaou, Shvartsman 06/09]
- General Quorum System
- Fast writes and Multiple slow reads per write (exp: 16% slow)
  - Allows concurrent fast reads with writes
  - How about the MWMR model?

Fastness in the MWMR Setting

Question

Can we allow some reads to be fast (single round reads) and still guarantee atomicity in the MWMR settings?

Answer: **YES!!**

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.

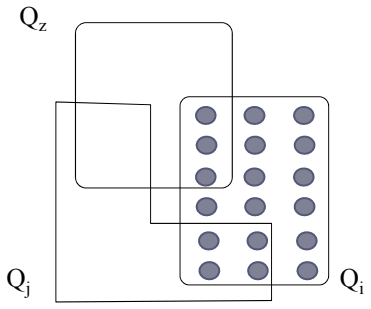
- Write State in the First Round of Read Operation

  Determinable => Read is Fast

  Undeterminable => Read is Slow

Determinable Write - Qview(1)

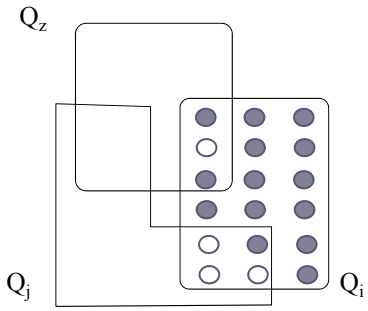
- All members of a quorum contain the maxTag



(Potentially) Write Completed

Determinable Write - Qview(2)

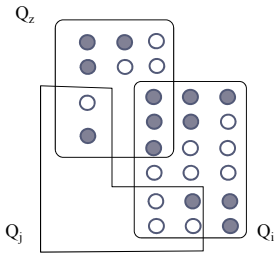
- Every intersection contains a member with tag<maxTag



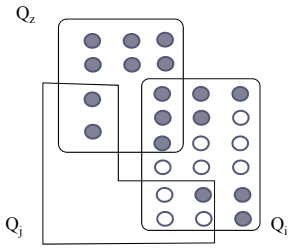
(Definitely) Write <maxTag,v> Incomplete

Undeterminable Write - Qview(3)

- There is intersection with all its members with tag=maxTag



qV(3) and Incomplete Write



qV(3) and Complete Write

Undeterminable => second Com. Round

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

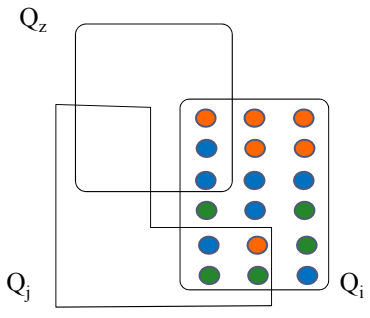
- Iterate to discover smallest completed write
- P1: receive replies from a quorum Q
  - $QView_Q(1)$ – **Fast**: return maxTag of current iteration
  - $QView_Q(2)$ – remove servers with maxTag and re-evaluate
  - $QView_Q(3)$ – **Slow**: propagate and return $maxTag_0$

Server Protocol: passive role

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

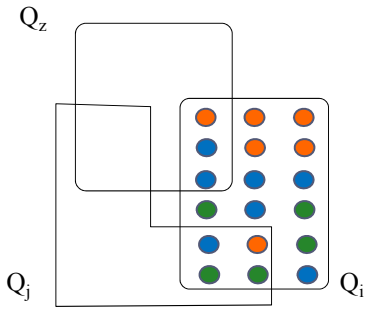
Quorum Views and Multiple Writers

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



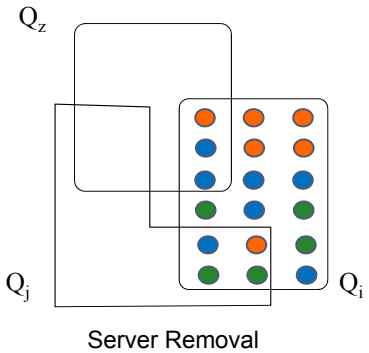
Idea: Uncover the Past

- 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

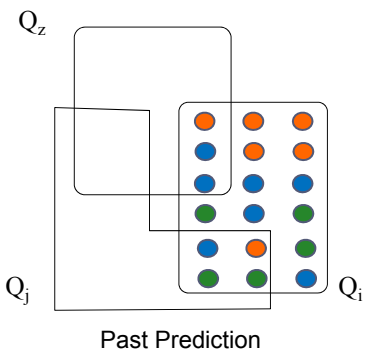


Read Iteration: Discard Incomplete Tags

- 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>



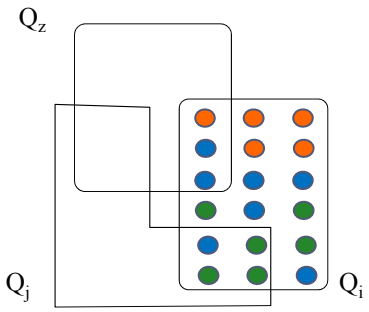
Server Removal



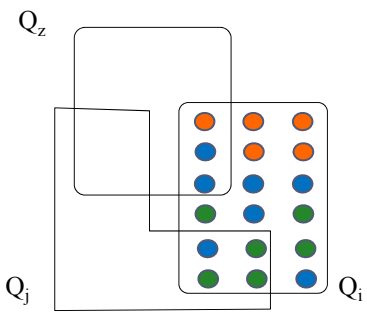
Past Prediction

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

What about writes?

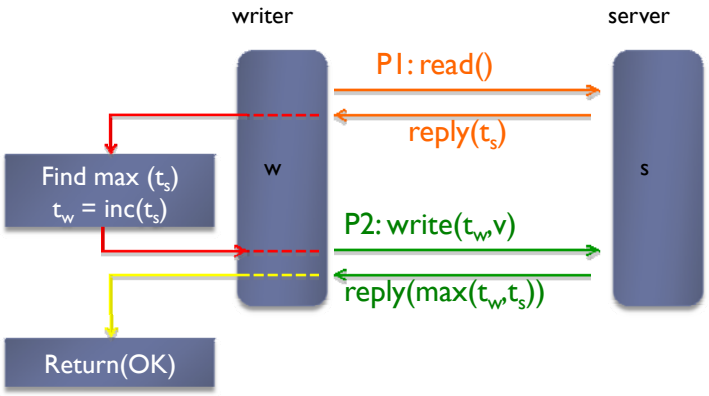
Can we devise MWMR atomic register implementations that allow executions that contain both fast read and write operations?

Answer: **YES!!**

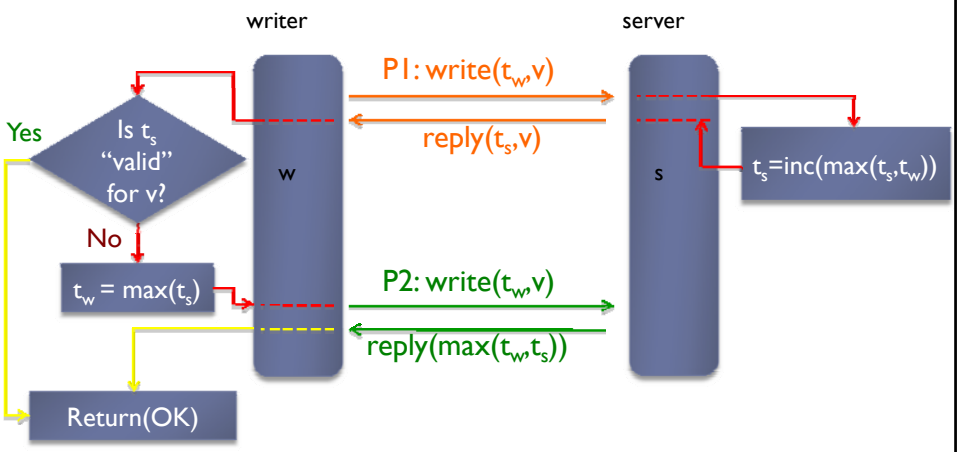
New Technique – SSO [Englert et. al OPODIS 2009]

- SSO: Server Side Ordering
  - Tag is incremented by the servers and not by the writer.
    - Generated tags may be different across servers
    - Clients decide operation ordering based on server responses
- SFW Algorithm
  - Deploys the SSO technique
  - Enables Fast Writes and Reads -- first such algorithm

Traditional Writer-Server Interaction



SSO Writer-Server Interaction



Definition: n-wise Quorum Systems

Definition: A quorum system Q is an *n*-wise quorum system, if:

$$\mathbf{Q} = \{Q : Q \subseteq S\} \; where \; \forall A \subseteq \mathbf{Q} : |A| = n \text{ and } \bigcap_{Q \in A} Q \neq \varnothing$$

Algorithm: SFW

Write Protocol: one or two rounds

- P1: send v and gather candidate tags from a quorum
  - Exists tag t propagated in a bigger than (n/2-1)-wise intersection (PREDICATE)
    - YES – assign t to the written value and return => **FAST**
    - NO - propagate the unique largest tag to a quorum => **SLOW**

Read Protocol: one or two rounds

- P1: collect list of writes and their tags from a quorum
  - Exists max write tag t in a bigger than (n/2-2)-wise intersection (PREDICATE)
    - YES – return the value written by that write => **FAST**
    - NO - propagate the largest confirmed tag to a quorum => **SLOW**

Server Protocol

- Increment tag when receive write request and record the latest writes
- Upon read/write request send the recording set

Predicates: Read and Write

Writer predicate for a write ω (PW): $\exists\ \tau, \mathbb{Q}^i, MS$ where: $\tau \in \{\langle ., \omega\rangle : \langle ., \omega\rangle \in m(\omega)_{s,w}.inprogress \ \wedge\ s \in Q\}$, $MS = \{s : s \in Q \ \wedge\ \tau \in m(\omega)_{s,w}.inprogress\}$, and $\mathbb{Q}^i \subseteq \mathbb{Q}, 0 \leq i \leq \lfloor \frac{n}{2} - 1 \rfloor$, s.t. $(\bigcap_{\mathcal{Q} \in \mathbb{Q}^i \cup \{Q\}} \mathcal{Q}) \subseteq MS$.

Reader predicate for a read ρ (PR): $\exists\ \tau, \mathbb{Q}^j, MS$, where: $\max(\tau) \in \bigcup_{s \in Q} m(\rho)_{s,r}.inprogress$, $MS = \{s : s \in Q \ \wedge\ \tau \in m(\rho)_{s,r}.inprogress\}$, and $\mathbb{Q}^j \subseteq \mathbb{Q}, 0 \leq j \leq \lfloor \frac{n}{2} - 2 \rfloor$, s.t. $(\bigcap_{\mathcal{Q} \in \mathbb{Q}^j \cup \{Q\}} \mathcal{Q}) \subseteq MS$.

Lower bound

Theorem: No execution of safe register implementation that use an N-wise quorum system, contains more than $N-1$ consecutive, quorum shifting, fast writes.

Remark

Remark: Algorithm SFW is near optimal since it allows up to $\left(\frac{N}{2}-1\right)$ consecutive, quorum shifting, fast writes.

Trade-offs in MWMR

Trades Write Speed for Fast Reads

- Quorum Views in the MWMR environment
- Algorithm CWFR
  - Traditional two round writes
  - Some single round reads – even when reads are concurrent with writes
  - Utilizes any General Quorum System

Trades Quorum Generality for Fast Writes

- Algorithm SFW
  - Server Side Ordering
  - Allows both single round reads and writes in MWMR
  - Fastness depends on n-wise quorum intersections
- n-1 consecutive fast writes are possible in MWMR
  - SFW near optimal – Allows n/2 consecutive fast writes

Recent Advancements

Problem: Communication vs Computation

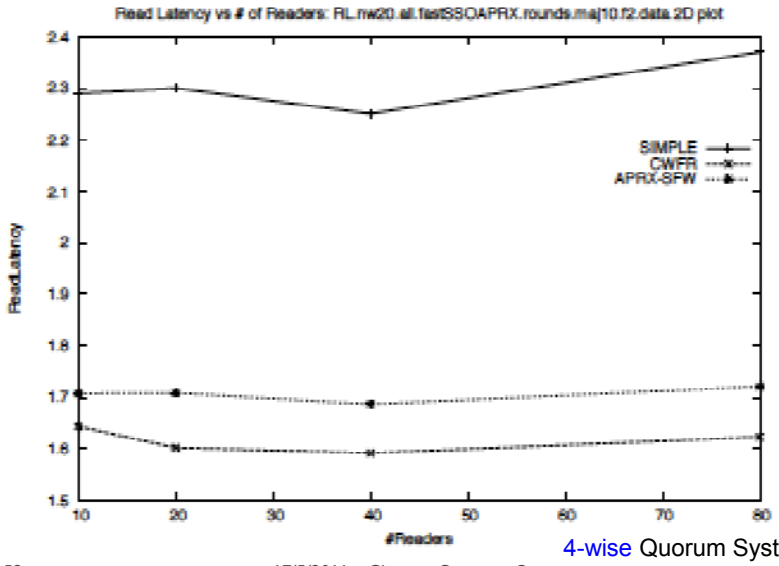
Theorem: The computation of the read and write predicates used by algorithm SWF is NP-complete.

We provide a reduction from 3-SAT

Approximation Algorithm: AprxSFW

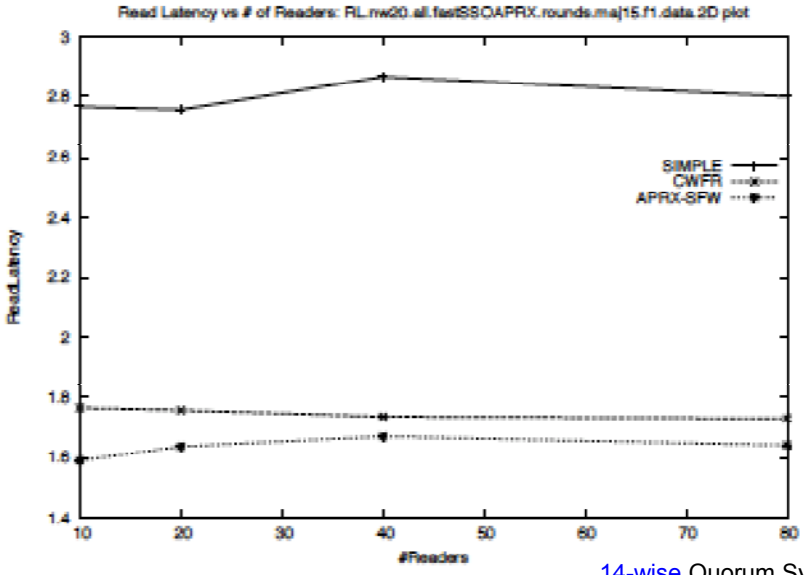
- Greedy algorithm
  - Uses Set Cover greedy approximation algorithm at its core for the predicate computation
- Log-Approximation
  - Invalidates RP and WP in a factor of log|S| times
  - A factor of log|S| more second communication rounds
  - Does not affect correctness
- Preliminary empirical evaluation using the NS2 simulator
  - Ongoing work: Evaluation on Planetlab

AprxSFW – CWFR Empirical Evaluation



4-wise Quorum System

AprxSFW – CWFR Empirical Evaluation



14-wise Quorum System

Gracias!