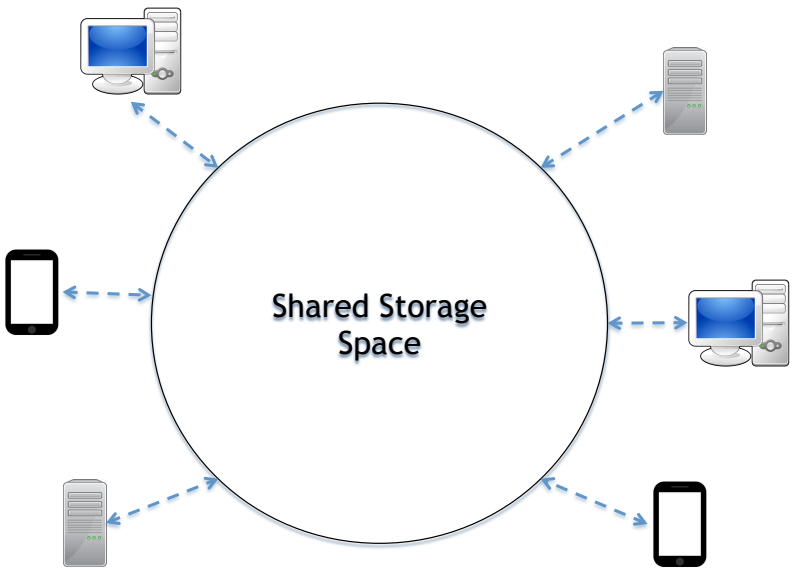
institute iMdea networks

Emulating Highly Consistent Shared Storage on top of Unreliable Message-Passing Nodes

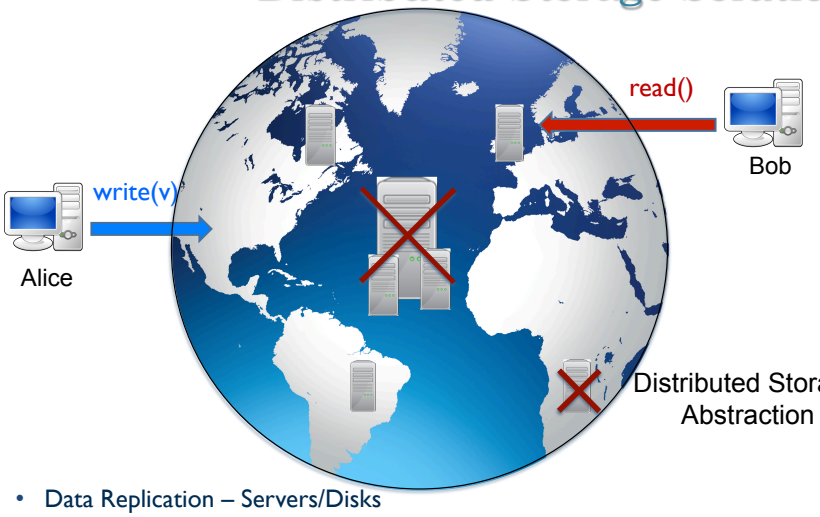
Nicolas Nicolaou

Developing the Science of Networks

Shared Storage Space



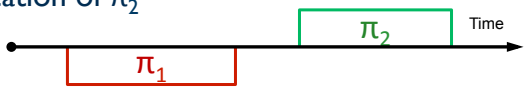
Distributed Storage Solution



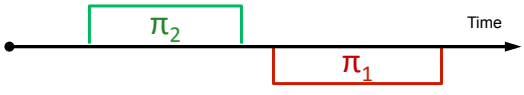
- Data Replication – Servers/Disks
  - Survivability and Availability
- Read/Write operations
- Consistency Semantics

Operation Relations

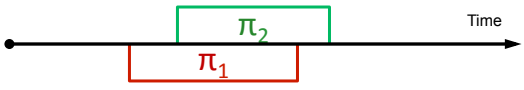
- Precedence Relations for two operations π_1 ,π_2:
  - π_1 precedes π_2 if the response of π_1 happens before the invocation of π_2



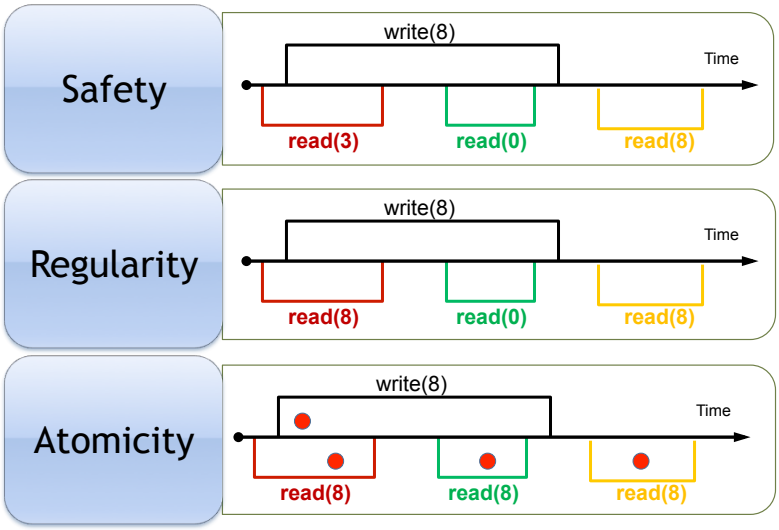
  - π_1 succeeds π_2 if the invocation of π_1 happens after the response of π_2



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



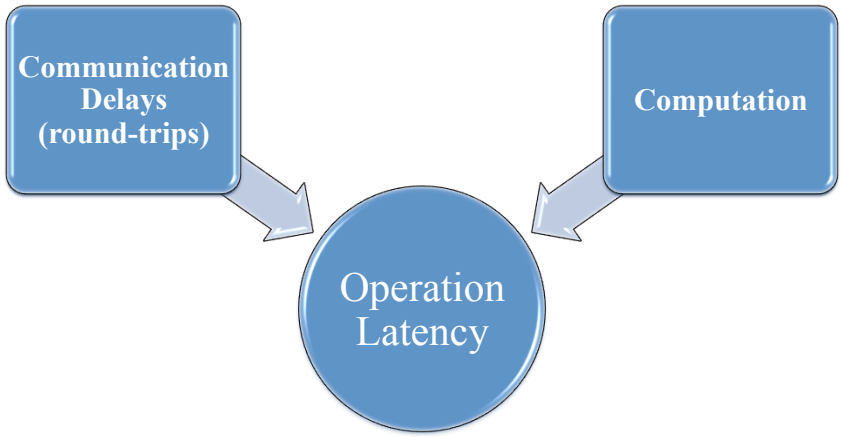
Consistency Semantics [Lamport86]



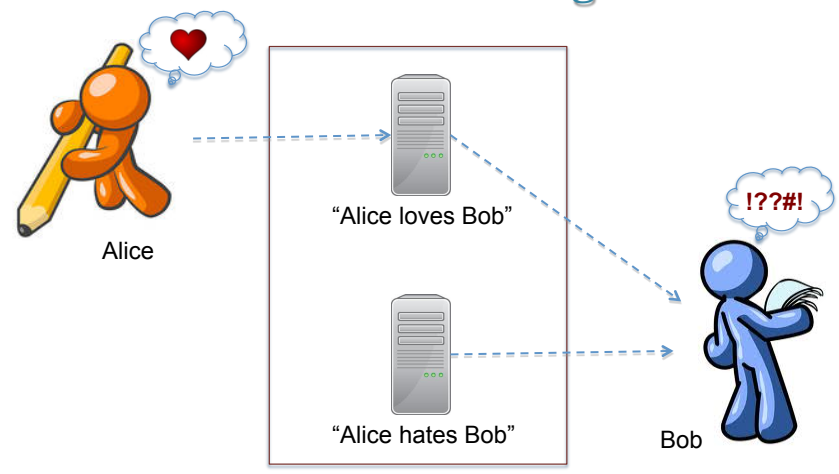
Model

- Asynchronous, Message-Passing model
  - Processes: writers W, readers R, servers S (replica hosts)
- Operations: read and write
- Shared Object: Read/Write Register
- Concurrency Environments:
  - SWMR: $|W|=1$, $|R|>1$
  - MWMR: $|W|>1$, $|R|>1$
- Failures: Crashes

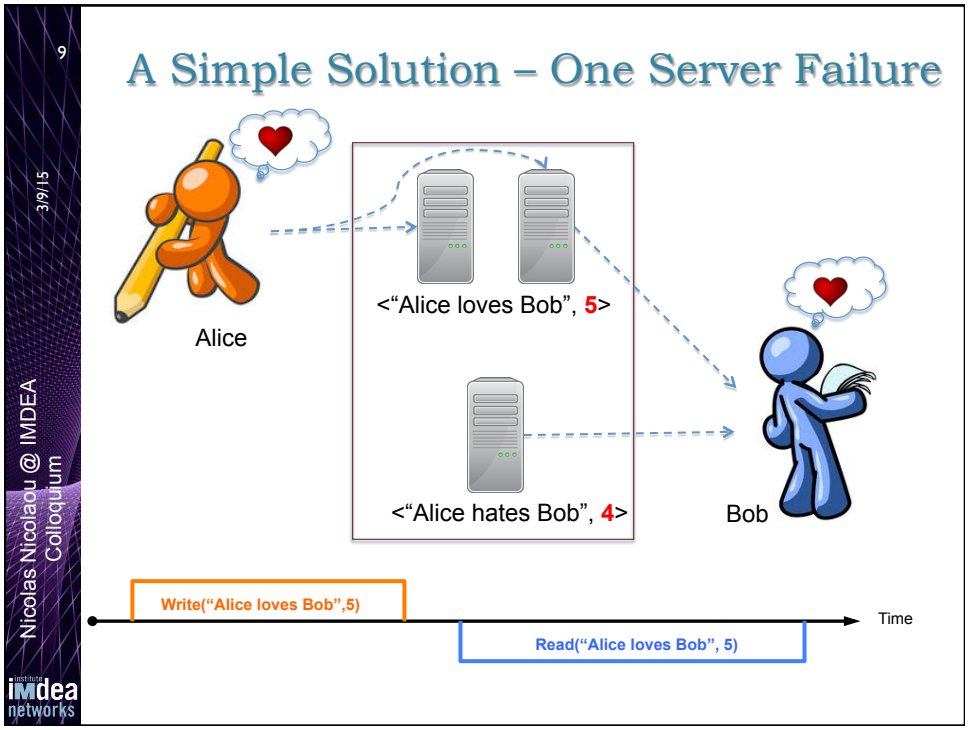
Complexity Measure

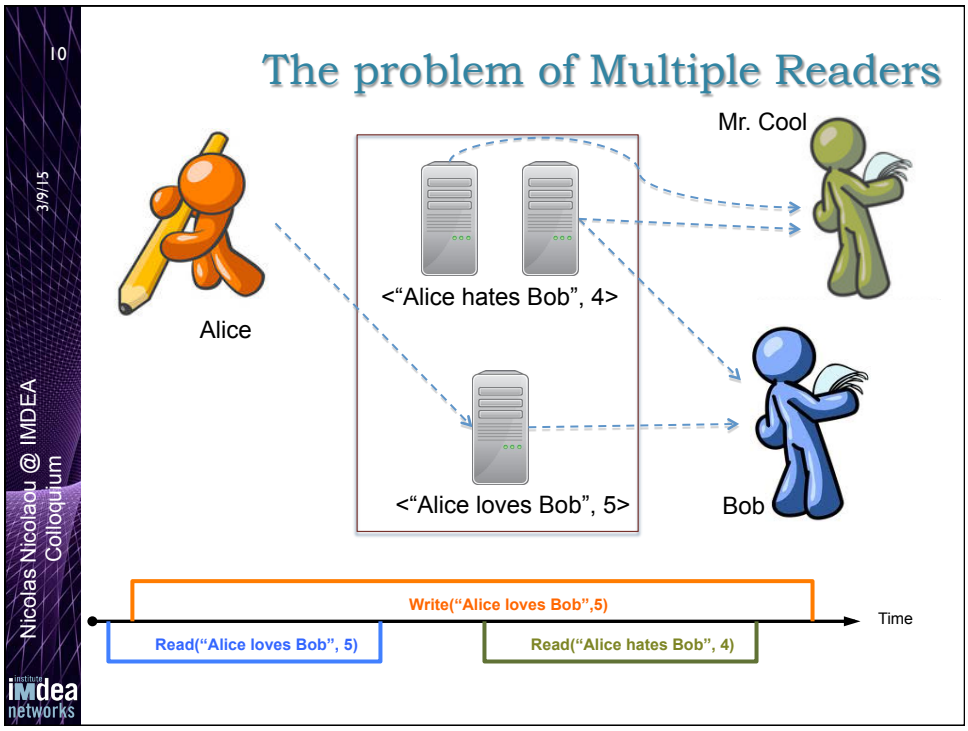


Understanding the Problem



Not Fault Tolerant and cannot distinguish the latest value


9
3/9/15
Nicolas Nicolaou @ IMDEA Colloquium
iMdea networks
A Simple Solution – One Server Failure
Alice
<"Alice loves Bob", 5>
<"Alice hates Bob", 4>
Bob
Write("Alice loves Bob",5)
Read("Alice loves Bob", 5)
Time


10
3/9/15
Nicolas Nicolaou @ IMDEA Colloquium
iMdea networks
The problem of Multiple Readers
Mr. Cool
Alice
<"Alice hates Bob", 4>
<"Alice loves Bob", 5>
Bob
Write("Alice loves Bob",5)
Read("Alice loves Bob", 5)
Read("Alice hates Bob", 4)
Time

[Attiya, Bar-Noy, Dolev '96]

Algorithm: ABD

Write Protocol: one phase

- P1: Increment your local timestamp, and send <ts, v> to a majority of the replicas

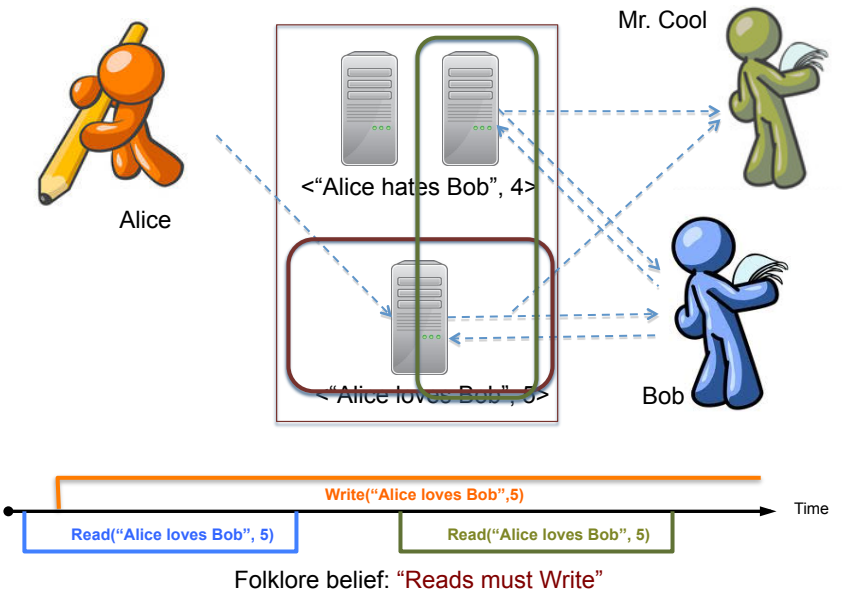
Read Protocol: two phases

- P1: query phase
  - Send read() to all the replicas
  - When a majority replies discover the largest <maxts,v>
- P2: propagation phase
  - Send write(<maxts,v>) to all replicas and wait for a majority to reply

Server Protocol: passive role

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

How it works in our example?



Folklore belief: "Reads must Write"

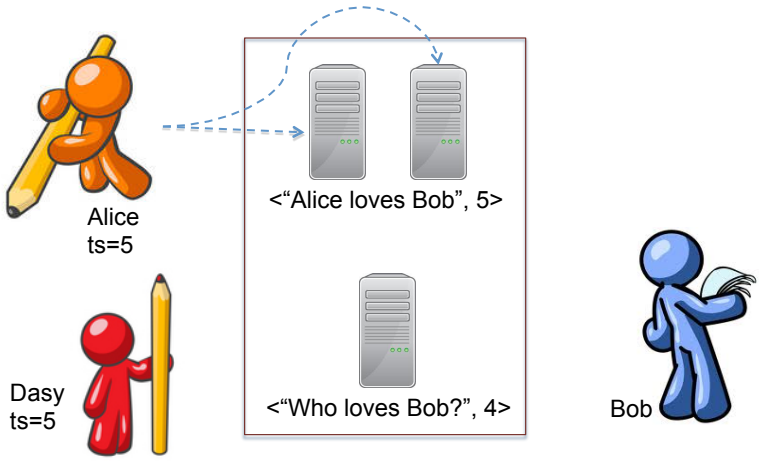
The Era of Fast Implementations….

SWMR Fast [Dutta et al. 2004]
- Single round (***fast***) writes and reads
- Bounded readers: $R<(S/f)-2$ where S servers & f failures
  - Impossible in MWMR model

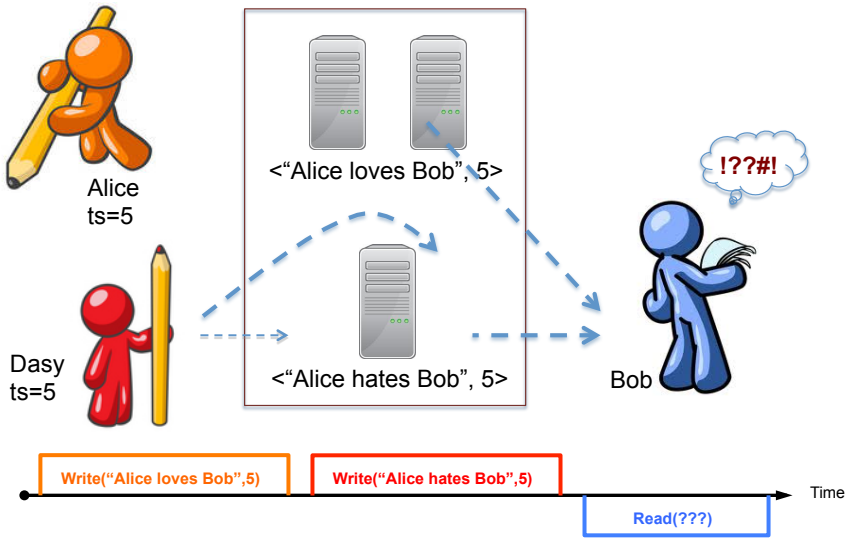
SWMR Semifast [Georgiou, Nicolaou, Shvartsman 2006]
- Single round writes
- Only a single complete 2-round (***slow***) read per write
  - Unbounded readers
  - Bounded Virtual Nodes: $V<(S/f)-2$
  - Impossible in the MWMR model

SWMR Weak-Semifast [Georgiou, Nicolaou, Shvartsman 2008]
- General Quorum System
- Fast writes and Multiple slow reads per write
- Allows concurrent fast reads with writes
- Unknown if applicable in MWMR model

What happens in the MWMR?



The problem of Multiple Writers



How to solve the problem?

- Writer needs to discover the latest timestamp in the system before incrementing it
- Each writer includes its id with the timestamp
  - Breaks the symmetry in case two writers read the same maximum timestamp
  - The <timestamp, id> pair is called TAG
  - tag1 > tag2 if either:
    - tag1.timestamp > tag2.timestamp, or
    - tag1.timestamp = tag2.timestamp AND tag1.id > tag2.id

[Lynch, Shvartsman 1997]

Algorithm: MWABD

Belief: "Writes must Read"

Write Protocol: two phases

- P1: Query the majority for the largest tag
- P2: Increment the largest tag, and propagate <newtag, v> to a majority

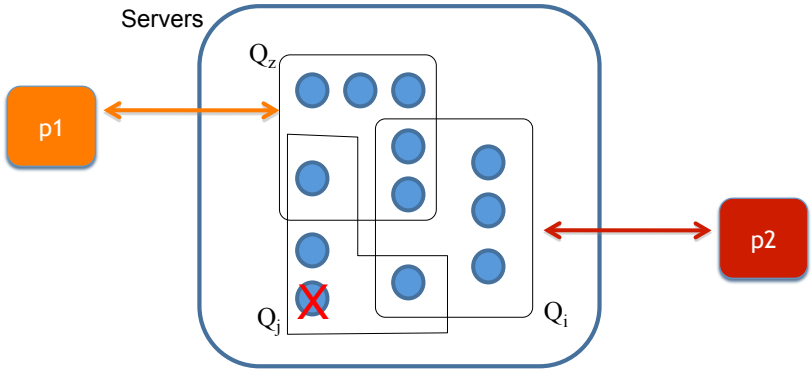
Read Protocol: two phases

- P1: Query the majority for the largest <maxtag, v>
- P2: Propagate (<maxtag, v>) to a majority

Server Protocol: passive role

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

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
  - **N-wise quorums systems:** every N quorums intersect for N>1
- Every R/W operation communicates with a single quorum
- Faulty Quorum: Contains a faulty process

[Englert, Georgiou, Musial, Nicolaou, Shvartsman 2009]

Are Fast Operations Possible in MWMR?

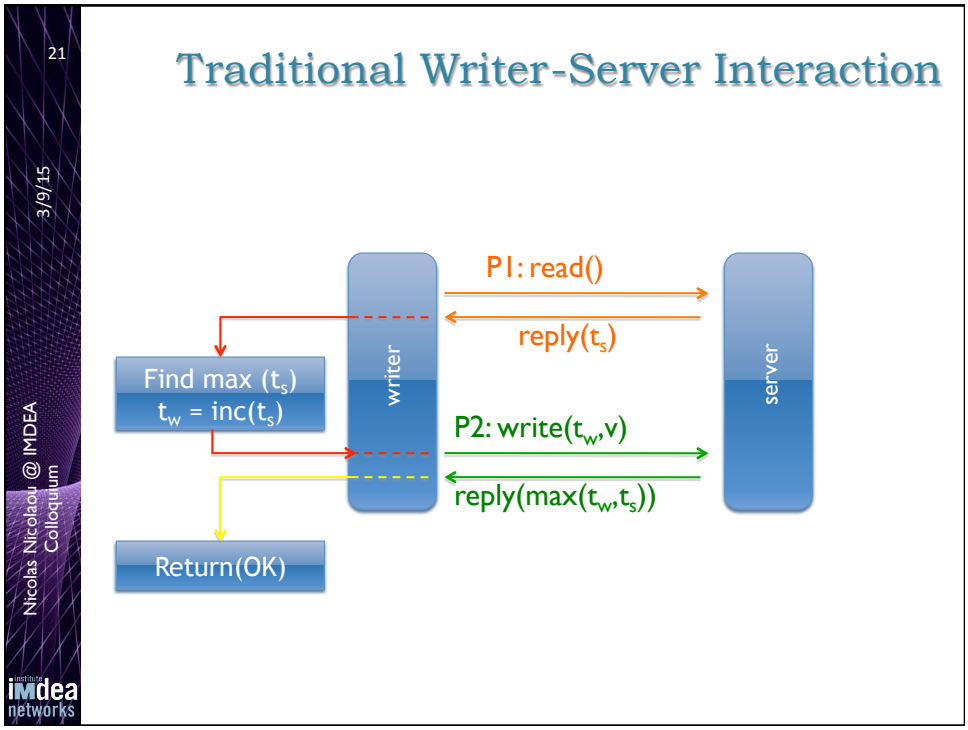
Theorem: No execution of safe register implementation that use an N-wise quorum system, contains more than $N-1$ consecutive, quorum switching, fast writes.

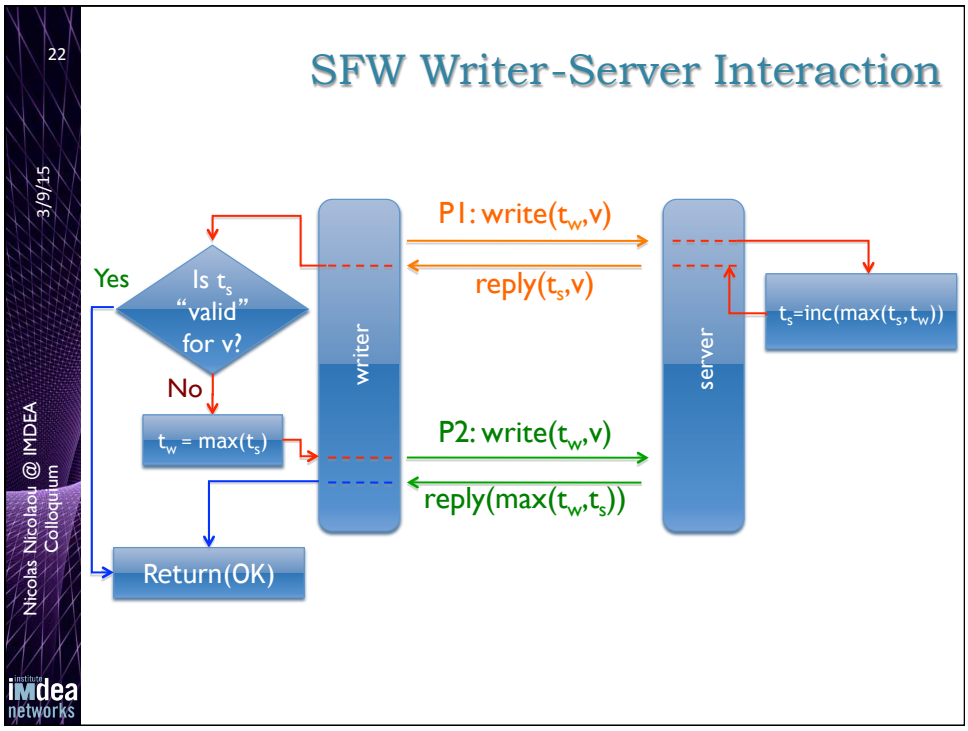
Theorem: It is impossible to obtain MWMR safe register implementations that exploit an N-wise quorum system, if

$$|W \cup R| > N-1$$

New Technique - SSO

- 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
  - Enables Fast Writes and Reads -- first such algorithm
  - Allows Unbounded Participation


21
3/9/15
Nicolas Nicolaou @ IMDEA Colloquium
institute IMdea networks
Traditional Writer-Server Interaction
P1: read()
reply(ts)
writer
server
Find max (ts)
tw = inc(ts)
P2: write(tw,v)
reply(max(tw,ts))
Return(OK)


22
3/9/15
Nicolas Nicolaou @ IMDEA Colloquium
institute IMdea networks
SFW Writer-Server Interaction
P1: write(tw,v)
reply(ts,v)
writer
server
Yes
Is ts "valid" for v?
ts=inc(max(ts,tw))
No
tw = max(ts)
P2: write(tw,v)
reply(max(tw,ts))
Return(OK)

[Englert, Georgiou, Musial, Nicolaou, Shvartsman 2009]

Algorithm: SFW (in a glance)

Write Protocol: one or two rounds

- P1: Collect candidate tags from a quorum
  - Exists tag t propagated in a bigger than (n/2-1)-wise intersection (PREDICATE PW)
    - 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 PR)
    - YES – return the value written by that write => **FAST**
    - NO – is there a confirmed tag propagated to (n-1)-wise intersection => **FAST**
    - NO - propagate the largest confirmed tag to a quorum => **SLOW**

Server Protocol

- Increment tag when receive write request and send to read/write the latest writes

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. $(\bigcap_{\mathcal{Q} \in \mathbb{Q}^j \cup \{Q\}} \mathcal{Q}) \subseteq MS$.

The Weak Side of SFW

- Predicates are Computationally Hard
  - NP-Complete
- Restriction on the Quorum System
  - Deploys N-wise Quorum Systems
  - Guarantees fastness iff n>3

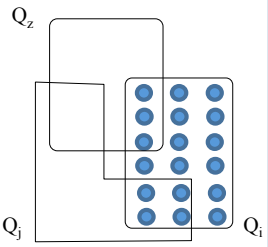
The Good News…

- Algorithm CWFR
  - Based on Quorum Views
    - SWMR prediction tools
  - Fast operations in General Quorum Systems
  - Trades Speed of Write operations
    - Two Round Writes

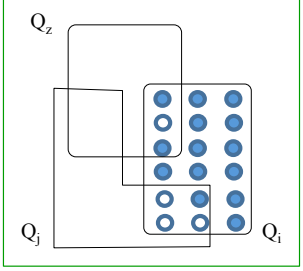
[Georgiou, Nicolaou, Shvartsman 2008]

Quorum Views

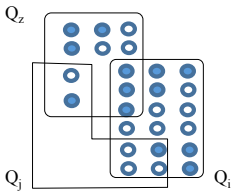
- Idea: Try to determine the status of the write operation based on the distribution of the max timestamp in the replied quorum.

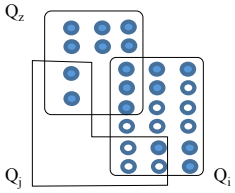


Qview(1): (Potentially) Write Completed



Qview(2): (Definitely) Write Incomplete

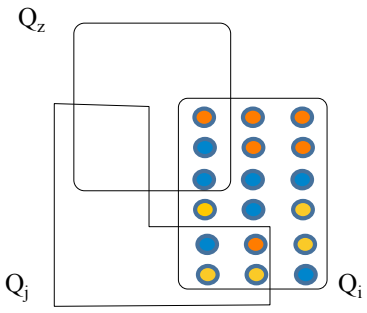




Qview(3): Undeterminable

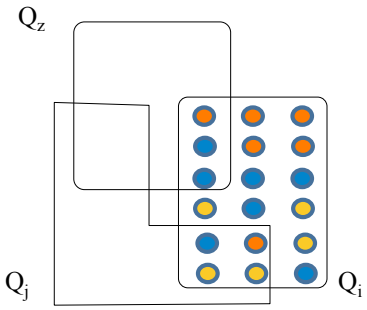
What happens in MWMR?

- 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



[Georgiou, Nicolaou, Russell, Shvartsman 2012]

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

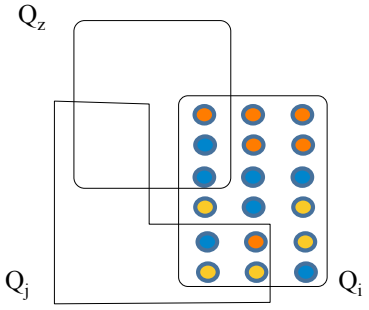
- Iterate to discover smallest completed write
- P1: receive replies from a quorum Q
  - $QView_Q(1)$ - **Fast**: return maxTag of current iteration
  - $QView_Q(2)$ - remove servers with maxTag and re-evaluate
  - $QView_Q(3)$ - **Slow**: propagate and return $maxTag_0$

Server Protocol: passive role

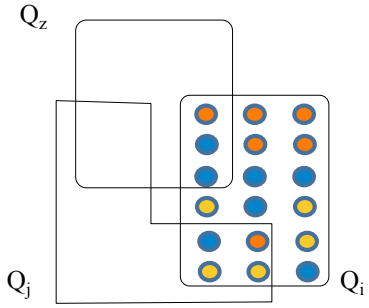
- Receive requests, update local timestamp and return <tag,v>

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>



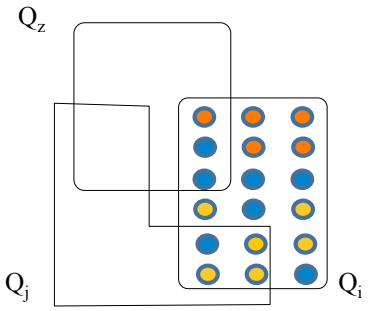
Server Removal



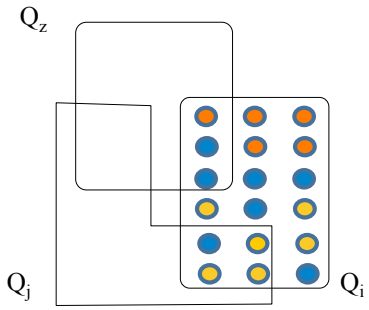
Past Prediction

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

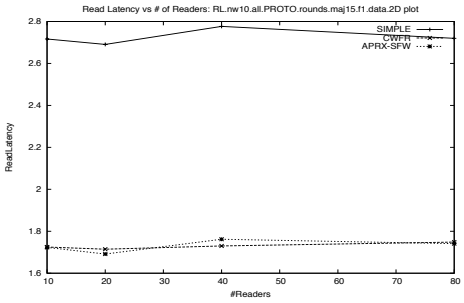


Past Prediction

Empirical Evaluation

- NS2: Single Processor Emulator
  - Fully Controlled Environment
  - Tested the efficiency of the algorithms under predefined environmental parameters
- Planet-Lab: Planetary Scale Real Time Platform
  - Real Time Network Environment (INTERNET)
    - Machines Disseminated Throughout the Globe
  - Tested practicality of the algorithms

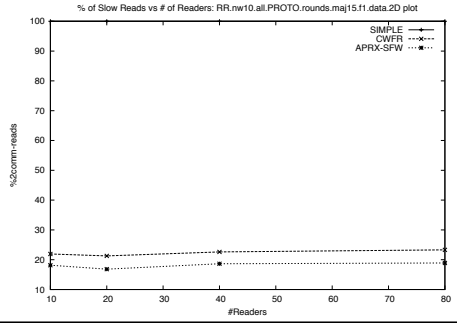
NS2: SIMPLE, APRX-SFW, CWFR



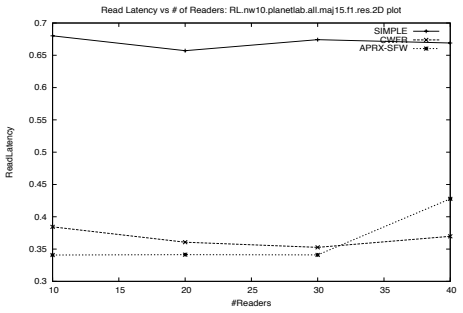
Latency

14-wise Quorum System

Rounds



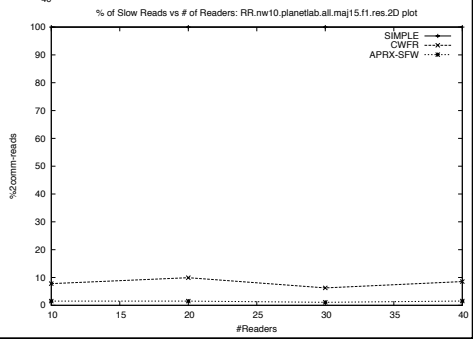
Planetlab: SIMPLE, APRX-SFW, CWFR



Latency

14-wise Quorum System

Rounds



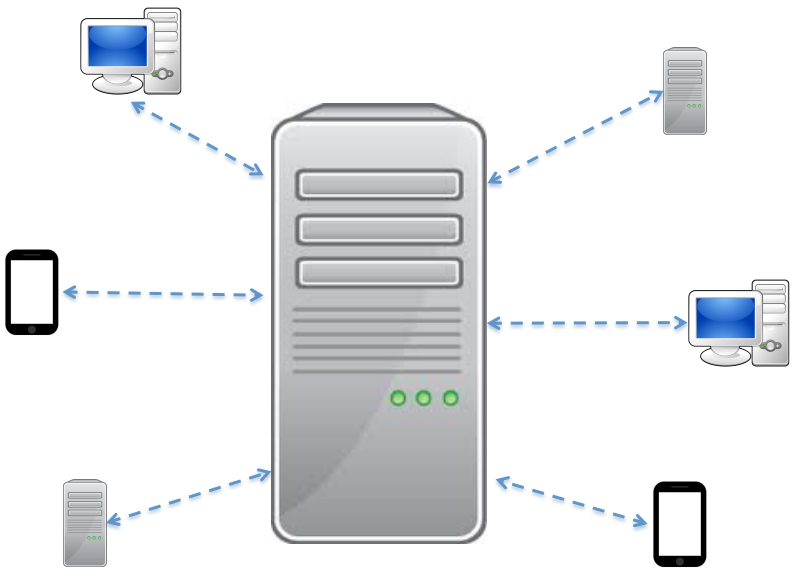
Survivability of Data

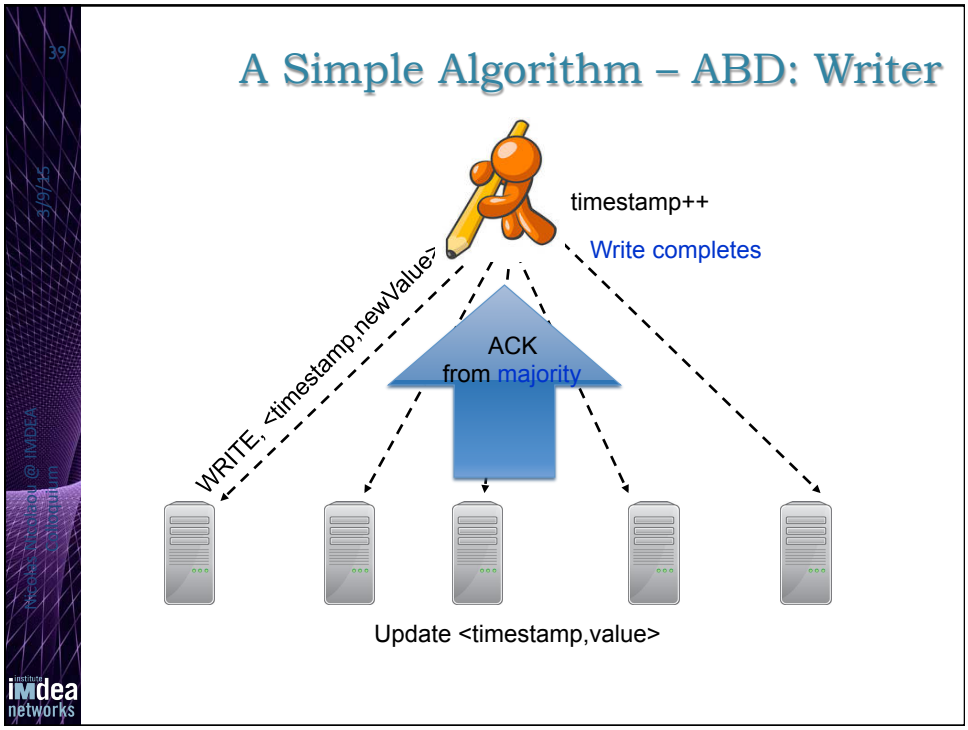
- Data Survivability is essential in today's systems and applications
- Popular approach: Redundancy
  - Use of a Redundant Array of Inexpensive Disks (RAID)
  - A RAID system contains
    - A single box
    - Residing on a single physical location
    - Servers are connected to clients via a single network interface

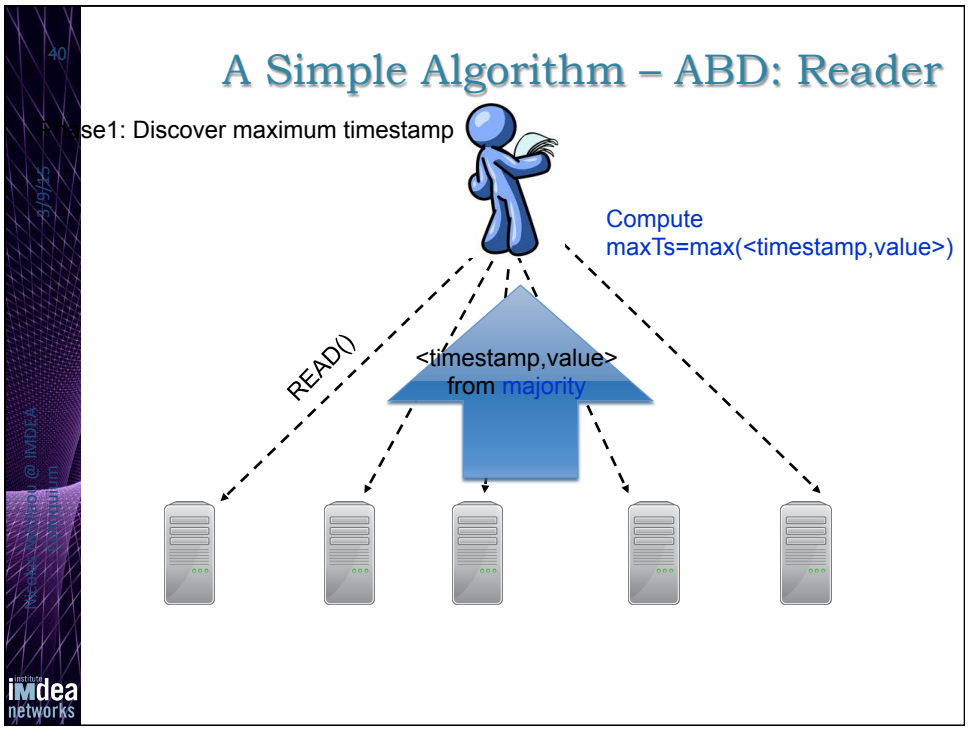
Single point of failure

*Use **Distributed Storage Systems** instead*

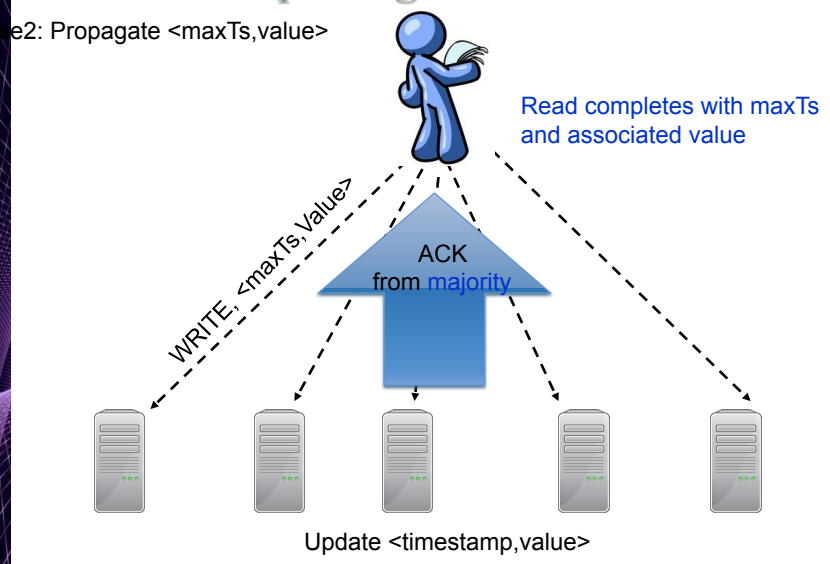
Centralized Solution??




39
A Simple Algorithm – ABD: Writer
timestamp++
Write completes
WRITE, <timestamp,newValue>
ACK
from majority
Update <timestamp,value>
iMdea networks


40
A Simple Algorithm – ABD: Reader
se1: Discover maximum timestamp
Compute
maxTs=max(<timestamp,value>)
READ()
<timestamp,value>
from majority
iMdea networks

A Simple Algorithm – ABD: Reader



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