Distributed Systems
Consistency & Replication

Alberto Montresor

University of Trento, Italy

November 12, 2012

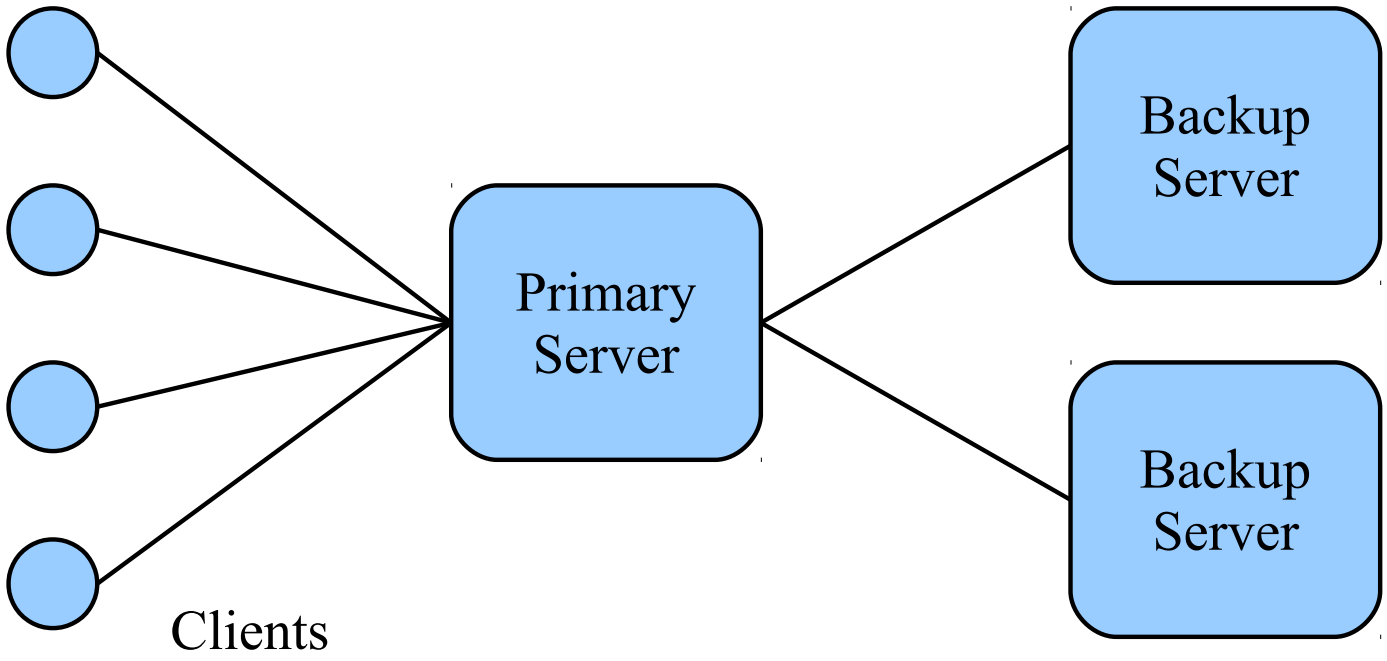
Redundancy is our main avenue of survival
Robert Silverberg, "Shadrach in the furnace"

Contents

1. Replication architectures
   - Overview
   - Primary-Backup
   - Quorum protocols
   - State machines
   - Client-centric consistency
2. CAP Theorem
3. Bibliography

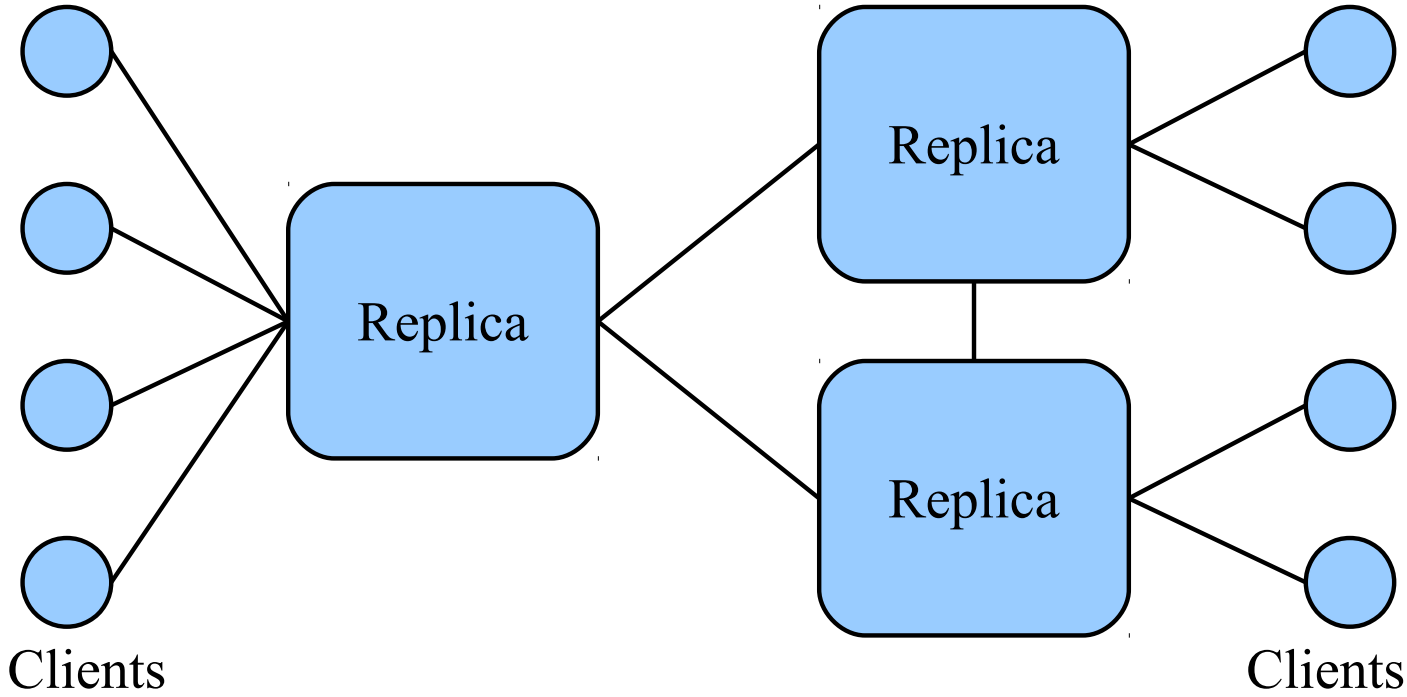
Passive replication

- Clients communicate with primary server
- Updates are forwarded from primary to backups
- Queries are replied by the primary



Active replication

- Several (all) replicas handle the invocation and send the response
- Updates must be applied in the same order – total order broadcast



Passive vs Active

Passive replication

- Computation is performed only at primary
- If state updates are large, can waste network bandwidth
- Can handle non-determinism

Active replication

- Small recovery delay after failures
- If operations are compute intensive, can waste computational resources
- Only deterministic

Consistency protocols

- Primary-based protocols
  - Definition
  - Lower bounds
- Replicated-write protocols
  - Majority, quorum-based
  - State machine approach
- Client-centric protocols
  - Monotonic reads
  - Read-your-writes

Primary-Backup

The idea

- Clients communicate with a single replica (the primary)
- The primary updates the other replicas (backup)
- Backups detect the failure of the primary using a timeout mechanism
- Clients learn from the service when the primary fails and the service "fail over" to a backup
- Note: non-deterministic events are executed only at the primary

How to evaluate a primary-backup protocol

Definition (Degree of replication)

Number of servers used to implement the service; the smaller, the better

Definition (Blocking time)

The worst-case period between a request and its response in any failure-free execution

Definition (Failover time)

The worst-case period during which request can be lost because there is no primary

Definitions

Definition (Service outage)

The service has a server outage at t if some correct client sends a request at time t to the service, but does not receive a response

Definition ((k, Δ)-bofo service - "bounded outage, finitely often")

A service in which all server outages can be grouped into at most k intervals of time, each of them at most length Δ

Specification

PB1 At any time, there is at most one server p_i that acts as a primary

PB2 If a client request arrives at a server that is not the current primary, then the request is ignored

PB3 There exist fixed values k and Δ such that the service behaves like a single (k, Δ)-bofo service

Primary-backup – Simple protocol

System model:

- point-to-point communication
- no communication failures
- upper bound δ on message delivery time
- FIFO channels
- at most one server crashes

Two servers:

- The primary p_1
- The backup p_2

Variables:

- At server p_i, $primary = \textbf{true}$ if p_i acts as the current primary
- At clients, $primary$ is equal to the identifier of the current primary

Primary-backup – Simple protocol

Protocol executed by the primary p_1

upon initialization **do**
  $primary \leftarrow$ **true**

upon receive $\langle \text{REQ}, r \rangle$ **from** c **do**
  $state \leftarrow \mathsf{update}(state, r)$ % Update local state
  send $\langle \text{STATE}, state \rangle$ **to** p_2 % Send update to backup
  send $\langle \text{REP}, \mathsf{reply}(r) \rangle$ **to** c % Reply to client

repeat every τ seconds
  send $\langle \text{HB} \rangle$ **to** p_2 % Heartbeat message

upon recovery after a failure **do**
  { start behaving like a backup }

Primary-backup – Simple protocol

Protocol executed by the backup p_2

upon initialization **do**
  $primary \leftarrow$ **false**

upon receive $\langle \text{STATE}, s \rangle$ **do**
  $state \leftarrow s$ % Update local state

upon not receiving a heartbeat for $\tau + \delta$ seconds **do**
  $primary \leftarrow$ **true** % Becomes new primary
  send $\langle \text{NEWP} \rangle$ **to** c % Inform the client of new primary
  { start behaving like a primary }

Primary-backup – Client code

Protocol executed by client c

upon initialization **do**
  $primary \leftarrow p_1$ % Initial primary

upon receive $\langle \text{NEWP} \rangle$ **from** p_2 **do**
  $primary \leftarrow p_2$ % Backup

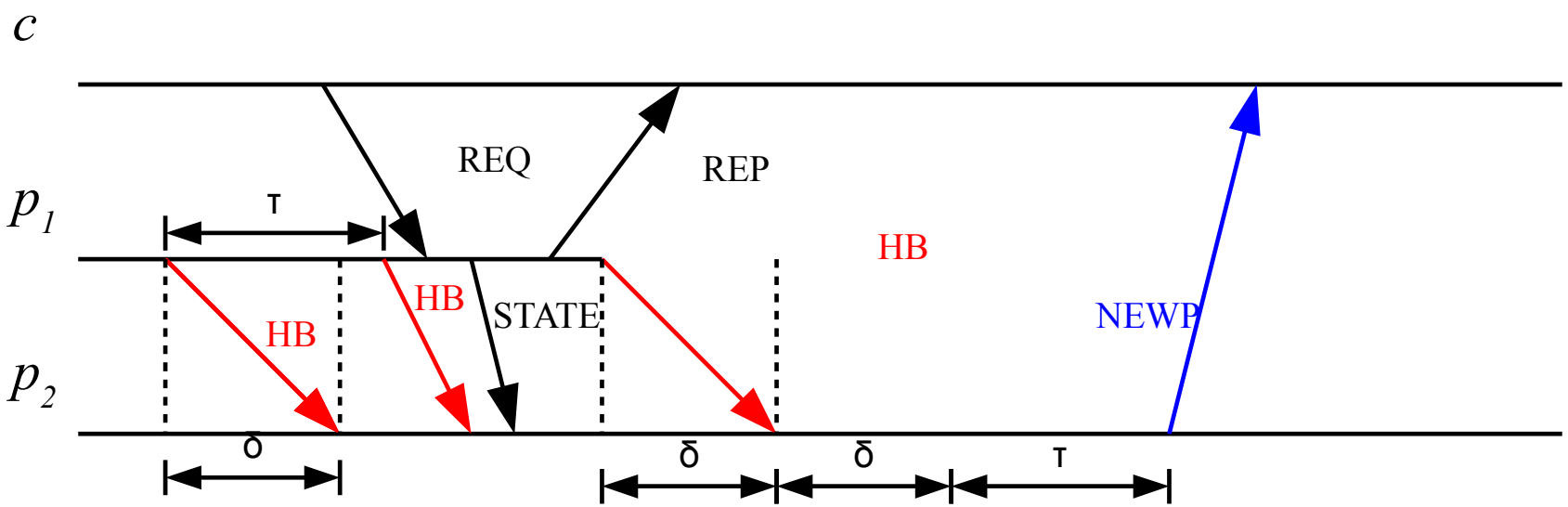
upon operation(r) **do**
  while not received a reply **do**
    send $\langle \text{REQ}, r \rangle$ **to** *primary*
    wait receive $\langle \text{REP}, v \rangle$ **or receive** $\langle \text{NEWP} \rangle$
  return v

Simple protocol – Proof of correctness

PB1 At any time, there is at most one server p_i that acts as a primary

Proof

- $primary_1 = \textbf{true} \wedge primary_2 = \textbf{false}$ until the failure of p_1
- $primary_2 = \textbf{false}$ until the expiration of the timeout
- $primary_2 = \textbf{true}$ after the expiration of the timeout
- Failover time: $\tau + 2\delta$



Simple protocol – Proof of correctness

PB2 If a client request arrives at a server that is not the current primary, then the request is ignored

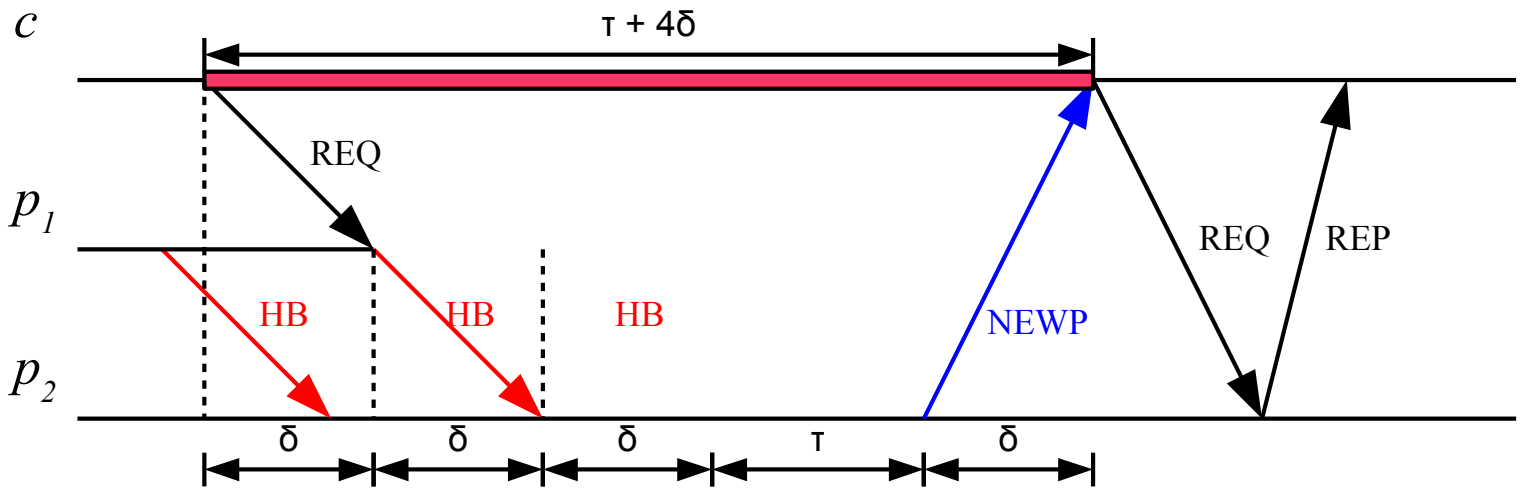
Proof Trivially follows from the protocol

Simple protocol – Proof of correctness

PB3 There exist fixed values k and Δ such that the service behaves like a single (k, Δ)-bofo service

Proof Find k, Δ

- At most one process can fail: $k = 1$
- $\Delta = \tau + 4\delta$:
  - ★ assume p_1 crashes at t_c
  - ★ any client request sent to p_1 at time $t_c - \delta$ or later may be lost
  - ★ p_2 may not become the new primary until $t_c + \tau + 2\delta$
  - ★ client may not learn that p_2 is new primary for another δ



Simple protocol – Questions

Question

What kind of consistency model is provided by this simple protocol?

Simple protocol – Questions

Question

What kind of consistency model is provided by this simple protocol?

Answer: Linearizability

An execution E is linearizable provided that there exists a sequence (linearization) H such that

- L1 H contains exactly the same operations that occur in E, each paired with the return value received in E
- L2 H is a legal history of the sequential data type that is replicated
- L3 the total order of operations in H is compatible with the real-time partial order $<_{E,rt}$

Primary-backup – Multiple backups

System model:

- point-to-point communication
- Perfect Channels
- perfect failure detector P
- FIFO channels
- at most $f < n$ servers crash

n servers:

- $p_1, \ldots, p_n$

Primary-backup – Multiple backups

Protocol executed by process p_i

upon receive $\langle \text{REQ}, id, r \rangle$ **from** c **do**
- $servers \leftarrow servers - \{p_j : p_j \in servers \wedge j < i\}$
- **if** $id \notin state$ **then**
  - $state \leftarrow \mathsf{update}(state, r)$
- **send** $\langle \text{STATE}, state, id \rangle$ **to** $servers$
- **wait receive**$\langle \text{STATE}, id \rangle$ **from** $servers$
- **send** $\langle \text{REP}, id, \mathsf{reply}(r) \rangle$ **to** c

upon $\mathsf{suspect}(p_j)$ **do**
- $servers \leftarrow servers - \{p_j\}$

upon receive $\langle \text{STATE}, id, s \rangle$ **from** p_k **do**
- $servers \leftarrow servers - \{p_j : p_j \in servers \wedge j < k\}$
- **if** $p_k \in servers$ **then**
  - $state \leftarrow s$
  - **send** $\langle \text{STATE}, id \rangle$ **to** p_k

Primary-Backup – Client code

Protocol executed by client c

upon initialization **do**
  MAP *response* ← **new** MAP();

upon receive ⟨REP, *id*, *v*⟩ **do**
  response[*id*] ← *v*

upon suspect(p_j) **do**
  servers ← *servers* − $\{p_j\}$

upon operation(r) **do**
  id ← newId()
  while *servers* ≠ ∅ **or** *response*[*id*] = **nil do**
    p_k ← min(*servers*)
    send ⟨REQ, *id*, *r*⟩ **to** p_k
    wait *response*[*id*] ≠ **nil or** $p_k \notin$ *servers*
  return *response*[*id*]

Primary-backup – Multiple backups

- How large is the failover time?

Primary-backup – Multiple backups

- How large is the failover time?
  $\tau + 2\delta$, as before (hidden in the Failure Detector)
- How large is the outage period Δ?

Primary-backup – Multiple backups

- How large is the failover time?
  $\tau + 2\delta$, as before (hidden in the Failure Detector)
- How large is the outage period Δ?
  $(\tau + 2\delta)(n - 1)$
- What kind of consistency model we obtain if all operations are handled by the primary?

Primary-backup – Multiple backups

- How large is the failover time?
  $\tau + 2\delta$, as before (hidden in the Failure Detector)
- How large is the outage period Δ?
  $(\tau + 2\delta)(n - 1)$
- What kind of consistency model we obtain if all operations are handled by the primary?
  Linearizability
- What kind of consistency model we obtain if only write operations are handled by the primary?

Primary-backup – Multiple backups

- How large is the failover time?
  $\tau + 2\delta$, as before (hidden in the Failure Detector)
- How large is the outage period Δ?
  $(\tau + 2\delta)(n - 1)$
- What kind of consistency model we obtain if all operations are handled by the primary?
  Linearizability
- What kind of consistency model we obtain if only write operations are handled by the primary?
  Sequential consistency

Lower bounds

- Assuming that no more than f components can fail, what are the smallest possible values (lower bounds) of
  - the degree of replication
  - the failover time?
  - the blocking time
- Knowing the lower bounds for a problem enables to evaluate the quality of a protocol
- Tight lower bounds $\rightarrow$ optimal protocols
- Components:
  - Processes
  - Point-to-point links
  - Up to f crash+link failures $\rightarrow$ at most f processes may crash or f links may crash or f_1 links + f_2 processes = f components

Lower bounds

Failure Model	Degree of Replication	Blocking Time	Failover Time
crash	$n > f$	0	$f\delta$
crash+link	$n > f + 1$	0	$2f\delta$
rec-omission	$\left\lfloor \frac{3f}{2} \right\rfloor$	2δ	$2f\delta$
send-omission	$n > f$	2δ	$2f\delta$
omission	$n > 2f$	2δ	$2f\delta$

Lower bounds

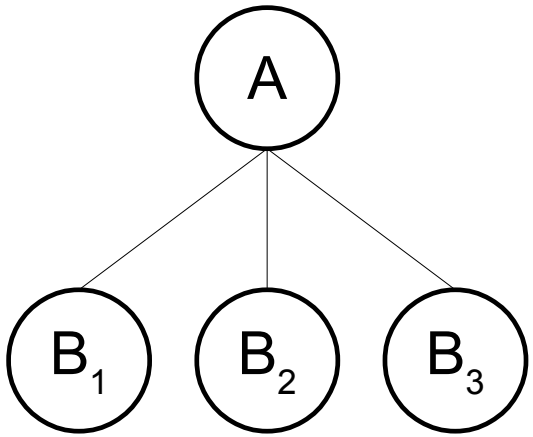
Crash+link

To tolerate up to f crash+link failures, more than $f + 1$ servers are needed

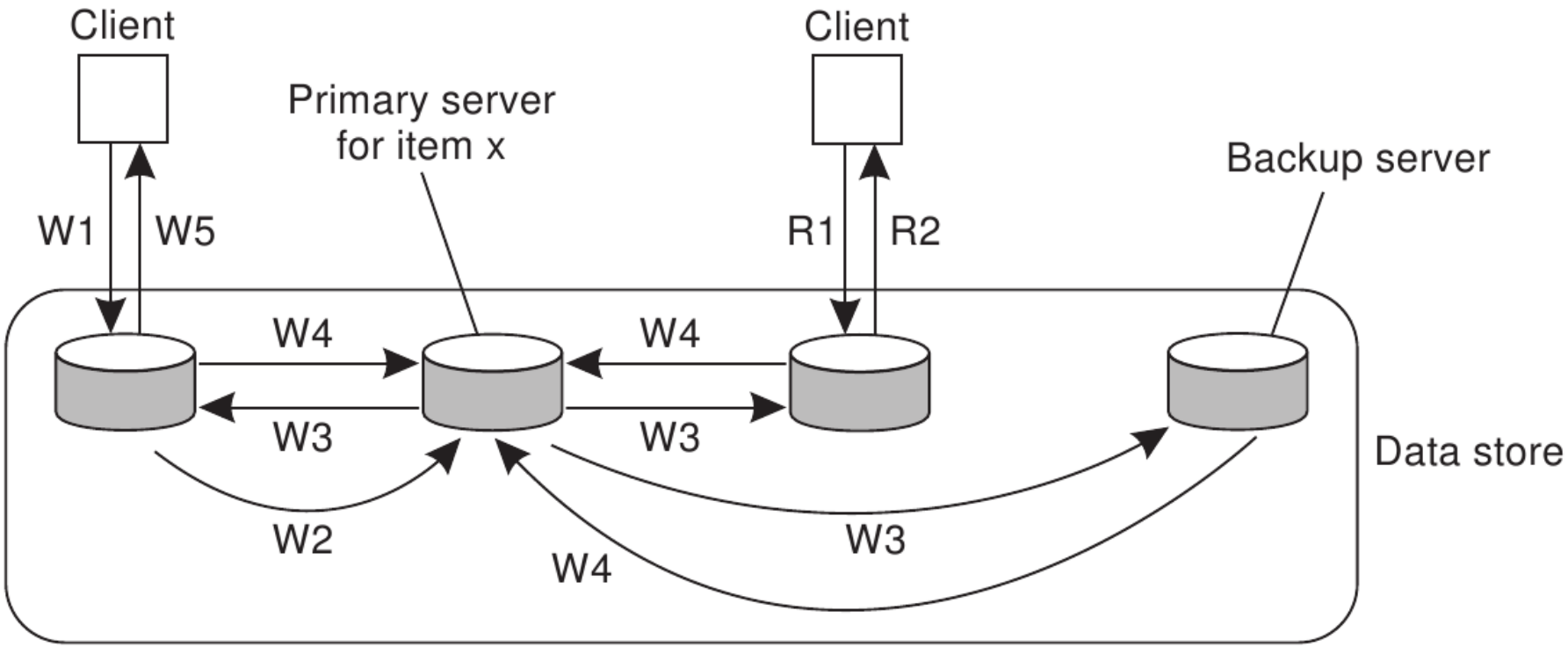
Proof – by contradiction

Suppose $n = f + 1$ servers is sufficient

- divide the n servers in two subsets A and $B_1 \dots B_f$
- if all server in B crash, A must become primary
- if A crashes, one of servers B_i must become primary
- what if all f links between A and B_i fails?



Multiple primaries



W1. Write request
W2. Forward request to primary
W3. Tell backups to update
W4. Acknowledge update
W5. Acknowledge write completed

R1. Read request
R2. Response to read

Quorum protocols (Gifford, 1979)

Definition

Quorum-based protocols guarantee that each operation is carried out in such a way that a majority vote (a quorum) is established.

- Write quorum n_W: the number of replicas that need to acknowledge the receipt of the update to complete the update
- Read quorum n_R: the number of replicas that are contacted when a data object is accessed through a read operation

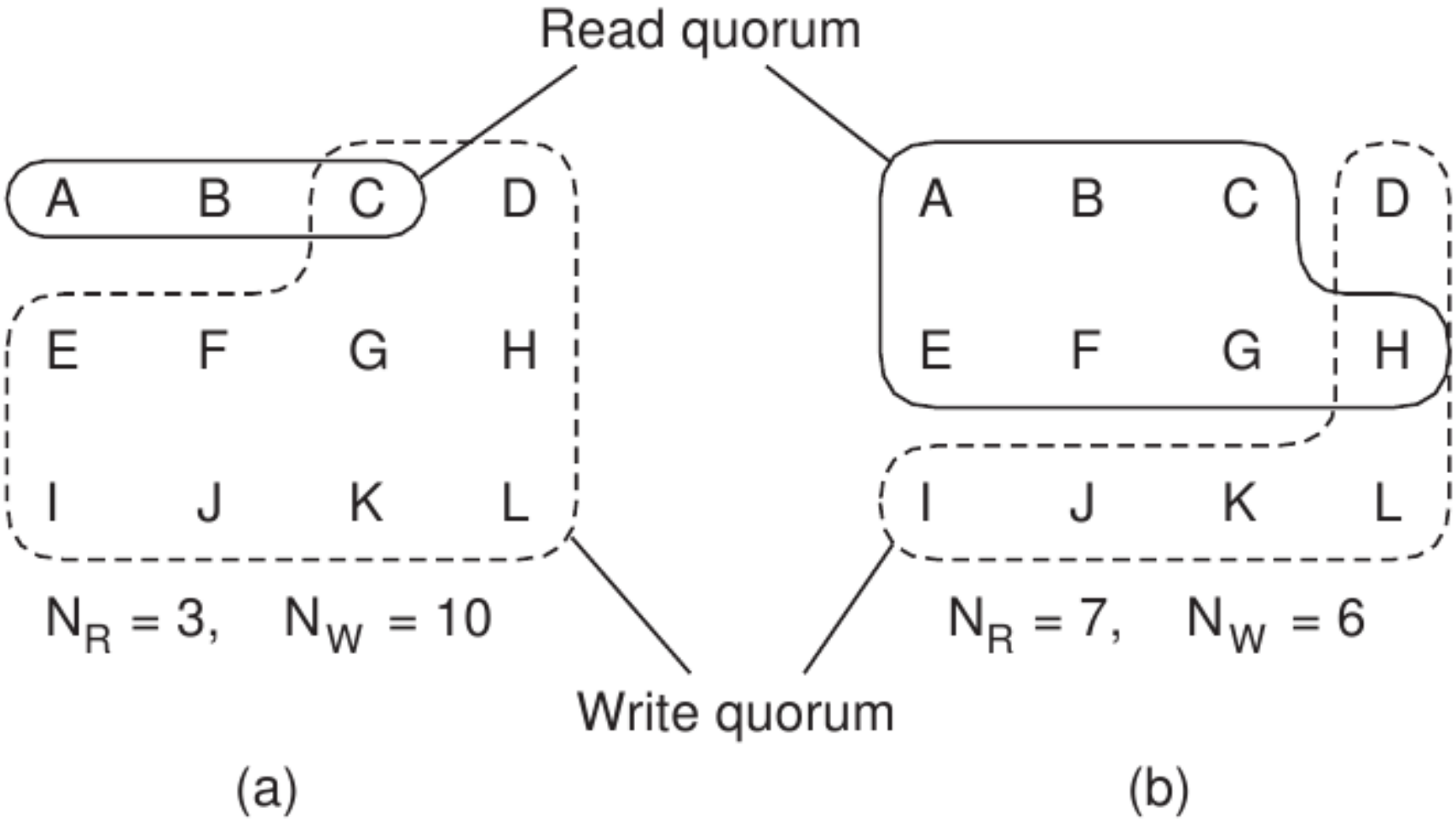
Constraints

- $n_R + n_W > n$ (prevent R-W conflicts)
- $n_W > n/2$ (prevent W-W conflicts)

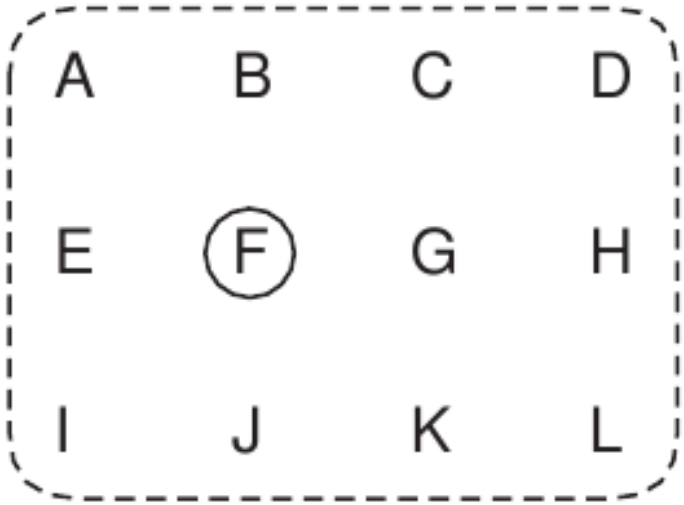
The algorithm

- To read, the most up-to-date entry is taken
- Quorums guarantee that the last written entry will be present

Quorum protocols (Gifford, 1979)



Quorum protocols (Gifford, 1979)



State machine

Definition (State machine)

A state machine consists of:

- State variables
- Commands which transforms its state
  - Implemented by deterministic programs
  - Atomic with respect to other commands

Specification

- Agreement: every correct replica receives the same set of commands
- Order: every non-faulty state machine processes the commands it receives in the same order

Implementing linearizability – General scheme

Implementation

- The initiator A-broadcasts all read, write requests to all servers
- When the message is A-delivered at the initiator, it replies to the client

Correctness

- All replicas execute read, write in the same order

Assumptions

- Synchronous system
- Asynchronous system with $\diamond S$ failure detector

Implementing sequential consistency – General scheme

Implementation

- The initiator A-broadcasts write requests to all servers
- When the message is A-delivered, the replica updates its local copy
- Read request are replied immediately by the initiator

Correctness

- Writes are executed in the same order everywhere
- Reads are consistent with local order

Assumptions

- Synchronous system
- Asynchronous system with $\diamond S$ failure detector

Implementing causal consistency – General scheme

Implementation

- The initiator C-broadcasts write requests to all servers
- When the message is C-delivered, the replica updates its local copy
- Read request are replied immediately by the initiator

Correctness

- Writes are executed in a causal order
- Reads are consistent with local (and causal) order

Assumptions

- Asynchronous system

Hypervisor-based fault tolerance

- Implement state machine on virtual machines running on the same instruction-set as underlying hardware
- Undetectable by higher layers of software
- One of the great come-backs in systems research!
  - CP-67 for IBM 369 [1970]
  - Xen [SOSP 2003], VMware
- State transition should be deterministic
- ...but some VM instructions are not (e.g. time-of-day)!
- Two types of commands
  - Virtual-machine instructions
  - Virtual-machine interrupts (with DMA input)
    Interrupts must be delivered at the same point in cmd sequence

Hypervisor-based fault tolerance

- Thomas C. Bressoud, Fred B. Schneider. Hypervisor-based Fault Tolerance. ACM TOCS, 14(1):80-107
- John R. Douceur an Jon Howell. Replicated Virtual Machines. Microsoft Research TR-2005-119
  - Technical paper associated to a patent
- Brendan Cully et al. Remus: High Availability via Asynchronous Virtual Machine Replication. NSDI'08.
  - Best paper award
  - Real implementation for XEN

Client-centric consistency - Naive implementations

- Each write operation is assigned a unique identifier
  - Done by the server where the operation is requested
- For each client c, we keep track of:
  - Read set WS_r: contains write operations relevant to the read operations performed by c
  - Write set WS_w: contains write operations relevant to the write operations performed by c
- For each server, we keep track of:
  - Write set WS: contains the write operations executed so far

Monotonic reads - Naive implementation

- To perform a read operation o_r, a client:
  - send o_r and its read set WS_r to the server
- The server
  - Checks whether all the writes in WS_r have been executed locally ($WS_r \subseteq WS$?)
  - If not, asks the appropriate servers the missing operations O
  - Applies the operations O and add them to WS
  - Returns the requested value and the WS set to the client
- The client
  - Adds WS to its local read set: $WS_r = WS_r \cup WS$

Read your writes - Naive implementation

- To perform a read operation o_r, a client:
  - send o_r and its write set WS_w to the server
- The server
  - Checks whether all the writes in WS_w have been executed locally ($WS_w \subseteq WS$?)
  - If not, asks the appropriate servers the missing operations O
  - Applies the operations O and add them to WS
  - Returns the requested value to the client
- To perform a write operation o_w, a client c
  - send o_w to the server
  - add o_w to the write set WS_w

Contents

1. Replication architectures
   - Overview
   - Primary-Backup
   - Quorum protocols
   - State machines
   - Client-centric consistency
2. CAP Theorem
3. Bibliography

CAP theorem

Theorem (Impossibility of CAP)

It is impossible for a web service to provide more than two of the following three guarantees:

- ***C**onsistency*
- ***A**vailability*
- ***P**artition-tolerance*

- This is the reason why Amazon Web Services only provide eventual consistency
  - W. Vogels. Eventual consistent. *Comm. of the ACM*, 52(1):40–44, 2009
- Similar stands have been taken for example by HP
  - HP. There is no free lunch with distributed systems. `http://www.disi.unitn.it/~montreso/ds/papers/NoFreeLunchDS.pdf`

CAP theorem

History:

- First introduced by Eric Brewer in a keynote at PODC'00
  - E. A. Brewer. Towards robust distributed systems (abstract). In *Proc. of the 19th ACM symposium on Principles of distributed computing*, PODC'00, page 7. ACM, 2000
- Formally proved by Gilbert and Lynch two years later
  - S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33:51–59, June 2002.
    `http://www.disi.unitn.it/~montreso/ds/papers/CapProof.pdf`

Contents

1. Replication architectures
   - Overview
   - Primary-Backup
   - Quorum protocols
   - State machines
   - Client-centric consistency
2. CAP Theorem
3. Bibliography

Reading material

- N. Budhiraja, K. Marzullo, F. Schneider, and S. Toueg. The primary-backup approach. In S. Mullender, editor, *Distributed Systems (2nd ed.)*. Addison-Wesley, 1993.
  `http://www.disi.unitn.it/~montreso/ds/papers/PrimaryBackup.pdf`

- F. Schneider. Replication management using the state machine approach. In S. Mullender, editor, *Distributed Systems (2nd ed.)*. Addison-Wesley, 1993.
  `http://www.disi.unitn.it/~montreso/ds/papers/StateMachine.pdf`