Quorum Systems

Navendu Jain

CS 395T: Design and Implementation of Trusted Services

Outline

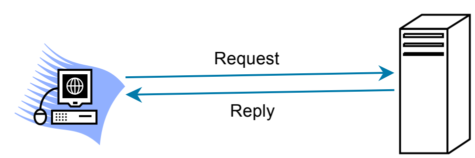
- Motivation
- Quorum Systems
- Tree Quorums
- Probabilistic Quorum Systems

Outline

- **Motivation**
- Quorum Systems
- Tree Quorums
- Probabilistic Quorum Systems

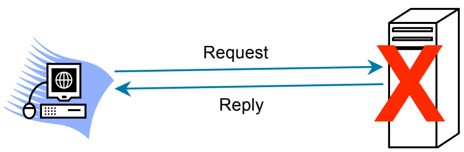
Introduction

- Object stored on a single server



Introduction

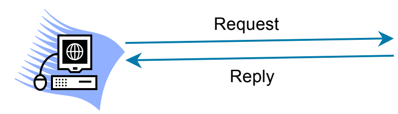
- Object stored on a single server

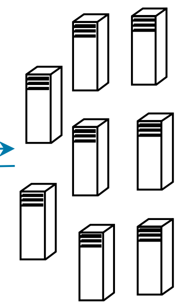


What if the server fails ?

Introduction

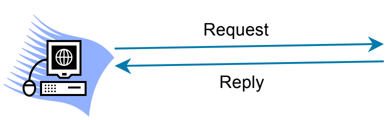
- Replicate the object
  - Availability ✓
  - Performance ✓

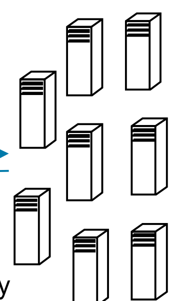




Introduction

- Replicate the object
  - Availability ✓
  - Performance ✓

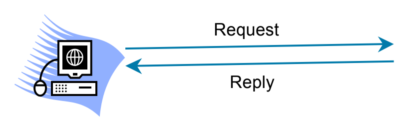


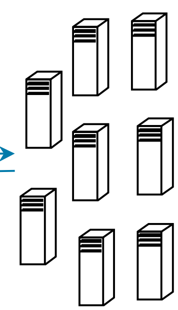


But what about consistency (failures, msg reordering) ?

Quorum Systems

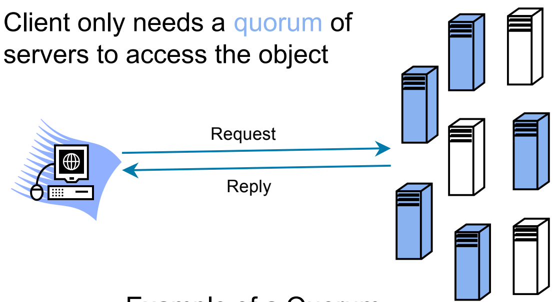
- Client only needs a quorum of servers to access the object





Quorum Systems

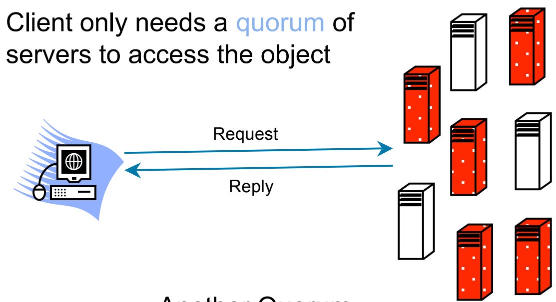
- Client only needs a quorum of servers to access the object



Example of a Quorum

Quorum Systems

- Client only needs a quorum of servers to access the object



Another Quorum

Outline

- Motivation
- **Quorum Systems**
- Tree Quorums
- Probabilistic Quorum Systems

Quorum Systems

Definition

Given a set of servers $\mathrm{P} = \{P_1, \ldots, P_n\}$

A *quorum system* $\mathcal{Q} \subseteq 2^P$ *is a set of subsets of* P *such that*

$$\forall Q_1, Q_2 \in \mathcal{Q} : Q_1 \cap Q_2 \neq \phi$$

Each $Q \in \mathcal{Q}$ is called a quorum

Coterie

Definition

Given a set of servers $P = \{P_1, \ldots, P_n\}$

A *coterie* $\mathcal{Q} \subseteq 2^P$ *is a quorum system such that*

$$\forall Q_1, Q_2 \in \mathcal{Q} : Q_1 \not\subset Q_2$$

Coteries are quorums of minimal size

Example Quorum Systems

Singleton $\mathcal{Q} = \{\{P_1\}\}$

Majority $P = \{P_1, \ldots, P_n\}$

$$\mathcal{Q} = \left\{ Q \subset P : |Q| = \left\lceil \frac{n+1}{2} \right\rceil \right\}$$

tolerates $t < \frac{n}{2}$ faulty servers

Weighted Majority $P = \{P_1, \ldots, P_n\}$

Every server s is assigned w_s votes

$$\mathcal{Q} = \left\{ Q \subset P : \sum_{q \in Q} w_q > \frac{\sum_{q \in P} w_q}{2} \right\}$$

Example Quorum Systems

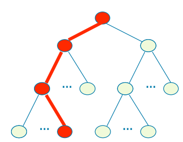
Tree Quorum

$P = \{P_1, \ldots, P_n\}$

$\mathcal{Q} = \{Q \subset P : a \text{ path from top to a leaf node}\}$

$\lceil \log n \rceil \le |Q| \le \left\lceil \frac{n+1}{2} \right\rceil$

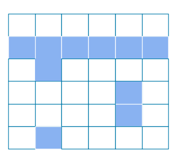
tolerates $t < n - \lceil \log n \rceil$ (*) faulty servers



Grid

A $\sqrt{n}$ x $\sqrt{n}$ grid of n servers

Quorum size $O(\sqrt{n})$



Voting and Quorums

Weighted Majority $P = \{P_1, \ldots, P_n\}$

Every server s is assigned w_s votes

$$\mathcal{Q} = \left\{ Q \subset P : \sum_{q \in Q} w_q > \frac{\sum_{q \in P} w_q}{2} \right\}$$

Majority Voting $P = \{P_1, \ldots, P_n\}$

Let V be the total number of votes

Define, *r* and *w*, the quorums required for read and write ops respectively

2w > V

w + r > n

Voting and Quorums

Vote assignments and quorums are not equivalent

Quorum systems are strictly more general than voting

$$Number\ \ of\ quorums = O(2^{2^{cn}})$$

$$Number\ \ of\ vote\ assigments = O(2^{n^2})$$

Measures on Quorum Systems

- **Load**
  Probability of accessing the busiest server in the best case (an optimal strategy of accessing the servers)
- **Resilience**
  Maximum number of faulty servers that the quorum system can tolerate
- **Failure Probability**
  Probability that at least one server of every quorum fails

Load

Access strategy w : probability distribution on elements of $\mathcal{Q}$

$$\sum_{Q \in \mathcal{Q}} P_w(Q) = 1$$

The load induced by strategy w on a server i

$$l_w(i) = \sum_{Q \in \mathcal{Q} : i \in Q} P_w(Q)$$

The load induced by w on $\mathcal{Q}$

$$L_w(\mathcal{Q}) = \max_{i \in P} l_w(i)$$

The system load (or load) on a quorum system $\mathcal{Q}$ is

$$L(\mathcal{Q}) = \min_{w} L_w(\mathcal{Q})$$

Comparison

$\mathcal{Q}$	$L(\mathcal{Q})$	$R(\mathcal{Q})$	$F_p(\mathcal{Q})$
Singleton	1	0	$p(>\frac{1}{2})$
Majority	$\frac{1}{2}$	$\left\lfloor \frac{n-1}{2} \right\rfloor$	$e^{-\Omega(n)}{}_{p<\frac{1}{2}}$
Tree	$\frac{2}{\log(n+1)+1}$	$n - \log n$	-
Grid	$O(\frac{1}{\sqrt{n}})$	$\sqrt{n} - 1$	≈ 1
PQS	$O(\frac{1}{\sqrt{n}})$	$n - l\sqrt{n}$	$e^{-\Omega(n)}{}_{p<\frac{1}{2}}$

How do quorums work ?

- *A quorum system implements a shared read-write register in an asynchronous point-to-point network*

Linearizability

- Each method call on an object should appear to
  - "take effect"
  - Instantaneously
  - Between invocation and response events
- Any such concurrent object is linearizable
- Two operations that
  - Non-Overlap: must be ordered in an order consistent with their real-time precedence
  - Overlap: can be ordered either way

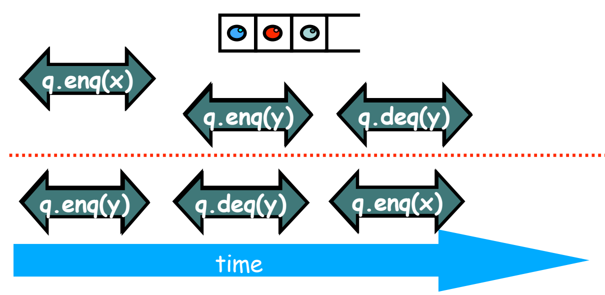
Sequential consistency

Two operations that

- Non-Overlap: must be ordered in that order (need not be their real-time precedence)
- Overlap: can be ordered either way

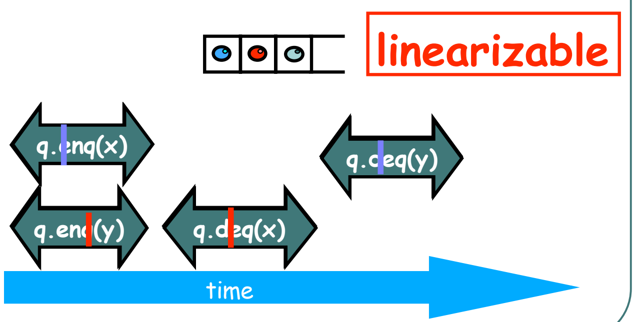
Linearizability is stronger

Sequentially consistent but not linearizable



(6)

Example



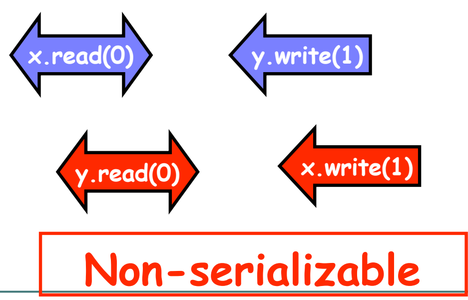
(6)

Serializability

- A transaction is a finite sequence of method calls to a set of shared objects
- Serializable if
  - transactions appear to execute serially
- Strictly serializable if
  - order is compatible with real-time
- Used in databases
- *Linearizability: single method, single object*

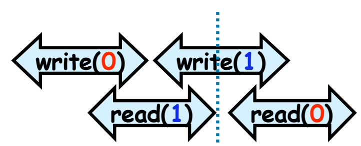
Strict Serializability even stronger

Transactions: A, B   Shared Objects: x, y



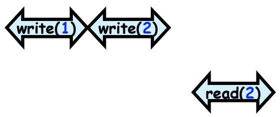
Semantics

- Safe:
  A read not concurrent with any write returns most recently written value
- Regular:
  Safe + a concurrent read (with a write) obtains either old or new value
- Atomic:
  Safe + reads and writes behave as if they occur in some definite order



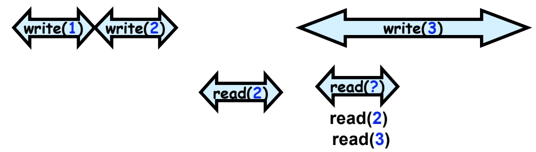
Semantics

- Safe:
  A read not concurrent with any write returns most recently written value
- Regular:
  Safe + a concurrent read (with a write) obtains either old or new value
- Atomic:
  Safe + reads and writes behave as if they occur in some definite order



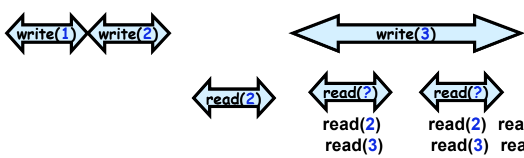
Semantics

- Safe:
  A read not concurrent with any write returns most recently written value
- Regular:
  Safe + a concurrent read (with a write) obtains either old or new value
- Atomic:
  Safe + reads and writes behave as if they occur in some definite order



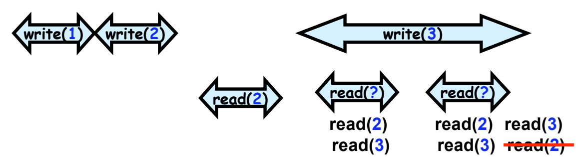
Semantics

- Safe:
  A read not concurrent with any write returns most recently written value
- Regular:
  Safe + a concurrent read (with a write) obtains either old or new value
- Atomic:
  Safe + reads and writes behave as if they occur in some definite order



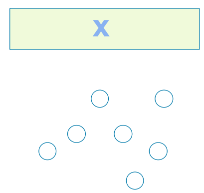
Semantics

- Safe:
  A read not concurrent with any write returns most recently written value
- Regular:
  Safe + a concurrent read (with a write) obtains either old or new value
- Atomic:
  Safe + reads and writes behave as if they occur in some definite order



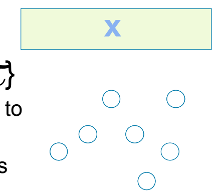
Shared Read-Write Register

- Replicated Variable X
- Each server stores (v, ts)
  - v – local copy of X
  - ts – timestamp
- Operations
  - Write (V, _)
  - Read (X)



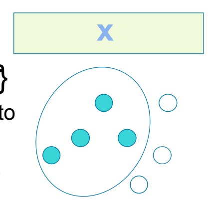
Shared Read-Write Register

- Writer W: Write (V, τ)
  - Picks a quorum Q to get ts
  - τ > max {({ts} from Q) ,prev τ}
  - Sends (Write, V, τ) operation to some quorum Q'
  - Each server checks ts < τ; sets X = V; ts = τ
  - W waits for |Q'| *acks* before terminating the write



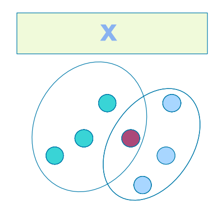
Shared Read-Write Register

- Writer W: Write (V, τ)
  - Picks a quorum Q to get ts
  - τ > max {({ts} from Q) ,prev τ}
  - Sends (Write, V, τ) operation to some quorum Q'
  - Each server checks ts < τ; sets X = V; ts = τ
  - W waits for |Q'| *acks* before terminating the write



Shared Read-Write Register

- Reader W: Read (X)
  - Sends (Read, X) to some quorum Q to get all (v, ts)
  - Selects (v, ts) such that ts = max {({ts} from Q)}
  - Writes (v, ts) to some quorum Q'



Issues

- Timestamp ts – break symmetry
  - E.g. Node id in lower bits
- Writer: τ > max {(ts) from Q, prev τ}
  - Concurrent writes by single/multiple writers
- Concurrent reads and writes
  - **Regular** semantics
  - Reader: writes to some quorum after reading

Can we do better ?

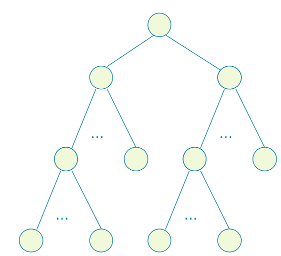
- Minimize quorum size
- Reduce communication cost
- Graceful degradation
  - more msgs only when failures increase

Outline

- Motivation
- Quorum Systems
- Tree Quorums
- Probabilistic Quorum Systems

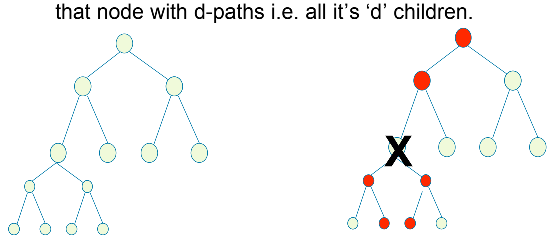
Tree Quorums (trade time + storage for communication)

- Algorithm
  - Impose a d-ary tree logical structure
  - Quorum calls
    - GrantsPermission (Site s)
      - Agrees to be in quorum
    - GetQuorum (Root T)
      - Initiate a quorum vote

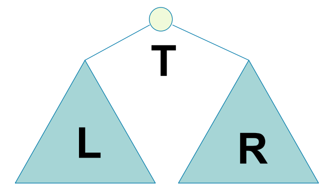


Tree Quorums

- Algorithm
  - Key Idea: On a failure, the algorithm substitutes for that node with d-paths i.e. all it's 'd' children.



Proof

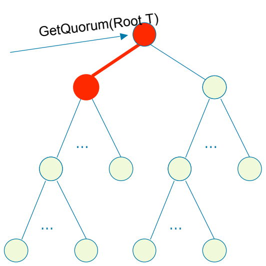


Quorum:

- T U L
- T U R
- L U R

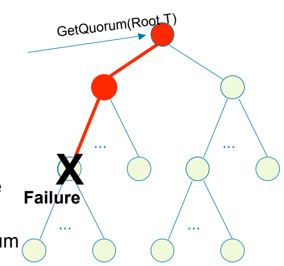
Tree Quorums

- Algorithm
  - Impose a d-ary tree logical structure
  - Quorum calls
    - GrantsPermission (Site s)
      - Agrees to be in quorum
    - GetQuorum(Root T)
      - Initiate a quorum vote



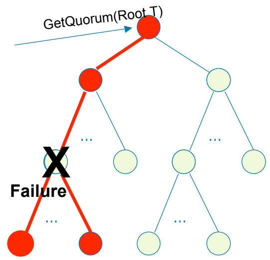
Tree Quorums

- Algorithm
  - Impose a d-ary tree logical structure
  - Quorum calls
    - GrantsPermission (Site s)
      - Agrees to be in quorum
    - GetQuorum(Root T)
      - Initiate a quorum vote



Tree Quorums

- Algorithm
  - Impose a d-ary tree logical structure
  - Quorum calls
    - GrantsPermission (Site s)
      - Agrees to be in quorum
    - GetQuorum(Root T)
      - Initiate a quorum vote
  - Quorum: Path starting from top to a leaf. Size = $\lceil \log n \rceil$



Can we do even better ?

- Quorum Q only functions when all |Q| nodes work
  - No quorum exists (if all |Q| elements of any Q fail)
- Implications:
  - large quorum sizes : more reliable.
  - Small quorum sizes : increase efficiency, reduce communication
- Any strict quorum system with optimal load of $\frac{1}{\sqrt{n}}$ has fault tolerance of only $O(\sqrt{n})$ [NW98]

tradeoff between low load and fault tolerance

Outline

- Motivation
- Quorum Systems
- Tree Quorums
- Probabilistic Quorum Systems

Probabilistic quorums [MRW97]

- Relax Intersection: Quorums may not intersect
- Property:

  Pairs of quorums chosen according to a specific access strategy w intersect w.h.p.

$$\forall Q_1, Q_2 \in \mathcal{Q} \ \Pr[\ Q_1 \cap Q_2\ ] \geq 1 - \varepsilon$$

- A probabilistic quorum system is defined
  - w.r.t a consistency guarantee ε
  - access strategy to achieve guarantee w

Example construction [MRW97]

$P = \{P_1, \ldots, P_n\}$.

The quorums are all the sets of size $l\sqrt{n}$ $(l \geq 1)$

$\mathcal{Q}\ Q \subseteq P : |Q| = l\sqrt{n}\};$

$\forall Q \in \mathcal{Q}\ w(Q) = \frac{1}{|\mathcal{Q}|}; \varepsilon = e^{-l^2}$

$\mathcal{Q}$	$L(\mathcal{Q})$	$R(\mathcal{Q})$	$F_p(\mathcal{Q})$
Strict	$O(\frac{1}{\sqrt{n}})$	$O(\sqrt{n})$	$\approx 1_{\ p>\frac{1}{2}}$
Probab.	$O(\frac{1}{\sqrt{n}})$	$\Omega(n)$	$e^{-\Omega(n)}{}_{\ p<\frac{1}{2}}$

References

- D. H. Gifford. Weighted voting for replicated data. In *Proceedings of the 7th ACM Symposium on Operating Systems Principles*. Pages 150--159, Asilomar Conference Grounds, Pacific Grove, CA USA, December 10--12, 1979. ACM.
- R. H. Thomas. A majority consensus approach to concurrency control for multiple copy databases. *ACM Transactions on Database Systems*, 4(2):180--209, June 1979.
- H. Garcia-Molina and D. Barbara. How to assign votes in a distributed system. *Journal of the ACM*, 32(4):841--860, October 1985.
- D. Agrawal and A. El Abbadi. An efficient and fault-tolerant solution for distributed mutual exclusion. *ACM Transactions on Computer Systems*, 9(1):1-20, February 1991.
- M. Herlihy. A quorum-consensus replication method for abstract data types. *ACM Transactions on Computer Systems*, 4(1):32-53, February 1986.
- [MRW97] D. Malkhi, M. Reiter, and R. Wright. Probabilistic quorum systems. In *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing*, pp. 267-273 June 1997.

References

- I. Abraham, D. Malkhi. Probabilistic quorum for dynamic systems. In *17th International Symposium on Distributed Computing (DISC 2003)*, Sorrento, Italy.
- [NW98] M. Naor and A. Wool. The load, capacity, and availability of quorum systems, *SIAM Journal of Comput.*, Vol. 27, No. 2, 423-447, April 1998.
- M. Maekawa. A √(n) algorithm for mutual exclusion in decentralized systems. *ACM Transactions on Computer Systems*, 3(2):145-159, 1985.
- L. Lamport. On interprocess communications (part ii: algorithms). *Distributed Computing*, 1:86-101, 1986.
- D. Malkhi. Quorum Systems. In *The Encyclopedia of Distributed Computing. Joseph Urban and Partha Dasgupta Editors*, Kluwer Academic Publishers.

Thanks