

CS603: Distributed Systems

Lecture 15: Quorum Systems

The State Machine Approach

- The system consists of clients that invoke commands on **deterministic state machines**.
- The state of a state machine depends only on its initial state and the sequence of (deterministic) commands it has been given.
- All non-faulty state machines, being deterministic, will give the same response to a command.



Challenges

- Main challenge is ensuring coordination between servers: requires **agreement on the request to be processed** and **consistent order of requests**.
- **IN REAL SYSTEMS, CLIENTS AND SERVERS CAN CRASH OR MISBEHAVE at ANY TIME.**
- How do clients submit a request to make sure it is processed by servers?
- How to guarantee correctness and termination between servers when any server can crash or misbehave?
- How can clients be sure that the received answer is correct?
- How to deal with clients that can misbehave?

About Consensus ... again

- **Asynchronous:** consensus can not be reached even if there is just one benign fault [FLP83]: participants can not distinguish between faults or delayed messages.
- Alternative? Synchronous models? **BUT REAL, PRACTICAL SYSTEMS ARE NOT SYNCHRONOUS !!!**
- Possible solutions:
 - Process groups: sacrifice liveness under the assumption that retransmissions will eventually be received from good participants, the protocol eventually terminates
 - Avoid consensus, use quorum systems
 - Use randomization, probabilistic guarantees

Process Groups Approach

- One way of building distributed fault-tolerant systems by organizing them in a group:
 - Ensure group membership
 - Ensure group multicast, with different ordering properties.
- Easier to work with when providing in the form of a toolkit.



Limitations of Process Groups Approach

- Need to respond to leader failure
 - Costly agreement on membership
- Virtual synchrony: simplify recovery from partitioned views
- Servers need to monitor for failures (correct but slow participants may be excluded)
- Reconfiguration

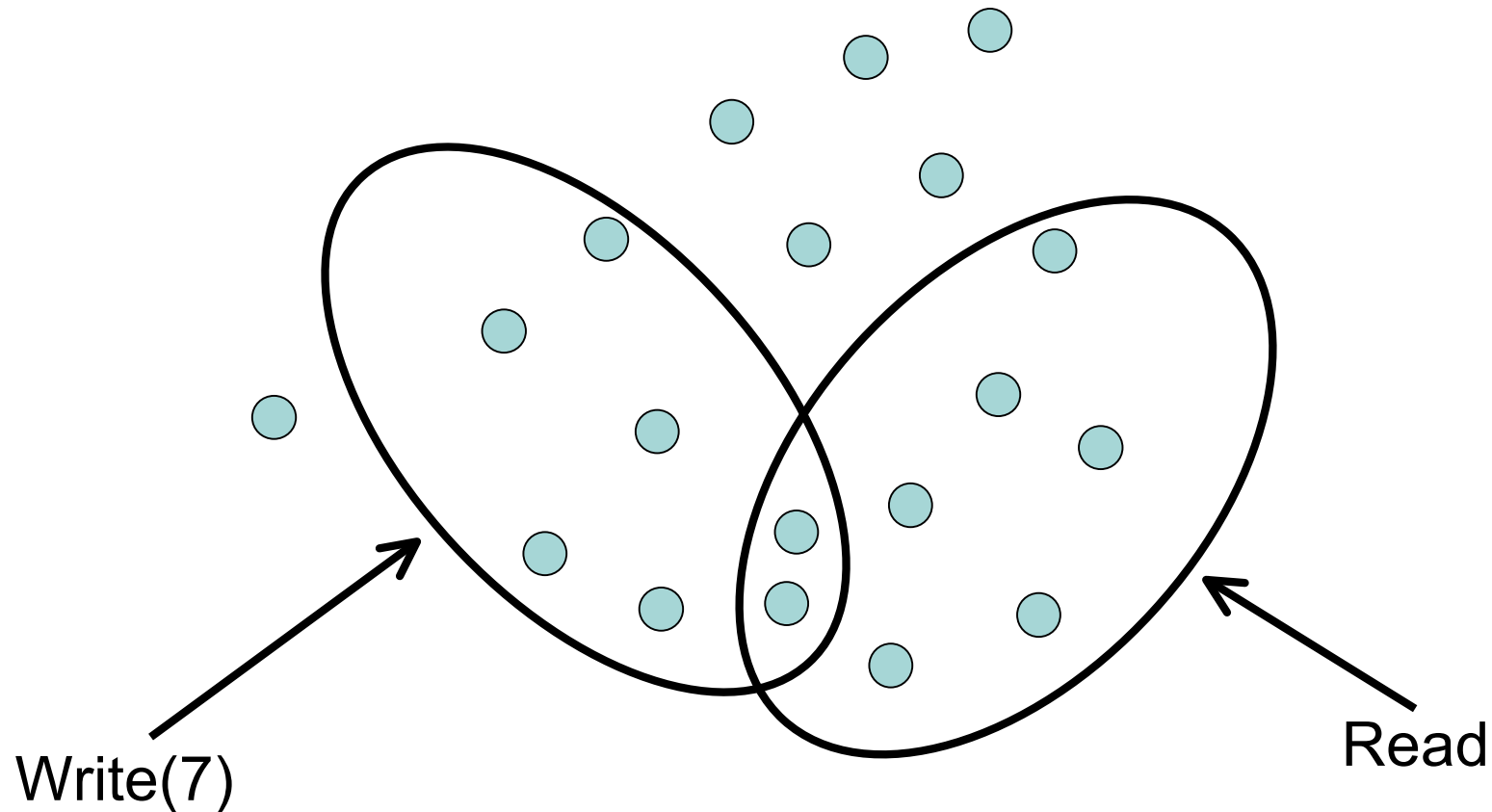
Quorum Systems

- In law, a quorum is **the minimum number** of members of a deliberative body necessary to conduct the business of that group
- When **quorum is not met, a legislative body cannot** hold a vote, and cannot change the status quo
- What does this have to do with distributed systems anyway?

Quorum Systems

- Increase **availability** and **efficiency** of replicated services
- **Availability**: Operations succeed in spite of failures; quorum systems can be defined to tolerate both benign and arbitrary/malicious failures
- **Efficiency**: Can significantly reduce communication complexity, do not require all servers in order to perform an operation, requires a subset of them for each operation

Using Quorums to Read and Write



The set of processors from which a variable is *read* must intersect the set of processors to which a variable was *written*.

Quorum Systems: Some Definitions

- **A quorum system for a universe of servers is a collection of subsets of servers, each pair of which intersect.**
- Formally:
 - Assume a universe U of servers, $|U| = n$, and an arbitrary number of clients. A *quorum system* $\mathcal{Q} \subseteq 2^U$ is a set of subsets of U , every pair of which intersect. Each $Q \in \mathcal{Q}$ is called a quorum.

Example: Shared Variable

- Use a quorum system to implement a multi-reader multi-writer shared variable, replicated across n servers.

Write:

- Client queries each server in some quorum (writing quorum) to obtain a set A of value/timestamp pairs;
- Choose a timestamp greater than **the highest value** in the set A and updates the value and the timestamp at each server in the writing quorum

Example: Shared Variable (II)

Read:

- Client queries each server in a quorum to obtain a set A of value/timestamp
- Then chooses the pair with the highest timestamp
- For both read and write each server updates its local variable and timestamp to the received values, only if received timestamp is greater than the one they had for that value

Replication with Quorums

- Replicated data items have “versions”, and these are numbered
 - I.e. can't just say “ $X_p=3$ ”. Instead say that X_p has timestamp $[7,q]$ and value 3
 - Timestamp must increase monotonically and includes a process id to break ties
 - We will see later that this process id is

Read Operation

- Send request and wait until Q_r (read quorum) processes reply
- Then use the value with the largest timestamp
 - Break ties by looking at the process id
 - For example
 - $[6,x] < [9,a]$ (first look at the “time”)
 - $[7,p] < [7,q]$ (but use process id as a tie-breaker)

Write Operation

- When a process initiates a write, it does not know if it will succeed in updating a quorum (writing quorum) of processes
 - Need to use a commit protocol
- Moreover, must implement a mechanism to determine the version number as part of the protocol.

Write Operation: Details

1. Propose the write: "I would like to set $X=3$ "
2. Members "lock" the variable against reads, put the request into a queue of pending writes, and send back:
"OK. I propose time $[t, pid]$ "
Here, time is a logical clock. Pid is the member's own pid
3. Initiator collects replies, hoping to receive Q_w (or more)

$\geq Q_w$ OKs

Compute maximum of proposed $[t, pid]$ pairs.
Commit at that time

$< Q_w$ OKs

Abort

Examples of quorum constructions

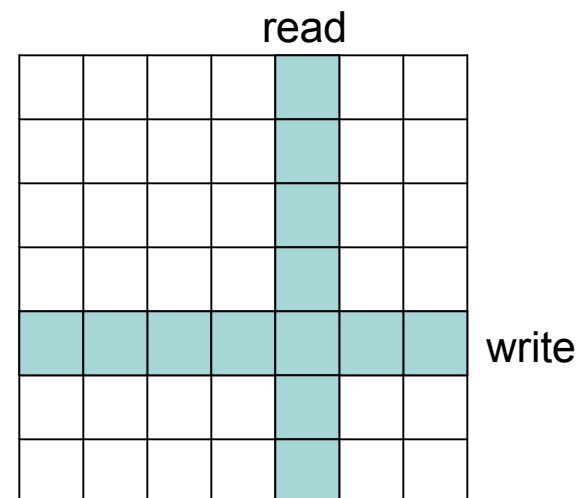
Weighted Majorities:

- Assume that every server s in the universe U is assigned a number of votes w_s .
- Then, the set system $\mathbf{Q} = \{Q \subseteq U: \sum_{q \in Q} w_q > 1/2 \sum_{q \in U} w_q\}$ is a quorum system called Weighted Majorities.
- When all the weights are the same, simply call this the system of Majorities.

Another Example

Grid

- Previous example was not very efficient, requiring more than half of the servers to be contacted.
- Arrange servers into a logical grid, and use rows/columns for writes/reads, respectively.
- Can cut the number of servers contacted in an operation.
- Can change row/column sizes to optimize for write-heavy/read-heavy scenarios.



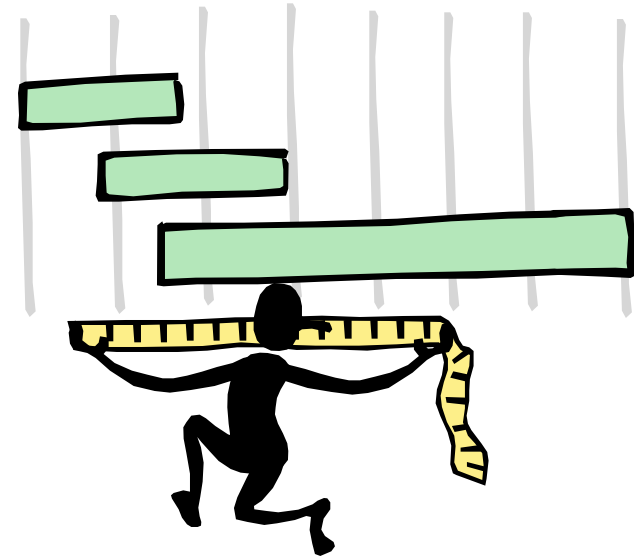
$$\sqrt{n} \quad \times \quad \sqrt{n}$$

Quorum Size

- Why access a large quorum if a subset of it is itself a quorum?
- Ideally we need quorum systems that cannot be reduced in size
- *Coterie*: a coterie $\mathcal{Q} \subseteq 2^U$ is a quorum system such that for any $Q, Q' \in \mathcal{Q}$: $Q \cap Q' \in \mathcal{Q}$
- *Domination*: suppose that $\mathcal{Q}, \mathcal{Q}'$ are two coterie, $\mathcal{Q} \neq \mathcal{Q}'$, such that for every $Q' \in \mathcal{Q}'$, there exists a $Q \in \mathcal{Q}$ such that $Q \subseteq Q'$. Then \mathcal{Q} dominates \mathcal{Q}'

How Do We Compare Quorum Systems

- **Load:** The probability of accessing the busiest server in the best case
- **Resilience:** The highest number of servers such that there is a quorum that has no failed servers contained in it
- **Failure Probability:** The probability of the event in which every quorum contains at least one crashed server
- **There is an inherent trade-off between load and resilience. In fact, it is not possible to achieve optimality in both simultaneously.**



Load: More Precise Definitions

- **Load:** The probability of accessing the busiest server in the best case
- Given a quorum system Q , an access strategy w , is a probability distribution on the elements of Q , $w(Q)$ is the probability that quorum Q will be chosen when the service is accessed

Definitions

- Let a strategy w be given for a quorum systems $Q = \{Q_1, \dots, Q_m\}$ over universe U . For u in U , the load induced by w is $L_w(u) = \sum_{u \in Q_i} w(Q_i)$
- The load induced by w on Q is $L_w = \max_{u \in U} \{L_w(u)\}$
- The system load is $L(Q) = \min_w \{L_w(Q)\}$

Load: cont.

- If $c(Q)$ is the size of the smallest quorum of Q , then $L(Q) \geq \max\{1/c(Q), c(Q)/n\}$.
Thus $L(Q) \geq 1/\sqrt{n}$

Resilience

- **Resilience:** The highest number of servers such that there is a quorum that has no failed servers contained in it
- Definition: The resilience f of a quorum system is the largest k such that for every set $K \subseteq U$, $|K| = k$, there exists Q s.t. $K \cap Q = \emptyset$
- The resilience is at most $c(Q) - 1$, where $c(Q)$ is the size of the smallest quorum
- One way to measure it is to assume that servers fail independently with probability p , then determine the probability that no quorum remains completely alive

Let's Compare some QS?

- Majorities have the highest possible resilience and asymptotically optimal failure probability, but poor load
- Grids have near optimal load, but mediocre resilience, and failure probability tends to 1 as n grows.

Byzantine Quorum Systems

- A quorum system tolerant of Byzantine failures is a collection of subsets of servers, each pair of which intersect containing sufficiently many correct servers to mask out the behavior of faulty servers
- Require intersection size to be at least $2b+1$ to tolerate b Byzantine faults
- Requires at least $4b+1$ servers in total
- The faults are “masked” by the majority of correct and most recent values in the intersection

Probabilistic Quorum Systems

- The previous constructions were deterministic.
- Probabilistic constructions are simpler, more adaptive, and may circumvent difficulties with the tradeoff between load and resilience.
- *Dynamic* quorum systems can handle servers joining and leaving.

Probabilistic Quorum Systems

- Observations
 - Resilience of any quorum system is bounded by half the number of systems
 - Inherent tradeoff between low load and good resilience
- Relax the intersection property of a quorum system so that “quorums” chosen according to a specified strategy intersect only with very high probability
- A small relaxation in consistency can yield a dramatic improvement in resilience and failure probability of the system (availability)

Quorum Systems in MANETs

- Previous constructions & protocols are unsuitable for the highly dynamic nature of mobile ad hoc networks.
- Need to leverage the dynamic topology, not fight it.
- Probabilistic constructions & protocols seem favorable.

Terminology

- A *quorum system* (also called a *construction*) is the set of subsets (quorums) that satisfy the intersection requirement.
- A *strict quorum system* is one that guarantees intersection between two quorums.
- When the phrase “a random quorum” is used, it refers to a randomly chosen subset (from the quorum system), **not** a randomly chosen set of nodes that constitutes a quorum.

Why Change Things?

- Strict constructions are highly sensitive in terms of failure probability (inconsistency).
- Since messages are expensive in wireless, we want to minimize the number of them.
- Practically speaking, construction of quorum systems is tricky enough in static networks. In a highly dynamic environment, it is inefficient to create static logical constructions since they can require global views of the system.
- Always-changing topology prevents logical constructions (e.g. a grid tied to node IDs) does not exploit locality.

Strict Quorum Systems in MANETs

- A lot of research avoids constructing quorums, instead assuming quorums are already constructed.
- The few explicit constructions are generally based on logical arrangements into a grid of some sort.
- Quorums are still selected randomly



Enter Probabilistic Constructions...

- Basic idea: Form the quorum itself probabilistically as part of the update/query protocols.
- Each update or query can independently form a quorum.
- Intersection (consistency) becomes w.h.p. instead of guaranteed.
- This idea is not new, probabilistic quorum systems have been studied in wired networks. What is new are the update/query protocols that accomplish it.

Examples

- If all nodes are known, pick nodes from entire network uniformly at random.
- Forward to neighbors based on direction (e.g. updates go north/south, reads east/west).
- Employ probabilistic gossip protocols to diffuse updates/reads to an appropriate “random” set.
- In all cases there is no actual quorum system, each update/read independently builds a random set of nodes.

Metrics

- Average completion time per query
- Correctness rate (percentage of inconsistent values)
- Average communication cost
- Throughput

Key Results

- Density helps...a lot.
- There is a tradeoff between larger quorum sizes (better correctness rate) and communication complexity.
- Probabilistic quorums actually maintain a higher correctness (vs. strict quorums) rate in many cases.
- Probabilistic quorum systems also maintain a higher throughput than existing strict constructions.

Applications

- Distributed storage
- Location Tracking / Mobility Management
- Distributed services
- ...and any other applications that carry over to wireless

References

- D. Malkhi **Quorum Systems**. Chapter in *The Encyclopedia of Distributed Computing*, Joseph Urban and Partha Dasgupta, editors, Kluwer Academic Publishers. To be published.
- D. Malkhi and M. Reiter. **Byzantine quorum systems**. In Proceedings of the 29th ACM Symposium on Theory of Computing (STOC), May 1997.
- D. Malkhi, M. Reiter, A. Wool and R. Wright. **Probabilistic quorum systems**. Preliminary version appears in Proceedings of the 16th ACM Symposium on Principles of Distributed Computing, pages 267--273, August 1997.
- S. Bhattacharya - Randomized Location Service in Mobile Ad-Hoc Networks
- Z. Haas & B. Liang - Ad-Hoc Mobility Management with Uniform Quorum Systems
- J. Welch H. Lee & N. Vaidya - Location Tracking Using Quorums in Mobile Ad-Hoc Networks
- I. Stojmenovic & P. Pena - A Scalable Quorum Based Location Update Scheme for Routing in Ad-Hoc Wireless Networks
- J. Luo J.P. Hubaux & P.T. Eugster - PAN: Providing Reliable Storage in Mobile Ad Hoc Networks with Probabilistic Quorum Systems