Distributed Storage Systems: Data Replication using Quorums

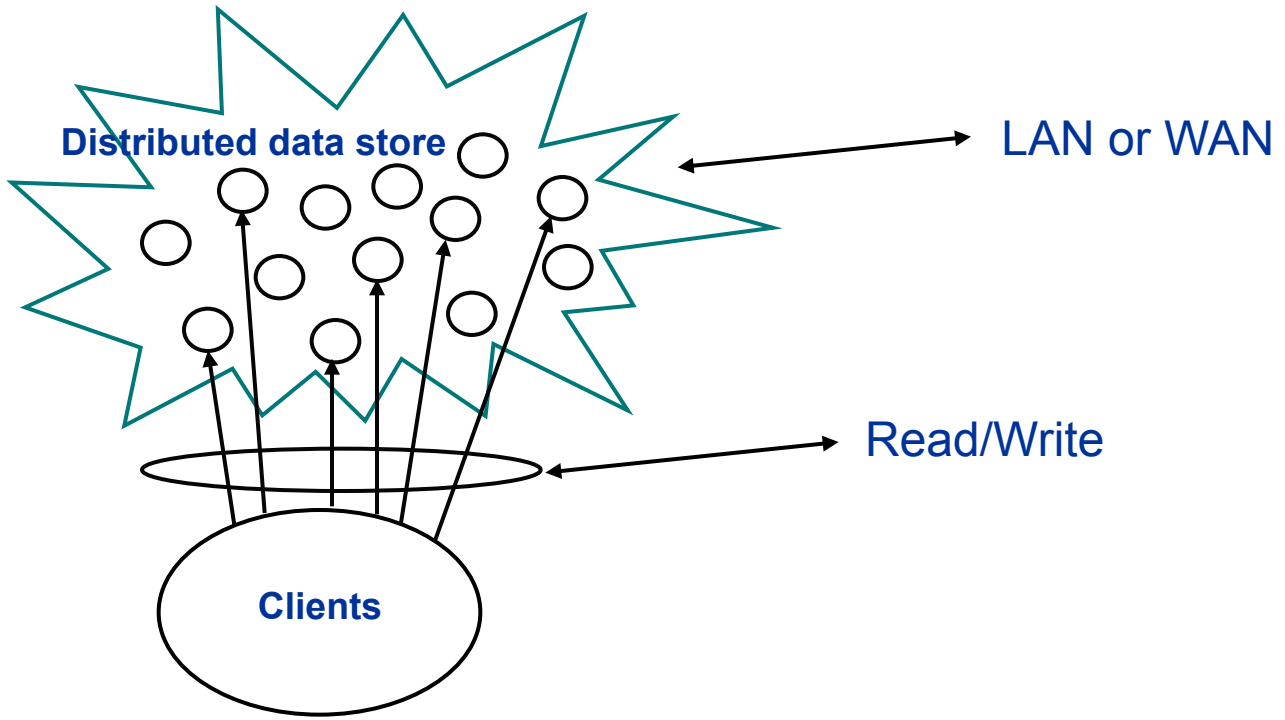
Georgia Institute of Technology

cercs

Background

- Software replication focuses on dependability of computations
- What if we are primarily concerned with integrity and availability (and perhaps confidentiality) of data?
- *Data replication* has several advantages
  - Simpler protocols (atomic ordering not necessarily required, no determinism assumptions)
  - Variety of consistency models provides efficiency/consistency tradeoffs
  - Performance better than software replication: simpler protocols can be implemented efficiently; caching and hashing

Distributed Storage Model



Georgia Institute of Technology

cercs

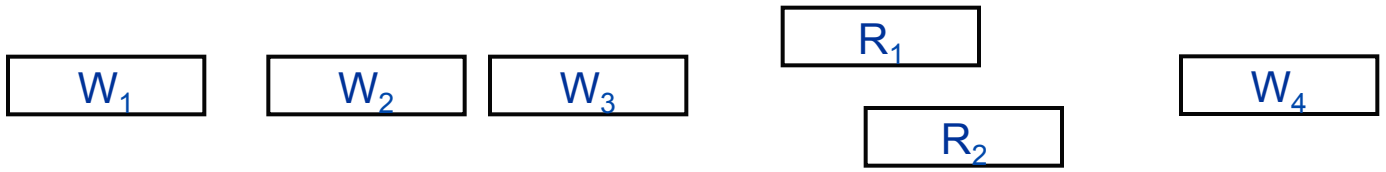
Basic Quorum Systems

- Gifford, "Weighted Voting for Replicated Data," *SOSP Proceedings*, 1979
- each client reads from *r* servers and writes to *w* servers
- if $r+w>n$, then the intersection of every pair of read/write sets is non-empty $\Rightarrow$ ***every read will see at least one copy of the latest value written***
- $r=1$ and $w=n \Rightarrow$ full replication (write-all, read-one): undesirable when servers can be unavailable because writes are not guaranteed to complete
- best performance (throughput/availability) when $1 < r < w < n$, because reads are more frequent than writes in most applications
- generalization: *r, w* vary across clients but non-empty intersection between all read/write sets is maintained

Basic Quorum Systems: Protocols

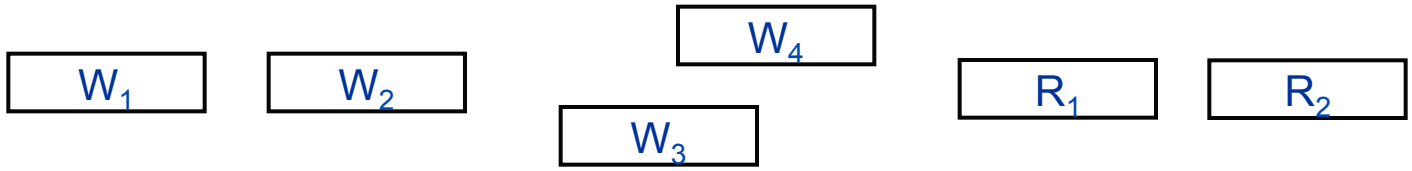
- time stamps are maintained for each object in store
- read protocol for a data object *V*:
  - read <*v, t*> from all servers in some read set
  - select *v* with latest time stamp *t*
- write protocol for a data object *V*:
  - read value according to above protocol to determine current time stamp *t*
  - write <*v, t'*> to all servers in some write set with time stamp *t' > t*
- guarantees *serial consistency* (sometimes called *safe semantics*)*:* a read operation that is not concurrent with any write operation on the same object returns the last value written in some serialization of the preceding writes to that object

Serial Consistency



(R_1, R_2 both read value written by W_3)

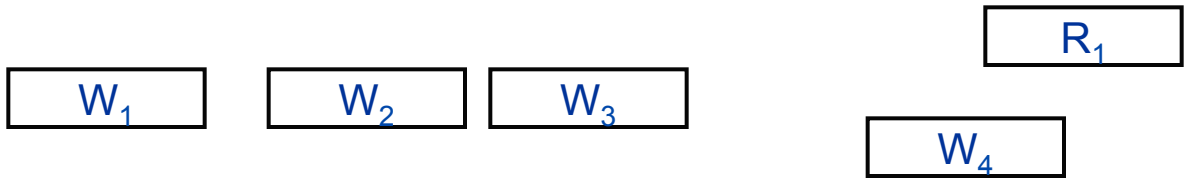
Serial Consistency, cont.



(R_1, R_2 both read ***same*** value, could be either the result of W_3 or W_4, i.e. serialization could be W_1, W_2, W_3, W_4 or W_1, W_2, W_4, W_3 but it must be seen the same way by R_1 and R_2)

- how can this be achieved?
- how are timestamps for W_3 and W_4 set?

Serial Consistency, cont.



(R_1 can return ***either*** the value written by W_4 ***or*** the value written by W_3 - serial consistency does not cover the case of a read that overlaps with a write)

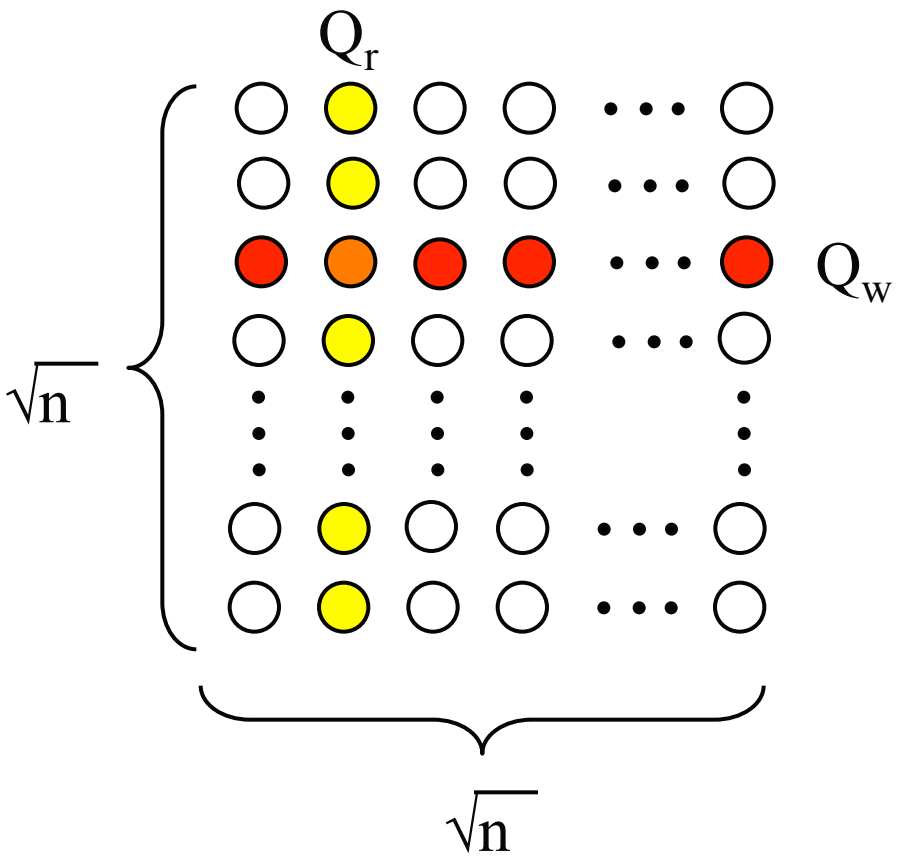
Data Replication: Performance Optimizations

- Data objects can be large, e.g. files, and many copies are transmitted during quorum protocol operation ⇒ extremely high bandwidth usage
- Hashing
  - Store hash of data at every server along with data
  - Return hash values and meta data (not data) during read operations
  - Do voting based on hash values
  - Once correct hash is determined, query a single server for data object and calculate its hash to verify data integrity
- Caching
  - Store data objects in client caches
  - Execute hashing protocol for a read
  - If object is in cache, calculate hash of cached object and do not query any server for object if cached copy is up to date

Quorum Systems: Formalization

- assume a set U of servers, where $|U| = n$
- An asymmetric quorum system $\mathcal{Q}_1, \mathcal{Q}_2 \subseteq 2^U$ is a pair of non-empty sets of subsets of U, where $\forall Q_r \in \mathcal{Q}_1$, $Q_w \in \mathcal{Q}_2$, $Q_r \cap Q_w \neq \emptyset$
- A symmetric quorum system $\mathcal{Q} \subseteq 2^U$ is a non-empty set of subsets of U, where $\forall Q_1, Q_2 \in \mathcal{Q}$, $Q_1 \cap Q_2 \neq \emptyset$ (read sets and write sets are identical)
- each $Q \in \mathcal{Q}$ is called a *quorum* (for asymmetric system, there are *read quorums* and *write quorums*)

Efficiency of Quorums: Grid Quorums



Grid Quorums are Asymmetric Quorum Systems!!

Georgia Institute of Technology

cercs

System Load

- load = avg. fraction of servers that must be contacted per read/write operation
- for grid quorum systems:
  - load = $\sqrt{n} / n = 1 / \sqrt{n}$
  - load $\rightarrow 0$ as $n \rightarrow \infty$
- compare to basic quorum systems:
  - load = $(cr + (1-c)\, w) / n \geq b$, where c is fraction of ops that are reads, b is a constant, and $r+w > n$

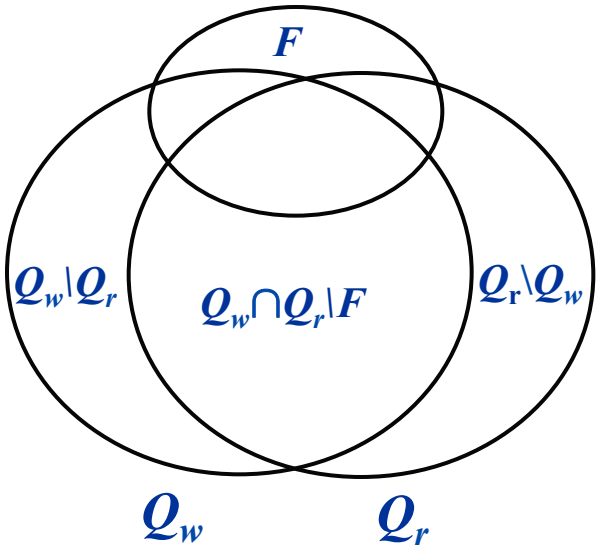
Georgia Institute of Technology

cercs

Byzantine Quorum Systems

- Malkhi and Reiter, *Distributed Computing*
- uses asynchronous system model and assumes servers can fail or be compromised, i.e. servers can experience Byzantine faults
- increase the size of the quorums' intersection to mask responses from faulty servers
- if intersection is at least $2f_{max}+1$, where f_{max} is max. number of faulty servers, then $f_{max}+1$ correct servers will match in any Q
- problem: with asynchronous system model, can not distinguish slow servers from faulty servers

Masking Quorum Systems



- $|(Q_w \cap Q_r) \setminus F| = f_{max}+1$ (correct, up-to-date)
- $Q_r \setminus Q_w$ can be out of date and F can be arbitrary
- if f_{max} faulty servers match out-of-date values in $Q_r \setminus Q_w$, then $f_{max}+1$ or more matching old values can exist
- not sufficient to accept the first $f_{max}+1$ values that match

Georgia Institute of Technology

cercs

Masking Quorum Systems (cont.)

- suppose a read operation has returned f_{max}+1 values that match but some values have not yet been returned
- if the servers that have not responded are correct but slow, the result might change after more values arrive; if the non-responding servers are faulty, they might never respond
- if ***all*** responses arrive from a quorum, pick the newest value that has at least f_{max}+1 representatives (in some cases, correct result can be determined without waiting for all responses)
- approach needs additional constraint: a quorum consisting solely of correct servers must exist at all times to ensure progress
- can contact all servers initially to guarantee a quorum of responses, or can contact different quorums sequentially until a full quorum responds

f-Masking Quorum Systems

- read protocol for a data object *V*:
  - read <*v,t*> from any $|Q_r| + f_{max}$ servers in *U*
  - wait until $|Q_r|$ responses have been received from some quorum Q_r
  - select value with latest time stamp that has at least f_{max} +1 representatives
- write protocol for a data object *V*:
  - read value according to above protocol to determine current time stamp *t*
  - write <*v,t'*> to all servers in some quorum Q_w with time stamp *t'*>*t*
- guarantees *serial consistency* as long as a "fully-correct" quorum exists at all times

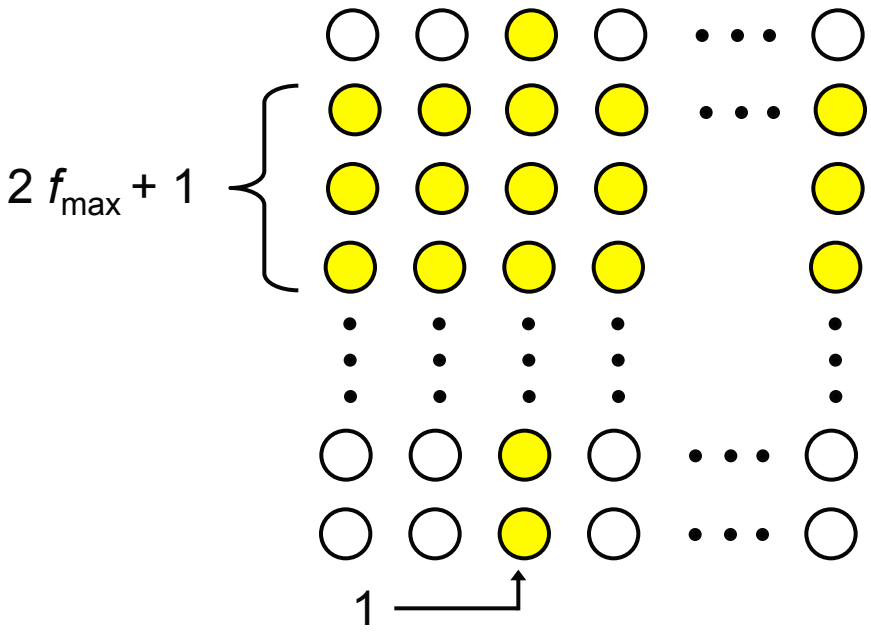
Georgia Institute of Technology

cercs

f-Masking Quorums: Sufficient Condition

- if $n > 4\ f_{max}$ and a symmetric quorum system is used with $|Q_r| = |Q_w| = \lceil (n+2\ f_{max} +1)/2 \rceil$, then the preceding read/write protocols are correct and guarantee progress
- $|Q_w \cap Q_r| \geq 2\ f_{max} +1$
- $n > 4\ f_{max} \Rightarrow \lceil (n+2\ f_{max} +1)/2 \rceil \leq n - f_{max} \Rightarrow$ at least one “fully correct” quorum exists at all time

Grid Masking Quorum Systems



Grid Masking Quorums are Symmetric Quorum Systems!!

- quorum is any 2 f_{max} + 1 rows and 1 column of servers
  $\Rightarrow$ intersection between any 2 quorums $\geq$ 2 f_{max} + 1

Georgia Institute of Technology

cercs

System Load

- load = avg. fraction of servers that must be contacted per read/ write operation
- for symmetric quorum systems, load = $|Q| / n$
  - threshold masking load $\approx (n+2 f_{max}+1) / 2n \in (0.5, 0.75)$
  - grid masking load $\approx (2 f_{max}+2) n^{1/2} / n \in O(f_{max} / n^{1/2}) \rightarrow 0$ as $n \rightarrow \infty$ (for small f_{max})
- for asymmetric quorum systems, load depends on frequency of reads vs. writes,

  $$\text{load} = [\, c\, |Q_r| + (1-c)\, |Q_w| \,] / n,$$

  where c = fraction of accesses that are reads
  - write-all, read-one load = 1-c + c/n > 1-c

Georgia Institute of Technology

cercs

Byzantine Quorum System Variations

- dissemination quorum systems
  - work with self-verifying data, e.g. signed data
  - sufficient that quorums' intersection contains at least one non-faulty server, i.e intersection size ≥ $f_{max}+1$
  - quorum size = $\lceil (n+f_{max}+1)/2 \rceil$, $n > 3\ f_{max}$
- opaque masking quorum systems
  - clients don't need to know f_{max} (harder for adversary to attack system)
  - read protocol simply does a majority vote among responses
  - $n > 5\ f_{max}$ (need to outvote faulty + out-of-date servers)
  - load for any opaque masking quorum system > 1/2
- dealing with faulty/malicious clients
  - can not prevent faulty but authorized clients from writing bad data of which they have write permission
  - have servers execute agreement protocol before installing write values to avoid faulty clients from leaving servers in inconsistent state