Computer Systems - Notes Week 11

Ruben Schenk, ruben.schenk@inf.ethz.ch

January 8, 2022

Chapter 21: Quorum Systems & Distributed Storage

What happens if a single server is no longer powerful enough to service all your customers? The obvious choice is to add more servers and to use the majority approach to guarantee consistency. However, even if you buy one million servers, a client still has to access more than half of them per request! While you gain faul-tolerance, your efficiency can at most be doubled. We used majorities because majority sets always overlap. But are majority sets the only sets that guarantee overlap? In this chapter we study the theory behind overlapping sets, known as *quorum systems.*

Let $V = \{v_1, ..., v_n\}$ be a set of nodes. A **quorum** $Q \subseteq V$ is a subset of these nodes. A **quorum system** $S \subset S^V$ is a set of quorums s.t. every two quorums intersect, i.e. $Q_1 \cap Q_2 \neq \emptyset$ for all $Q_1, Q_2 \in S$.

Remarks:

- When a quorum system is being used, a client selects a quorum, acquires a lock (or ticket) on all nodes of the quorum, and when done releases all locks again. The idea is that no matter which quorum is chosen, its nodes will intersect with the nodes of every other quorum.
- A quorum system S is called **minimal** if $\forall Q_1, Q_2 \in S : Q_1 \not\subseteq Q_2$.
- The simplest quorum system imaginable consists of just one quorum, which in turn just consists of one server. It is known as a **singleton.**
- In the **majority** quorum system, every quorum has $\lfloor \frac{n}{2} \rfloor + 1$ nodes.

21.1 Load and Work

An **access strategy** Z defines the probability $P_Z(Q)$ of accessing a quorum $Q \in S$ s.t. $\sum_{Q \in S} P_Z(Q) = 1$.

Definition of **load:**

- The *load* of access strategy Z on a node v_i is $L_Z(v_i) = \sum_{Q \in S;\, v_i \in Q} P_Z(Q)$. The load is the probability that $v_i \in Q$ if Q is sampled from S.ß
- The *load* included by access strategy Z on a quorum system S is the maximal load induced by Z on any node in S, i.e. $L_Z(S) = \max_{v_i \in S} L_Z(v_i)$.
- The *load* of a quorum system S is $L(S) = \min_Z L_Z(S)$.

Definition of **work:**

- The *work* of a quorum $Q \in S$ is the number of nodes in Q, $W(Q) = |Q|$.
- The *work* induced by access strategy Z on a quorum system S is the expected number of nodes accessed, i.e. $W_Z(S) = \sum_{Q \in S} P_Z(Q) \cdot W(Q)$.
- The *work* of a quorum system S in $W(S) = \min_Z W_Z(S)$.

Example: We illustrate the above concepts with a small example. Let $V = \{v_1, v_2, v_3, v_4, v_5\}$ and $S = \{Q_1, Q_2, Q_3, Q_4\}$, with $Q_1 = \{v_1, v_2\}$, $Q_2 = \{v_1, v_2, v_3\}$, $Q_3 = \{v_2, v_3, v_5\}$, $Q_4 = \{v_2, v_4, v_5\}$. If we choose the access strategy Z s.t. $P_Z(Q_1) = \frac{1}{2}$ and $P_Z(Q_2) = P_Z(Q_3) = P_Z(Q_4) = \frac{1}{6}$, then node with the highest load is v_2 with $L_Z(v_2) = \frac{1}{2} + \frac{1}{6} + \frac{1}{6} = \frac{5}{6}$, i.e. $L_Z(S) = \frac{5}{6}$. Regarding work, we have $Z_Z(S) = $\frac{1}{2} \cdot 2 + \frac{1}{6} \cdot 3 + \frac{1}{6} \cdot 3 + \frac{1}{6} \cdot 3 = \frac{15}{6}$.
If every quorum Q in a quorum system S has the same number of elements, S is called **uniform.**

Theorem 21.6: Let S be a quorum system. Then $L(S) \geq \frac{1}{\sqrt{n}}$ holds.

21.2 Grid Quorum System

Assume $\sqrt{n} \in \mathbb{N}$ and arrange the n nodes in a square matrix with side length of $\sqrt{n}$, i.e. in a grid. The basic **grid quorum system** consists of $\sqrt{n}$ quorums, with each containing a full row i and the full column i, for $i \leq i \leq \sqrt{n}$.

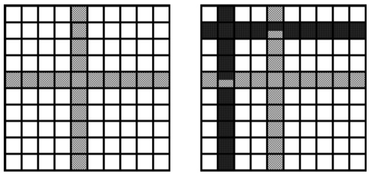

Figure 21.8: The basic version of the Grid quorum system, where each quorum Q_i with $1 \leq i \leq \sqrt{n}$ uses row i and column i. The size of each quorum is $2\sqrt{n}-1$ and two quorums overlap in exactly two nodes. Thus, when the access strategy Z is uniform (i.e., the probability of each quorum is $1/\sqrt{n}$), the work is $2\sqrt{n}-1$, and the load of every node is in $\Theta(1/\sqrt{n})$.

```
# Algorithm 21.10: Sequential Locking Strategy for a Quorum Q
Attempt to lock the nodes one by one, ordered by their identifiers
Should a node be already locked, release all locks and start over
```

> ***Theorem 21.11:*** If each quorum is accessed by Algorithm 21.10, at least one quorum will obtain a lock for all of its nodes.

```
# Algorithm 21.12: Concurrent Locking Strategy for a Quorum Q
Invariant: Let v_Q in Q be the highest identifier of a node locked by Q s.t. all nodes v_i in Q with v_i <
repeat:
    Attempt to lock all nodes of the quorum Q
    for each node v in Q that was not able to be locked by Q do:
        exchange v_Q and v_Q' with the quorum Q' that locked v
        if v_Q > v_Q' then:
            Q' releases lock on v and Q acquires lock on v
        end if
    end for
until all nodes of the quorum Q are locked
```

> ***Theorem 21.13:*** If the nodes and quorums use Algorithm 21.12, at least one quorum will obtain a lock for all of its nodes.

21.3 Fault Tolerance

If any f nodes from a quorum system S can fail s.t. there is still a quorum $Q \in S$ without failed nodes, then S is **f-resilient.** The largest such f is the **resilience** $R(S)$.

> ***Theorem 21.15:*** Let S be a grid quorum system where each of the n quorums consists of a full row and a full column. S has a resilience of $\sqrt{n}-1$.

Assume that every node works with a fixed probability p. The **failure probability** $F_p(S)$ of a quorum system S is the probability that at least one node of every quorum fails. The **asymptotic failure probability** is $F_p(S)$ for $n \to \infty$.

A version of a **Chernoff bound** states the following: Let $x_1, ..., x_n$ be independent Bernoulli-distributed random variables with $Pr[x_i = 1] = p_i$ and $Pr[x_i = 0] = 1 - p_i$, then for $X := \sum_{i=1}^{n} x_i$ and $\mu := \mathbb{E}[X] = \sum_{i=1}^{n} p_i$, then the following holds:

$$\text{for all } 0 < \delta < 1 : Pr[X \leq (1-\delta)\mu] \leq e^{-\mu\delta^2/2}.$$

> ***Theorem 21.18:*** The asymptotic failure probability of the majority quorum system is 0, for $p > \frac{1}{2}$.

> ***Theorem 21.19:*** The asymptotic failure probability of the grid quorum system is 1 for $p > 0$.

Consider $n = dhr$ nodes, arranged in a rectangular grid with $h \cdot r$ rows and d columns. Each group of r rows is a band, and r elements in a column restricted to a band are called a mini-column. A quorum consists of one mini-column in every band and one element from each mini-column of one band; thus every quorum has $d + hr - 1$ elements. The **B-grid quorum system** consists of all such quorums.

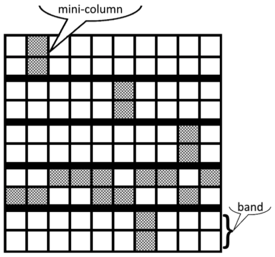

Figure 21.21: A B-Grid quorum system with $n = 100$ nodes, $d = 10$ columns, $h \cdot r = 10$ rows, $h = 5$ bands, and $r = 2$. The depicted quorum has a $d + hr - 1 = 10 + 5 \cdot 2 - 1 = 19$ nodes. If the access strategy Z is chosen uniformly, then we have a work of $d + hr - 1$ and a load of $\frac{d+hr-1}{n}$. By setting $d = \sqrt{n}$ and $r = \ln d$, we obtain a work of $\Theta(\sqrt{n})$ and a load of $\Theta(1/\sqrt{n})$.

Theorem 21.22: The asymptotic failure probability of the B-grid quorum system is 0, for $p \geq \frac{2}{3}$.

Measure	**Singleton**	**Majority**	**Grid**	**B-Grid** *
Work	1	$\simeq n/2$	$\Theta(\sqrt{n})$	$\Theta(\sqrt{n})$
Load	1	$\simeq 1/2$	$\Theta(1/\sqrt{n})$	$\Theta(1/\sqrt{n})$
Resilience	0	$\simeq n/2$	$\Theta(\sqrt{n})$	$\Theta(\sqrt{n})$
F.Prob. **	$1 - p$	$\rightarrow 0$	$\rightarrow 1$	$\rightarrow 0$

The table above shows and overview of the different quorum systems regarding resilience, work, load, and their asymptotic failure probability:

- *: Setting $d = \sqrt{n}$ and $r = \ln d$
- **: Assuming prob. $q = 1 - p$ is constant but significantly less than $\frac{1}{2}$

21.4 Byzantine Quorum Systems

Byzantine nodes make life more difficult, as they can pretend to be a regular node, i.e. one needs more sophisticated methods to deal with them. We need to ensure that the intersection of two quorums always contains a non-byzantine (correct) node and furthermore, the byzantine nodes should not be allowed to infiltrate every quorum.

A quorum system S is **f-disseminating** if (1) the intersection of two different quorums always contains $f + 1$ nodes, and (2) for any set of f byzantine nodes, there is at least one quorum without byzantine nodes.

A quorum system S is **f-masking** if (1) the intersection of two different quorums always contains $2f + 1$ nodes, and (2) for any set of f byzantine nodes, there is at least one quorum without byzantine nodes.

Theorem 21.26: Let S be an f-disseminating quorum system. Then $L(S) \geq \sqrt{(f+1)/n}$ holds.

Theorem 21.27: Let S be an f-masking quorum system. Then $L(S) \geq \sqrt{(2f+1)/n}$ holds.

A **f-masking grid quorum system** is constructed as the grid quorum system, but each quorum contains one full column and $f + 1$ rows of nodes, with $2f + 1 \leq \sqrt{n}$.

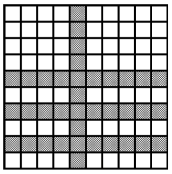

Figure 21.29: An example how to choose a quorum in the f-masking Grid with $f = 2$, i.e., $2 + 1 = 3$ rows. The load is in $\Theta(f/\sqrt{n})$ when the access strategy is chosen to be uniform. Two quorums overlap by their columns intersecting each other's rows, i.e., they overlap in at least $2f + 2$ nodes.

The **M-grid quorum system** is constructed as the grid quorum as well, but each quorum contains $\sqrt{f+1}$ rows and $\sqrt{f+1}$ columns of nodes, with $2f+1 \leq \sqrt{n}$.

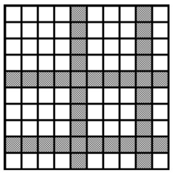

Figure 21.31: An example how to choose a quorum in the M-Grid with $f = 3$, i.e., 2 rows and 2 columns. The load is in $\Theta(\sqrt{f/n})$ when the access strategy is chosen to be uniform. Two quorums overlap with each row intersecting each other's column, i.e., $2\sqrt{f+1}^2 = 2f+2$ nodes.

The f-masking grid quorum system and the M-grid quorum system are f-masking quorum systems.

A quorum system S is ***f*-opaque** if the following two properties hold for any set of f byzantine nodes F and any two different quorums Q_1, Q_2:

$$|Q_1 \cap Q_2 \setminus F| > |(Q_1 \cap F) \cup (Q_2 \setminus Q_1)| F \cap Q = \text{ for some } Q \in S$$

Theorem 21.35: Let S be an f-opaque quorum system. Then, $f < n/5$.

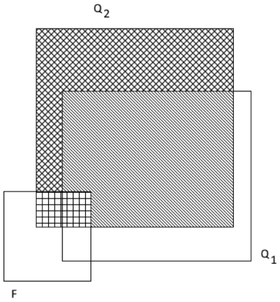

Figure 21.34: Intersection properties of an opaque quorum system. Equation (21.33.1) ensures that the set of non-byzantine nodes in the intersection of Q_1, Q_2 is larger than the set of out of date nodes, even if the byzantine nodes "team up" with those nodes. Thus, the correct up to date value can always be recognized by a majority voting.

Theorem 21.36: Let S be an f-opaque quorum system. Then $L(S) > \frac{1}{2}$ holds.

21.5 Consistent Hashing

How do you store 1 million movies, each with a size of about 1GB, on 1 million nodes, each equipped with a 1TB disk? Simply store the movies on the nodes, arbitrarily, and memorize (with a global index) which movie is stored on which node. What if the set of movies or nodes changes over time?

```
# Algorithm 9.1: Consisten Hashing
Hash the unique file name of each movie x with a known set of hash functions h_i(x) -> [0, 1), for i =
Hash the unique name of each node with the same hash function h(u) -> [0, 1)
Store a copy of movie x on node u if h_i(x) \simeq h(u), for any i. More formally, store moview x on n
```

In expectation, each node in Algorithm 9.1 stores km/n movies, where k is the number of hash functions, m is the number of different movie, and n is the number of nodes.

A version of a **Chernoff bound** states the following: Let $x_1, ..., x_n$ be independent Bernoulli-distributed random variables with $Pr[x_i = 1] = p_i$ and $Pr[x_i = 0] = 1 - p_i = q_i$, then for $X := \sum_{i=1}^{n} x_i$ and $\mu := \mathbb{E}[X] = \sum_{i=1}^{n} p_i$, then the following holds:

$$\text{for any } \delta > 0 : Pr[X \geq (1+\delta)\mu] < \Big(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\Big)^{\mu}$$

21.6 Hypercubic Networks

In this section we present a few overlay topologies of general interest.

Our virtual network should have the following properties:

- The network should be somewhat *homogeneous:* no node should play a dominant role, no node should be a single point failure.
- The nodes should have *IDs,* and the IDs should span the universe $[0, 1)$, such that we can store data with hashing, as in Algorithm 9.1.
- Every node should have a small *degree,* if possible polylogarithmic in n, the number of node.
- The network should have a small *diameter,* and routing should be easy. If a node does not have the information about a data item, then it should know which neighbor to ask.

Let $m, d \in \mathbb{N}$. The (m, d)**-mesh** $M(m, d)$ is a graph with node set $V = [m]^d$ and edge set

$$E = \Big\{\{(a_1, ..., a_d), (b_1, ..., b_d)\} \mid a_i, b_i \in [m], \sum_{i=1}^{d} |a_i - b_i| = 1\Big\},$$

where $[m]$ means the set $\{0, ..., m-1\}$. The (m, d)**-torus** $T(m, d)$ is a graph that consists of an (m, d)-mesh and additionally wrap-around edges from nodes $(a_1, ..., a_{i-1}, m-1, a_{i+1}, ..., a_d)$ to nodes $(a_1, ..., a_{i-1}, 0, a_{i+1}, ..., a_d)$ for all $i \in \{1, ..., d\}$ and all $a_j \in [m]$ with $j \neq i$. In other words, we take the expressions $a_i - b_i$ in the sum modulo m prior to computing the absolute value. $M(m, 1)$ is also called a **path,** $T(m, 1)$ a **cycle,** and $M(2, d)$ a **d-dimensional hypercube.** Figure 9.7 presents a linear array, a torus, and a hypercube:

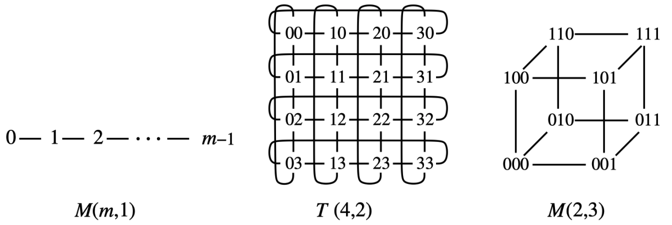

Figure 9.7: The structure of $M(m, 1)$, $T(4, 2)$, and $M(2, 3)$.

Remarks:

- Routing on a mesh, torus, or hypercube is trivial. On a d-dimensional hypercube, to get from a source bitstring s to a target bitstring t one only needs to fix each "wrong" bit, one at a time; in other words, if the source and the target differ by k bits, there are $k!$ routes with k hops.
- As required, the d-bit IDs of the nodes need to be mapped to the universe $[0, 1)$. One way to do this is by turning each ID into a fractional binary representation. For example, the ID 101 is mapped to 0.101_2 which has a decimal value of $0 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} = \frac{5}{8}$.

Let $d \in \mathbb{N}$. The **d-dimensional butterfly** $BF(d)$ is a graph with node set $V = [d+1] \times [2]^d$ and an edge set $E = E_1 \cup E_2$ with

$$E_1 = \{\{(i, \alpha), (i+1, \alpha)\} \mid i \in [d], \alpha \in [2]^d\}$$

and

$$E_2 = \{\{(i, \alpha), (i+1, \beta)\} \mid i \in [d], \alpha, \beta \in [2]^d, \alpha \oplus \beta = 2^i\}$$

A node set $\{(i, \alpha) \mid \alpha \in [2]^d\}$ is said to form **level** i of the butterfly. The **d-dimensional wrap-around butterfly** W-BF(d) is defined by taking the $BF(d)$ and having $(d, \alpha) = (0, \alpha)$ for all $\alpha \in [2]^d$.

Remarks: - Figure 9.9 shows the 3-dimensional butterfly $BF(3)$. The $BF(d)$ has $(d+1)2^d$ nodes, $2d \cdot 2^d$ edges and maximum degree 4. It is not difficult to check that if for each $\alpha \in [2]^d$ we combine the nodes $\{(i, \alpha) \mid i \in [d+1]\}$ into a single node then we get back the hypercube. - Butterflies have the advantage of a constant node degree over hypercubes, whereas hypercubes feature more fault-tolerant routing.

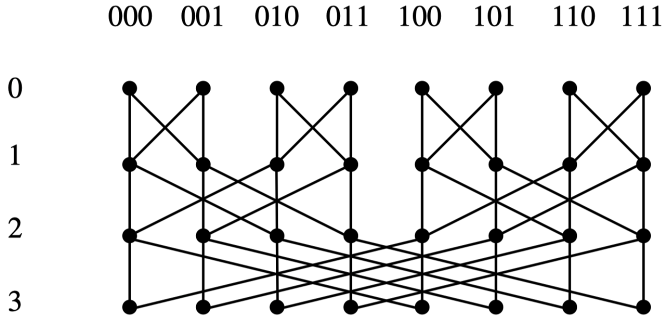

Figure 9.9: The structure of BF(3).

Let $d \in \mathbb{N}$. The **cube-connected-cycles** network $CCC(d)$ is a graph with node set $V = \{(a,\, p) \,|\, a \in [2]^d,\, p \in [d]\}$ and edge set

$$E = \{\{(a,\, p),\, (a,\, (p+1) \mod d)\} \,|\, a \in [2]^d,\, p \in [d]\} \cup \{\{(a,\, p),\, (b,\, p)\} \,|\, a,\, b \in [2]^d,\, p \in [d],\, a \oplus b = 2^p\}.$$

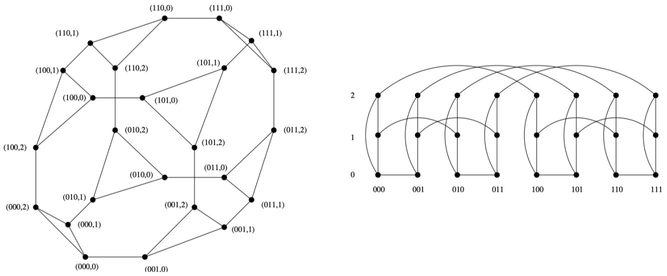

Figure 9.11: The structure of CCC(3).

Let $d \in \mathbb{N}$. The d**-dimensional shuffle-exchange** $SE(d)$ is defined as an undirected graph with node set $V = [2]^d$ and an edge set $E = E_1 \cup E_2$ with

$$E_1 = \{\{(a_1, ..., a_d).\, (a_1, ..., \bar{a}_d)\} \,|\, (a_1, ..., a_d) \in [2]^d,\, \bar{a}_d = 1 - a_d\}$$

and

$$E_2 = \{\{(a_1, ..., a_d),\, (a_d,\, a_1, ..., a_{d-1})\} \,|\, (a_1, ..., a_d) \in [2]^d\}.$$

Figure 9.13 shows the 3- and 4-dimensional shuffle-exchange graph.

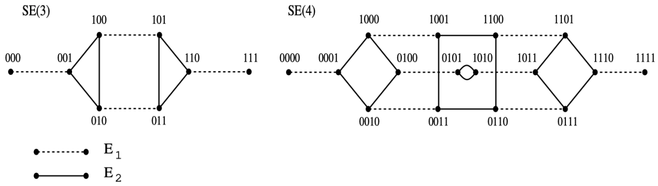

Figure 9.13: The structure of SE(3) and SE(4).

The b**-ary DeBruijn graph of dimension** d $DB(b,\, d)$ is an undirected graph $G = (V,\, E)$ with node set $V = [b]^d$ and edge set $E = \{\{(a_1, ..., a_d),\, (x,\, a_1, ..., a_{d-1}) \,|\, (a_1, ..., a_d) \in [b]^d,\, x \in [b]\}\}$.

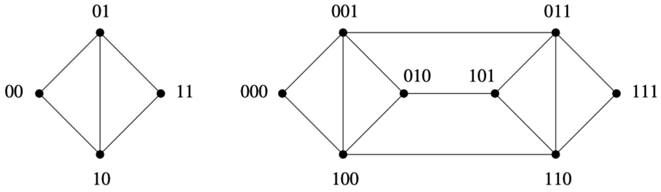

Figure 9.15: The structure of $DB(2,2)$ and $DB(2,3)$.

The **skip list** is an ordinary ordered linked list of objects, augmented with additional forward links. The ordinary linked list is the level 0 of the skip list. In addition, every object is promoted to level 1 with probability $\frac{1}{2}$. As for level 0, all level 1 objects are connected by a linked list. In general, every object on level i is promoted to the next level with probability $\frac{1}{2}$. A special start-object points to the smallest/first object on each level.

> ***Theorem 9.17:*** Every graph of maximum degree $d > 2$ and size n must have a diameter of at least $\lceil (\log n)/(\log(d-1)) \rceil - 2$.

21.7 DHT & Churn

A **distributed hash table (DHT)** is a distributed data structure that implements a distributed storage. A DHT should support at least (i) a search for a key and (ii) an insert (key, object) operation, possibly also (iii) a delete key operation.

Remarks:

- A DHT has many applications beyond storing movies, e.g. the Internet domain name system (DNS) is essentially a DHT.
- A DHT can be implemented as a hypercubic overlay network with nodes having identifiers such that they span the ID space [0, 1).
- A hypercube can directly be used for a DHT. Just use a globally known set of hash functions $h_i$, mapping movies to bit strings with d bits.

```
# Algorithm 9.19: DHT
Given: a globally known set of hash functions $h_i$, and a hypercube (or any other hypercubic network)
Each hypercube virtual node ("hypernode") consists of O(log n) nodes
Nodes have connections to all other nodes of their hypernode and to nodes of their neighboring hyperno
Because of churn, some of the nodes have to change to another hypernode such that up to constant facto
If the total number of nodes n grows or shrinks above or below a certain threshold, the dimension of t
```

We have a fully scalable, efficient distributed storage system which tolerates $O(\log n)$ worst-case joins and/or crashes per constant time interval. As in other storage systems, nodes have $O(\log n)$ overlay neighbors, and the usual operations take time $O(\log n)$.