ETH zürich Distributed Computing

Slides last updated:
26.11.24

Computer Systems

Exercise Session 10
HS 2024

Program

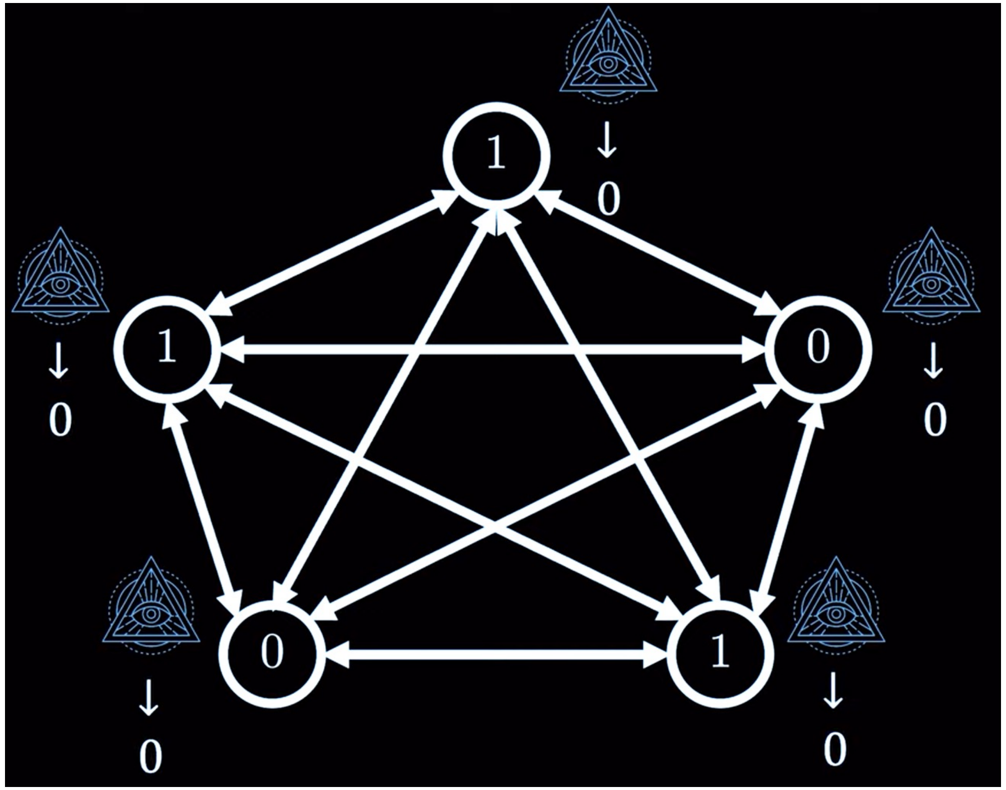
1. Chapter 19 – Shared Coins
2. Chapter 20 – Quorum Systems
3. Algorithms overview
4. Assignment preview

ETH zürich Distributed Computing

Chapter 19 – Shared Coins

Shared Coins – Motivation

- Worst-case runtime of our randomized algorithms were limited by the probability that all nodes sample the same value in a round
- **Idea:** use random oracle such that all nodes always sample the same value
- **Gain:** constant expected number of rounds in asynchronous byzantine agreement algorithm (Ben-Or)
- **Major problem:** random oracles do not exist
- **Solution:** shared coin algorithm
  - Outputs 0 or 1 with constant probability



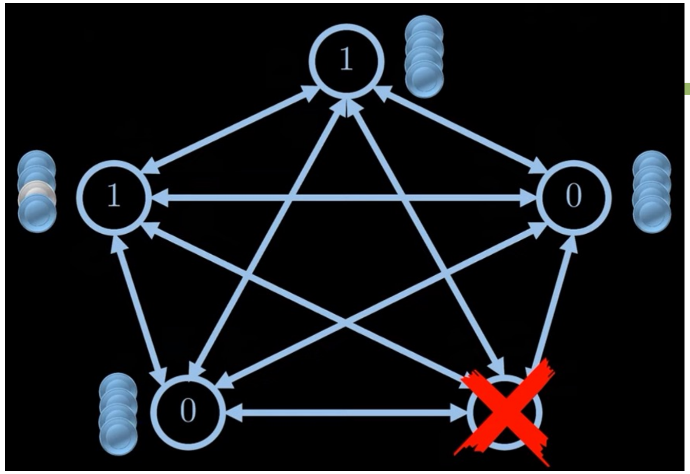
Shared Coin Algorithm ($f < {}^{n}/_{3}$)

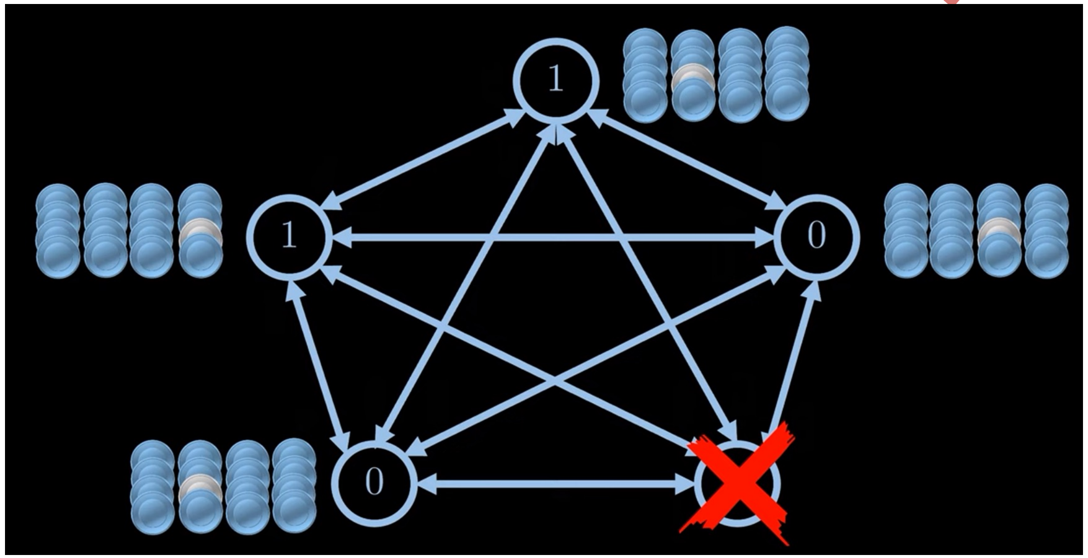
- Every node samples a biased coin
  - Samples 0 with probability ${}^{1}/_{n}$
- Nodes share their coin sets
- Success probabilities:
  - 0 with probability $1 - (1 - e)^{1/3} \approx 0.28$
  - 1 with probability ${}^{1}/_{e} \approx 0.37$
- Let W be the set of coins that a node receives from $f + 1$ different coin sets
  - W always contains coins from at least $f + 1$ distinct nodes

Algorithm 19.8 Shared Coin (code for node u)

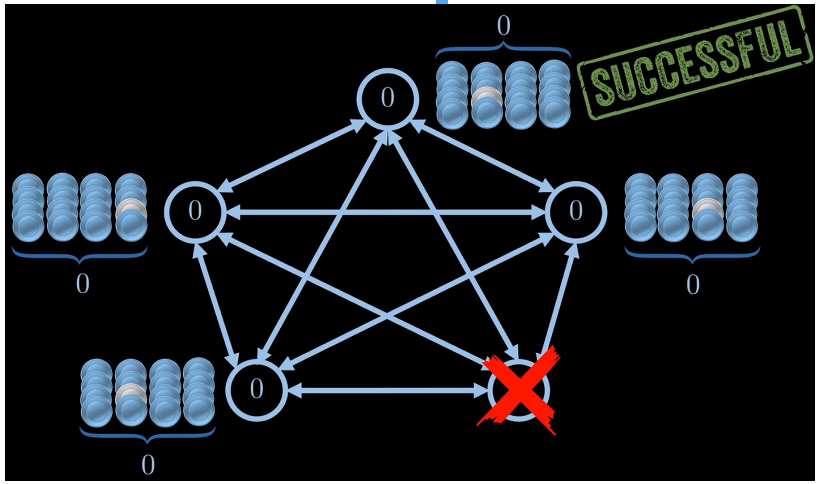
```
Choose local coin c_u = 0 with probability 1/n, else c_u = 1
Broadcast myCoin(c_u)
Wait for n − f coins and store them in the local coin set C_u
Broadcast mySet(C_u)
Wait for n − f coin sets
if at least one coin is 0 among all coins in the coin sets then
   return 0
else
   return 1
end if
```

Shared Coin Algorithm ($f < n/3$)





1. Choose local coin $c_u = 0$ with probability $1/n$, else $c_u = 1$
2. Broadcast $\texttt{myCoin}(c_u)$
3. Wait for $n - f$ coins and store them in the local coin set C_u
4. Broadcast $\texttt{mySet}(C_u)$
5. Wait for $n - f$ coin sets
6. **if** at least one coin is 0 among all coins in the coin sets **then**
7.   return 0
8. **else**
9.   return 1
10. **end if**



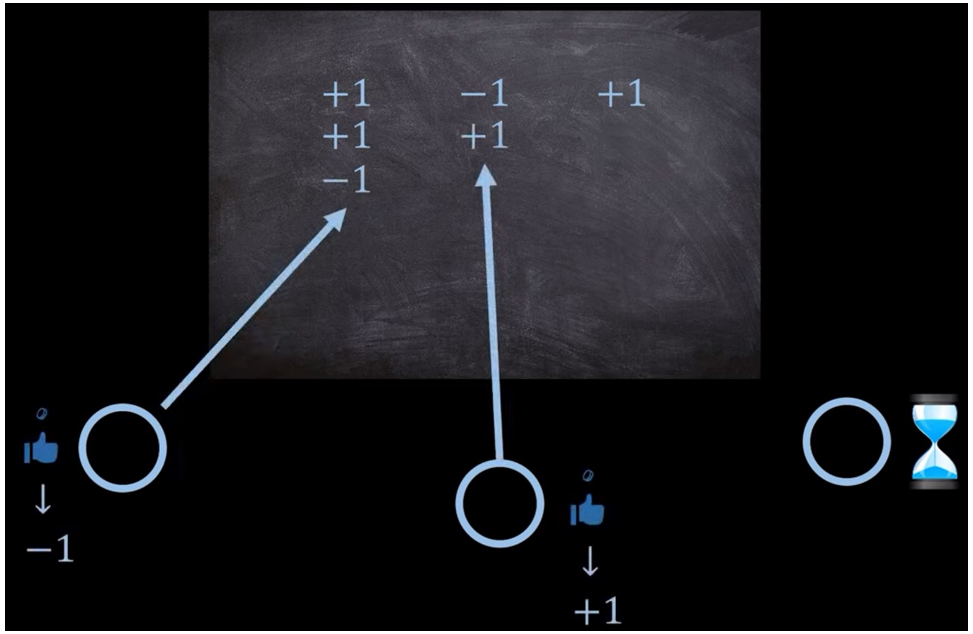
Blackboard Algorithm ($f < n$)

- Nodes write result of n^2 fair coin flips to a blackboard
- Outcome is the sign of the sum after a node sees $\geq n^2$ results
- Outcome will be the same for all nodes if $|\text{sum}(C)| > n$
- **Probability** that all nodes have the same outcome (0 or 1) is at least
  $$1 - \Phi(1) > 0.15$$
  - Proof by applying Central Limit Theorem

- What is the **problem** of this approach?
  - It requires a trusted central authority

Algorithm 19.15 Crash-Resilient Shared Coin with Blackboard (for node u)

while true **do**
    Choose new local coin $c_u = +1$ with probability $1/2$, else $c_u = -1$
    Write c_u to the blackboard
    Set C = Read all coin flips on the blackboard
    if $|C| \geq n^2$ **then**
        return sign(sum(C))
    end if
end while



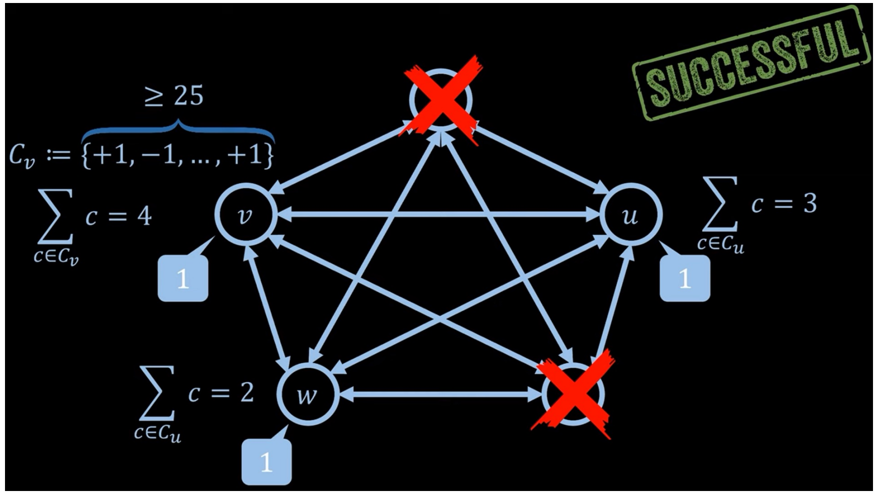
Message passing algorithm ($f < {}^{n}/_{2}$)

- Using FIFO broadcast to replace trusted central authority
- **Gain:** Save against crash failures and worst-case scheduling

- What is the problem of this algorithm?
  - It doesn't work with byzantine nodes
  - It has higher communication complexity

Algorithm 19.19 Crash-Resilient Shared Coin (code for node u)

$\mathrm{r} = 1$
while true **do**
    Choose local coin $c_u = +1$ with probability $1/2$, else $c_u = -1$
    FIFO-broadcast $\texttt{coin}(c_u, r)$ to all nodes
    Save all received coins $\texttt{coin}(c_v, r)$ in a set C_u
    Wait until accepted own $\texttt{coin}(c_u, r)$
    Request C_v from $n - f$ nodes v, and add newly seen coins to C_u
    if $|C_u| \geq n^2$ **then**
        return $\mathrm{sign}(\mathrm{sum}(C_u))$
    end if
    $\mathrm{r} := \mathrm{r} + 1$
end while



Secret Sharing algorithm ($f < t$)

- Generate random polynomial and distribute distinct points on it to all nodes
- Can reconstruct secret value at $p(0)$ using t points

- Why does this work?
  - Any polynomial of degree $t-1$ can be reconstructed using t points
    - Also works over some finite field $\mathbb{F}_q$
- What can go wrong?
  - Dealer must be a trusted central authority
  - Shares must be distributed securely using a private communication channel

Algorithm 19.23 (t, n)-Threshold Secret Sharing

1: Input: A secret $s \in \{0, \ldots, q\}$ for some prime number $q > n$.

Secret distribution by dealer d

2: Generate $t-1$ uniformly random values $a_1, \ldots, a_{t-1} \in \mathbb{F}_q$
3: Obtain a polynomial p of degree $t-1$ with $p(x) = s + a_1 x + \cdots + a_{t-1}x^{t-1}$
4: Distribute share $\texttt{msg}(p(1))_d$ to node v_1, …, $\texttt{msg}(p(n))_d$ to node v_n

Secret recovery

5: Collect t shares $\texttt{msg}(p(u))_d$ from at least t nodes
6: Use Lagrange's interpolation formula to obtain $p(0) = s$

Algorithm 19.24 Preprocessing Step for Algorithm 19.25 (code for dealer d)

1: **for** $i = 1, \ldots, \lambda$ **do**
2: Choose coin flip c_i, where $c_i = 0$ with probability $1/2$, else $c_i = 1$
3: Using Algorithm 19.23, generate n shares $(p(1)), \ldots, p(n))$ for c_i
4: **end for**
5: Send shares $\texttt{msg}(p(1))_d, \ldots, \texttt{msg}(p(n))_d$ to node u

Algorithm 19.25 Shared Coin using Secret Sharing

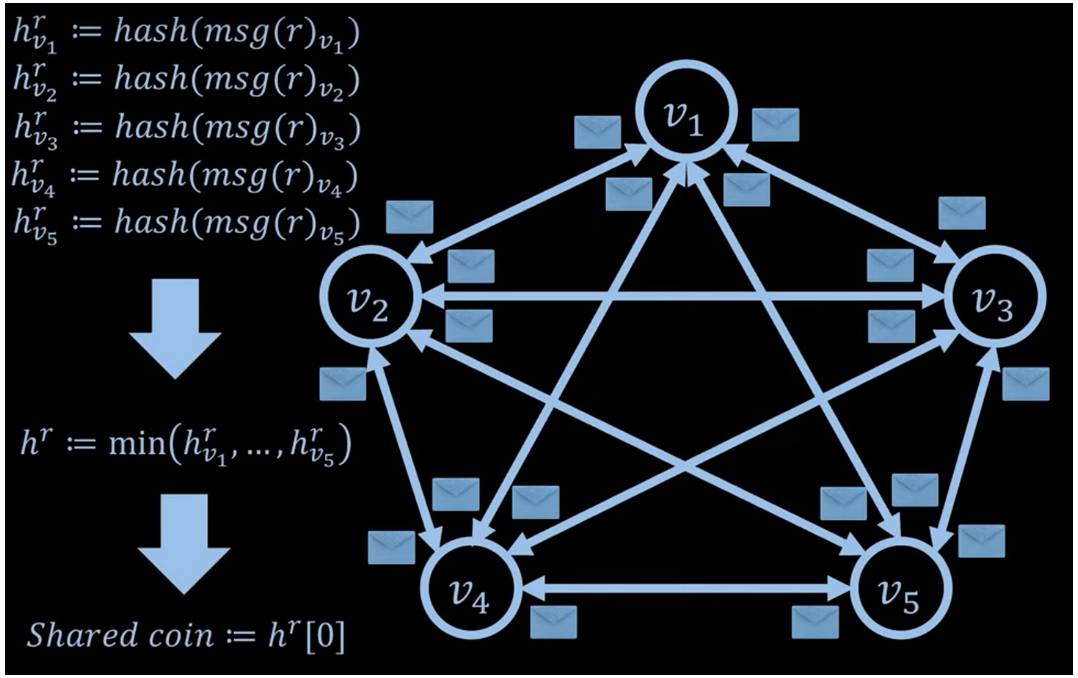
1: Request shares for c_i from at least $f+1$ nodes
2: Using Algorithm 19.23, let c_i be the value reconstructed from the shares
3: **return** c_i

Synchronous Byzantine Shared Coin ($f < n/3$)

- Broadcast current round number signed with private key
- Decide on LSB of smallest hash received
- Uses signatures
  - Every node can only broadcast a single value as inconsistency will be detected because of signatures
  - It must be hard to compute different signatures for the same message
- What is the drawback of this approach?
  - It requires cryptographically strong hash functions

Algorithm 19.28 Simple Synchronous Byzantine Shared Coin (for node u)

1: Each node has a public key that is known to all nodes.
2: Let r be the current round of Algorithm 17.19
3: Broadcast $\texttt{msg}(r)_u$, i.e., round number r signed by node u
4: Compute $h_v = \text{hash}(\texttt{msg}(r)_v)$ for all received messages $\texttt{msg}(r)_v$
5: Let $h_{min} = \min_v h_v$
6: **return** least significant bit of h_{min}



How to use these shared coin algorithms?

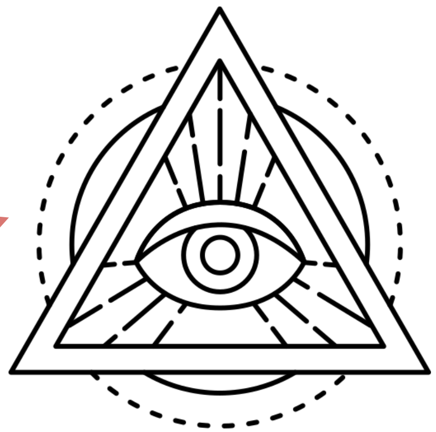
- We can replace the sampling for random values with an instance of a shared coin algorithm
- Which algorithm to use depends on our setting

Async. Byzantine Agreement with Random Oracle ($f < n/10$)

- Replace standard coin flip by the shared coin algorithm
- **Gain:** Runtime becomes constant
- Proofs for validity and agreement still hold
- The proof for termination must be changed to account for the changed probability that all coins will give the same result

Algorithm 17.19 Asynchronous Byzantine Agreement (Ben-Or, for $f < n/10$)

```
x_u ∈ {0,1}                    ◁ input bit
round = 1                      ◁ round
while true do
   Broadcast propose(x_u,round)
   Wait until n − f propose messages of current round arrived
   if > n/2 + 3f propose messages contain same value x then
      Broadcast propose(x,round + 1)
      Decide for x and terminate
   else if > n/2 + f propose messages contain same value x then
      x_u = x
   else
      choose x_u randomly, with Pr[x_u = 0] = Pr[x_u = 1] = 1/2
   end if
   round = round + 1
end while
```



Icon from https://www.flaticon.com/free-icon/eye_14432782

Randomized Consensus with Shared Coin ($f < {}^{n}/_{3}$)

- Replace simple coin flip by the shared coin algorithm
- **Gain:** Termination in expected 3 rounds
- **Drawback:** Can only deal with $f < {}^{n}/_{3}$ crashes instead of $f < {}^{n}/_{2}$ crashes

Algorithm 16.28 Randomized Consensus (assuming $f < n/2$)

$v_i \in \{0,1\}$ ◁ input bit
round $= 1$
while true **do**
  Broadcast **myValue**(v_i, round)
  Propose
  Wait until a majority of **myValue** messages of current round arrived
  if all messages contain the same value v **then**
    Broadcast **propose**(v, round)
  else
    Broadcast **propose**($\perp$, round)
  end if
  Vote
  Wait until a majority of **propose** messages of current round arrived
  if all messages propose the same value v **then**
    Broadcast **myValue**(v, round + 1)
    Broadcast **propose**(v, round + 1)
    Decide for v and terminate
  else if there is at least one proposal for v **then**
    $v_i = v$
  else
    Choose v_i randomly, with $Pr[v_i = 0] = Pr[v_i = 1] = 1/2$
  end if
  round = round + 1
end while

Algorithm 19.8 Shared Coin (code for node u)

Choose local coin $c_u = 0$ with probability $1/n$, else $c_u = 1$
Broadcast **myCoin**(c_u)
Wait for $n - f$ coins and store them in the local coin set C_u
Broadcast **mySet**(C_u)
Wait for $n - f$ coin sets
if at least one coin is 0 among all coins in the coin sets **then**
  return 0
else
  return 1
end if

Byzantine Agreement using Secret Sharing ($f < {}^{n}/_{10}$)

Distributed Computing

- Uses Shared Coin using Secret Sharing algorithm
- Terminates in 3 rounds in expectation
- It can be shown that Asynchronous Byzantine Agreement algorithm to requires only λ shared coins with probability $2^{-\lambda}$

Algorithm 17.19 Asynchronous Byzantine Agreement (Ben-Or, for $f < n/10$)

$x_u \in \{0, 1\}$ ◁ input bit
round = 1 ◁ round
while true **do**
  Broadcast **propose**(x_u,round)
  Wait until $n - f$ **propose** messages of current round arrived
  if $> n/2 + 3f$ **propose** messages contain same value x **then**
    Broadcast **propose**(x,round + 1)
    Decide for x and terminate
  else if $> n/2 + f$ **propose** messages contain same value x **then**
    $x_u = x$
  else
    choose x_u randomly, with $Pr[x_u = 0] = Pr[x_u = 1] = 1/2$
  end if
  round = round + 1
end while

Algorithm 19.25 Shared Coin using Secret Sharing

Request shares for c_i from at least $f + 1$ nodes
Using Algorithm 19.23, let c_i be the value reconstructed from the shares
return c_i

ETH zürich

Byzantine Agreement using Synchronous Shared Coin ($f < {}^{n}/_{10}$)

Distributed Computing

- Uses Synchronous Byzantine Shared Coin
- Takes $2\,{}^{2}/_{9}$ rounds in expectation

Algorithm 17.19 Asynchronous Byzantine Agreement (Ben-Or, for $f < n/10$)

$x_u \in \{0,1\}$ ◁ input bit
round $= 1$ ◁ round
while true **do**
  Broadcast `propose`(x_u,round)
  Wait until $n - f$ `propose` messages of current round arrived
  if $> n/2 + 3f$ `propose` messages contain same value x **then**
    Broadcast `propose`(x,round $+ 1$)
    Decide for x and terminate
  else if $> n/2 + f$ `propose` messages contain same value x **then**
    $x_u = x$
  else
    choose x_u randomly, with $Pr[x_u = 0] = Pr[x_u = 1] = 1/2$
  end if
  round $=$ round $+ 1$
end while

Algorithm 19.28 Simple Synchronous Byzantine Shared Coin (for node u)

Each node has a public key that is known to all nodes.
Let r be the current round of Algorithm 17.19
Broadcast $\texttt{msg}(r)_u$, i.e., round number r signed by node u
Compute $h_v = \text{hash}(\text{msg}(r)_v)$ for all received messages $\texttt{msg}(r)_v$
Let $h_{min} = \min_v \ h_v$
return least significant bit of h_{min}

ETH zürich

Fast Synchronous Byzantine Agreement ($f < {}^{n}/_{4}$)

- 2 communication rounds per iteration
- `coin_toss` is the min hash algorithm from the simple synchronous algorithm
- Termination in $5\,{}^{3}/_{4}$ rounds in expectation
- Success probability

$$\Pr[C = 0] = \Pr[C = 1] > {}^{27}/_{64}$$

Algorithm 19.30 Fast Synchronous Byzantine Agreement

$x_u \in \{0, 1\}$.
while *true* **do**
  broadcast `propose`(x_u)
  $x_u :=$ most frequently received value
  if $\geq n - f$ `propose` messages contain the same value x_u **then**
    decide on x_u
    broadcast `propose`$(x_u, \mathit{decided})$
    terminate
  else
    broadcast `propose`(x_u)
    $x_u :=$ most frequently received value
    if $< n - f$ `propose` messages contain the same value x **and** `coin_toss()` $= 0$ **then**
      $x_u := 0$
    end if
  end if
end while

Shared Coins Quiz

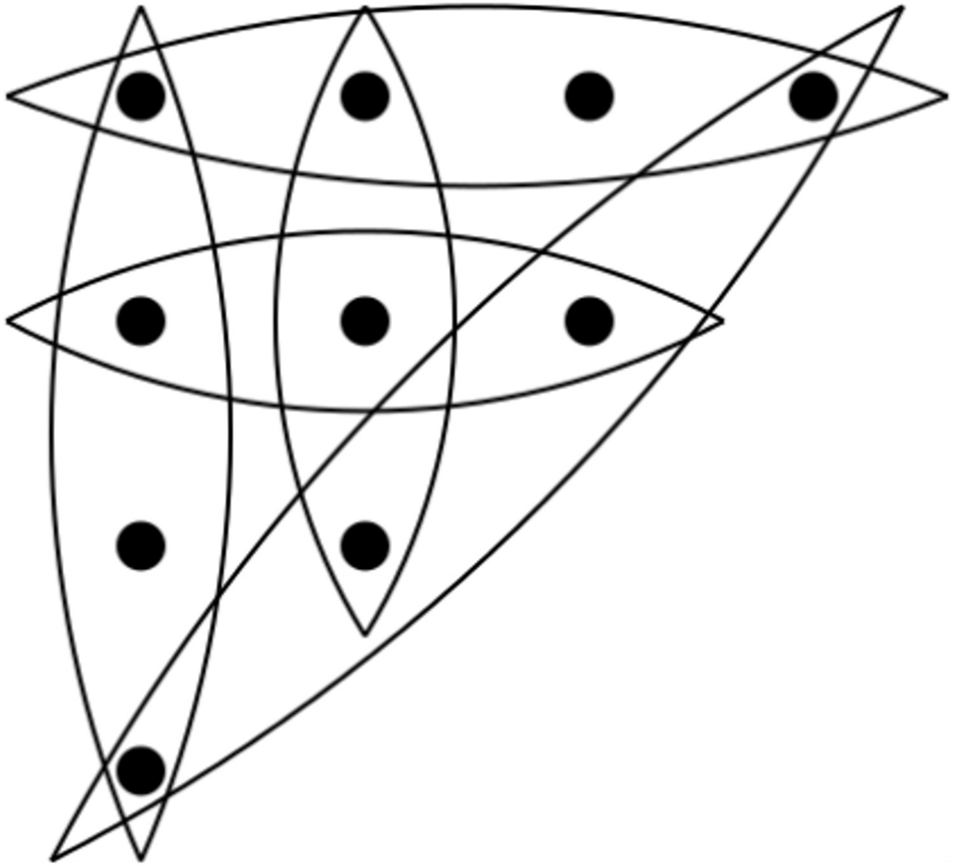
1. Let p_0 be the probability that a shared coin algorithm outputs 0 and p_1 the probability that it outputs 1. Do we always have $p_0 + p_1 = 1$?
   - No. A shared coin algorithm is allowed to fail with constant probability.
2. We can use a shared coin in the asynchronous randomized consensus algorithm (16.28). How many of the nodes are allowed to crash?
   - $f < {}^{n}/_{3}$. Even though the randomized consensus algorithm can handle $f < {}^{n}/_{2}$ crashes, the shared coin algorithm tolerates $f < {}^{n}/_{3}$ crash failures.
3. In the shared secret algorithm: Can we approximate the secret value using $t - 1$ values?
   - No. If we have $t - 1$ values of p we still have one degree of freedom. Therefore, $p(0)$ still can take on any value.
4. Is the Fast Synchronous Byzantine Agreement Algorithm optimal regarding expected number of rounds?
   - It depends. With a random oracle (that do not exist) it would be possible to achieve agreement in expected 6 rounds.

ETH zürich Distributed Computing

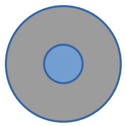
Chapter 20 – Quorum Systems

Quorum Systems

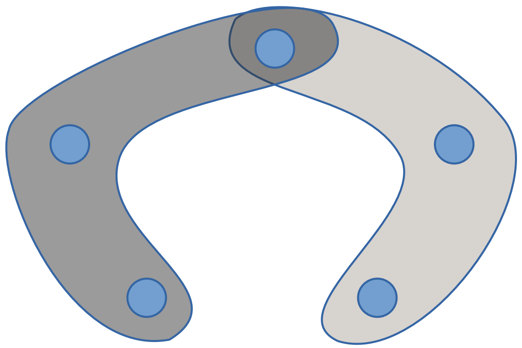
- Get a lock using a quorum system:
  - Client selects a free quorum
  - Requests lock from all nodes of the quorum
  - Client releases all locks
- Must make sure to request locks in a predefined order
- This example is **not** a quorum system. Why?
  - The two vertical quorums do not share a node



Singleton and Majority Quorum Systems



Singleton quorum system



Majority quorum system
(all sets consist of $n / 2 + 1$ nodes)

Load and Work

- An **access strategy** Z defines the probabilities $P_Z(Q)$ of accessing a quorum $Q \in S$ such that:

$$\sum_{Q \in S} P_Z(Q) = 1$$

Quorum metrics

- **Work:** How many servers need to be accessed
- **Load:** Workload of busiest server
- **Resilience:** Largest number of servers that can fail such that there still exists a complete quorum
- **Failure probability:** Probability that at least one server of every quorum fails, assuming every server works with probability p
- **Asymptotic failure probability:** Failure probability for $n \to \infty$

Load and Work

Load of access strategy Z on a node v_i $L_Z(v_i) = \sum_{Q \in S; v_i \in Q} P_Z(Q)$

Load induced by Z on quorum system S $L_Z(S) = \max_{v_i \in S} L_Z(v_i)$

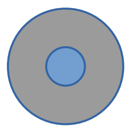
Load of quorum system S $L(S) = \min_Z L_Z(S)$

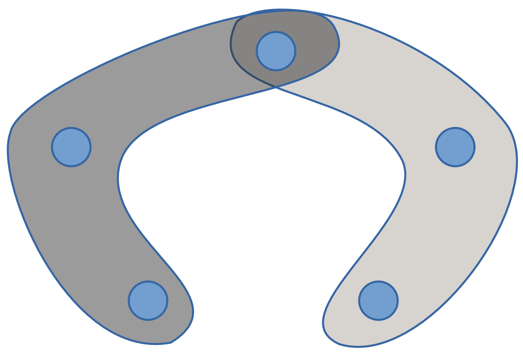
Work of quorum Q $W(Q) = |Q|$

Work induced by Z on quorum system S $W_Z(S) = \sum_{Q \in S} P_Z(Q) \cdot W(Q) = \sum_{v \in V} L_Z(v)$

Work of quorum system S $W(S) = \min_Z W_Z(S)$

Load and Work



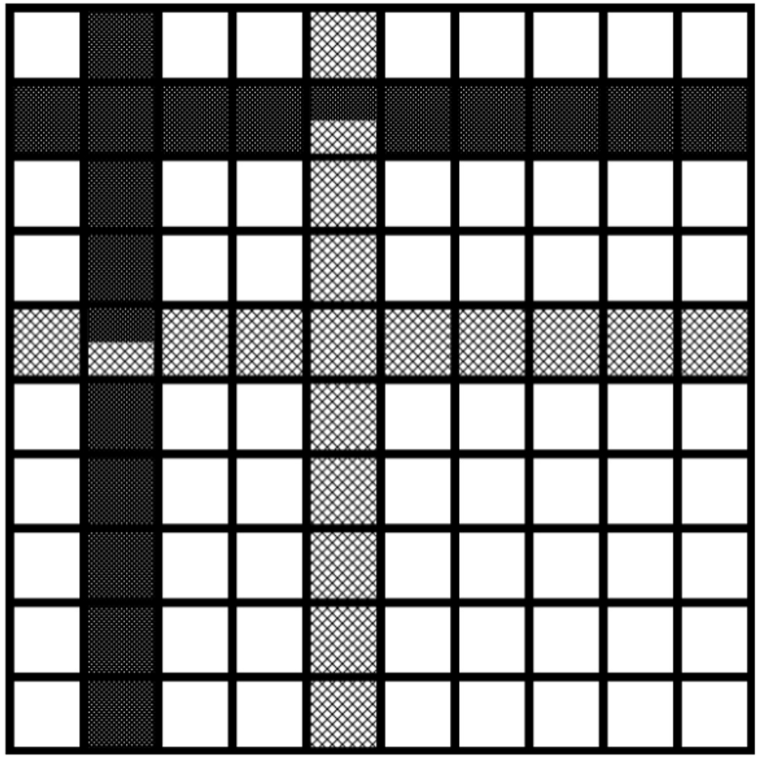


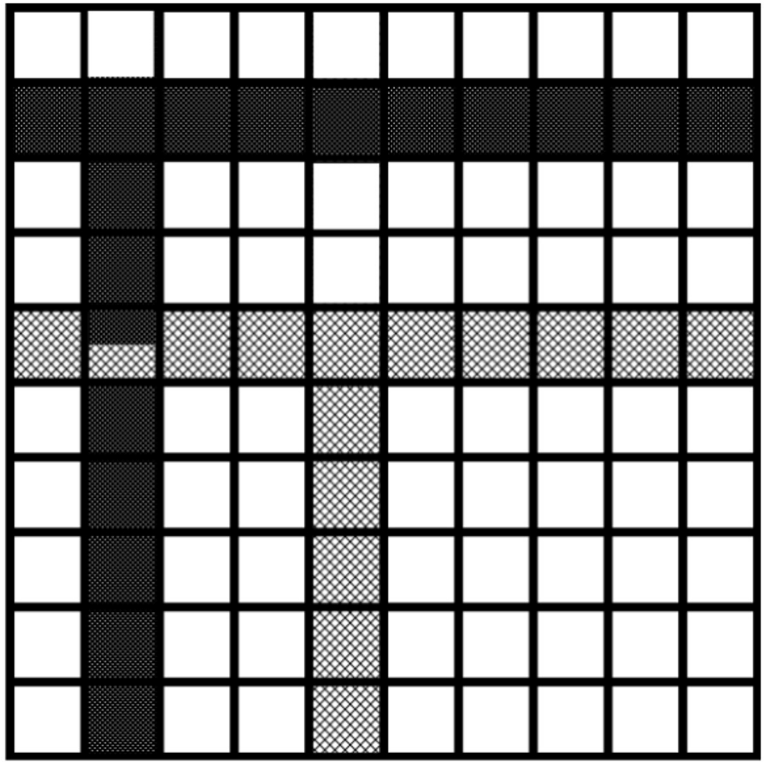
Singleton quorum system

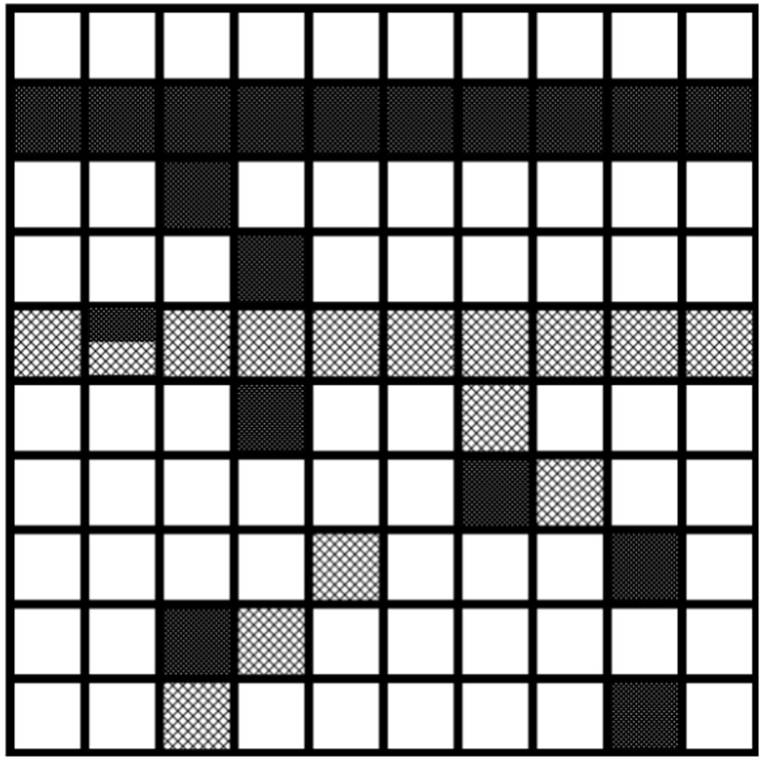
Majority quorum system
(all sets consist of $n / 2 + 1$ nodes)

	Singleton	Majority
How many servers need to be contacted? **(Work)**	1	> n/2
What’s the load of the busiest server? **(Load)**	100%	≈ 50%
How many server failures can be tolerated? **(Resilience)**	0	< n/2

Grid Idea ($n = d^2$)







Size of quorum: $2d - 1$ $\leq 2d - 1$ $\leq 2d - 1$

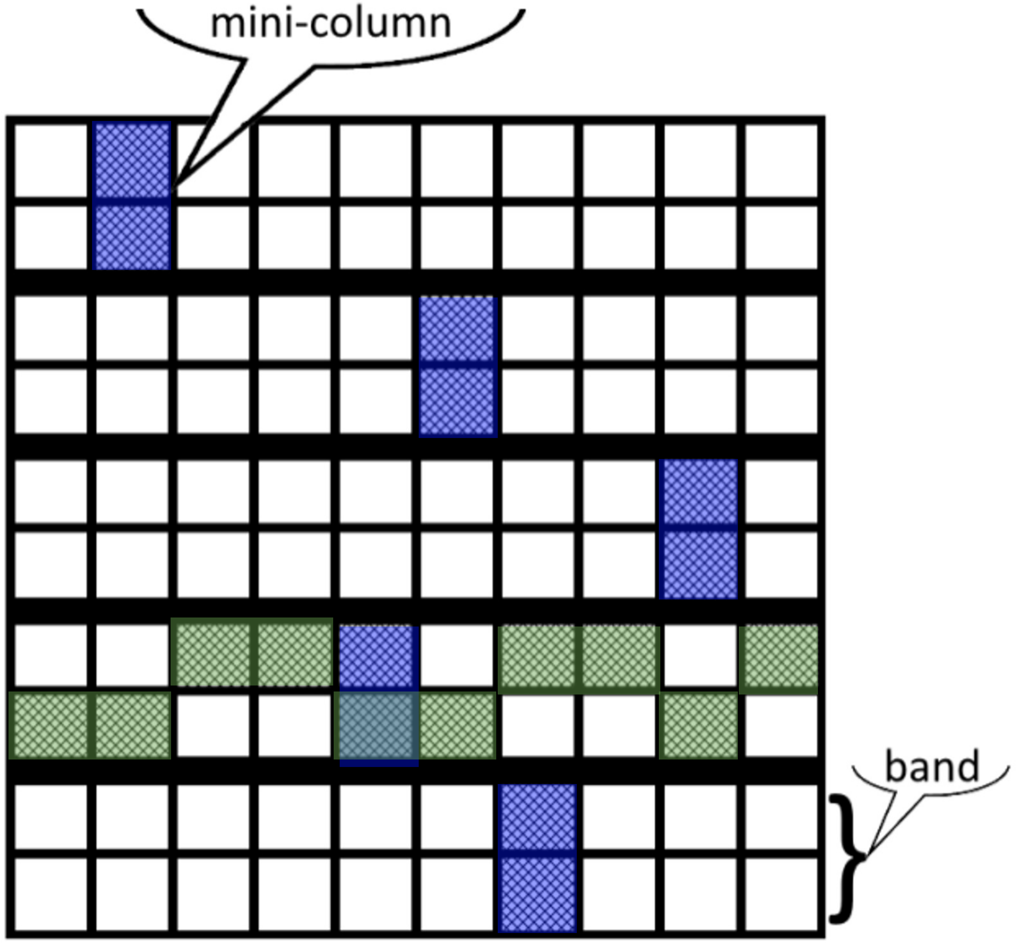
# of intersects: 2 1 1

Problem: Failure probability $F_p(\mathcal{S}) \geq \Pr[\text{at least one failure per row}] = (1 - p^d)^d \geq 1 - ndp^d \xrightarrow{n\to\infty} 1$

B-Grid Quorum System

- Nodes arranged in rectangular grid with $h \cdot r$ rows and d columns
- **Band:** Group of r rows
- **Mini-column:** Group of r elements in the same column and band
- A quorum consists of
  - one mini-column in every band
  - for one band: one element from each mini-column

- **Size of quorum:** $|Q| = r \cdot h + d - 1$



Different quorum systems

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

- Theorem: for any quorum system S we have $\mathrm{load}(S) \geq \frac{1}{\sqrt{n}}$
  - Load of Grid is asymptotically optimal

Quorum System Quiz

1. Does a quorum system exist which can tolerate that all nodes of a specific quorum fail?
   - No, because any two quorums intersect. So, when one quorum fails, all quora fail
2. Consider a quorum system, which is made up of n different quorums, each containing $n-1$ servers. We use a uniform access strategy: $P_Z(Q) = \frac{1}{n}$. What is the work, load, and resilience?
   - Work: $n-1$ Because all quora are of the same size and the access strategy is uniform
   - Load: $\frac{n-1}{n}$ For all nodes v_i we have: $\sum_{Q \in S; v_i \in Q} P_Z(Q) = \sum_{Q \in S; v_i \in Q} \frac{1}{n}$
   - Resilience: 1 If 2 nodes fail, all quora fail
3. Can you think of a quorum system that contains as many quorums as possible?
   Hint: it does not have to be minimal.
   - Construction: Pick a node v and create quorum for all possible sets containing that one node v.
     - All quora share at least one node. Which one?
     - Maximality (informally): of any quorum Q and its complement $\overline{Q}$ at most one quorum can be in the system. This system maximizes this number.

ETH zürich Distributed Computing

Algorithms Overview

Shared Coin Algorithms

	Shared Coins (19.8)	Blackboard (19.15)	Message Passing (19.19)	Secret Sharing (19.23)	Synchronous Shared Coin (19.28)
Number of failures	$f < {}^{n}/_{3}$	$f < n$	$f < {}^{n}/_{2}$	$f < n$	$f < {}^{n}/_{3}$
Min number of nodes	$3f + 1$	$f + 1$	$2f + 1$	$f + 1$	$3f + 1$
With byzantine nodes?	❌	❌	❌	✅ (unless it's the dealer)	✅
Asynchronous model?	✅	✅	✅	✅	❌
Runtime (worst-case)	n^{f+1} rounds	n^2 round	n^2 rounds	1 round	$2\,{}^{4}/_{7}$ rounds
Success probabilities	$\mathrm{P}[C = 0] \approx 0.28$ $\mathrm{P}[C = 1] \approx 0.37$	$\mathrm{P}[C = 0] > 0.15$ $\mathrm{P}[C = 1] > 0.15$	$\mathrm{P}[C = 0] > 0.15$ $\mathrm{P}[C = 1] > 0.15$	$\mathrm{P}[C = 0] = {}^{1}/_{2}$ $\mathrm{P}[C = 1] = {}^{1}/_{2}$	$\mathrm{P}[C = 0] = {}^{7}/_{18}$ $\mathrm{P}[C = 1] = {}^{7}/_{18}$
Drawbacks	Exponential expected runtime	Requires trusted central authority	Not robust against byzantine behaviour	Requires trusted dealer and/or cryptography	Only in the synchronous setting

Byzantine Agreement Algorithms with Shared Coins

	Byzantine Agreement with Random Oracle (19.2)	Byzantine Agreement using Secret Sharing (19.25)	Synchronous Algorithm using Shared Coin (19.28)	Fast Synchronous Byzantine Agreement (19.30)
Number of failures	$f < n/10$	$f < n/10$	$f < n/10$	$f < n/4$
Min number of nodes	$10f + 1$	$10f + 1$	$10f + 1$	$4f + 1$
With byzantine nodes?	✅	✅	✅	✅
Asynchronous model?	✅	✅	❌	❌
Expected runtime	3 rounds	3 rounds	$3 + 2/9$ rounds	$< 5\,3/4$ rounds
Success probabilities of shared coin	$\Pr[C = 0] = 1/2$ $\Pr[C = 1] = 1/2$	$\Pr[C = 0] = 1/2$ $\Pr[C = 1] = 1/2$	$\Pr[C = 0] = 9/2 \cdot 10$ $\Pr[C = 1] = 9/2 \cdot 10$	$\Pr[C = 0] > 27/64$ $\Pr[C = 1] > 27/64$
Shortcomings	Random Oracles do not exist	Requires trusted dealer and/or cryptography	Analysis uses Random Oracle Model	Uses cryptography

ETH zürich Distributed Computing

Assignment Preview

Task 1 – Quorum Systems

1.1 The Resilience of a Quorum System

a) Does a quorum system exist, which can tolerate that all nodes of a specific quorum fail? Give an example or prove its nonexistence.

b) Consider the *nearly all* quorum system, which is made up of n different quorums, each containing $n-1$ servers. What is the resilience of this quorum system?

c) Can you think of a quorum system that contains as many quorums as possible?
Note: the quorum system does not have to be minimal.