Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Systems@ETH Zürich

Autumn Term 2020 Prof. T. Roscoe, Prof. R. Wattenhofer

Computer Systems

— Solution to Assignment 10 —

1 Quorum Systems

1.1 The Resilience of a Quorum System

a) No such quorum system exists. According to the definition of a quorum system, every two quorum of a quorum system intersect. So at least one server is part of both quorums. The fact that all servers of a particular quorum fail, implies that in each other quorum at least one server fails, namely the one which lies in the intersection. Therefore it is not possible to achieve a quorum anymore and the quorum system does not work anymore.

b) Just 1 - as soon as 2 servers fail, no quorum survives.

c) Imagine a quorum system in which all quorums overlap exactly in one single node. Each element of the powerset of the remaining $n-1$ nodes joined with this special node is a quorum. This gives 2^{n-1} quorums.
Can there be more? No! Consider a set from the powerset of n servers. Its complement cannot be a quorum as well, as they do not overlap. So, from each such couple, at most one set can be part of the quorum system. This gives an upper bound of $2^n/2 = 2^{n-1}$.

1.2 A Quorum System

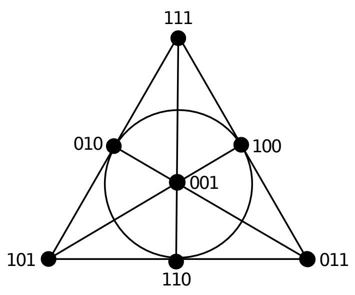

Figure 1: Quorum System

a) This quorum system consists of 7 quorums. As work is defined as the minimum (over all access strategies) expected number of servers in an accessed quorum, this system's work is 3 (all strategies induce the same work on a system where all quorums are the same size). The best access strategy consists of uniformly accessing each quorum (you will prove this for a more general case in exercise 3), so its load is 3/7.

b) Its resilience $R(\mathcal{S}) = 2$. Proof: every node is in exactly 3 quorums, so 2 nodes can be contained in at most $2 \cdot 3 = 6 < 7 = |\mathcal{S}|$ quorums, thus if no more than 2 nodes fail, there will be at least 1 quorum without a faulty node. If on the other hand for example the nodes 101, 010 and 111 fail, no other quorum can be achieved; see also exercise 1a).

1.3 S-Uniform Quorum Systems

Definitions:

s-uniform: A quorum system $\mathcal{S}$ is *s-uniform* if every quorum in $\mathcal{S}$ has exactly s elements.

Balanced access strategy: An access strategy Z for a quorum system $\mathcal{S}$ is *balanced* if it satisfies $L_Z(v_i) = L$ for all $v_i \in V$ for some value L.

Claim: An s-uniform quorum system $\mathcal{S}$ reaches an optimal load with a balanced access strategy, if such a strategy exists.

a) In an s-uniform quorum system each quorum has exactly s elements, so independently of which quorum is accessed, s servers have to work. Summed up over all servers we reach a total load of s, which is the work of the quorum system. As the load induced by an access strategy is defined as the maximum load on any server, the best strategy is to evenly distribute this work on all servers.

b) Let $V = \{v_1, v_2, ..., v_n\}$ be the set of servers and $\mathcal{S} = \{Q_1, Q_2, ..., Q_m\}$ an s-uniform quorum system on V. Let Z be an access strategy, thus it holds that: $\sum_{Q \in \mathcal{S}} P_Z(Q) = 1$. Furthermore let $L_Z(v_i) = \sum_{Q \in \mathcal{S}; v_i \in Q} P_Z(Q)$ be the load of server v_i induced by Z.

Then it holds that:

$$\begin{aligned}\sum_{v_i \in V} L_Z(v_i) &= \sum_{v_i \in V} \sum_{Q \in \mathcal{S}; v_i \in Q} P_Z(Q) = \sum_{Q \in \mathcal{S}} \sum_{v_i \in Q} P_Z(Q) \\ &= \sum_{Q \in \mathcal{S}} P_Z(Q) \sum_{v_i \in Q} 1 \overset{*}{=} \sum_{Q \in \mathcal{S}} P_Z(Q) \cdot s = s \cdot \sum_{Q \in \mathcal{S}} P_Z(Q) = s\end{aligned}$$

The transformation marked with an asterisk uses the uniformity of the quorum system.

To minimize the maximal load on any server, the optimal strategy is to evenly distribute this load on all servers. Thus if a balanced access strategy exists, this leads to a system load of s/n.

Note: A balanced access strategy does not exist for example for the following 2-uniform quorum system: $V = \{1, 2, 3\}$, $\mathcal{S} = \{\{1, 2\}, \{1, 3\}\}$. We have $\min\{L_Z(2), L_Z(3)\} < L_Z(1) = 1$ for any access strategy on this system.

2 Eventual Consistency & Bitcoin

2.1 Delayed Bitcoin

a) It is true that naturally occurring forks of length l decrease exponentially with l, however this covers naturally occuring blockchain forks only. As there is no information how much calculation power exists in total, it is always possible a large blockchain fork exists. This may be the result of a network partition or an attacker secretly running a large mining operation.

This is a general problem with all “open-membership” consensus systems, where the number of existing consensus nodes is unknown and new nodes may join at any time. As it is always possible a much larger unknown part of the network exists, it is impossible to have strong consistency.

In the Bitcoin world an attack where an attacker is secretly mining a second blockchain to later revert many blocks is called a 51% attack, because it was thought necessary to have a majority of the mining power to do so. However later research showed that by using other weaknesses in Bitcoin it is possible to do such attacks already with about a third of the mining power.

b) The delay in this case prevents coins from completely vanishing in the case of a fork. Newly mined coins only exist in the fork containing the block that created them. In case of a blockchain fork the coins would disappear and transactions spending them would become invalid as well. It would therefore be possible to taint any number of transactions that are valid in one fork and not valid in another. Waiting for maturation ensures that it is very improbable that the coins will later disappear accidentially.

Note that this is however only a protection against someone accidentially sending you money that disappears with a discontinued fork. The same thing can still happen, if someone with evil intent double spends the same coins on the other side of the fork. You will not be able to replay a transaction of a discontinued fork on the new active chain if the old owner spent them in a different transaction in the meantime. To prevent theft by such an attacker you need to wait enough time to regard the chance of forks continuing to exist to be small enough. A common value used is about one hour after a transaction entered a block (∼6 blocks).

2.2 Double Spending

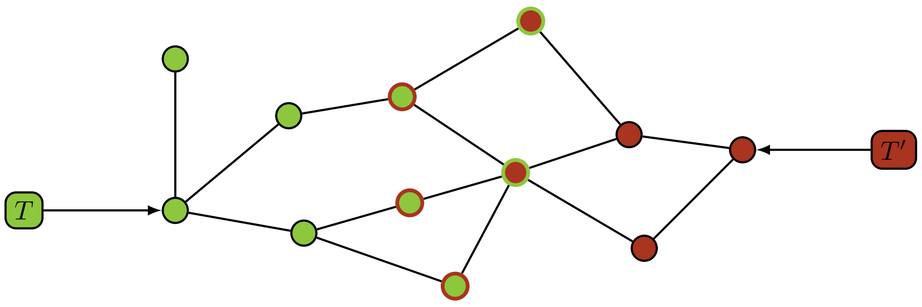

Figure 2: Random Bitcoin network

a) Figure 2 depicts the final situation. 7 nodes have seen T first and 5 nodes have seen T' first. The 5 nodes at the edge cut between the green and the red cut have seen both transactions.

b) Each node has 1/12 of all computational resources, hence the probability of T being confirmed is $7/12 \approx 58\%$, while T' has a $5/12 \approx 42\%$ chance of being confirmed. The higher connectivity from the first node seeing T resulted in the transaction spreading faster, increasing the probability of winning the doublespend.

c) The first node that sees T' now has 20% of the computational resources. T' therefore has a probability to win of $2/10 + 1/11 \cdot 8/10 \cdot 4 \approx 49\%$. The distribution of computational resources in the network therefore matters. The goal of an attacker is to spread the transaction that she wants to have confirmed to a majority of the computational resources, which may not be the same as spreading it to a majority of nodes.

2.3 The Transaction Graph

a) See Figure 3.

b) See red outputs in Figure 3.

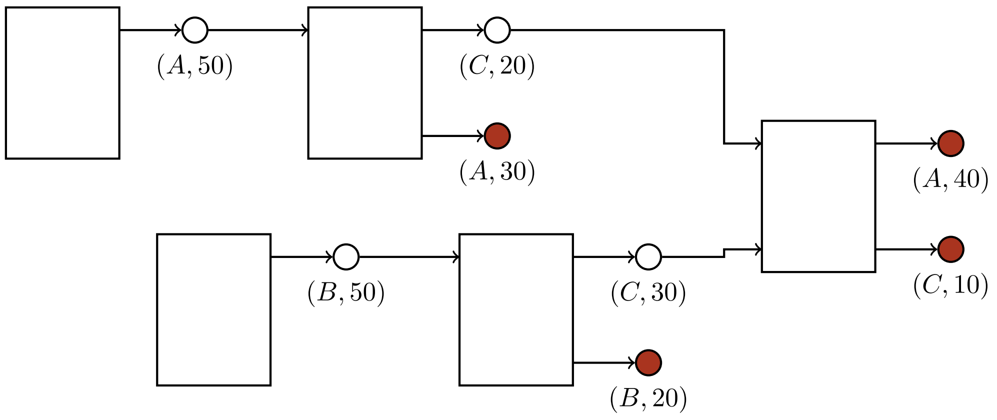

Figure 3: Transaction Graph. The red outputs are UTXOs.

c) Fully spending an output simplifies the bookkeeping considerably as an output can only be in two possible states: spent or unspent. This means that it is easy to detect conflicts, because two transactions spending the same output are conflicts. If we were to partially spend outputs, allowing multiple transactions to spend the same output until the coins on that output were completely spent, then the conflicts become more complicated. Assume an output with value 1 bitcoin. When partially spending outputs we could create 3 transactions claiming 0.5 bitcoins from that output, two of them are valid and the third will be invalid, but there are 3 possible combinations that are valid. So the simple answer is: it makes conflicts evident and reduces combinations for conflicts. The number of possible combinations increases rapidly with the number of transactions.

There are many points in the Bitcoin software where this would complicates things. A miner needs to construct a valid block from the known transactions, however this becomes more difficult if arbitrary combinations of transactions suddenly conflict with each other. Furthermore with complex conflicts attackers can create a situation where most nodes do not agree on the transactions which will be in the next block. However nodes are optimized to quickly be able to forward new blocks which look "expected". If the nodes do not agree at least loosely on the transactions to be committed in the next block, the propagation delays become much larger as many transactions need to be retransmitted, which finally results in smaller total transaction processing capability.

Note: As you might have realized the flow of money can be nicely followed with the transaction data from the Bitcoin blockchain. If some hacker steals your valuable coins, you can watch him buy things with it and see where the vendors are spending this money too! For this reason it is often possible to "buy" larger amounts of Bitcoin with less Bitcoin. The larger amount you can "buy" has very likely been involved in some crime and is being tracked by the police, thus the owner is eager to exchange them for other coins. Look for "Bitcoin doubling" in your favorite list of darknet services!

2.4 Bitcoin Script

a) Transactions are instantly finalized, so the large confirmation delay of the blockchain is irrelevant. Only the signatures of both parties are needed, then the money has effectively changed the owner. Furthermore no transaction fees have to be paid to miners for replacing a transaction.

b) Without the opening transaction A could just spend the money with a transaction without a timelock to a different address owned by himself. Requiring both signatures prevents this and gives security to B. In this construction B can trust that the funds will be available after the first timelock runs out.

Note that if B wants to access the funds earlier, it is still possible for A and B together to sign a transaction which directly executes the latest state. As long as both agree it is thus not necessary to wait for the timelocks. The timelocks are only necessary to ensure the last state in case there is disagreement.

c) A "kickoff" transaction can be introduced after the opening. Only the opening is executed (i.e., sent to the blockchain) at the beginning to secure the funds. Now transactions can be replaced and if someone wants to close the channel he can execute the kickoff. This starts the timers on the subsequent transactions. See Figure 4 for the new transactions.

In detail the protocol is the following.

Setup:

(a) A creates all transactions of the setup (opening, kickoff and first state).
(b) A sends these together with signatures for the kickoff and first state to B.
(c) B signs the kickoff and first state and sends the signatures to A.
(d) A signs the opening transaction and executes it on the blockchain.

Updating:

(a) A creates a new transaction spending the kickoff output with a lower timelock.
(b) A signs it and sends it to B.
(c) B sends his signature to A.

After the update both have a signed version of the new state and can terminate the channel in this state.

Closing:

(a) A or B proposes a settlement transaction that directly spends the locked-in funds in the opening output.
(b) The other party signs and sends it to the blockchain.

This is the cooperative closing case. If some dispute happens, either of the two parties can always send the kickoff and latest state to the blockchain.

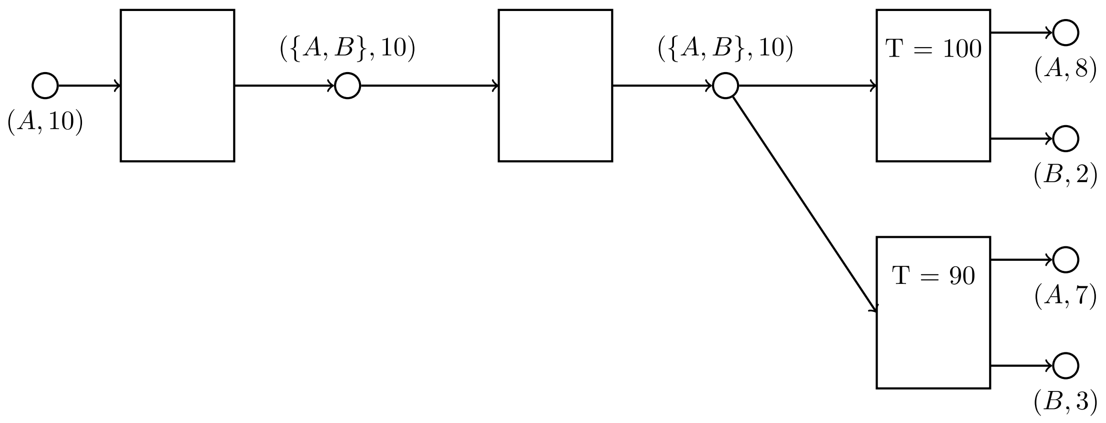

Figure 4: A "payment channel". A and B both have to sign to spend the output in the middle. The upper transaction can only be committed starting from blockheight 100, the lower one starting from blockheight 90.