ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Systems@ETH Zürich

Prof. T. Roscoe, Prof. R. Wattenhofer

Computer Systems

— Solution to Assignment 9 —

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 quorums 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; i.e. 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 $2^n/2 = 2^{n-1}$ quorums.

1.2 A Quorum System

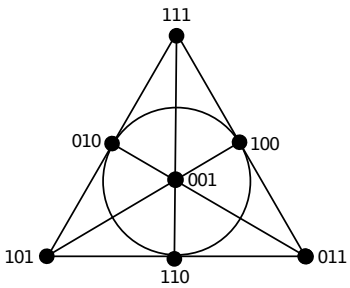

Figure 1: Quorum System

a) This quorum system consists of 7 quorums. As work is defined as the minimum expected number of servers in an accessed quorum (over all access strategies), this system's work is 3 (all strategies induce the same work on a system where all quorums are the same size). Observe that all nodes are in precisely 3 quorums, so the uniform access strategy induces the same load on all nodes. Since the quorum system is also 3-uniform, by exercise 3 it follows that the uniform strategy is optimal; it's load being 3/7.

b) The resilience is $R(\mathcal{S}) = 2$. 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 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 would be to evenly distribute this work on all servers. If such a strategy exists, then it is therefore optimal.

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:

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

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

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

Note: A balanced access strategy does not always 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 Approximate Agreement

Quiz

2.1 Asynchronous protocols in synchronous networks

a) Lemma 20.16 from the lecture notes ensures that there is no such strategy.

b) If a correct node v accepts $\texttt{msg}(x)$ at some point in time τ, then $f+1$ correct nodes have sent $\texttt{ready}(x)$ by time τ. Therefore, since the network is synchronous, all correct nodes receive these $f + 1$ messages $\texttt{ready}(x)$ within one additional communication round, and therefore send $\texttt{ready}(x)$. These messages are afterwards received within one more communication round, hence within two communication rounds after time τ.

c) Note that, if the sender v_S is correct, every correct node accepts $\texttt{msg}(x_S)$ in round 4. This is because all nodes receive the sender's value by the beginning of round 2. In round 2, all correct nodes send $\texttt{echo}(x_S)$, and these $n - f$ messages get delivered by the third round. Then, round 3, all correct nodes send $\texttt{ready}(x_S)$, and therefore all correct nodes accept the sender's value in round 4.

Then, if a node did not receive the sender's value by the end of round 4, then the sender must be a byzantine node.

Basic

2.2 From Approximate Agreement to Byzantine Agreement

a) Yes. If all correct nodes have the same input bit b, correct-range validity ensures that all correct nodes obtain value $x = b$.

b) No. If the correct nodes have distinct input bits, then they can obtain any ε-close values in $[0, 1]$. It is possible that a correct node obtains $x = 0.5 - \varepsilon/3$, and therefore its final output is 0, while another correct node obtains $x = 0.5 + \varepsilon/3$ and therefore its final output is 1.

c) We set $\varepsilon = 1/2 \cdot 10^{-2023}$. The nodes join the approximate agreement algorithm with their input bits as initial values. When a node obtains a value x, it queries the shared coin. Once $f+1$ nodes (hence at least one correct node v) query the coin, the random value r is decided and all nodes learn it eventually. When a node has obtained both the random value r and a value x via approximate agreement, if outputs 0 if $x < r$ and 1 otherwise.

The outputs x obtained via approximate agreement are ε-close, and agreement only fails if r is between the lowest and the highest values x obtained by correct nodes via approximate agreement. Then, if the first correct node that queries the coin has obtained value x, all correct nodes obtain outputs in the interval $[x - \varepsilon, x + \varepsilon]$. This means that only values $r \in [x - \varepsilon, x + \varepsilon]$ may lead to disagreement. Such a value r is obtained with probability at most $2 \cdot \varepsilon = 10^{-2023}$.

Advanced

2.3 Unbounded Input Space: Quick Fix

a) Nodes could simply define the number of iterations $I = \lceil \log_2((\max X - \min X)/\varepsilon) \rceil$.

b) Similarly to *correct-input validity*, one cannot distinguish between a correct node and a byzantine node that follows the algorithm correctly, but with an input of its own choice.

c) The algorithm proceeds as follows: every node sends its value to all nodes. Node v computes its estimation $\texttt{max\_range}_v$ as the difference between the highest value received (which is at least $\max X$, since all correct values were received), and the lowest value received (which is at most $\min X$, also because all correct values were received).

Note: removing the lowest f and the highest f values might discard some correct values and make the algorithm stop too early.

d) Nodes first run the algorithm from Task **c)**. Each node obtains an estimation $\texttt{max\_range}_v$. It computes $I_v = \lceil \log_2(\texttt{max\_range}_v/\varepsilon) \rceil$ as a sufficient number of iterations.

Then, the for loop goes up to I_v instead of the hardcoded number of iterations I. When node v computes its last value x_{I_v}, it sends a halting message $(\texttt{halt}, x_{I_v})$ to everyone. If node v receives a halting message $(\texttt{halt}, x_{I_u})$, it pretends it got x_{I_u} from u in each of its following iterations.

Algorithm 1 Synchronous Approximate Agreement: Unbounded Input Space

1: Code for node v with input x.
2: Send v to all nodes.
3: Add every received value to X.
4: $\texttt{max\_range}_v = \max X - \min X$.
5: $I_v = \lceil \log_2(\texttt{max\_range}_v/\varepsilon) \rceil$.
6: $x_0 = x$.
7: **for** i in $1...I_v$ **do**
8: Send x_{i-1} to all nodes.
9: Add every received value to R_i.
10: If node u sent $(\texttt{halt}, x_{I_u})$ (now or in some previous iteration), add x_{I_u} to R_i.
11: T_i = the multiset obtained by removing the lowest f and the highest f values in R_i.
12: $x_i = (\min T_i + \max T_i)/2$.
13: **end for**
14: Send $(\texttt{halt}, x_{I_u})$ to all the nodes.
15: Output x_{I_u}.

Let I_u denote the lowest correct estimation on the number of iterations, obtained by some correct node u. Since every correct node obtains $\texttt{max\_range}_v \geq \max X - \min X$, correct values obtained in iteration I_u already satisfy ε-Agreement (and correct-range validity). Then, once node u sends its `halt` message, we only need to ensure that all the following iterations maintain the range of correct values obtained in iteration I_u (and don't need to guarantee anything about convergence). This can be proven with the help of Lemma 20.6.

e) With the mechanism above, not really. The values $\texttt{max\_range}_v$ are essentially chosen by the byzantine nodes.