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 Shared Coins

1.1 Adaptive Adversaries

a) This algorithm is similar to the King Algorithm. Validity holds since the nodes only update their values to other nodes' inputs. Agreement holds since we have more phases than crashing nodes, which means that in some round j, the broadcasting node j is correct. In this round, every node updates its value v_j. Afterwards, the nodes never change their values again, and so they all output v_j.

b) We run the loop for λ iterations, rather than $f+1$. In each iteration, in the first round, the nodes choose a random node $k \in \{1, \ldots, n\}$ using `dice_toss()`. This node broadcasts v_k. Then, each node i updates its value to v_k if it receives the value v_k from node k. If in any round the chosen node k does not crash, then every node updates its value to v_k, and at the end the algorithm outputs v_k.

The probability the algorithm fails is thus upper bounded by the probability that we choose a crashing node every round, which is $\left(\frac{f}{n}\right)^{\lambda} < 2^{-\lambda}$.

Algorithm 1 Altered synchronous consensus algorithm: Code for node i

```
1:  v_i ∈ R                                  ◁ input
2:  for j = 1, ..., λ do
3:      k ← dice_toss()
4:      if you are node k then
5:          broadcast v_i
6:      end if
7:      if you receive v_k from node k then
8:          v_i := v_k
9:      end if
10: end for
11: output v_i
```

c) The answer of course depends on your algorithm. However, if your solution was the same as ours, then no, it does not work.

In each iteration, the adaptive adversary can wait until the nodes choose a random node k, learn k via the nodes' communication, and crash node k before it can broadcast its value. With this strategy, the nodes fail to reach consensus unless they choose $f+1$ separate kings in at least $f+1$ separate iterations. What we assumed when we designed the algorithm is that there is a fixed set of potentially crashing nodes, in other words that before the nodes begin running the algorithm, we can fix a set F of at most f nodes such that only the nodes in F can crash while the algorithm runs. The adaptive adversary breaks this assumption.

Note: Adversaries which let us fix such a set F are called "static" in the literature, and it is easier to design randomized algorithms against *static* adversaries. However, there do exist randomized algorithms that handle *adaptive* adversaries effectively.

2 Quorum Systems

2.1 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.

2.2 A Quorum System

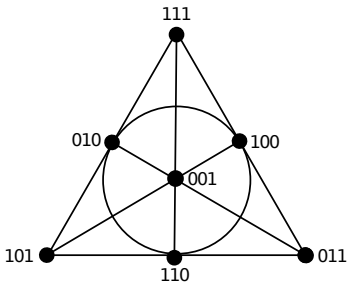

Figure 1: Quorum System

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).

2.3 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.