Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

HS 2011 B.Keller / S. Welten / Prof. R. Wattenhofer

Distributed Systems Part II

Solution to Exercise Sheet 6

1 A Quorum System

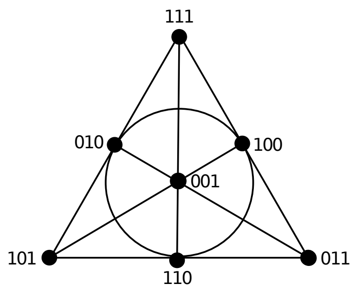

Figure 1: Quorum System

a) This quorum system consists of 7 quorums. As work is defined as the maximum number of servers in a quorum, its work is 3. The best access strategy consists of uniform accessing each quorum. So its load is 3/7.

b) Its resilience $R(Q) = 2$. If for example the nodes 101, 010 and 111 fail, no other quorum can be achieved.

c) Its failure probability $f(Q) = 7p^3(1-p)^4 + 28p^4(1-p)^3 + 21p^5(1-p)^2 + 7p^6(1-p) + p^7$.

2 The Resilience of a Quorum System

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.

3 S-Uniform Quorum Systems

Definitions:

S-uniform: A quorum system Q is *s-uniform* if every quorum in Q has exactly s elements.
Balanced access strategy: An access strategy W for a quorum system Q is *balanced* if it satisfies $l_W(i) = L$ for all $P_i \in P$.

Claim: An s-uniform quorum system Q reaches an optimal load with a balanced access strategy.

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

b) Let $P = \{p_1, p_2, ..., p_n\}$ be the set of servers and $Q = \{q_1, q_2, ..., q_m\}$ an s-uniform quorum system on P. W is a strategy, therefore it holds that: $\sum_{i=1}^{m} w_i = 1$ Furthermore let $l_w(p_i) = \sum_{q \in Q; p_i \in q} P_w(q)$ be the load of server p_i with strategy w.
Then it holds that:

$$\sum_{p_i \in P} \sum_{p_i \in q_j} w_j = \sum_{j=1}^{m} w_j \sum_{p_i \in q_j} 1 = s \sum_{j=1}^{m} w_j = s$$

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

4 Chubby

a) Coarse grained locking means that locks are held for "relatively" long periods of time (minutes to hours or even days). This is in contrast to locks that are held for very short periods of time, e.g. just for one short write operation (acquire-write-release). In Chubby, a master node is elected that does most of the work. This approach provides a good performance as long as everything goes well. However, if a master fails, a considerable amount of work is required (waiting for lease timeouts, electing a new master, etc.). If a client system expects fine grained locking, this work could not be completed in the expected time frame. An alternative might be to get rid of the master. However, if all consistency requirements had to be fulfilled without a master, a considerably larger amount of communication would be required within a Chubby cell at each operation. Thus, the coarse grained approach provides an interesting performance vs. consistency tradeoff.

b) A name service offers a mapping from names to addresses (for example domain names to IPs). To realize this service using Chubby, a node can reserve its name by acquiring a lock for the corresponding name. The meta-data stored aside this lock is then the address (such as the IP-address) of the corresponding node. If some node now wants to know the address that corresponds to a given name, it just requests the lock with this name (together with the meta-data). If the lock is held, the meta-data contains the required address (if the lock is not held, no node with the requested name exists in the name service).

c) Having more servers would imply more work to guarantee consistency, as the master has to communicate with more replicas (observe that the maximum load is at the master in normal operation). However, more node failures can be tolerated. Having fewer servers implies less work to guarantee consistency, but less node failures can be tolerated (only 1 in case of 3 servers). 5 seems to be a reasonable number: One server can be shut down for maintenance, and another server may fail at the same time for some (random) reason.