DISTRIBUTED SYSTEMS - EXERCISES

Exercise 1. Let C_1 and C_2 be two consistent cuts. Show that the intersection of C_1 and C_2 is consistent.

Solution.

$$e \to e',\ e' \in C_1 \Rightarrow e \in C_1$$
$$e \to e',\ e' \in C_2 \Rightarrow e \in C_2$$
$$e' \in C_1 \cap C_2 \Rightarrow e \in C_1 \cap C_2$$

Exercise 2. Let C_1 and C_2 be two consistent cuts. Show that the union of C_1 and C_2 is consistent.

Solution.

$$e \to e',\ e' \in C_1 \Rightarrow e \in C_1$$
$$e \to e',\ e' \in C_2 \Rightarrow e \in C_2$$
$$e' \in C_1 \cup C_2 \Rightarrow e \in C_1 \cup C_2$$

Exercise 3. Show that every consistent global state can be reached by some run.

Solution. TODO

Exercise 4. Label with proper vector clock all the events of this distributed computation

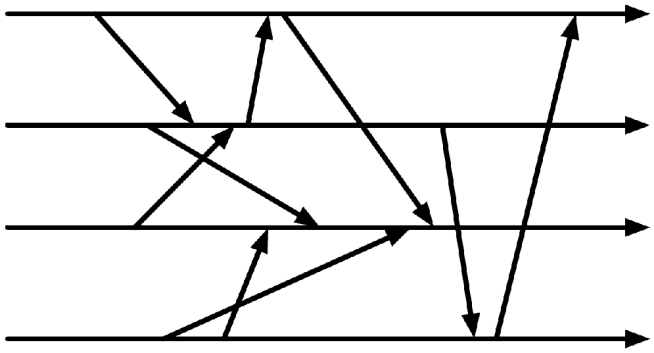

FIGURE 1. vector clock graph

Solution.

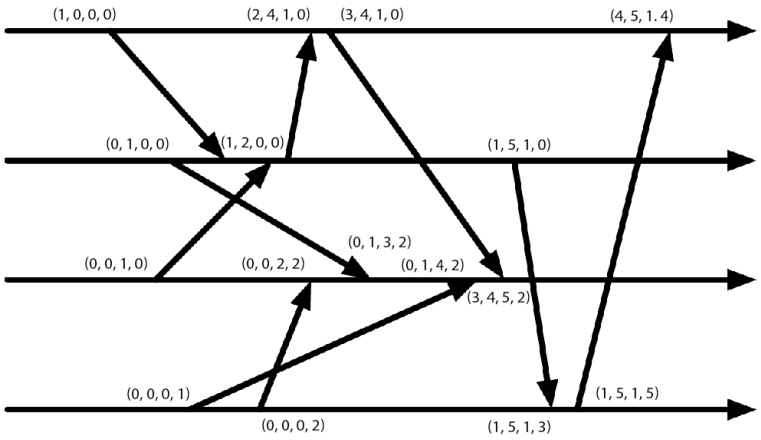

FIGURE 2. vector clock graph

Exercise 5. Show that the Chandy-Lamport Snapshot Protocol builds a consistent global state.

Solution. Assume the channels are FIFO (messages from same sender are delivered in the same order in which they were sent). If we take two events $e, e' \quad e \to e' \wedge e' \in S \Rightarrow e \in S$ and the Chandy-Lamport algorithm builds a consistent global state.

Exercise 6. What good is a distributed snapshot when the system was never in the state represented by the distributed snapshot? Give an application of distributed snapshots.

Solution. Assuming channels are FIFO, when a process receive its first "take snapshot" message, it records its state and broadcasts the message to every other process. The protocol terminates after the last process sends its last "take snapshot" message. Since channels are FIFO, it cannot happen that a message sent after recording its state is received before the "take snapshot" message from the same process. So, the global state is consistent.

Exercise 7. Consider a distributed system where every node has its physical clock and all physical clocks are perfectly synchronized. Give an algorithm to record global state assuming the communication network is reliable (note that your algorithm should be simpler than the Chandy–Lamport algorithm).

Solution. This means we have access to a real-time global clock. Assuming message delays are bounded, the monitor process starts the protocol by broadcasting a "take snapshot" message at time t_0 for time $t \geq t_0 + \delta$. At time t, each process sends its snapshot to the monitor, which can reorder them in increasing timestamp order. Time complexity goes from $O(n^2)$, required by the Chandy-Lamport algorithm, to $O(n)$.

Exercise 8. What modifications should be done to the Chandy–Lamport snapshot protocol so that it records a strongly consistent snapshot (i.e. all channel states are recorded empty).

Solution. A cut is strongly consistent if, for all sending events $e \in C$ to another process, we have also the receiving event $e' \in C$ (i.e., the message is no longer in transition). When a process receives the first "take snapshot" message, it broadcasts it to every other process and stop sending other messages, but does not record its local state yet. After a process receives its last "take snapshot" message, it records its local state and sends it back to the monitor. Since channels are FIFO, the last message sent from every process is a "take snapshot" message, so no other kind of messages can come after any process recorded its local state.

Exercise 9. Show that, if channels are not FIFO, then Chandy-Lamport snapshot algorithm does not work.

Solution. If channels are not FIFO, we may have $e, e' \;\; e \to e' \wedge e' \in S \not\Rightarrow e \in S$, which would not build a consistent global state. For example, it can happen that p_1 sends its "take snapshot" e and after that it sends a message e'. Then p_2 receives first e' and then e. So, $e' \in C$, $e \to e'$, but $e \notin C$.

Exercise 10. Let S_0 be the global state when the Chandy-Lamport snapshot protocol start, S the global state build by the protocol, and S_1 the global state when the protocol ends. Show that S is reachable from S_0 and S_1 is reachable from S.

Solution. S_0 is the state in which the first "take snapshot" message is sent. From S_0 we reach S when the "take snapshot" message is received by every process. Then, we reach S_1 from S when every process sends the "take snapshot" message to each other.

Exercise 11. Give an ACP that also satisfies the converse of condition AC_3. That is, if all processes vote Yes, then the decision must be Commit. Why is it not a good idea to enforce this condition?

Solution. Suppose every process votes YES except P, for which we do not know its decision (maybe it crashed or its message takes too long to deliver). If the decision is COMMIT anyway and P recovers, it may not have voted YES before and unilaterally decide ABORT. Thus the protocol would not be safe.

Exercise 12. Consider 2PC with the cooperative termination protocol. Describe a scenario (a particular execution) involving site failures only, which causes operational sites to become blocked.

Solution. Suppose the coordinator sends a VOTE request, every process votes YES and then the coordinator fails just before sending the decision to every other process. This would block the site.

Exercise 13. Show that Paxos is not live.

Solution. Suppose $n - f$ processes send an $\langle$ACCEPT$\rangle$ message for round i, but some of them fails afterward. Those processes formed a majority, so no subsequent round $i' > i$ can succeed until enough of them recovers. This might never happen and Paxos is therefore not live.

Exercise 14. Assume that acceptors do not change their vote. In other words, if they vote for value v in round i, they will not send learn messages with value different from v in larger rounds. Show that Paxos does not work any more (it can reach a livelock).

Solution. If acceptors keep sending their vote in rounds $i + 1 \ldots$ (without being able to change it) to the learners, the latter wont be able to collect a quorum of $n - f$ votes and no value can be chosen.

Exercise 15. How many messages are used in Paxos if no message is lost and in the best case? Is it possible to reduce the number of messages without losing tolerance to failures and without changing the number of proposers, acceptors, and learners?

Solution. Basic Paxos requires 4 communication rounds. It is possible to reduce the communication rounds (keeping the above requirements) to 3 if we run multiple instances of Paxos with a coordinator. This translates to $3n + n \cdot l$, where n is the number of acceptors and l is the number of learners in a round.

Exercise 16. Assume that you remove the property that every round is associated to a unique proposer. After collecting a quorum of $n - f$ promises (where n is the number of acceptors and f is such that $n = 2f + 1$), the proposer chooses one of the values voted in max round in the promises (of course it is not unique, the proposer chooses just one in an arbitrary way). Show that Paxos is not safe any more.

Solution. Suppose we have $n = 7$ acceptors and $f = 3$ failures for round i. The proposer collects a quorum of $n - f = 4$ promises, where some last voted for value 1 in round j and some last voted for value 2 in round j. But we still do not know the last vote of the other three acceptors and these might be enough to form a quorum of $n - f = 4$ votes in round j either on value 1 or value 2. Then the proposer cannot make a safe choice for round i.

Exercise 17. Assume that all proposers are learners as well. Let even rounds be assigned to proposers with the rules that we know. If round $2i$ is assigned to proposer p, then also round $2i + 1$ is assigned to proposer p. Odd rounds are "recovery" rounds. If round $2i$ is a fast round and if the proposer of round $2i$ sees a conflict (it is also a learner), then the proposer immediately sends an accept for round $2i + 1$ with the value that has been most voted in round $2i$, without any prepare and any promise. Is safety violated? If yes, show an example. If not, demonstrate safety.

Solution. From the safety theorem of Paxos, if an acceptor a has voted for value v at round i, then no value $v' \neq v$ can be chosen in any previous round $j < i$. In Fast Paxos, a coordinator which recognize a conflict in round i, can still start a new round with a number greater than $i + 1$ to fix it.

Exercise 18. You are an optimization freak. You realize that, in some cases, it is not necessary that the proposer collects $n - f'$ (the Fast Paxos quorum) promises to take a decision. Which is the minimum quorum and under what hypothesis this minimum quorum is enough to take a decision?

Solution. Fast Paxos requires that any three fast roud quorums Q_f intersect, i.e. $\forall Q, Q', Q'' \in Q_f : Q \cap Q' \cap Q'' \neq \emptyset$, meaning that a quorum of 2/3 of acceptors is need to make safe choices. However, we can differentiate between phase-1 (prepare, promise) quorums Q_1 and phase-2 (accept, learn) fast round quorums Q_{2f}. We then observe that quorum intersection is only required between phase-1 quorums Q_1 and any pair of phase-2 fast round quorums Q_{2f}, i.e. $\forall Q \in Q_1, \forall Q', Q'' \in Q_{2f} : Q \cap Q' \cap Q'' \neq \emptyset$. The weakened intersection requirements show that a simple majority is sufficient for phase-1 of a fast round, instead of requiring 2/3 of the acceptors.

Exercise 19. For each of the six ordered pairs of problems among: the Byzantine agreement problem, the Consensus problem, and the Interactive consistency problem, demonstrate a reduction from the former to the latter.

Solution. Reduction $A \leq B$, where n is the number of processes

- $IC \leq BA$. Run BA for n times with process p_i as coordinator, where i is the iteration number
- $BA \leq IC$. Run IC and, from the returned vector, pick the value associated with the coordinator
- $CP \leq IC$. Run IC, then agree on a value in the vector by majority
- $IC \leq CP$. Run CP for n times, then pick a value for each process p_i at each iteration i
- $BA \leq CP$. Run CP with a coordinator, then run a consensus protocol to agree upon coordinator's value
- $CP \leq BA$. Run BA and obtain the coordinator proposed value, which will be the consensus value for CP. If the leader is byzantine, agree on any other value

Exercise 20. Modify the "consensus algorithm for crash failures (synchronous systems)" to design an early-stopping variant, that terminates within $f' + 1$ rounds, where f', the actual number of stop-failures, is less than f. Prove the correctness of your algorithm (hint: a process can be required to send a message in each round, even if the value was sent in the earlier round. Processes should also track the other processes that failed, which is detectable by identifying the processes from which no message was received).

Solution. Correctness proof:

- termination: satisfied by synchronous model assumption
- validity: local consensus variable can be updated only with proposed values
- agreement: in $f + 1$ rounds, there must be at least one crash-free round f', where alive processes succeed in broadcasting their values and update their local consensus variable. If $f' < f$, the algorithm can stop earlier in $f' + 1$ rounds, otherwise it behaves like the original one.

```
f <- max number of failures tollerated
x <- local consensus value
y[r] <- number of processes from wich P_i received
            messages in round r
early <- wether to stop earlier
process P_{1≤i≤n} executes the consensus algorithm for up to f crash failures:
    y[0] <- n
    early <- FALSE
    FOR round r=1 to f+1 do
         broadcast(x, early)
         IF early THEN RETURN x  // terminate in f'+1 rounds
         x <- min_{∀j}(x, {consensus values from processes P_j})
         y[r] <- number of messages received by process P_i
         decide <- equal_{∀j}({early values from processes P_j})
         IF (y[r-1] = y[r]) OR decide THEN early <- TRUE  // early-stop
    RETURN x  // terminate in f+1 rounds
```

Exercise 21. Prove that the leader election problem is not solvable under a crash failure (you can use the FLP result without proving it).

Solution. The leader election problem can be reduced to the consensus problem. But we know, by the FLP result, that consensus is not solvable (in an asynchronous system) even if ony one crash-failure is present. Therefore the leader election is not solvable under a crash-failure.

Exercise 22. Make Paxos live with a eventually perfect failure detector $\diamond$-S.

Solution. A $\diamond$-S failure detector has the following properties

- strong completeness. Eventually every faulty process is permanently suspected by every non-faulty process
- eventually strong accuracy. There is a time after which non-faulty processes are not suspected by any non-faulty process

An eventually perfect failure detector $\diamond$-S can notify each proposer about the status (alive or faulty) of each other proposer and, after a (unknown) time t, faulty processes can be accurately detected. If it is used to elect a coordinator, eventually, every proposer agrees on which is the current coordinator and the system is guaranteed to proceed.