NAME : VIVEK BANSAL

STUDENT ID : 112044493

MAIL ID : VIVEK.BANSAL@STONYBROOK.EDU

COURSE NUMBER : CSE 535 (ASYNCHRONOUS SYSTEMS)

ASSIGNMENT NUMBER : 4

DUE DATE : 22 OCTOBER 2018

ANS 1:

Liveness Problems and their solutions in Basic Paxos:

(i) **Race Condition or Competitive Nature of Processes:** Race condition can happen in basic paxos. Suppose there are 2 processes P0 and P1 in the system. P0 sends a proposal request with number n0 and completes its phase1. Before completing phase 2, another process P1 sends a proposal request with proposal number n1 > n0. So acceptors sends a reject to process P0 in phase 2 since they have the request of higher proposal number. Now suppose before P1 completes its phase 2, process P0 sends another request of proposal number n2 > n1. So acceptors send reject to process P1 since they have the higher proposal number request. So this thing can go forever and system will not make any progress.

Solution: To prevent race condition, select a random leader after specified period of time. There is one more solution of exponential random backoff. In this case, proposer will not send another request for a specified period of time if it realizes that no one is getting a majority. So this thing will ensure that someone will get majority after certain time.

ii) **<u>No one is getting the majority due to message loss</u>**: One of the liveness problem which can happen in basic paxos is due to message loss. Suppose the messages are getting lost in the network, like sometimes prepare messages are getting dropped and sometimes accept requests from the acceptors are getting dropped. So in this system no one will get majority and system will not progress. At the end, no proposer will get the majority and no consensus is reached.

<u>Solution:</u> To prevent this problem, do a timeout at the proposer end. There should be the implementation of timeout mechanism at each of the processes which are waiting for the messages. This will ensure that another round of consensus will start if there is timeout at any of the process.

ANS 2:

There are 4 performance problems in the paper by Van Ranesse in his "paxos made moderately complex" paper:

1) **<u>Fast Growing Set Accepted:</u>** In this algorithm, we can have many states possible since it is keeping all the triples associated with a ballot for each slot. This is really inefficient for a real-time system like distributed system. This inefficiency made the working of the algorithm extremely slow as we are keeping the older states in the system.
<u>Solution:</u> Rather than keeping all the triples associated with a slot, we should keep only the triple with the maximum ballot number for each slot. This will lead to drastic state reduction of both the leaders and the acceptors.

2) **<u>Unnecessary competition of leaders:</u>** This type of problem occurs when all processes are competing to become the leader with the highest ballot number. When a process is pre-empted it immediately tries the next round to issue the next request with a greater ballot number.
<u>Solution:</u> This problem can be solved using failure detection. Once a process is preempted, it will periodically ping the leader with the maximum ballot number (as it got that number from acceptor) to check if that process is alive or not. It will issue a new request only when this process with the highest ballot number is no more a leader.

3) **Keeping States on Disk:** All the states are kept on the disk as the number of states are too large and power outrages can happen as well. The problem with this approach is that disk accesses are slow and it is sometimes not possible to update the data structures on the disk.
   Solution: The solution to this problem is Write-Ahead Logging in which updates are logged sequentially to the disk and memory caching all our data structures as well. The log is written to the disk periodically to update the data structures and log checkpoints are created.

4) **Garbage Collection:** This type of problem arises as too much information is stored about the slots that have already been decided. Some information is necessary but some of the information from state is redundant. The leader maintains information about the slots for which it has a proposal and acceptor maintains state for each slot. But since f +1 replicas have learned about the decision of some slots, so it is not required for the leaders and acceptors to store these states.
   Solution: Replica periodically updates the leaders and acceptors about the slots. Once leaders and acceptors learn that f + 1 replicas have the information about some slot s, then all information for the slots lower numbered than s can be garbage collected.
