DSM: Tutorial 4

Question 1

Players in a game move around a shared scene. The game state is replicated for all players and a server controls collision detection, etc. Updates are broadcast to all replicas.

- *Players may throw projectiles and be hit by them, debilitating them for some time. What kind of ordering is required for 'throw', 'hit' and 'revive' events?*
- *Players may pick up magical items to assist them. What kind of ordering is required for 'pick-up' and 'use-item' events?*

At a minimum, causal ordering is required for the first set of actions. Hits should not occur before throws, and revivals should not occur before hits. For the second set of items, FIFO ordering may be enough: all replicas should know that a player has picked up an item before using it.

Depending on the game's functionality, total ordering may be required in both cases. For instance, if the concurrency of non-causal events is important then total ordering will be needed to make sure all players see the same scene.

Question 2

Pros and cons of primary backup and state machine approaches. How might cons be handled?

The primary backup approach is simple to implement and makes it easy to ensure consistency. However, it introduces a single point of failure and it does concentrate load on one RM. The single point of failure can be mitigates by proper failover procedures and primary re-election. Concentrated load can be mitigated with horizontal and/or vertical partitioning, depending on the data access pattern.

The state machine approach is more complex, but more robust because all RMs are equal and there is no single point of failure. It can require more complexity on the client side, but this can be mitigated with a server-side implementation.

Question 3

What happens in the following situation, and how may it be dealt with?

- *Client* A *sends* R_{A1}.
- *Client* B *sends* R_{B1}.
- RM_1 *receives and responds to* R_{A1} *before* R_{B1}.

- *RM_2 receives and responds to R_{B1} before R_{A1}.*
- *Each RM gives 1 and 2 for the first and second candidate IDs.*
- *Each client chooses the maximum candidate ID and informs the RMs.*

The following sequence of events will happen:

- RM_1 will propose 1 for R_{A1} and then 2 for R_{B1}.
- RM_2 will propose 1 for R_{B1} and then 2 for R_{A1}.
- Client A will receive candidates 1 and 2, and will select 2.
- Client B will receive candidates 1 and 2, and will select 2.
- Both clients inform both RMs that they have chosen 2 as their supposedly-unique IDs.

This situation could be avoided by ensuring that **different RMs don't propose the same numbers** (in a system of n RMs, each could start from a different offset and increment their candidate IDs by n).

In this example, RM_1 could propose $1, 3, 5, ...$ and RM_2 could propose $2, 4, 6, ...$

Question 4

*How do the RM approaches covered provide replication **transparency**?*

By implementing a front-end between clients and RMs, clients do not need to know how many replicas exist and are functional.