1.

a.

For the order 1->2, if the reliable multicast fails, other processes will not receive the message and the current process will not deliver the message to the local application.

For the order 2->1, if the message is delivered to the local application, then if the following multicast fails, other processes will not receive the message.

b.

**FIFO:** Assume there are N processes. Let each process maintain an array of N integers.

When sending a multicast message at process Pj, set Pj[j]+=1, attach the new array in the multicast message.

When receiving a multicast message, if Pi receives from Pj with the number at position j being S,

if (S==Pi[j]==1) then deliver message to the application. Set Pi[j]+=1

Else buffer this multicast until above condition is met.

**Total ordering:** elect a special process as a leader

When sending multicast message at process Pi: send multicast message M to the group and the leader

Leader process:

Maintains a global sequence number S (initially 0). When it receives a multicast message M, it sets S+=1, and multicast <M, S>

When receiving a multicast message at Pi:

Pi maintains a local received global sequence number Si(initial 0). If Pi receives a multicast M from Pj, it buffers it until both of the following conditions are met:

1. Pi receives <M, S(M)> from the leader 2. Si+1=S(M)

Then the message is delivered to the application and Si+=1.

**Causal Ordering:** Each receiver maintains a vector of per-sender sequence numbers, Pi maintains a vector Pi[1…N] initially all zeroes. Pi[j] is the latest sequence number Pi has received from Pj.

When sending a multicast message at Pj, set Pj[j]+=1, attach the new vector array in the multicast message as its sequence number.

When receiving a multicast message, if Pi receives a multicast message from Pj with vectors P’j[1…N] in message, buffer it until the following conditions are met:

The message is what Pi is expecting from Pj, that is to say, P’[j]=Pi[j]+1; All multicasts, anywhere in the group, which happened-before M have been received at Pi, i.e. for all k!=j: P’[k]<=Pi[k].

Then deliver the message to application and set Pi[j]+=1

2.

a.

Each server has only one vote for one specific term. The algorithm ensures that a candicate is legitimate only when it contains all the committed entries and the candidate wins an election only if it receives votes from a majority of the servers in the full cluster for the same term. This ensures that there can’t be two candidates who both have the majority of votes from the cluster. So election safety property holds.

b. e is elected as leader for term 4. Then it receives four entries and handled two before crashing. D is elected as leader for term 7 but isn’t able handle it before crashing. C is elected for term 6 and crashed after successfully handling 3 entries.

c. “the RPC includes information about the candidate’s log, and the voter denies its vote if its own log is more up-to-date than that of the candidate.” This adds the restriction that a server only votes for candidates that have logs more up-to-date than its own, thus avoiding wasting votes on unqualified servers. So It’s better than the fist come first served approach.

d. The assumption is that U is the smallest term whose leader doesn’t store the entry.

e.

The broadcast time should be an order of magnitude less than the election timeout so that the leader will have enough time to send heartbeat message to followers, avoiding constant change of server states. This also makes split votes less possible, as inequality will enable the broadcast to end before the occurrence of another server timeout. The number of concurrent leader elections by different candidates will be reduced.

Election timeout should be a few orders of magnitude less than MTBF to allow the system to make stready progress. If a large portion of the running time of the server is spent on leader election. The system will not have enough opportunity to grow.

3.

I will do the project 1. I will build a elastic mapreduce app using Akka Cluster.