ECE 428 MP1

[Xiao Shuhong 3180111551] [Ma Zicheng 3180112691]

1. Problem Description

In this mp, we are supposed to set up a reliable distributed system that used to keep the same balance of bank account in different local servers. The requirements list as follow:

```
1. Achieve a total ordered multicast to ensure the right
ordering of balance update.
2. Detect and handle potential failures of any network nodes in the system.
```

2. Ensures total ordering

Basically, we use ISIS algorithm to achieve this goal. The algorithm can be described as follow: first, sender multicasts message to everyone; second, every alive acceptor reply with proposed priority and store the message in its priority queue; third, sender chooses agreed priority, re-multicasts message with agreed priority; finally, every sender receive agreed (final) priority.

In our design, use the 3 node system as an example. For every message, each node would see it 3 times due to the design of multicast, each time the node see this message, we can use the priority number it contains to update our queue. As all the node alive will see exactly the same 3 message, we do not need to send the agreed final priority again as described in the ISIS algorithm. For any message stored in the queue, we record how many reply it has received, if all reply is received, we use this message to update our balance.

For each message in our design, it will contains: a "SenderNodeName" denotes who multicast this message; Content stores the account update process; "MessageID" which is unique for every message generated, in the form of nodex.t, x as the node id, t as the timestamps when the message is generated; a "sequence_number" as the priority the node who multicast this message would like to give. We also design struct for priority queue in each node: a "queue" will be used to store all message get, the order will be keep every time some message is inserted, deleted or updated; a "numitem" to keep track of the message number; a "recv_feedback" dictionary to record the node replied.

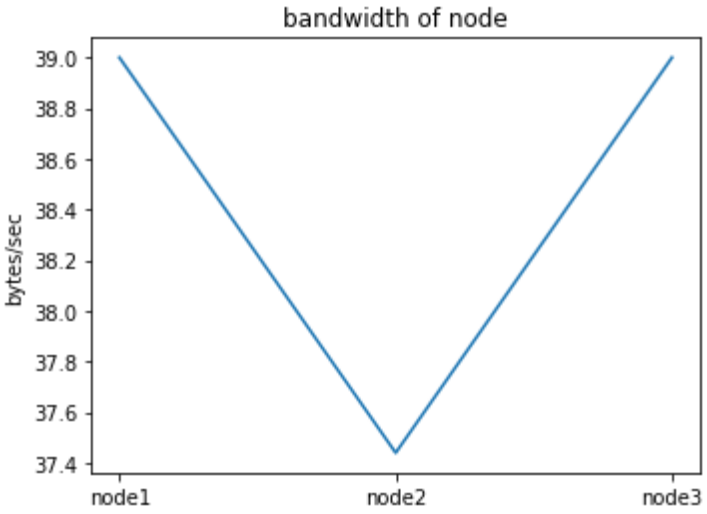
3. Ensures reliable delivery under failures

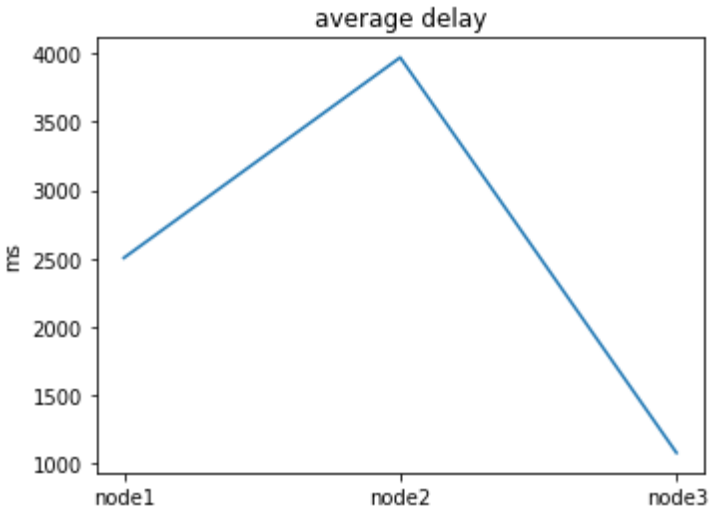
To ensure reliable delivery, we achieve R-multicast. A node will multicast any message to every member in the group if this message has not been seen before. For each node, we will maintain all the connection it has. We detected the failures at the sender side, if the message sending is failed, we

know that the listen socket in the other side may not work, so we delete that connection and check whether message in the head of queue has received all other reply and can be delivered.

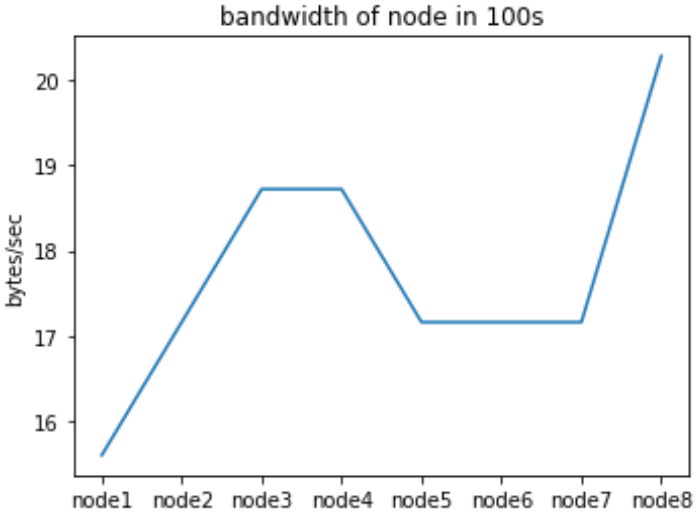
4. Performance evaluation graphs

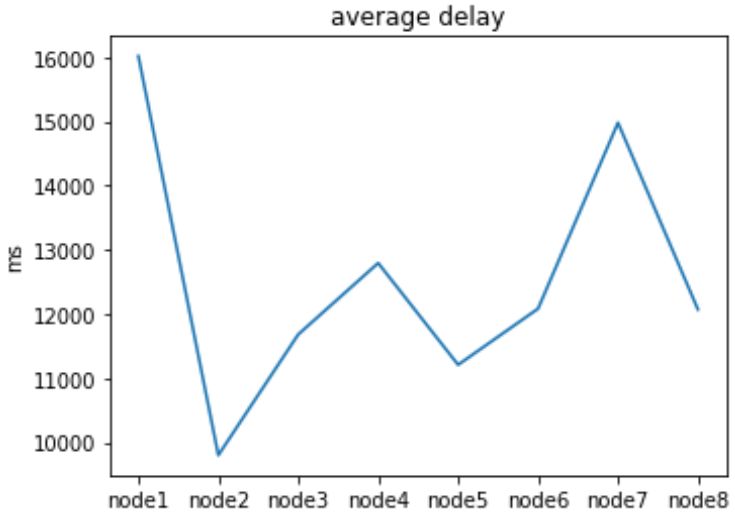
1. 3 nodes, 0.5 Hz each, running for 100 seconds



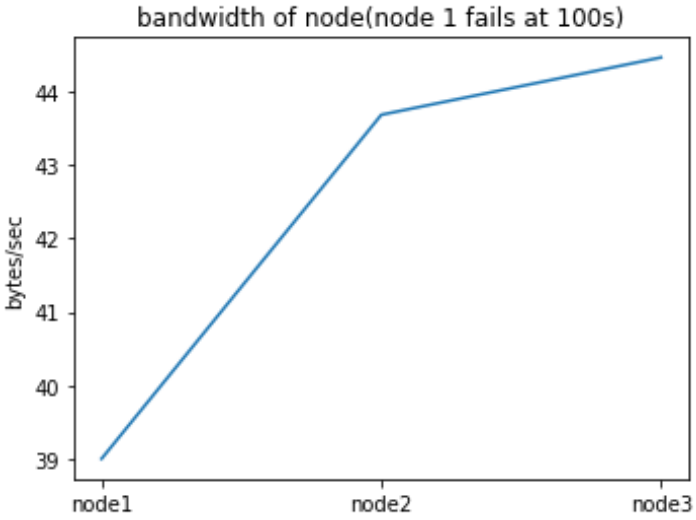


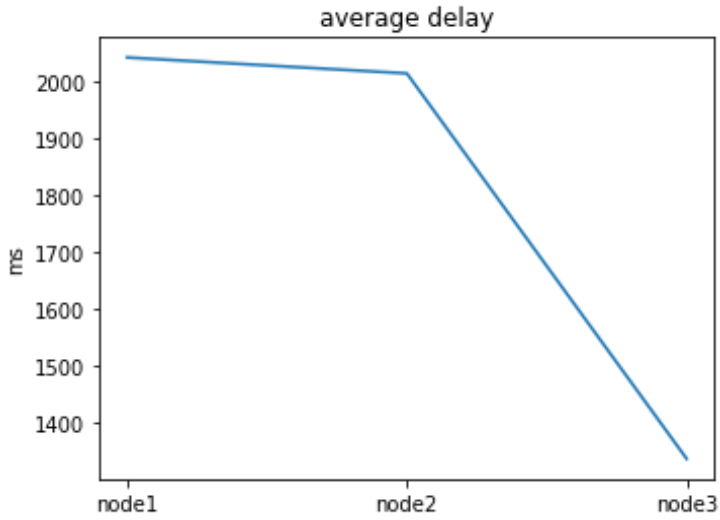
2. 8 nodes, 5 Hz each, running for 100 seconds



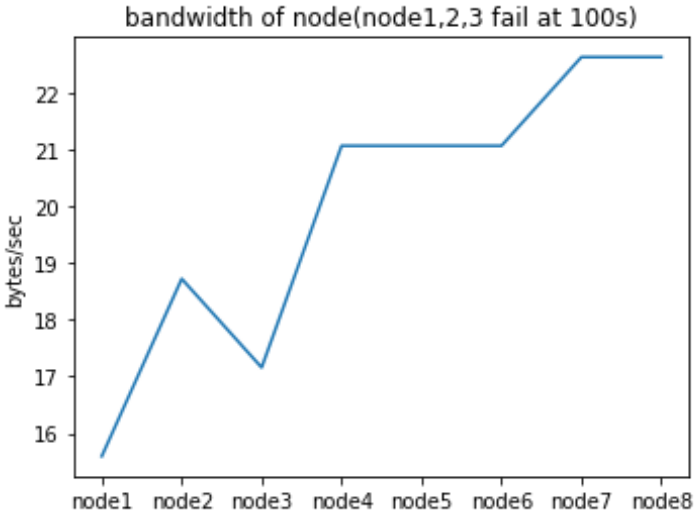


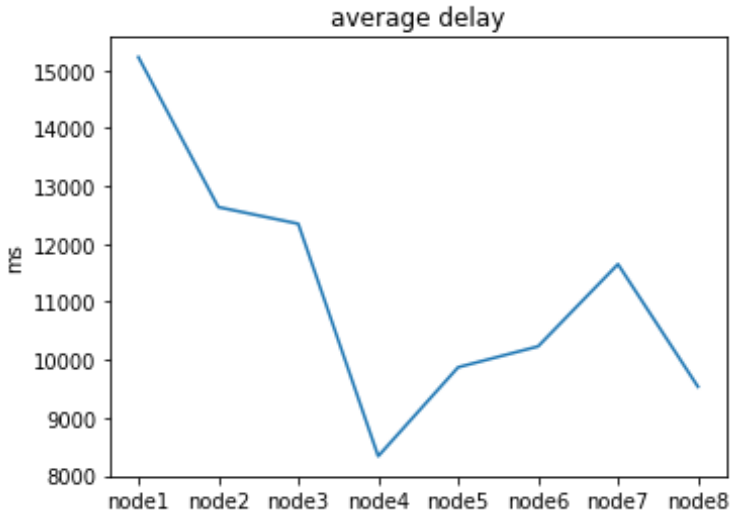
3. 3 nodes, 0.5 Hz each, runing for 100 seconds, then one node fails, and the rest continue to run for 100 seconds.(node 1 fails at 100s)





4. 8 nodes, 5 Hz each, running for 100 seconds, then 3 nodes fail simultaneously, and the rest continue to run for 100 seconds.(node 1,2,3 fails at 100s)





5. Run our code

- 3 node version is in **localtest_scripts.txt**
- 8 node version is in **localtest_scripts_8 node.txt**