MP1: Event Ordering

Group name: coredump

Group member: Xu Xiong(xuxiong2), Hanqi Wang(hanqiw3)

Cluster name: g05

Design Algorithm

In MP1, we use the **ISIS algorithm**. This algorithm can guarantee messages totally ordered in the asynchronous, decentralized case. Each node in this system must be connected with each other.

ISIS algorithm can be simply decomposed in the steps below:

1. A node *S* multicasts a desired message to other nodes.
2. All other nodes in this system will propose a sequence number and send it back to *S*.
3. The node *S* compare all received sequence numbers(including itself) and choose the largest one.
4. The node *S* multicast this largest sequence number to all other nodes and itself.
5. After all nodes receive this sequence number, they assign it to the message and mark it deliverable.
6. Each node sorts the consequence numbers from smallest to largest, and then delivers messages in turn.

Design Documents

`gentx.py`: A file to automatically generate random input messages(deposit or transfer)

`main.py`: We implement ISIS Algorithm in sequence number generation. In Class Message, we define some functions for how we define a message. In Class Process, we define a 'node' and we store accounts in the process as a dictionary. The main loop which spawns threads and handles things such as message passing corresponding to depositing and transferring money.

`config.txt`: provide the node_num and the address of other nodes.

Class Message

A message is defined by the following parameters:

source: the originating process.

messageId: a hash of the message content a global message counter and the process

id this message originated from.

sequence: the proposed sequence attached to the message.

type: The type of message (deposit or transfer).

amount: The amount of money to be transferred.

sender: The source account the money is being transferred from.

receiver: The target account the money is being transferred to.

isDeliverable: This marks whether we have received a sender reply representing.

whether this message's sequence number has been finalized.

isRecvReply: A boolean telling us if this message is a receiver reply.

isSendReply: A boolean telling us if this message is a sender reply.

Class Process

A process is defined by the following parameters:

pid: Process id

totalProcessNum: The total number of process in the cluster

sendSeq: The current proposed sequence number

agreedSeq: The current agreed sequence nubmer

holdback: A list of messages in our holdback queue

delivered: A list of messages which have been delivered. This is never used as we process and discard the message immediately after updating account information

replyDict: A dictionary where we store receiver replies to messages we own and have multicast. The key is the message id and the value is a list of receiver replies messages

ISIS

There are utility functions for generating the correct proposed sequence or largest proposed

sequence from a list of messages. These are then used when creating a new message, creating

a receiver reply, or when receiving a sender reply. In addition to those functions, we also placed

a reordering function here to help us reorder the messages in the holdback queue of our

process.

Ensuring Total Ordering and Reliability

To achieve total ordering we implemented the ISIS algorithm together with multicast. In order

to ensure that we maintain data coherency we use `multiprocessing.lock()` and `queue.Queue()`.

As discussed above, the protocol are implemented in `main.py`. In order to ensure

that our multicast is reliable we use multiprocessing.connection which is built on top of

TCP therefore guaranteeing us reliability.

Instructions for running code

On every node please run the command as follows:

```
python3 -u gentx.py 0.5 | python3 main.py {node id} {port} {config file}
```

Below is an example:

```
python3 -u gentx.py 0.5 | python3 main.py node1 2000 config1.txt
```

Data Analysis Methodology

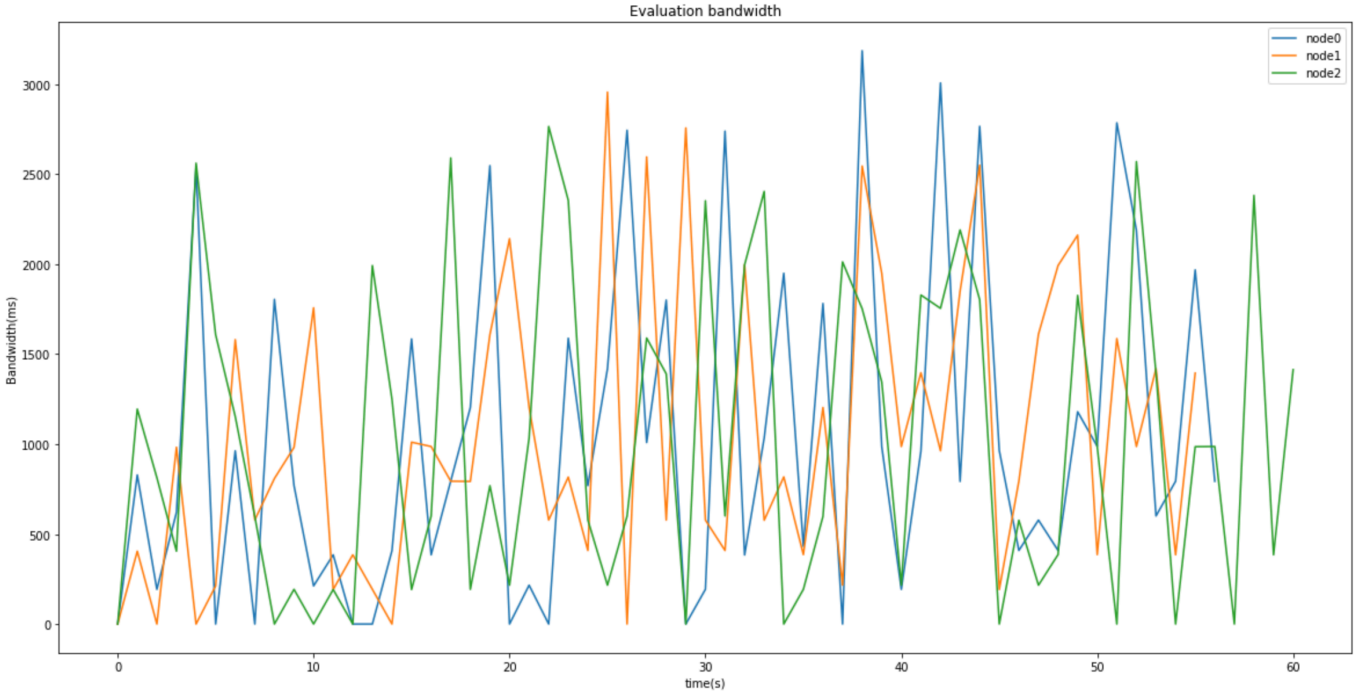
Total ordering is verified by unique `messageId` generated from `hashlib`.

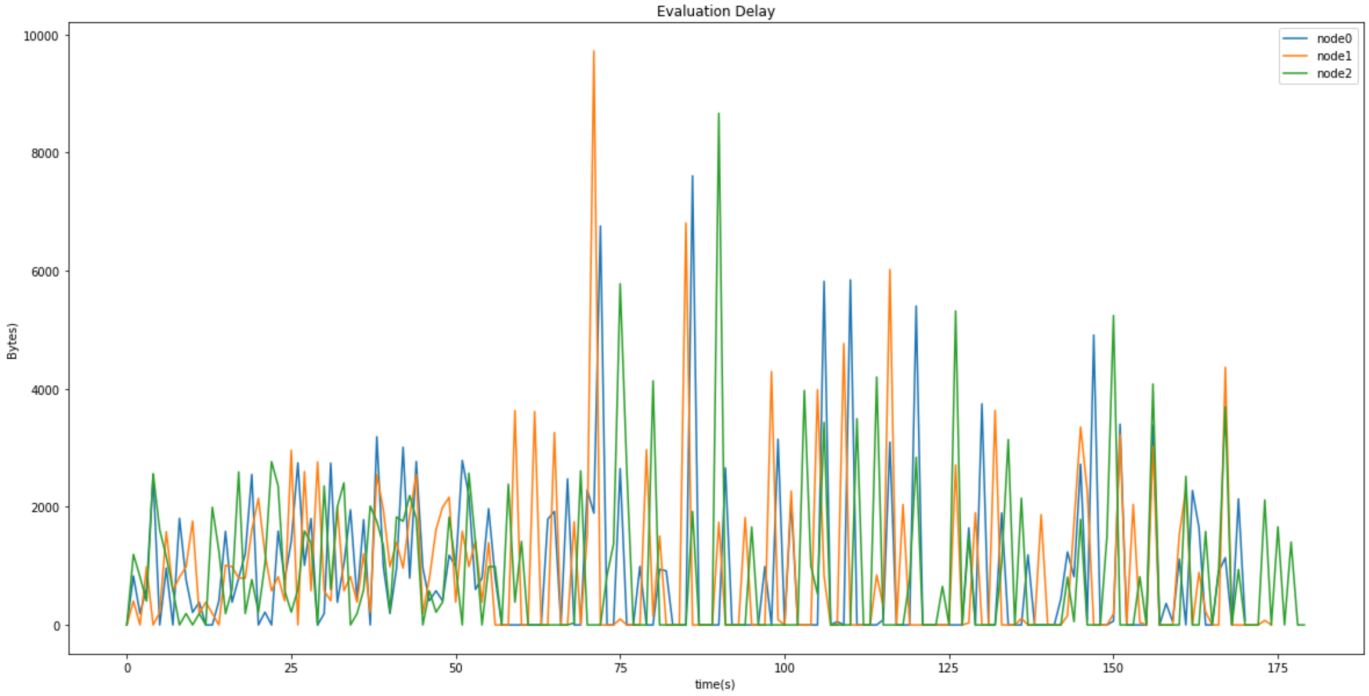
Bandwidth is collected by detecting the size of the data downloaded from the sockets. We checked the size of the data and summed it by each second.

Transaction delay is obtained by recording the timestamp when each message is delivered.

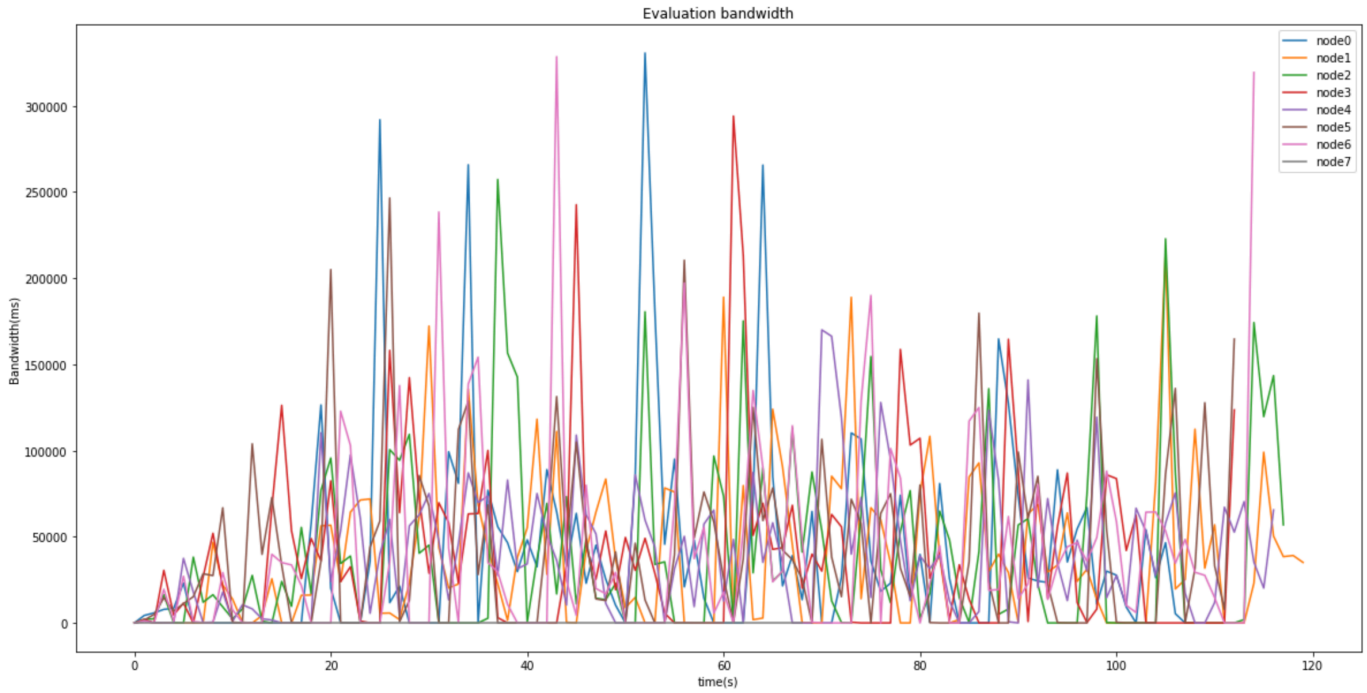
Delay and Bandwidth Measurement

Scenario1 :3 nodes, 0.5 Hz each, running for 100 seconds.

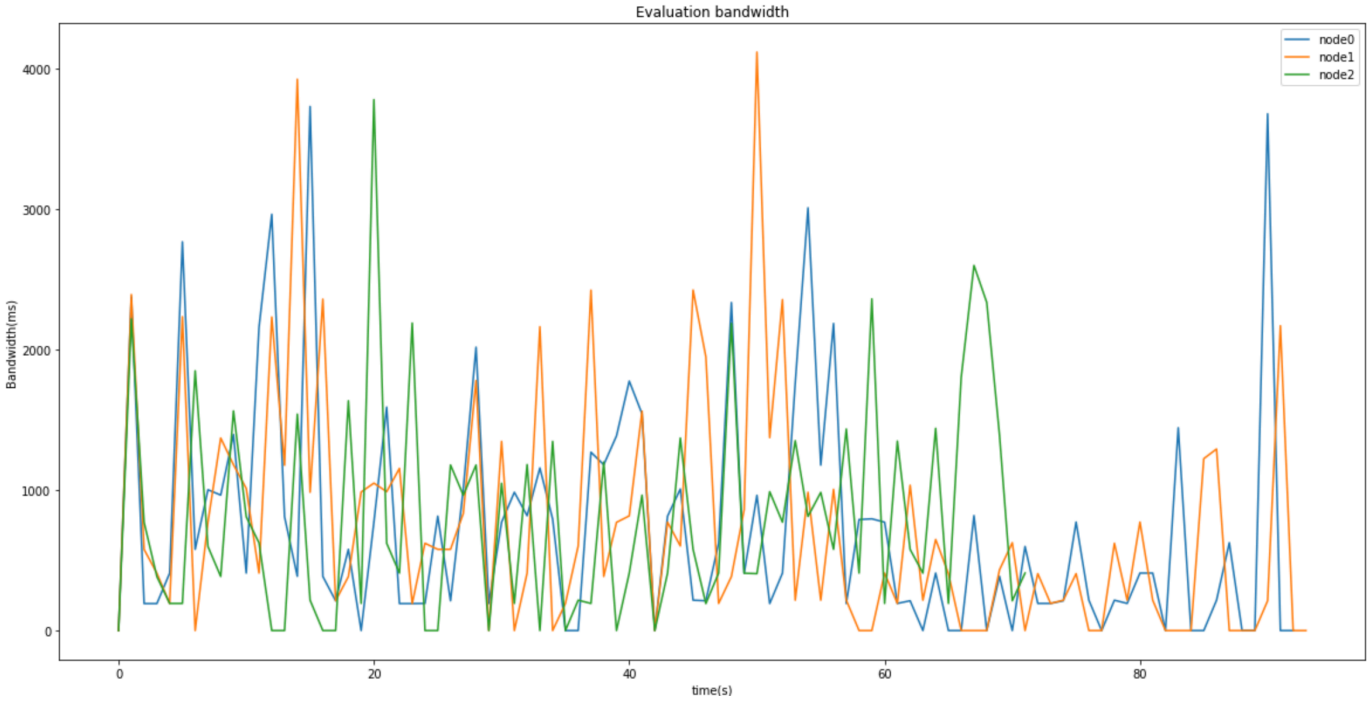


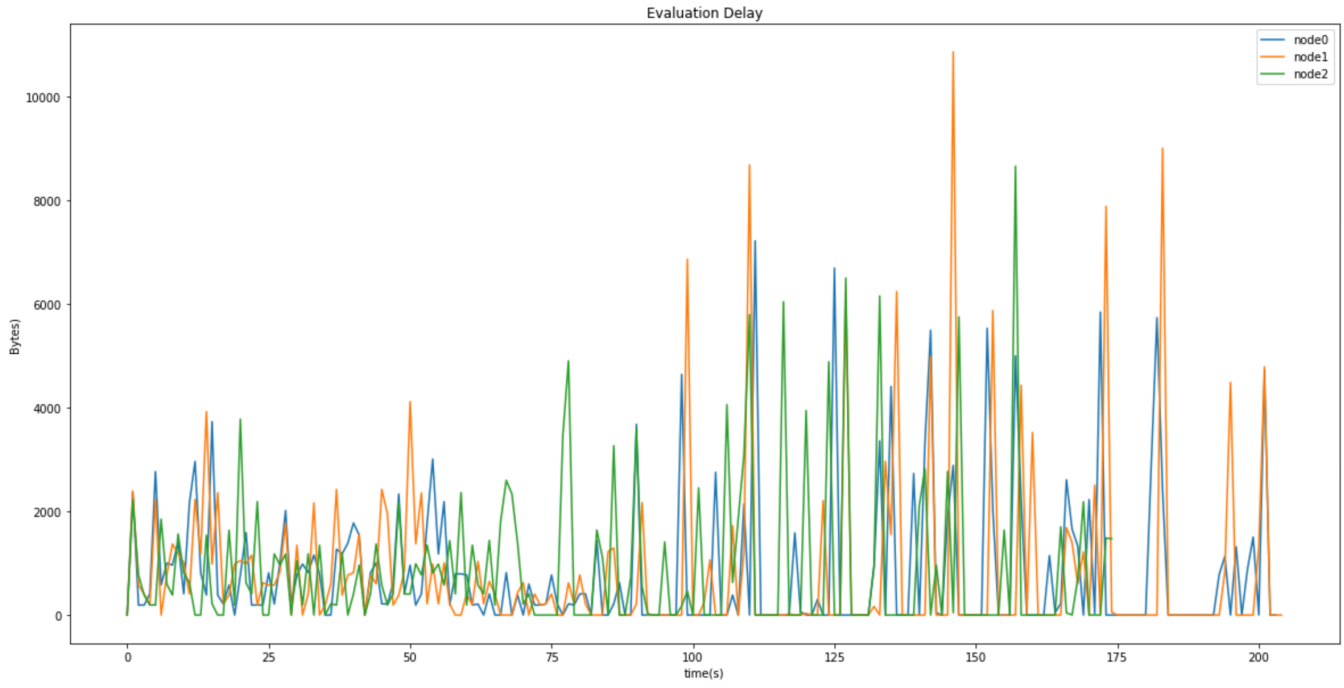


Scenario2: 8 nodes, 5 Hz each, running for 100 seconds

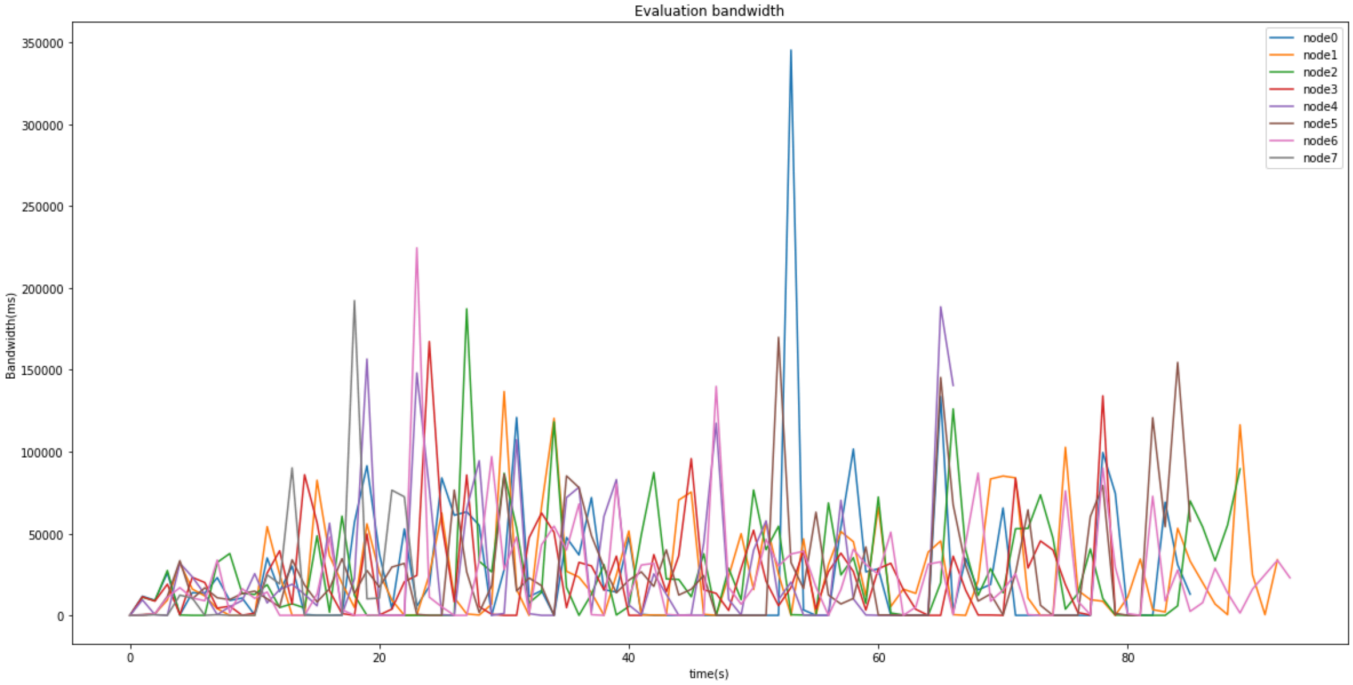


Scenario3 : 3 nodes, 0.5 Hz each, runing for 100 seconds, then one node fails, and the rest continue to run for 100 seconds





Scenario4: 8 nodes, 5 Hz each, running for 100 seconds, then 3 nodes fail simultaneously, and the rest continue to run for 100 seconds.


Evaluation bandwidth
Bandwidth(ms)
time(s)
node0
node1
node2
node3
node4
node5
node6
node7
350000
300000
250000
200000
150000
100000
50000
0
0
20
40
60
80