CS425 MP1

Christopher Coronado | ccoron7

Lishen He | lhe10

1. Information of the Project

URL: https://gitlab.engr.illinois.edu/lhe10/cs425-mp1.git

Version Number: 41444381706d4dbb6e1dee3880d24c6efee3c86b

Running instructions: python3 -u gentx.py 5.0 | python3 mp1\_node.py 3 [port #]

1. Design Document
   1. Protocol

We use a modification of ISIS algorithm that is more robust and decentralized. It is more robust because there are less corner cases in terms of node failures. It is more decentralized because we do not need the original sender node to be in charge of calculating priority of its message. Rather, we have each node to make its calculation. To justify this, as we are using R-multicast, each node’s reply will reach each other as long as one node successfully receives. Thus, each node has all the information it needs to make calculations.

A detailed protocol is described as follows.

Protocol

1. A newly generated message is r-multicasted to everyone else
2. A response is r-multicasted to everyone else if this is first time a node sees the message
3. A update on the priority of a specific message is given whenever a new response is seen
4. When an update on the priority is performed, we check if messages with highest priorities have received all responses from current living nodes. If so, mark it/them as “deliverable”
5. Deliver proper messages determined by ISIS algorithm

Message design

This protocol uses the following message pattern.

#msg = content + Org. Sender IP + Org. Sender Priority + Responder IP +\

Responder's Priority

#Example:

# DEPOSIT a 10, "172.22.156.169", 3, "172.22.156.170", 5

Processing incoming messages

To process incoming messages, we use the following algorithm.

"""

\* If the message is never seen[in terms of hash]

- add message hash into SEEN\_MSGS

- increment SELF\_PRIO by 1

- compare the priority in the data and calcualte max\_priority

- update metadata and save message/hash with the current maximum priority

- append meta-data to message and send to all

\* If the message has been seen[in terms of hash]

- If message only comes from self, then no key established for responses

-- update message metadata

- If message is recorded, but current response is new

-- calculate max priority and update if necessary

-- save this new response to a list/dict to a specific message(hash)

Note that the message generated by self will be seen here as well.

"""

* 1. Reliability

The reliability of this r-multicast communication is built on top of a B-multicast from Python socket() package. Whenever a node is r-multicast, it loops all current living communication channels, which it maintains, and send messages via B-multicast.

This means whenever a message is sent successfully to one node, it will be sent to all nodes.

* 1. Total Ordering

Our algorithm calculates total order in a slightly different(robust and easier to implement) fashion than the vanilla ISIS algorithm. The traditional ISIS algorithm has multiple stages for sender and responder and put certain responsibility to the sender.

Thus, to handle node failure properly, we need to consider node failure in many scenarios.

* 1. Node Failures

In our design, a node failure will be reflected in a global variable called “ALL\_CONNECTIONS”. This is a list of all the IPs that are available. To change a message from “undeliverable” to “deliverable”, the content in the list is checked against to make sure all the message has collected all responses from current living nodes.

1. Metrics and Graphs
   1. Bandwidth

Low scale(# nodes = 3, frequency = 0.5)

High scale(# nodes = 8, frequency = 5.0)

* 1. Delay for message

Low frequency(# nodes = 3; frequency = 0.5)

A close up of a pencil

Description automatically generated

High frequency(# nodes = 3; frequency = 5.0)