CS425 Distributed System MP2 Report - Group Membership

Tanqiu Liu (tanqiul2), Dehua Chen (dchen51)

Design - The group membership system uses a SWIM-like protocol. The Member Class is defined in Member.py.

(1) Failure detection: We use ping-ack messages to check whether a node is running or not. There are two threads running inside the daemon. One of them is the "sending" thread, which handles sending out pings and checking acks periodically. The other one is the "receiving" thread which continuously receiving messages from others. The two thread shares two queues which are protected by the thread lock, ackQueue and eventQueue.

For each period, the sending thread sends a ping message to a target from its member list. If it does not receive the ack (i.e. not in the ackQueue) from the target within pingTimeout, it randomly select pingReqK members from its member list as the indirect node and send indirect pings to the target. If the ack is not received from any of them by the beginning of the next period, the target is marked as "failed" and add that "fail" as an event into its eventQueue. Events generated or received within a period is piggybacked in all its sending-out message and sent to everyone who communicates with it in that period.

When the receiving thread receives a message, it handles different conditions according to the types of the message (e.g. PING, ACK, PINGREQ, etc). If it receives an ack message, it adds the ack to the ackQueue. If there are events piggybacked in the message, it adds the events to its eventQueue.

We use Google's Protocol Buffer to build our inter-process communication protocol to fulfill the marshaled message format. The message protocol is defined in membership.proto. A message will have required fields including sourceId, seqNum, msgType. An optional field call targetId only appears in indirect ping messages. And an events array which carries the information of FAIL, JOIN, LEAVE events.

It scales to N because our failure system uses a swim like protocol to ping one other node in its membership list per period of time time. It also sends back acks to the nodes that ping it within at least 1 period so that other nodes know it is alive. The events propagate allowing other nodes to receive info about other nodes in a gossip infection style once information is disseminated from the introducer

Mp1 helped us to query a log called membership.log where we used it to help find Joins, Fails, and Leaves along with the sourceId and timestamp. It also helped for measurements and plotting.

(2) Join & leave: A leaving event is different from crash since the leaving node sends pings to all other nodes in the membership list telling others about it leaving and then exiting gracefully.

A joining node knows the Introducer so it adds the introducer into its membership list. The node then sends a ping with a join event into the introducer so the introducer knows to add the new node into its membership list. The introducer then sends a ping to every node in its membership list that a new node has joined in the membership list and also sends a copy of its membership list.

For finding the bandwidth usage, we logged the size of the serialized packet for pinging and acking along with a timestamp of when the packet was sent or received. We took a time period that made sense for the situation such as a time way after the 4 vms joined each other to check for background bandwidth usage when no membership list changes are being made. Repeat for JOIN, LEAVE, and FAIL.

In Bytes [73, 17, 17, 73, 73, 72, 72, 73, 73, 73, 73, 73, 17, 17, 73, 73] 3.65135s timespan

i) On average for 4 machines after all machines joined: 257.986416752 Bps. We found out that when our encoded message is serialized to byes, our id is not serialized and it is very long because it is made from the IP, Port, and a timestamp thus giving an increase in Byte size to the size of Pings and Acks.

ii) JOIN: [183, 73, 183, 127, 357, 183, 183, 183] 0.76844286918s timespan

The join command averages around 1938.98605 Bps for the 4th vm joined. The burst in one packet is because of the new joining node getting a ping with all the nodes in the membership list currently and also all nodes getting a ping with the join event for the new node.

LEAVE: [72, 72, 182, 72, 72, 72, 72, 72, 73] 1.56346 timespan

The average of leave is around 485.46173 Bps for each machine that receives the leave event. This is because once the leave event is received the member is popped from the local memberlist and everything is back to normal.

Fail: [72, 72, 149, 72, 182, 236, 72, 149, 72, 72] 2.35043215752s timespan

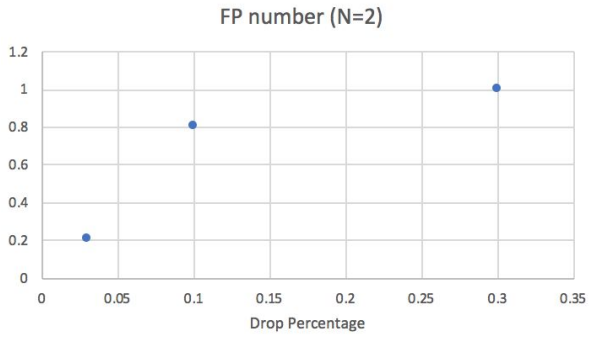
The average Bps for when one of the 4 vms failed is 488.42081 Bps. This is the Bps that each vm receives when there is a FAIL event in the pings that propagate around.

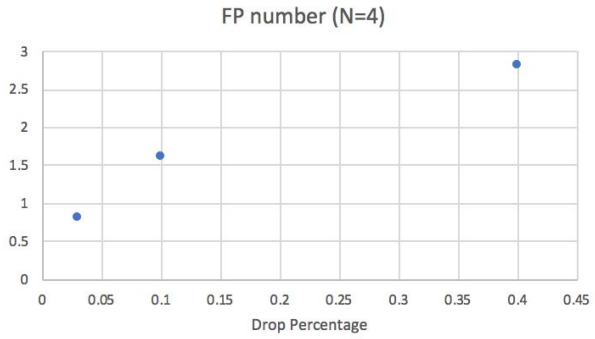
2 Machines #False positives within 5 seconds

% Drop	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Mean	Conf. Intv. (95%)	Std. Dev
3	0	0	0	1	0	0.2	[-0.355, 0.755]	0.447
10	1	1	0	1	1	0.8	[0.245, 1.355]	0.447
30	1	1	1	1	1	1	[1, 1]	0

4 Machines #False positives within 5 seconds

% Drop	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Mean	Conf. Intv. (95%)	Std. Dev
3	2	0	1	1	0	0.8	[-0.239, 1.839]	0.837
10	2	1	2	2	1	1.6	[0.92, 2.28]	0.548
30	3	3	3	2	3	2.8	[2.245, 3.355]	0.447





This trend makes sense in that once the percent drops become higher the false positive rates start rising along with a shorter interval of time to reach those false positives as well. We used a period of 0.5 and a

ping timeout of 0.15 so around 2 packets were sent per vm for one machine. Increasing the size of the machines will increase the false positive rates as well because each machine added is basically another chance for the false positive rate to go up.