Jiachen Yuan & Sizheng Zhang 09/16/23

CS-425 MP2 Report

Design & Algorithm Overview

We designed two variants of heartbeat-style failure detectors (*Gossip* & *Gossip+S*) using the introducer-based approach. Each node is spawned with two coroutines, one that periodically refreshes the membership list to filter out LEFT/FAILED nodes and sends the latest list to a fixed fanout of random nodes, and the other updates its own membership list with the received list from other nodes. When a node joins, it first contacts the hardcoded introducer node to fetch information about other nodes in the cluster. We chose Google's Protocol Buffers as our serialization and deserialization method because it is a more bandwidth-efficient scheme for other obvious choices such as json or xml. All gossip messages, including messages related to join and leave, are sent using UDP. External controls to the entire group (drop_rate change and suspicion mode toggle) are sent using TCP instead for simplicity. Our system adopts the gossip-style multicast, in which a group message can disseminate in O(log(N)) periods, and it has a time-bounded completeness of 5 seconds and relatively low false-positive rate with manually introduced packet drop rates.

Time-Bounded Completeness:

In the steady state of our system, every node sends a heartbeat every 500 milliseconds to random 2 peers. Given the $O(logN)$ complexity of gossip-style multicast, our dissemination time is $(network\ latency + machine\ latency + period\ time) * O(log_2 N) \approx 0.5 * 4 = 2\ seconds$

In the *Gossip* mode, we set our T_FAIL[1] to be 4 seconds, which is smaller than 5 seconds. In the *Gossip + S* mode, we set our T_FAIL and T_SUSPECT[2] to be 2 seconds, which adds up to 4 seconds. From these numbers we can tell that our messages are spreaded quick enough and every failure will be catched at around 4 seconds if a node's heartbeat is not received for around 4 times, which is within 5 seconds, in both modes of operation.

Debugging

We used MP1's distributed log querier to analyze logs across machines in the cluster. It has helped us pinpoint multiple problems including socket blocking and incorrect state transfers.

Evaluation

We ran our experiment on 10 VMs for both part 1 and part 2 of the report. In part 1, we fixed our detection time bound to around 4 seconds. In the *Gossip* mode, we set T_FAIL to be 4 seconds; in the *Gossip + S* mode, we set our T_FAIL and T_SUSPECT to be 2 seconds respectively.
In part 2, we fixed our average background bandwidth to roughly 750 Bps by restricting the T_GOSSIP to be 1000ms for both *Gossip* and *Gossip + S* mode. Since we don't have the 5 seconds restriction in detection time in part 2, we set T_SUSPECT to be 4 seconds and T_FAIL to be 2 seconds in order to reduce false positive rate for the *Gossip+S* mode.

Part 1

a) Figure 1 shows the bandwidth usage (incoming and outgoing) of the two modes. The plot shows that the bandwidth of *Gossip+S* is slightly larger than that of *Gossip* for almost all drop rates. This is because by experiments we found that we can slightly decrease the frequency of gossiping for *Gossip* mode but still achieve a similar detection time to that of *Gossip+S,* thus slightly decreasing its bandwidth consumption compared to *Gossip+S.* Both modes have diminishing bandwidth consumption until 0KB/s at 0.7 loss rate because as drop rate increases, more and more nodes are disconnected from the group until every node is disjoint and no longer participates in the gossiping.

[1] Time after which a node will be marked FAILED locally if no update is heard from the node
[2] Time after which a node will be marked SUSPECTED locally if no update is heard from the node

b) Figure 2 exhibits the false positive rate[3] of the two modes. When the message drop rate increases, the false positive rate will also increase, but the number of false positives in the *Gossip+S* mode is fewer than that in *Gossip* mode because the SUSPECTED stage serves as a buffer for false positives. However, when the message drop rate goes to 0.7, the false positive rate of both modes arrives around 110 because almost all nodes were marked as FAILED and deleted from each other's membership list, so that no FAILED gossip messages will be disseminated in the cluster.

c) Figure 3 shows the relationship between simultaneous failures and failure detection time in the two modes. Despite the measurement fluctuations and random errors at around 4.3s detection time, the detection time of two modes stay within 5% of each other. The downward trend of *Gossip+S* is explained in detail in section 2 a).

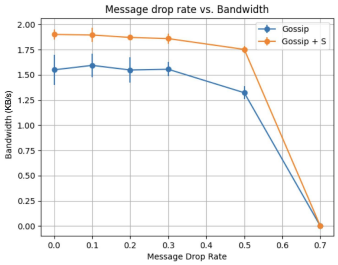

Figure 1

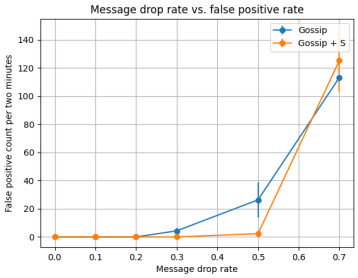

Figure 2

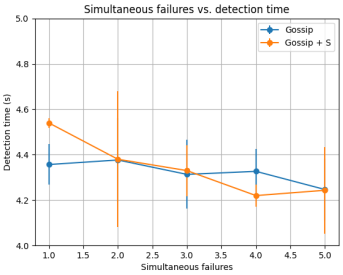

Figure 3

Part 2

a) Figure 4 shows the relationship between simultaneous failures and failure detection time in the two modes. The failure detection time of *Gossip + S* is about 2 seconds more than that of *Gossip* due to the existence of T_SUSPECT. Also, as the number of failures increase, failure detection times show a slightly downward trend in *Gossip + S* because as the number of alive node decreases, it is quicker for SUSPECT gossips to be disseminated among the rest of the nodes, so it might take the rest of the nodes slightly less time to mark SUSPECTED nodes as FAILED.

b) Figure 5 exhibits the false positive rate of the two modes. It shows a similar relationship and trend to figure 2 for similar reasons. The advantage of the *Gossip+S* mode is shown here as it can absorb more false positives than the *Gossip* mode.

c) Figure 6 shows the total bandwidth usage of the two modes after restricting the bandwidth cap. No matter what the message drop rate is, the bandwidth of the two modes are within 5% of each other. (The downward trend was also explained in answer 1 a).

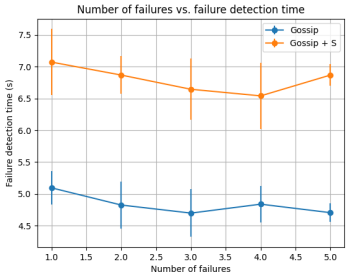

Figure 4

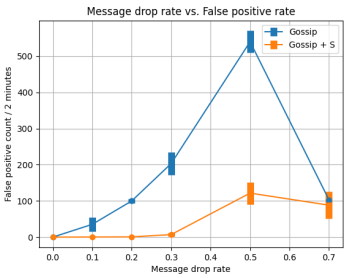

Figure 5

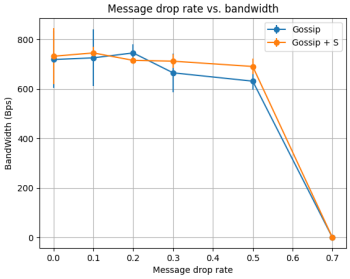

Figure 6

[3] We defined false positive rate as the number of false positive reports within 2 minutes