Report

CS425-MP2-Group 9

Design Overview

In this MP we implemented Failure detector module as a set of daemon threads along with the main application thread. We spawned three threads in total apart from the main application. Two of those threads were responsible for sending and receiving failure detector module messages over UDP, and one final thread for TCP connection only used to get membership list from introducer when a process joins the group.
We used a special introducer node which was responsible for making new processes join the group. Although we used a special introducer node in our implementation it is fault tolerant in the way that, even if introducer fails rest of the processes can continue failure detector module without any problem, and introducer failure only limits joining new processes in the group. Our application is fast *(detects failure within 3 seconds and disseminates within 6 sec),* complete and consistent with very low false positive rate. We took into account various cases involving introducer failure, high network latencies and scalability while implementing the MP.

Algorithm Used

Our failure detector module is based on SWIM algorithm as described in the class. Apart from the failure detector algorithm of SWIM, we used a special introducer node for the joining process. When a process needs to join a group it initiates a tcp connection with the introducer and it receives the membership list from introducer. The introducer then disseminates the join information to other nodes. Also, when a process needs to leave the group it just disseminates a leave message along with it's ping and awk messages for *O(log(N)*T_protocol)* time and then exits.
For sending messages in the network we serialized the message information into strings and deserialized the string received into relevant message information at the receiver.

Scalability

In our implementation, per node message load is constant, along with constant expected failure detection time and O(log(N)) dissemination time. Introducer used is just any node but one extra function of joining and our implementation doesn't break in case of it's failure. TCP is used sparingly only for joins which makes our algorithm highly scalable and robust.

Debugging

We used information in our logs to debug our application, which was made possible with the help of distributed grepClient built in MP1. Apart from being a great help in debugging MP2, grepClient was also used to gather all the statistics from logs in our experiments.

Experiments

Bandwidth

We measure the background bandwidth usage by using a counter to count the total number of bytes sent per protocol time period (1 sec). Number of nodes is chosen as 4 for all bandwidth experiments. The **background bandwidth measured 44 bytes/sec**.
In case of node failure, The measured bandwidth goes up to **206 bytes/sec for the first detector** of failure and for other nodes it goes up to **92 bytes/sec**. This increase in case of first detector can be attributed to the fact that the the first detector sends ping-requests to k (=2 in our case) random other nodes. It should also be noted that the ping-requests are longer messages themselves. The increase in bandwidth for other nodes can be attributed to the flooding of failure information piggy-backed with pings and acks. In case of node leave as well as node join, we see that the bandwidth used goes up

to **92 bytes/sec** in both cases. This increase can again be attributed to the flooding of failure information piggy-backed with pings and acks.

Lossy network

To measure false positive rate in case of lossy network, we implement a method which decides on whether to drop a received packet based on given probability distribution parameters. The plots shown below report the mean of percentage FPR, confidence interval of percent FPR and standard deviation of percent FPR respectively per protocol time.

Let us discuss the trends seen in the graph. The mean FPR increases as packet loss percentage increases. This is expected because as packets are lost at a higher rate, it becomes more likely that for a ping or a ping-request, an ack is not received because packet is dropped in between. The confidence intervals and standard deviations also increase as packet loss percentage increases.

For N=2, the mean FPR is almost same as the packet loss rate which means everytime a packet is lost, a false failure is detected. This is because for N=2, The ping-requests are not sent because there is no other 3rd member node to send ping-request to. So essentially there are no ping-requests and ack-requests. Implementing a suspect mechanism could improve this, but our present version does not have a suspect mechanism. For N=4, we can see that the FPR is lower than the packet loss rate. This is because, now with more than 2 members in the system, ping-requests and ack-requests become functional which reduce the FPR. We also see that the confidence intervals and standard deviations of N=4 readings are smaller than that of N=2. This is because these functions are dependent on the probability of not receiving ack for a ping/ping-request. For N=2, this probability is higher than N=4 because of non-functional ping-request mechanism.