# **Team55 (yixin10, yangt2) – MP2 Report**

## **Algorithm Design**

In our system, machines are organized as a bi-directional ring, where each machine’s neighbors consist of two predecessors and two successors. Thus, each process is monitored by four neighbors, and using this topology, our system will function as usual in the case when there are three simultaneous failures. All the messages in our system are transmitted through UDP.

When a process is started, its status should be initialized as *LEAVE* while the introducer (VM 1 in our system) is initialized as *ALIVE*. When a process receives *Join* command, it will start a Join Thread, and will send *JOIN* message directly to introducer. When the introducer receives *JOIN* message, it will first update its own membership list and mark the newly joined machine as *ALIVE*, and then send its own complete membership list to joined machine so that the joined machine obtains the membership list at current stage, and finally it uses multicast to notify all other alive processes to update their membership list.

When a process wants to leave, it will send *LEAVE* message to all its alive neighbors, who will update the membership list and mark the left machine as *LEAVE*, and disseminate the *LEAVE* message to their neighbors, until all the alive processes perceive the departure of the machine.

In addition, there is a thread named Sender which periodically sends *PING* message to its neighbors and waits for their *ACK* responses. To check process timeout, it records the time when the *PING* message is sent for each neighbor. And there is another thread called TimeoutChecker which checks the *PING* records periodically by comparing the current time and the recorded time, and if the difference is greater than *TIMEOUT* value (in our design, 1 seconds), it will erase the record and mark the process as *FAILED*.

## **Scalability of System**

Our system scales to large N because, each process sends all types of messages only to its two predecessors and two successors (except that introducer multicasts *JOIN* message when new process joins the group), thus there is equal load on each member, and there is no bottleneck on any process. In addition, there is no single point failure problem in our system, and the failure of any member will not affect the functionality of the system.

## **Marshaled Message Format**

All messages in our system are transmitted in the form of JSON string. Depends on specific situation, JSON string contains different key-value information. Basically, it contains information such as type of message, hostname of the left, failed, or joined machine and the time when the message is sent.

## **Satisfaction of Three Completeness Requirements**

**(1) 5 second completeness:** consider the worst case when one process fails right after it responds *ACK* message to *PING* sender. After 1 second, the sender will send another *PING* message to the failed process, without receiving *ACK* within 1 second. At this point, since TimeoutChecker checks record every 1 seconds, and if we consider the worst case, TimeoutChecker waits for 1 seconds and then begins checking and will detect failed process. Therefore, the failure of process can be detected by the sender within at most 5 seconds.

**(2) Meets completeness up to 3 failures:** Since in our system each process is monitored by four neighbors, even in the worst case where three adjacent machines fail simultaneously, all the failure can still be detected Thus, the system meets completeness up to 3 failures.

**(3) Violation of completeness with 4 simultaneous failures:** Let’s consider the worst case. For example, there are 5 adjacent machines a, b, c, d, e, and machines a, b, d and e fail simultaneously. In this case, the failure of c will never be detected because all its neighbors are failed. Thus, the system will violate completeness with 4 simultaneous failures in the worst case.

## **How to Use MP1**

When Server receives messages, it will generate a log file locally, we can then use MP1 to get the information we want.

1. **Bandwidth Measurements**

We used **iftop** to monitor the bandwidth of our group:

**(1) Bandwidth Usage for N=6 machines:** Every machine needs to both send PING to alive neighbors and send ACK to respond to PING, the average bandwidth usage is 10.5Kb/s.

**(2) Average bandwidth usage for a node joins:** The newly joined machine sends JOIN message, then introducer sends back membership list, and multicast. Average bandwidth usage is 18.8Kb/s.

**(3) Average bandwidth usage for a node leaves:** The leaving machine sends LEAVE message to neighbors, who again disseminate LEAVE message until all alive machines detect the left machine. Average bandwidth usage is 13.2Kb/s.

**(4) Average bandwidth usage for a node fails:** The first machine that detects the failure will send FAIL message to neighbors, who again disseminate FAIL message until all alive machines perceive the failed machine. Average bandwidth usage is13.6Kb/s.

## **False Positive Rate**

In this part, we assume the message loss rate is 3% and 30% for N = 2 and N = 6 groups. We used the probability to randomly drop messages before it is sent. Figure 1 shows average false positive rate for each situation, and Figure 2 shows standard deviation of each situation. Table 1 also shows the data.

Table 1 – False positive rate information

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **N=2, 3% packet loss** | **N=2, 30% packet loss** | **N=6, 3% packet loss** | **N=6, 30% packet loss** |
| **Average** | 0.0135 | 0.2374 | 0.0401 | 0.3033 |
| **Standard Deviation** | 0.0179 | 0.1617 | 0.0206 | 0.1553 |
| **Confidence Interval** | [0.0006, 0.0263] | [0.1217, 0.3531] | [0.0253, 0.0548] | [0.1922, 0.4144] |

Chart, bar chart

Description automatically generated Chart, bar chart

Description automatically generated

Figure 1 – Average false positive rage Figure 2 – Standard deviation of false positive rage

## **Discussion About Data**

In our design, we set the timeout for ACK is 1 second, and we will Ping a node’s neighbors every second. It can be seen from the plot that when N is small the false positive rate is relatively small, and when N increases, message will have more chances to be dropped, so we have higher false positive rate than N = 2. And because there is a huge gap between message loss rate of 3% and 30%, we can see that the average false positive rate can be very different when N is equal. But when the loss rate is 30%, we can see the difference between N = 2 and N =6 is relatively small, and this is because the loss rate is large, so it can easily report a wrong false failure detection even when N = 2. And the 95% confidence interval is calculated for each data point.