MP 1

Created by Borisov, Nikita, last modified by Mehar, Jayasi on Jan 31, 2017

Due: Thu, Feb 9, 11:59 pm (CP1), Tue, Feb 28, 11:59 pm (CP2)

Demos will be conducted on the day after the MPs are due. Details will be given later.

You are encouraged to start this MP timely since it involves considerable design, development, debugging and experimentation from your end. Whether you are doing the MP individually or as a team, the expectations and requirements are the same. We recommend not leaving the work for the days before the deadline. Please use Piazza for any clarifications.

In this MP you will implement a chat system that supports **totally ordered multicast** and **failure detection**. The MP will be carried out in two checkpoints.

Basic functionality

You will implement a *chat room* functionality. Each client should implement two functions: take messages as input and display messages from all participants in the chat room (*including* messages from oneself). The messages should be displayed in a **total ordering**, i.e., all clients should see the same messages displayed in the same order. Any client in the chat room can leave or fail at any point of time and other clients in the chat room should be able to detect these failures.

Checkpoint 1

For the first check point, you will implement a version of the chat room with **reliable nodes** over a **reliable** network with **no ordering guarantees**. This checkpoint is intended to make sure you are comfortable sending messages back and forth and printing them out. We encourage you to consider the design of the next checkpoint while working on this one so that you can reuse as much of your implementation as possible.

You will need to start a group of 5 chat nodes that all connect to each other. The list of the nodes in the chat group can be static and hard-coded in your implementation. You may want to add a start-up delay before trying to make connections to allow all nodes to come online. Use socket communication to make sure that a message sent by any of the clients is received by all other clients in the group. Your nodes may use any type of message passing protocol, such as TCP or UDP. You do not need to worry about failures of the channel or node failures.

The demo for this checkpoint will consist of showing that you can start up to **5** chat clients that can connect to each other and that each can see each other's messages.

Checkpoint 2

For the second checkpoint, you will need to enforce **total ordering** over an **unreliable network** and deal with **node failures**. Total ordering means that the order of received messages should appear in the same order as before. Your protocol should also be able to tolerate the failure of **any** nodes, and the failure of **two or more** nodes. You should implement a failure detector that notices when a chat node has failed; all remaining live nodes should notify their users that the corresponding chat nodes have failed. This notification **does not** have to be totally-ordered with the rest of the messages or other, i.e., node A might print that node C has failed before showing message X, whereas node B might show these in the opposite order. Any messages sent by the failed node, however, should always be delivered **before** the failure notification. E.g., the following transcript should never happen:

Transcript
A: Hi everyone!

B: Hello, how is it going?

C: My computer seems unstable

C has left conversation

C: I think it might crash any minute...

Nodes that have failed will never rejoin the chat.

During the demo you will need to show that your chat program supports up to **8** nodes communicating together, with messages being delivered in a total order. For testing, we will use a network that introduces randomized delays between nodes such that messages may arrive at different orders. You also need to show that your nodes properly detect one or more failed nodes, and continues functioning.

Instructions

- You have an option of working with Java, Python, Go or C++ for your MPs.
- Please create a private git repository on Gitlab between you and your team mate. Please don't make your source code public till the end of the semester.
- The code submitted on Gitlab before the deadline would be counted as your submission.
- We will be running plagiarism detection on all your code.
- You are required to add a README.md file to your repository which has all the instructions about how to set up your code. The TAs might run your code again after the demo.
- Add all the course staff to your repository with **Reporter** access. The following are the net ids to be added:
  - `nikita, jmehar2, ideshpa2, tzhu13, xzhan160`
- Every team has been assigned a VM cluster, the details of which can be found on Piazza. It is your responsibility to ensure that your code is functional on the VMs that are assigned to you. If you face any issues with the VMs, please contact Engr-IT. It may take them a short while to respond, so make sure not to leave testing on VMs until the end.
- Add the design document to your repository and bring a print out of the same to the demo, the details of which are in the following section.

Design Document

In addition to the demo, you will be required to submit a design document of 2–3 pages, describing your design for reliable, ordered multicast and failure detection. This has to be submitted only for the second checkpoint. You will need to describe the algorithm you chose and explain why you chose it over other potential alternatives, as well as how some of the implementation details and parameters were decided on.

You are required to plot suitable metrics with respect to number of nodes in the cluster to show that your system is scalable. Explain these graphs. Take multiple readings and plot averages as well as confidence intervals.

You should also describe your approach to testing your design.

Grading rubric

CP1 Demo	
Demo chat functionality with 2 nodes	10%
Demo chat functionality with 5 nodes	10%
CP2 Demo	
Demo chat functionality with 8 nodes	10%
Demo total ordering with randomized network delays	20%

Demo tolerating 1 node failure	10%
Demo tolerating multiple node failures	10%
Design document	20%
Code quality	10%

Academic Integrity

You are not allowed to post solutions, ideas or any code on Piazza. **Violations of academic integrity, including any sharing of code outside of your group, will be punished severely**; see Section 1-402 of the Student Code as well as the Computer Science Honor Code for more information. Minimum penalty: 0 grade on the MP. Maximum penalty: expulsion. All cases will be reported to CS, your college and the senate committee.

Start early and happy coding!

No labels