CS425, Distributed Systems: Fall 2020
Machine Programming 1 – Distributed Group Membership

Released Date: September 4, 2020

Due Date (Hard Deadline): Sunday September 27, 2020 (Code+Report due at 11.59 PM)

Demos on Monday September 28, 2020

Covfefe! Inc. (MP1) just got acquired by the fictitious FakeNews Inc. (this company's business model is to detect, not generate, fake news – go figure!). Anyway, this company liked your previous work while you were hired by Covfefe! Inc., so they've commissioned you to build a distributed group membership service for them.

You must work in groups of up to 3 for this MP. You can (and should) start on this MP early, and be able to make substantial progress even before you have VM access.

This service maintains, at each machine in the system (at a daemon process), a list of the other machines that are connected and up. This membership list is a full membership list, and needs to be updated whenever:

1. A machine (or its daemon) joins the group;
2. A machine (or its daemon) voluntarily leaves the group; and
3. A machine (or its daemon) crashes from the group (you may assume that the machine does not recover for a long enough time).

There is only one group at any point of time. Since we're implementing the crash/fail-stop model, when a machine rejoins, it must do so with an id that includes a timestamp – this distinguishes successive incarnations of the same machine (these are the ids held in the membership lists). Notice that the id also has to contain the IP address.

A machine failure must be reflected in at least one membership lists within 5 seconds (assuming synchronized clocks) – this is called *time-bounded completeness*, and it must be provided no matter what the network latencies are. A machine failure, join or leave must be reflected within 6 seconds at *all* membership lists, assuming small network latencies.

You're told that at most **three** machines can fail simultaneously for gossip-style heartbeating and any number of simultaneous failures can occur for all to all heartbeating. After a set of back-to-back failures, the next set of failure(s) don't happen for at least 20 seconds. Your system must ensure completeness for all such failures (up to three simultaneous failures).

Your algorithm must be scalable to large numbers of machines. For your experiments however, you may assume that you have N > 5 machines in the group at any given time. Note that this is not a limit on the set of machines eligible to be group members. Typical runs will involve about 7-10 VMs.

You **must use** the heartbeating style of failure detection. Implement **two variants**: gossip-style heartbeating (as discussed in class), and all to all heartbeating (as discussed in class). You will also get to compare them experimentally (see Report section). Think of the parameter settings you need (frequency of heartbeats and gossip, timeouts, etc.) to achieve the time bounds shown above.

Design first, then implement. Keep your design (and implementation) as simple as possible. Use the adage "KISS: Keep It Simple Si…". Otherwise FakeNews Inc. may generate fake news about you, give you fake points, and then have KISS sing it. (I know, that's probably a Dad joke. Bear with me. I'm trying to keep our lives entertaining here.)

For the failure detection or leaves, you cannot use a master or leader, since its failure must be detected as well. However, to enable machines to join the group, you can have a fixed contact machine that all potential members know about, which you already know is called the "introducer". When the introducer is down, no new members can join the group until the contact has rejoined – but the rest of the group should proceed normally including failures should still being detected, and leaves being allowed.

Pay attention to the format of messages that are sent between machines. Ensure that any platform-dependent fields (e.g., ints) are marshaled into a platform-independent format. An example is Google's Protocol Buffers (this is not a requirement, especially since it is not installed on CS VM Cluster). You can invent your own, but do specify it clearly in your report.

Make your implementation bandwidth-efficient. Your implementation must use UDP (cheap).

Create logs at each machine, and use MP0 (if you chose to do MP0) to debug. These logs are important as we will be asking you to grep them at demo time. You can make your logs as verbose as you want them, but at the least you must log: 1) each time a change is made to the local membership list (join, leave or failure) and 2) each time a failure is detected or communicated from one machine to another. We will request to see the log entries at demo time, via local grep or remote grep (or MP0 if you implemented it).

We also recommend (but don't require) writing unit tests for each of the join, leave, and failure functionalities. At the least, ensure that these actually work for a long series of join/leave/fail events.

Machines: We will be using the CS VM Cluster machines. You will be using 7-10 VMs for the demo. The VMs do not have persistent storage, so you are required to use git to manage your code. To access git from the VMs, use the same instructions as MP1.

Demo: Demos are usually scheduled on the Monday right after the MP is due. The demos will be on the CS VM Cluster machines. You must use all VMs for your demo (details will be posted on Piazza closer to the demo date). Please make sure your code runs on the CS VM Cluster machines, especially if you've used your own machines/laptops to do most of your coding. Please make sure that any third party code you use is installable on CS VM Cluster. Further demo details and a signup sheet will be made available closer to the date.

Language: Choose your favorite language! Student groups in the past have used C/C++/Java/Go. We will release "Best MPs" from the class in these languages only (so you can use them in subsequent MPs). Other languages used by students include Rust.

Report: Write a report of less than 1 page (12 pt font, typed only - no handwritten reports please!). For each of the following categories measure and draw a plot that contains two lines (or two bar graphs), with standard deviation bars: one line/bar for all-to-all and one for gossip-style heartbeating: (i) background bandwidth usage (in Bps not messages per second) vs. number of machines up to the max (assuming no membership changes), (ii) false positive rate of your membership service when the message loss rate goes from 1% to 20%.
For each plot, choose at least 5 values on x axis. For each data point take at least as many readings as is necessary to get a non-zero false positive rate (at least 5 readings each), and plot averages **and** standard deviations (and, if you can, confidence intervals). Discuss your plots, don't just put them on paper, i.e., discuss trends briefly, and whether they are what you expect or not (why or why not). (Measurement numbers don't lie, but we need to make sense of them!) Stay within page limit – for every line over the page limit you will lose 1 point!

Submission: There will be a demo of each group's project code. Submit your report (softcopy) as well as working code by the deadline time. Please include a README explaining how to compile and run your code. Submission instructions are similar to previous MPs.

When should I start? Start NOW. You already know all the necessary class material to do this MP. Each MP involves a significant amount of planning, design, and implementation/debugging/experimentation work. **Do not** leave all the work for the days before the deadline – **there will be no extensions**.

Evaluation Break-up: Demo [40%], Report (including design and plots) [40%], Code readability and comments [20%].

Academic Integrity: You cannot look at others' solutions, whether from this year or past years. We will run Moss to check for copying within and outside this class – first offense results in a zero grade on the MP, and second offense results in an F in the course. There are past examples of students penalized in both those ways, so just don't cheat. You can only discuss the MP spec and lecture concepts with the class students and forum, but not solutions, ideas, or code (if we see you posting code on the forum, that's a zero on the MP).
We recommend you stick with the same group from one MP to the next (this helps keep the VM mapping sane on IT's end), except for exceptional circumstances. If you are in a group of size > 1, we expect all group members to contribute about equally to the overall effort. If you believe your group members are not, please have "the talk" with them first, give them a second chance. If that doesn't work either, please approach Indy.
FakeNews Inc. is watching and if you cheat, it will be very Sad!

Happy Membership (from us and the fictitious FakeNews Inc.)!