CS425, Distributed Systems: Fall 2023
Machine Programming 2 – Distributed Group Membership

Released Date: September 11, 2023
Due Date (Hard Deadline): Sunday September 24, 2023 (Code and report due at 11.59 PM)
Demos on Monday September 25, 2023

The time for this MP is rather short! So please start early! Start now!
You must work in groups of two for this MP. Please stick with the groups you formed for MP1. (see end of document for expectations from group members.)

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!). 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 two for this MP.

This service maintains, at each machine in the system (at a daemon process), a list of the group, i.e., 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 a node id that includes not only IP and port but also a timestamp or version number which distinguishes successive versions of the same machine (this node id is what is held in the membership lists). This timestamp/version is needed because our model is a fail-stop model and not a fail-recovery model, so subsequent rejoins of the same VM must be distinguishable. (Don't confuse these timestamps/version numbers with suspicion incarnations, which mean something different).

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**, and after three back-to-back failures, the next set of failure(s) don't happen for at least 20 seconds (i.e., enough time for your system to converge back to a good topology). Your system must ensure completeness for all such failures (up to three simultaneous failures). That is, whenever a process fails (and there are up to 3 simultaneous failures in the system), all membership lists that contained the process must be updated within 20 s.

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**:

I. **Gossip**: Gossip-style heartbeating (as discussed in class), and
II. **Gossip+S**: Gossip-style heartbeating with a Suspicion mechanism (like SWIM, but without any pinging).

DO NOT use ping-ack style failure detection like SWIM (though you can borrow the Suspicion style from SWIM's design). You will also get to compare these variants experimentally (see Report section). Think of the parameter settings you need (frequency of heartbeats and gossip, timeouts, etc.) to achieve the time bounds mentioned 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 (the musical group) sing it in a duet with Taylor Swift. (pa-dam-dishoom!)

In your report, justify your design choices and why your design meets 5 second completeness. Your algorithm must use a small bandwidth (defined as Bytes per second, and NOT as messages per second) needed to meet the above requirements.

For the failure detection or leaves, you cannot use a leader (archaic: master), 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 (joining) 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 (converted) 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 if possible, use MP1 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 the MP1's querier. Thus, make sure you integrate MP2 with MP1 to make this possible.
You should also use your MP1 solution for debugging MP2 (and mention how useful this was in the report). (If this is not possible for MP2 because you didn't finish MP1, then you can cook up a simple grep engine for MP2. But for future MP3 and onwards, make sure MP1 is integrated.)

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.

For the demo, please create commands so that: (1) you can switch your system between Gossip and Gossip+S, with the same membership list (i.e., without needing to re-add nodes, reboot, etc.), and (2) induce a specified message drop rate (default = 0%).

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.
We expect both partners to contribute equivalent amounts of effort during the entire MP execution (not just in the demo).

Language: Choose your favorite language! We recommend C/C++/Java/Go/Rust. We will release "Best MPs" from the class in these languages only (so you can use them in subsequent MPs).

Report: Write a report of less than 2 pages (12 pt font, typed only - no handwritten reports please!). For each of the following questions measure and draw a plot that contains two lines (or two bar graphs), with standard deviation bars: one line/bar for Gossip and one line/bar for Gossip+S:

1. Given a fixed detection time bound (5 seconds) that applies to both Gossip and Gossip+S (average detection times within 5% of each other is ok), draw plots to compare: (a) the bandwidth between Gossip vs. Gossip+S, in the no-failure scenario, and (b) false positive rate when there are no failures (you may have to introduce artificial message drops to induce false positives). (c) Also draw a plot comparing their detection times as a function of number of failures.

2. Ignore the 5 second requirement. Say base background bandwidth = background bandwidth under no failure/zero message loss rate, i.e., without suspicions. Given a (fixed) cap on average base background bandwidth usage (in Bps not messages per second) – note that this means that both algorithms should have identical (within 5%) average base bandwidth usage – compare (a) the detection time for failures, as a function of number of simultaneous failures, (b) false positive rate when there are no failures (to induce false positives, introduce artificial message drops to induce false positives – to do so, drop messages on the receiving end rather than the sending end, with the same drop probability applied across all messages; vary this drop probability at small values 0%, 1%, 5%, 10%, etc.). (c) Also give the bandwidth usages of Gossip and Gossip+S (and verify that they are within 5% of each other).

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. On Gradescope submit your report by the deadline (11.59 PM Sunday). In the git, also by the same deadline, submit your report as well as working code; please include a README explaining how to compile and run your code. Other 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). FakeNews Inc. is watching and will be very Sad!

We recommend you stick with the same group from one MP to the next (this helps keep the VM mapping sane on EngrIT's end), except for exceptional circumstances. We expect all group members to contribute about equivalently 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.

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