CS425, Distributed Systems: Fall 2016
Machine Programming 2 – Distributed Group Membership

Released Date: September 13, 2016
Due Date (Hard Deadline): Sunday October 2, 2016 (Code due at 11.59 PM)
Demos on Monday October 3, 2016

You'reFired! Inc. (MP1) just got acquired by the fictitious TheTribeHasSpoken Inc. (this company's business model is to throw people off of islands – go figure!). This company liked your previous work while you were hired by You'reFired! 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 other machines that are connected and up. This membership list 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 3 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 **two** machines can fail simultaneously, and after two back-to-back failures, the next set of failure(s) don't happen for at least 20 seconds.

Your algorithm must be scalable to large numbers of machines. For your experiments however, you may assume that you have N > 4 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 VMs.

You **must use** the **ping-ack** style of failure detection discussed in class (SWIM-style, though the indirect pings are optional). DO NOT use heartbeating. Think of the "topology" (graph) you create among the processes for the failure detection. Think about how the topology is reorganized when you have failures.

Your algorithm must use a small bandwidth (messages per second) needed to meet the above requirements. So, for instance, don't use all to all pinging (it's an overkill, and if you do it, TheTribeHasSpoken Inc. will throw you off the island.).

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 (the "introducer"). When the contact is down, no new members can join the group until the contact has rejoined – but failures should still be detected, and leaves 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 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).

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 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 7 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! We recommend C/C++/Java. 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!). Briefly describe your design (algorithm used, why it scales to large N, and marshaled message format), very briefly how useful MP1 was for debugging MP2, and measurements of (i) the background bandwidth usage for 4 machines (assuming no membership changes), (ii) the expected bandwidth usage whenever a node joins, leaves or fails (3 different numbers), and (iii) plot the false positive rate of your membership service when the message loss rate is 3%, 10%, 30% (you can simulate message losses at the sending end by dropping messages before sending them out to the network). Do this for a group with N=2,4 machines (6 total data points). 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 confidence intervals. Discuss your plots, don't just put them on paper, i.e., discuss trends, 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!)

Submission: There will be a demo of each group's project code. Submit your report (softcopy) as well as working code. Please include a README explaining how to compile and run your code. Submission instructions are similar to previous MPs. Online students must submit full instructions on how to run their code (the TA will execute demo tests on their code, so make the TA's life easier, so they don't give you a low score!).

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). TheTribeHasSpoken Inc. is watching!

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