MP3 – Distributed Group Membership – CS425 T3 (3 hours)

Kevin Peters (klpeter2)
Jack Montag (montag1)

Design

For the distributed group membership system, we decided to stick to the acronym *KISS* (*Keep It Simple, Stupid!*) and design the system with simplicity, yet robustness and efficiency in mind. Distributed systems can get complicated very quickly; thus it's best to keep the stuff working "under the hood" as simple as possible for ease of maintenance and portability. The distributed group membership system consists of $N > 0$ processes, each maintaining a synchronized membership list that stores the state of other processe(s) on the network. The protocol used to detect alive/crashed/left processes is a variant of the "ping-ack" protocol we saw in class. Specifically, all-to-all heart beating is used with the UDP protocol. In the all-to-all heart beating scheme, each process pings every other process on its membership list every T units of time (we used $T = 2$ sec.). When a process receives a ping, it then acks (acknowledges) the pinging process by sending it a message confirming that it is up and running. When an ack message is received, the receiving process updates its membership list by setting the time saved in its membership list for the sending process as the time the message was received. If an ack is not received within a time constraint of 5 seconds since it was last updated, then the process is removed from the membership list and thus the network by every other process. Every message that is sent from process to process is in the form of a char array. Each message contains a tag "P", "A", "J", or" L" for "Ping", "Ack", "Join", and "List" respectively that tells a process how to handle the message upon receipt and a tuple in the form of (ip, timestamp, port). The tuple consists of the sending machines address, time (locally) it sent the message, and its port number that it can be contacted on. Messages "P" and "A" are handled based on the "ping-ack" protocol we discussed above. The "J" message is received whenever a new process wishes to join the network. After a "J" message is received, an "L" message is sent back to the new process containing tuples corresponding with the current membership list. The process receiving an "L" message can then initialize its membership list with the current processes on the network. These messages are parsed with regular expressions from the portable Boost C++ library. These messages will be parsed correctly on both linux and windows machines (provided we've built the program for windows). Unfortunately, the messaging scheme we implemented will fail when passing them from big to little endian processes. We were going to provide this interoperability, but ran into many problems on the receiving end when using the Boost library.

The ping-ack scheme is continuously running on a separate thread while > 1 processes are on the network. In total, $N(N - 1)$ total messages are sent on the network every T units of time. This scales to $O(N*N)$ which is efficient because it is quadratic time (polynomial, not exponential). Of these N processes, there must be a designated *introducer process* alive in order for new processes to join the network. When a new process joins the network, the introducer process alerts all of the other processes to update their membership lists to include the new process. We believe that the all-to-all heart beating

scheme we decided to implement makes our system both robust and reasonably efficient (data being sent is small).

Implementation and Usage

To begin running the distributed group membership system, log into a computer that you would like to be the *introducer process*. In the main.cpp file in the "mp3/src" folder, there is a constant declared at the top of the main method called "INTRODUCER_IP". Its value must be set to the machine that you'd like the *introducer process* to be. The *introducer process* must be running for any new processes to join the network. So for example, if I log into "linux6.ews.illinois.edu", I'd change the "INTRODUCER_IP" constant to "linux6.ews.illinois.edu". Now any other process can connect to the network through this *introducer process*. To begin running the actual program, the executable "mp3" takes 1 argument: the ip address of the current machine you're on. So for example, if I'm on machine "linux6.ews.illinois.edu", to start the exe I would type "./mp3 linux6.ews.illinois.edu". Each process will print out to the console and to a log generated in the working directory, information whenever a new process connects to the membership list, or times-out/fails. To kill a process for testing purposes, you can press "Ctrl+c" to kill the process.

We decided to output generated logs to the working directory so the grader(s) could see them with ease. However, when we were debugging, we saved them to the "~/tmp/" directory and were able to use the distributed log querier (mp2) to access their content. This helped when we were debugging, testing the ping-ack scheme at a much higher frequency (lower time *T*) because we couldn't just eye-ball the results on the console - we had to diff them with the expected results!

Measurements

Ping and ack messages each have a length of 36 bytes. If there are 3 processes running on the network, then 108 bytes will be sent over the network every 2 seconds. This scales O(*N*N).*

When a new process joins the network with 2 other processes already on the network, on average of 143 bytes will be transferred until the membership list on all processes reflect all 3 processes on the network. The membership list sent from the *introducer process* to the new process takes up the most memory in this situation.

When a process fails/leaves the network with 3 processes, 108 bytes will be sent over the network before the process fails. This is because the failed process doesn't send a message back to the other processes saying it has failed, thus saving us another 72 bytes for notifying the 2 other processes. It simply times out.

Compilation

To compile mp3, go into the "mp3/Debug" folder and open a terminal and type "make". This will compile and generate the executable named "mp3".