Lab Notebook

Design Decisions

When implementing the model of a small synchronous distributed system, one of the design decisions we had to make was how exactly to model these virtual machines, in a way that was technically feasible and at the right level of abstraction for this design exercise. At first, we considered adapting code from the gRPC assignment from last exercise, but we agreed that this was probably overkill. Instead, we went to office hours and found Varun's demo extremely helpful. We ultimately decided to represent each machine as its own process as opposed to having all the machines run on the same process because as was described in office hours, the former approach more closely simulates what its like to build distributed systems in the real world where processes will run on physically different machines.

Another design decision was how to navigate being able to always listen to incoming messages that is not limited to clock ticks versus being able to perform an action which is limited to clock ticks. We initially considered grouping this all into one massive function so that they could easily use shared variables like the message queue but we ultimately decided to split this up into 2 methods each with its own thread, one for listening to incoming messages and one performing actions and used a few global variables instead. We also learned that separate processes do not share memory but the threads within a process can share memory of where a global variable is stored and if locks are used appropriately, there won't be complications with using global variables. This was because it was easier for us to implement the code this way and would make our code easier for reviewers and others to understand. Moreover, this decision also made sense because we later realized that the consumer had to be constantly listening for messages to add to the message queue which was not limited to clock ticks whereas the performance of actions was limited to clock ticks.

Another series of key design decisions we had to make was whether to represent something as a concept as an object, a method, or a variable. The most pertinent decision was around the concept of a clock. We realized that each virtual machine had its own clock and quickly realized that a universal clock variable did not accommodate the plurality of clocks. We decided to create a clock object that had to be passed into various methods. The tradeoff here was that it was less efficient in terms of argument size.

Another key design decision was how to store the connections to the other machines so that we could send messages to all the other machines. We initially tried to do this using a dictionary which would keep track of all the connections but we ran into a few bugs with not all of the connections being added appropriately to the dictionary. To address this, we decided to get rid of the dictionary and just pass in the ports of the other machines that need to be communicated with into the function that is sending messages. This made our code much easier to understand and to implement.

Observations

Initial Experiments

We ran the scale model 5 times for one minute each time. Please find the logs for our 5 runs in our repository, in the *logs* folder.

For reference, these were the clock speeds randomly generated in the five runs:

Run #	**Machine 2056**	**Machine 3056**	**Machine 4056**
1	3	2	5
2	6	6	5
3	5	3	4
4	4	5	5
5	2	1	2

Based on our 5 runs above, it is clear that when the machines were running at similar clock speeds, the queues of all the machines were roughly the same length which makes sense since the machines are also sending messages to each other at similar rates. For machines that had more ticks per second, these tended to have smaller queue sizes on average while machines that had less ticks per second tended to have queue sizes that were a bit larger on average. This observation makes since a machine with less ticks per second needs to sleep longer before being able to pop off a message from the queue while a machine with more ticks per second can more quickly pop messages off the queue since it sleeps for a smaller amount of time between each iteration which means that the queue will not become too large. So for instance, in run 1 where there is a difference between the speeds of machine 4056 and the other 2 machines, we see that the queue size for machine 4056 on average was smaller than the queue size for the other machines.

For machines with less ticks per second, these tended to have larger jumps in their logical clocks when at least some of the other machines had more ticks per second and vice-versa is true for machines with the most ticks per second relative to the other machines. This is because machines with less ticks per second will have a lower logical clock value at most times relative to machines with more ticks per second and once a machine with more ticks per second sends a message to a machine with less ticks per second, the slower machine will update its logical clock based on the logical clock value of the faster machine. This is why we see jumps in the logical clock values of slower machines quite frequently and the jumps are sometimes quite large.

Another observation is because the probability of an internal event is high, and since no messages get sent during internal events, the length of the queues did not become too large in general because although adding messages to the queue is not limited by the ticks per second, most of the time, the machines are not sending messages to each other and so the occasional times when messages are sent to each other, each machine likely has enough time to pop messages off the queue and keep the queue somewhat small. As the probability of an internal event increases, we see that the queues for machines that are slower (or less ticks per second) tend to get large more frequently and quicker and even for machines with slightly more ticks per second, their queues get a bit larger but not as much.

Note that increasing the clock cycles by an order of magnitude or decreasing the probability of the event being internal will just make the patterns discussed above more apparent and more dramatic. So for instance, having some machines that are really fast with high ticks per second, like 50 or 100 ticks per second, communicate with machines that are really really slow with few ticks per second, like 1 tick per second, the jump in the logical values for the slow machine will be more frequent and larger and the queue size of the slow machine will get very large.

Further Experiments

Please find the documentation for these extensive experiments fancily presented in the following Observable file:

https://observablehq.com/d/f00be23afec823cd

Thank you for reading!