Design Exercise 2
Engineering Notebook

Adarsh Hiremath and Andrew Palacci

March 8, 2023

I. Introduction

In our design exercise, we implemented a model of a small, asynchronous distributed system. In this system, each of the three simulated machines run at different clock rates assigned randomly at initialization. Each machine listens on a socket for incoming messages from other machines and updates the corresponding logical clock and message queue for each machine. In this engineering notebook, we intend to describe our design decisions and the results from the following experiments:

- Five iterations of the scale model for exactly one minute.
- A single run of the scale model for two minutes.
- Modifying the the probabilities that machines send messages to other machines.

The source code for our chat application can be found here.

II. Model Distributed System Design

Building our scale model for the simulated machines required implementing a protocol for the three machines to communicate. Otherwise, it would be impossible to receive logical clock information, send messages, or update the message queue. Thus, the first question Andrew and I sought to answer in the design process was what protocol we wanted to implement for our scale model.

Initially, Andrew and I were inclined to implement our communication protocol using a simple peer-to-peer (P2P) networking library such as Python's P2P. The idea of P2P networking was appealing since there would be no single point of failure; each machine could act as both a client and server, allowing for a more dynamic and flexible system. Additionally, with P2P, we thought that we could cut the number of connections in half because every connection would be bidirectional.

However, we decided to use web sockets instead for a couple of reasons:

- By using web sockets, we could partially reuse implementations from the first design project instead of learning P2P networking for the first time for this project.
- The problem specification recommended that each machine in the simulation send and receive information using web sockets.
- Completely decentralizing communication is not realistic, especially at scale. It is far easier to accomodate more machines in a distributed system without dramatically revamping the code base when machines can ping a server without anything other than an endpoint.

Instead of using a P2P system, we decided to implement a server and a client on each machine. In our code, the server exclusively listens for messages from other machines and the client sends messages to other machines in the model. Since we implemented the previous design project in Python, we did the same for this design project as well.

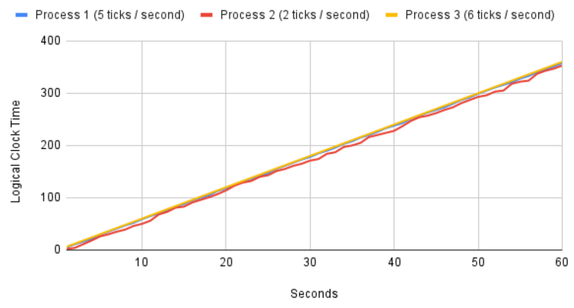
III. Five Scale Model Iterations

We ran five simulations of our model distributed system, each for approximately a minute. During each simulation, we logged the message queue size and logical clock time for each machine after every completed tick of the machine.

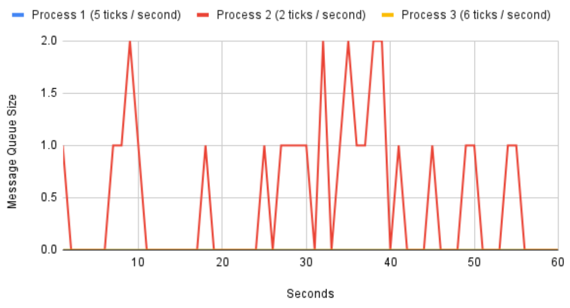
Initially, we logged the message queue size and logical clock time using Python's time module. However, unreliability in the time module and differences across machines prompted us to utilize the tick times from the simulations themselves. We initialized a variable, `currTime`, to 0 at the start of every thread. Then, at every operation, we incremented `currTime` until it hit the randomly generated `tickSize` for each machine. Then, we set `currTime` to 0 once again, repeating the process. Since each machine runs at a clock rate of 0 to 6 ticks per second, we were effectively able to simulate one "real" second on each simulated machine to log any necessary values for producing our graphs.

We generated the following graphs for each trial:

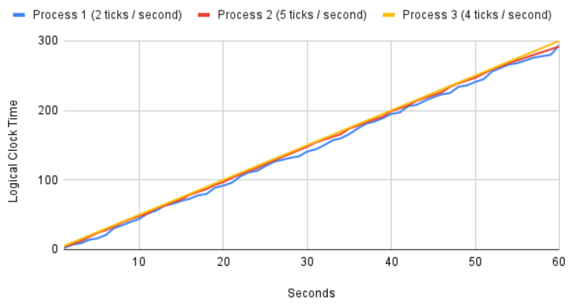
Trial 1: Seconds vs. Logical Clock Time



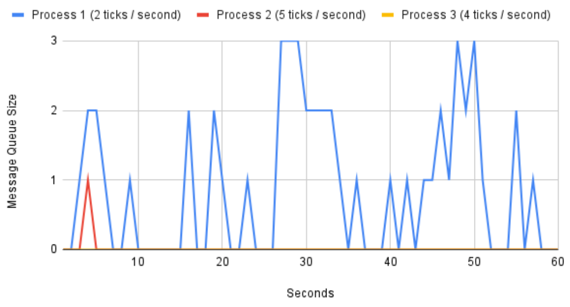
Trial 1: Seconds vs. Message Queue Size



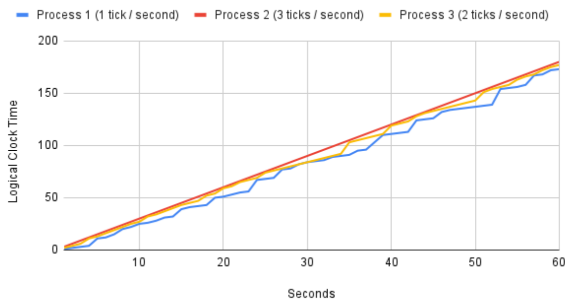
Trial 2: Seconds vs. Logical Clock Time



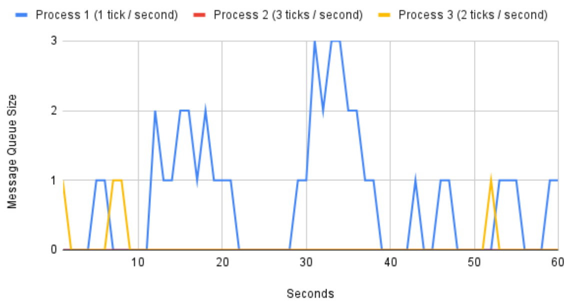
Trial 2: Seconds vs. Message Queue Size



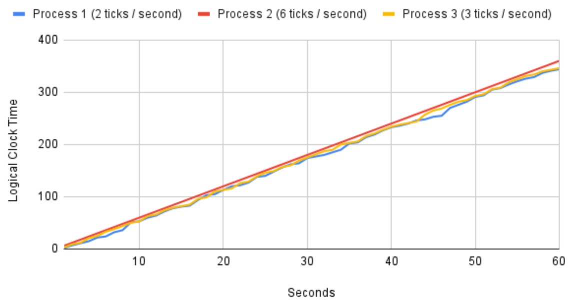
Trial 3: Seconds vs. Logical Clock time



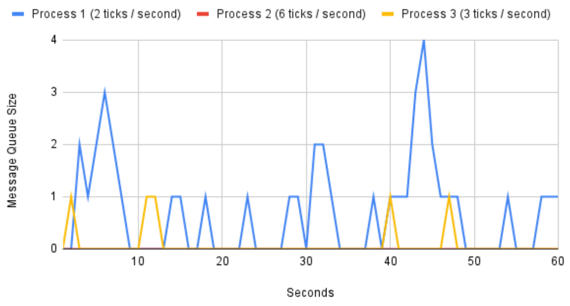
Trial 3: Seconds vs. Message Queue Size



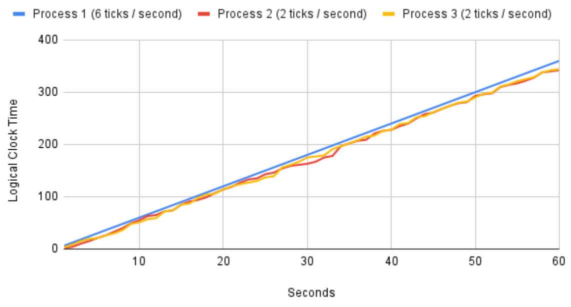
Trial 4: Seconds vs. Logical Clock Time



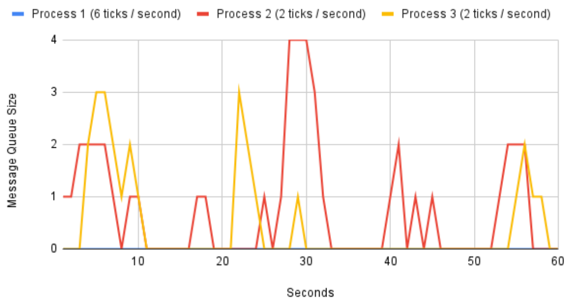
Trial 4: Seconds vs. Message Queue Size



Trial 5: Seconds vs. Logical Clock Time



Trial 5: Seconds vs. Message Queue Size



After experimenting with our scale models and analyzing the above graphs, we noticed the following important trends:

- Processes runnning with less ticks per second had much larger jumps in logical clock time. This is best illustrated on the graph for trial 3 where process 1 ran at 1 tick / second, process 2 ran at 3 ticks / second, and process 3 ran at 2 ticks / second. The line for process 1 is the most jagged, demonstrating larger deviations from the line of best fit. The line for process 3 is the second most jagged, with the line for process 2 following after. The jaggedness in these lines exponentially reduces, demonstrating that faster tick speeds also reduce variations in logical clock time.
- Processes running with less ticks per second also struggled to manage the message queue size. This is illustrated well in the message queue graph for trial 1, where every process except the one running at 2 ticks / second had a queue length of 0 for the entirety of the simulation. This is logical because systems operating at a slower clock speed struggle to process inbound messages of peers running at comparatively higher clock speed that also send more messages.

IV. Longer Scale Model Iterations

Per the assignment recommendation, we ran another iteration of our model for 2 minutes instead of 1 minute to verify the behavior of the logical clock and message queue size.