Chatbot Implementation Notebook

Date/time: 03/01/2025 - > Start Date

Goals: Distributed system where multiple VMs run at different rates, the machines should be able to communicate, efficient logging of messages, observation of what happens.

High-Level Design

Virtual Machine 1 - > Clock rate, Message Queue, Logical Clock
Virtual Machine 2 - > Clock Rate, Logical Clock, Message Queue
Virtual Machine 3 -> Clock Rate, Logical Clock, Message Queue

These virtual machines have different speeds and all have different logical clocks that track event ordering. Communication is done using sockets and incoming messages are stored in a queue.

The message flow would involve, for example, Virtual Machine 1 with logical clock 5 sending a message to Virtual Machine 2 and if the logical clock in Virtual Machine 2 is set at 15, then the update would be done based off of the maximum of the received clock, the clock for Virtual Machine 2 and the internal process.

Design choices

Logical Clock and Event Processing

Based on project requirements, we use Lamport's logical clock algorithm.
Our implementation specifications are:
- Each virtual machine maintains its own logical clock.
- When a message is received, the clock is updated using the formula: max(local_clock, message_clock) + 1. This ensures that events are ordered consistently across machines.
- Three types of events are handled:
    . Internal Events: Simply increment the logical clock.
    . Send Events: Increment the clock, then send a message that includes the new clock value.
    . Receive Events: Update the clock based on the received message's clock value.

Logical Clock and Event Processing

We decided to use Python for our code base, and TCP prototocol through sockets for the communication.
- TCP server thread is started for each virtual machine, which listens on a port which we define in the config.
- Server accepts incoming connections and spawns a new thread to process each connection.
- Incoming data is processed line by line, and any line starting with "clock:" is treated as a message with a logical clock value.
example transmission:
2025-03-05 12:46:04 | RECEIVE | Clock: 5 | Message clock: 4, Queue length: 0

- Client side: machine attempts to connect to each peer listed in config.
- If connection attempt fails, the socket is closed to prevent resource leaks and the connection is retried.

Logging and Resource Management:
- Each virtual machine writes event logs to its own file (log_1.txt for machine 1). Each log entry includes the system time, event type, current logical clock, and event details.
- A shutdown() method is provided to ensure that both the server socket and any peer sockets are closed when the machine stops running, thereby preventing resource leaks.

Script for Running a Virtual Machine

We decided to encapsulate the script for running a single machine separately for easier maintenance.
Command-Line Interface:
- The script reads a machine ID from the command line. If no machine ID is provided, it displays a usage message and exits.
- Once a valid machine ID is supplied, a VirtualMachine instance is created with that ID.

- We also included a script to run all of the machines to prevent race conditions.

Implementation Process

Based off of the design discussions above, the system was implemented. We create a VM instance in machine.py with a logical clock, network connection and queue. We also send a message that contains its logical clock value and receive a message and update its logical clock value and log all the messages afterwards. We run a loop in each iteration where it is decided if the messages should be processed from the queue, should the message be sent to another machine and do we need internal clock updates.

How did we run the experiments?
First approach considered was manually running three terminals but that turned out to be too tedious. Starting them all instantly might also result in expected findings. Hence we introduced a tiny delay to ensure that each machine is running in sequence. The start_all.py file introduces that delay and start all the VMs instantly.

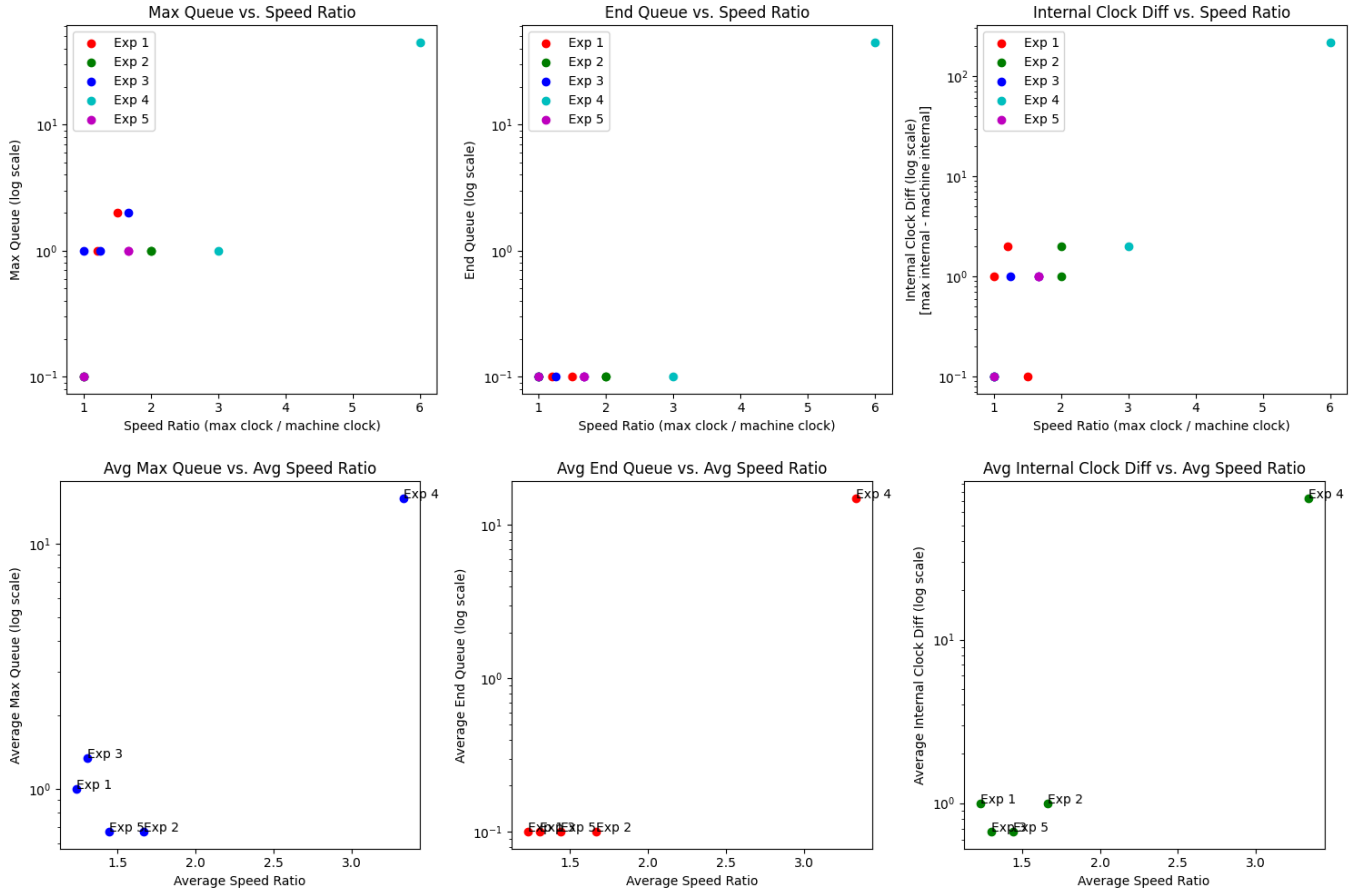
Experiment Findings

Data Excerpts

Experiment	Machine	Clock Rate	Max Queue	End Queue	Internal Clock Diff
1	Machine 1	4	2	0	0
1	Machine 2	5	1	0	2
1	Machine 3	6	0	0	1
2	Machine 1	1	1	0	2
2	Machine 2	2	0	0	0
2	Machine 3	1	1	0	1
3	Machine 1	5	1	0	0
3	Machine 2	4	1	0	1
3	Machine3	3	2	0	1
4	Machine1	6	0	0	0
4	Machine2	1	45	45	218
4	Machine3	2	1	0	2
5	Machine1	3	1	0	1
5	Machine2	3	1	0	1
5	Machine3	5	0	0	0

Visualization of Experiment Results

We decided to explore the data through visualization. For x axis, we decided to display speed ratio of each machine's clock which we defined as (max clock / machine clock). We then compared the clocks based on max queue, end queue value to compare experiment outcomes, and internal clock differences at the end of each test interval.



Findings

1. **Clock Rate Disparities:**
    - Experiments with more balanced clock rates (e.g., Experiments 1, 3, and 5) have lower ratio differences—i.e., the machine with the highest clock rate is only marginally faster than the others.
    - Experiments with very low clock rates on one machine (e.g., Experiment 4, where one machine runs at 1 compared to a maximum of 6) show a high ratio difference (~0.83).

2. **Maximum Queue Length:**

- Machines with higher ratio differences tend to experience higher maximum queue lengths. For example, in Experiment 4 the machine with a ratio difference of ~0.83 had a max queue of 45.
- In experiments with balanced clock rates, maximum queue values are relatively low (typically 0–2).

3. **End Queue:**
   - In most experiments, the end queue (i.e. the queue size at the end of the run) is 0.
   - Experiment 4 is an exception where one machine maintained a high end queue (45), indicating persistent backlog when clock rate differences are extreme.

4. **Internal Clock Differences:**
   - When clock rates are similar, the differences in the final (internal) clock values are minimal.
   - Significant disparities in clock rates (as in Experiment 4) result in large differences in the final internal clocks (e.g., a difference of 218 between the fastest and slowest machines).

5. **Finding of Note:**
   - In experiment 4, even though machine 3 had a clock rate of 2, which is significantly slower than machine 1's rate of 6, the queue it had was minimal, and the queue load was on machine 3 only.

Special Cases

Smaller probability in internal operation

- Clock rates: 3, 4, 6

After changing the rate of the internal operations to 0.2, more send operations occurred. This resulted in any differences in clock speeds to have significantly greater effect on queues and internal clock differences. In this scenario, machine with clock rate 3 queue length reached 48. Additionally, the machine with que rate of 4 had often queue length of 1, sometimes 2 which happened with lesser frequency in a similar clock setup with the internal probability being low.

Small Variation in Clock Cycle

- Clock rates: 5,6,6

To observe behavior of the distributed system where the differences in clock cycle rates are not as pronounced, we manually selected rates of 5,5, and 6 for our virtual machines. The results shown that there is no significant difference in behavior between the machines with our chosen magnitude of difference. There was no message congestion as most of the queue lengths were 0 with occasional 1 message waiting to be read across all machines. For machine with the slowest clock speed of 5, there was one occasion however, where the queue was maintained at 1 for four cycles, which shows that there can be edge cases, depending on probability, where there can be a small congestion event. In some situations, this can potentially be exacerbated further.

Additional Custom Implementations

- Fractional clocks
    - Clock rates: 3.48, 3.78, 3.25

We tried narrowing the difference between machines to see if any emergent behavior takes place and if some queue still exists. We found that internal clocks for each machine were within margin of error, so they appeared identical. Maximum queue length across all the virtual machines was 1, which occurred rarely. This makes sense as the presence of internal operations gives time to each machine for processing of queue. Each machine's internal time is then updated through messages.

- Network Delays
    - Implemented with: delay = random.uniform(0.1, 0.5); time.sleep(delay)

We also tried simulating network delays by unform distribution between 0.1 and 0.5 seconds. With moderate clock differences (3,4,6), no worsening of behavior was observed. That is, the worst-case scenario was momentary queue of 1-2 members. This was slightly surprising to us as we expected performance to get work, especially since delay of up to 0.5 seconds is possible, which delays the performance of all of the operations.

Take-aways

1. Even differences in clock rates that seem large initially do not necessarily lead into long queues across 1–2-minute run times.
2. Large absolute differences create the longest queues in the slowest machines.
3. If a next slowest machine is at least 2x faster than the slowest, queue is not necessarily going to be long for the second slowest machine, even though the fastest machine has 3x rate of the second slowest.
4. Network delays (uniform, relatively minor) do not worsen clock difference effect in our experiment

Gen AI Usage Statement:

Gen AI powered copilot was used in the implementation process for debugging, coding help and asking questions.