Report HW3: Loggy - a logical time logger

Artem Sliusarenko

September 27, 2023

1 Introduction

This assignment covered the topic of logical clocks; a mechanism of capturing the order of events between processes.

2 Main problems and solutions

Understanding how to identify wether a message is safe to print or not was challenging. After some consideration I decided to rely on the latest logical time of all processes. However, that might prove to be a bad decision, because if one worker received significantly more messages than any other worker, some of its logs might be left in the holdback queue.

3 Evaluation

After running some tests as described in part 3 of the assignment it is clear that messages are printed in the wrong order. For example, test run with Sleep=100 and Jitter=50 had log 'na george received,hello,68' printed way before log 'na george sending,hello,68' was printed. It is clear that the happened-before relation, 'na george sending,hello,68' -¿ 'na george received,hello,68', is not true in this case. This was addressed and explained further in section 4 below.

1. Lamport time (Section 4)

   Section 4 of of assignment requires us to introduce logical time to the worker module. A new module called 'time' will handle Lamport clocks. To keep it simple Lamport clocks timestamp will be represented as an integer values (0, 1, 2, ..., N).

   To answer questions from part 4:

   1. How do you identify messages that are in the wrong order? Messages printed in the wrong order can be identified by looking at the log.

If 'sending X' log appears after 'receiving X' we know that messages are in the wrong order.

2. What is always true, and what is sometimes true? Timestamp of each message sent or received by a worker A is unique.

3. How do you play it safe? To play it safe we can se tthe Jitter to 0. This way all messages seems to be logged in proper order.

2. Observations

   The 'time' module is exposing the API to manage the logical clocks mechanism. API exposed by the 'time' module allows us to capture the happened-before relationshop using few simple rules:

   1. Increment logical time before each event in the process.

   2 a. When process sends a message logical time is sent with it.

   2 b. When process receives a message we need to merge received and local logical time and take the max and then repeat step 1.

   Entries out of order in the first implementation of Loggy were detectable by looking at the log message. For example, if 'received,hello,68' is printed before 'sending,hello,68'. Final log tells us that we can clearly track the journey of any given message through the network of processes. That information would be very useful to improve the eobservability of the distributed application. Though we can trust the ordering of the events in the log it doesn't mean that all events appeared in that exact order displayed. We have 4 workers that could be sending/receiving messages simulteneously, however we have only one logger, therefore logs are printed as fast as logger can process it. Also, logs could be printed in a slightly wrong order, according to the timestamp, unless we sort the holdbackqueue. Given that our test script runs only for 5000 miliseconds we can find the maximum size of the holdbackqueue by runninga test with Sleep = 1 and Jitter = 0. Maximum holbackqueue size observed was 12.

4 Conclusions

Lamport time allows us to order the communication between processes by applying the concept of logical time. Logical clocks unlike physical clocks is not defined by the outside factors. We are in control of the logical clocks representation. Our implementation of the logical clocks is represented by the API exposed by the 'time' module. Logical clocks can benefit the observability of the distributed system.