Daniel Kudrow
Oct. 24, 2014

CS 271 Homework #1 Report

Assumptions

- Because I could not determine my machine's clock drift, I assumed it was 15ms per min as per the lab instructions. Thus, resynchronizing every 2s was sufficient to maintain drift < 1 ms/min. Because the RTT was large (~200ms) and there were 5 timeservers, the resync rate could not be much higher than this anyway.
- I assume all timeservers are up (i.e. it will hang forever waiting for a response).
- It appears that simple python statements (assigment, arithmetic, &c.) seem to take on the order of $10_6 s$ to complete. When performing calculations between clock readings, I assume no time has passed.

Making Timeserver Requests

The main challenge in implementing time synchronization is the inability to obtain concurrent timestamps from remote timeservers. One approach would have been to create a thread pool and attempt to make timeserver requests in parallel. However, thread scheduling is not transparent and on a shared resource like CSIL there is no way of reliably binding threads to separate cores. Even then, an implementation that requires one core per timeserver will not scale and is useless. My approach was to include a local timestamp with each server request. Thus each response from the server contains

1. the remote server's timestamp
2. the roundtrip time (calculated with the local clock)
3. local time of request

An approximation of the current time on a remote server can be calculated by

$$T_{server} = T_{local} - \left(T_{request} + \tfrac{T_{rtt}}{2}\right) + T_{timestamp}$$

This calculation is clearly subject to the local clock drift however this calculation occurs very shortly ($< 1s$) after the timeserver request so the actual error is acceptable (i.e. negligible compared to network latency, &c.)

Clock Synchronization

I implemented two methods of clock synchronization, both of which take a list of timeserver responses as input. The first, called *syncMinRTT*, selects the closest server (lowest RTT) and uses the timestamp from that server to reset the clock using

$$T_{new} = T_{server} + \tfrac{T_{rtt}}{2}$$

The second is *syncMarzullo* which implements Marzullo's algorithm. A timeserver's confidence interval is defined as the approximate current time on the server (as described above) plus or minus half of the RTT. This allows for any amount of network asymmetry between the local clock and the timeserver. In the event that there is no overlap among the time serve responses, the timeserver with the slowest clock is used to perform the resynchronization.