Clock Synchronization

Implications of having no global time: `make` command checks for a source file's last modified times to assess (last modified time is higher than creation time or modified time for the output file) if it needs to recompiled. Otherwise if a program consists of `100` files, every file would have to recompiled.

Physical Clocks

Nearly all computers hav circuitary for keeping track of time, but these are not actually clocks in the usual sense. **Timer** is a better word. A computer timer is actually a precisely machined quartz crystal. Quartz crystals oscillate at a well defined frequency.

Associated with each crystal are 2 register, **counter** and a **holding register**. Each oscillation of the crystal decreases the counter by `1`. When counter = `0`, an interrupt is generated and counter is reset. `60` oscillations = 1 second and 1 interrupt called **clock tick**.

Universal coordinated time = UTC

Clock sync Algorithms

If one machine has a UTC receiver, then the goal is to keeping all the other machines synced to it. If no UTC receiver, each machine tracks its own time.

Goal of clock syncs algo is to keep deviation between time within a certain bound called **precision** (π)

$$\mid C_p(t) - C_q(t) \mid \leq \pi$$

When talking about an external source, we use **accuracy** (α)

$$\mid C_p(t) - C_q(t) \mid \leq \alpha$$

Keeping clocks precise is called **internal synchronization** and accurate is called **external synchronization**.

Because hardware clock's frequency is not perfect (subject temperature, etc) it can drift which is called **clock drift**.

$$\forall t : (1-\rho) \leq \frac{F(t)}{F} \leq (1+\rho)$$

ρ = max drift rate
$F(t)$ = frequency at t

F = ideal frequency

Network Time Protocol

Common approach is to let clients contact a time server equipped with a UTC receiver and an accurate clock. But message delays can outdate the reported time. The trick is to find a good estimation of these delays.

In this case, A will send a request to B, timestamped with value T_1. B, in turn, will record the time of receipt T_2 (taken from its own local clock), and returns a response timestamped with value T_3, and piggybacking the previously recorded value T_2. Finally, A records the time of the response's arrival, T_4. Let us assume that the propagation delays from A to B is roughly the same as B to A, meaning that $\delta T_{req} = T_2 - T_1 \approx T_4 - T_3 = \delta T_{res}$. In that case, A can estimate its offset relative to B as

$$\theta = T_3 + \frac{(T_2 - T_1) + (T_4 - T_3)}{2} - T_4 = \frac{(T_2 - T_1) + (T_3 - T_4)}{2}$$

In **NTP**, this is set up pairwise between the servers.

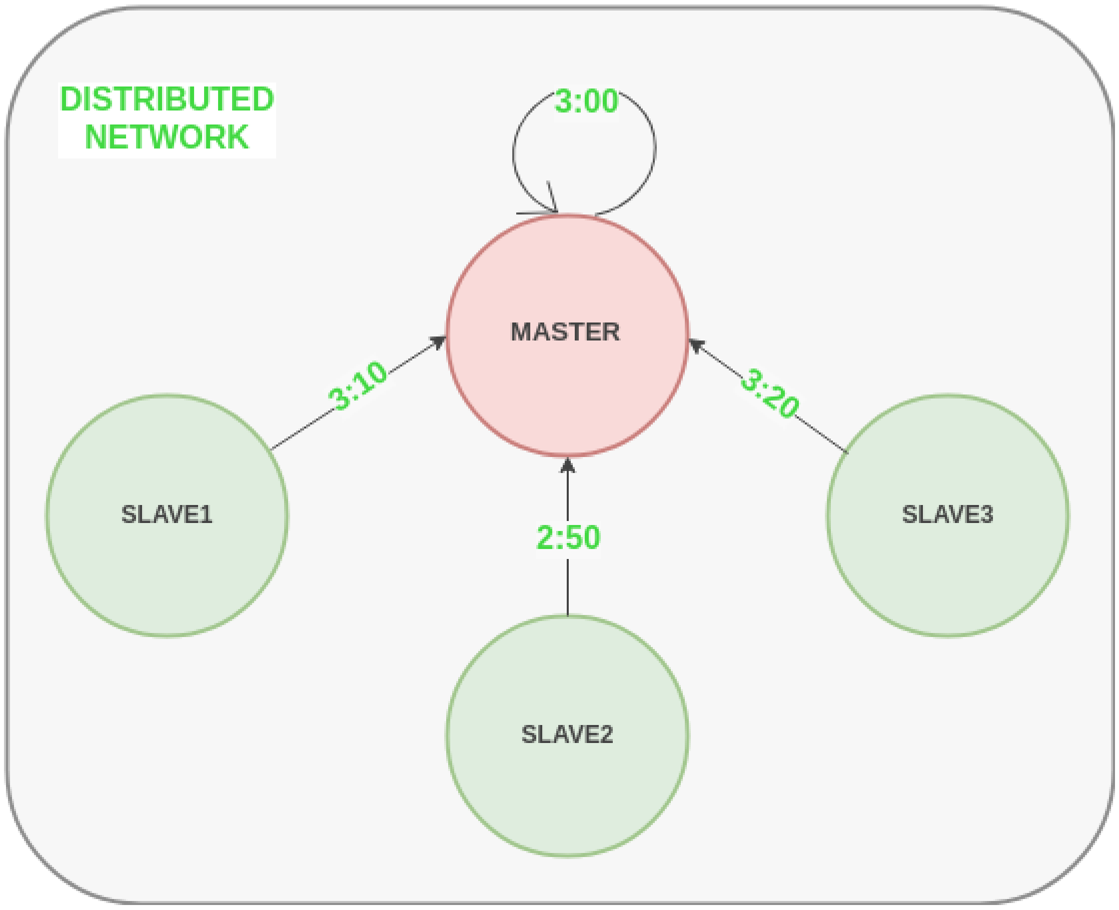
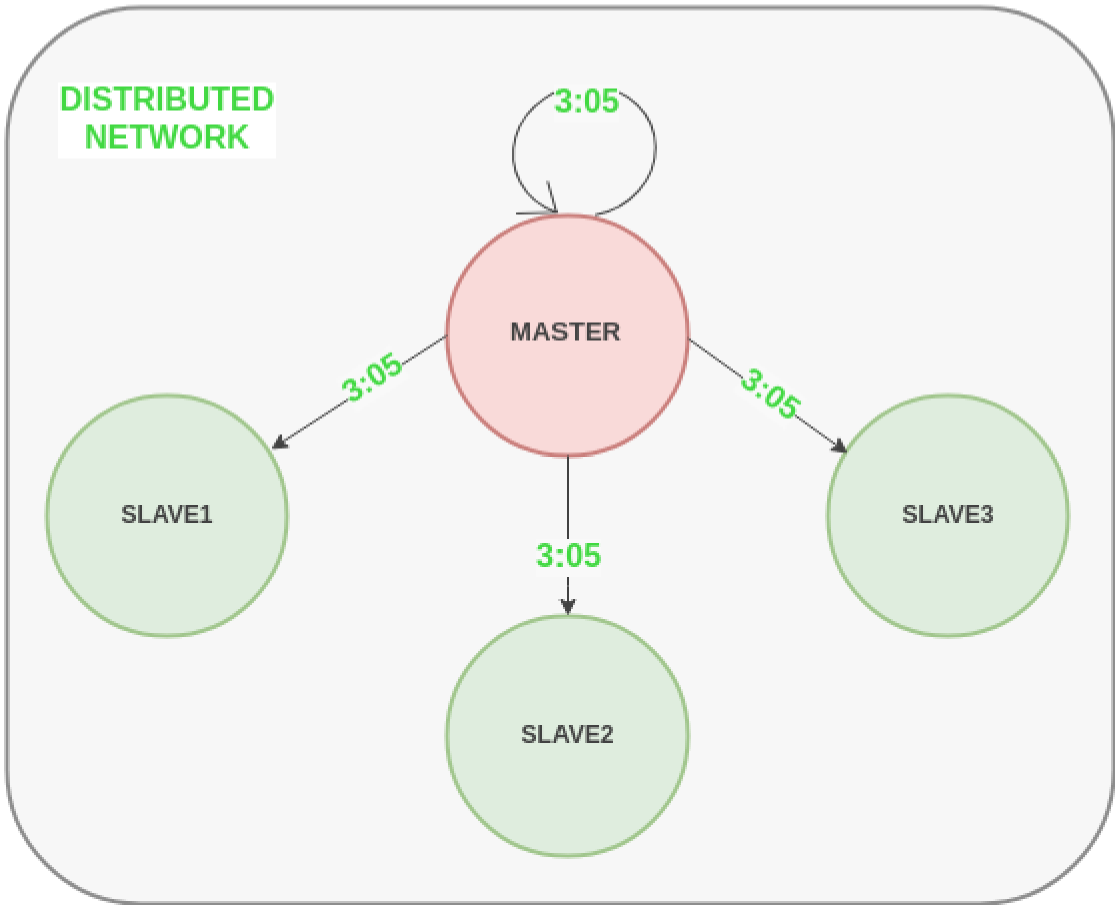
$$\delta = \frac{(T_4 - T_1) - (T_3 - T_2)}{2}$$

8 pairs of (δ, θ) are calculated taking $\min(\delta)$ for best estimation.

Berkeley's Algo

Berkeley's Algorithm is a clock synchronization technique used in distributed systems. The algorithm assumes that each machine node in the network either doesn't have an accurate time source or doesn't possess a UTC server.
Algorithm:

1. An individual node is chosen as the **master node** from a pool node in the network. This node is the main node in the network which acts as a master and the rest of the nodes act as **slaves**. The master node is chosen using an **election process/leader election algorithm**.
2. Master node periodically pings slaves nodes and fetches clock time at them using Cristian's algorithm.
3. Master node calculates the **average time difference between all the clock times** received and the clock time given by the master's system clock itself. This average time difference is added to the current time at the master's system clock and broadcasted over the network.


DISTRIBUTED NETWORK
3:00
MASTER
3:10
3:20
2:50
SLAVE1
SLAVE3
SLAVE2

DISTRIBUTED NETWORK
3:05
MASTER
3:05
3:05
3:05
SLAVE1
SLAVE3
SLAVE2