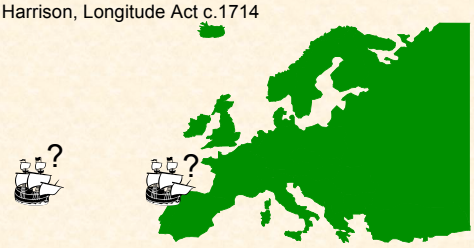
Time and Coordination

Distributed Systems 1

Measuring Longitude

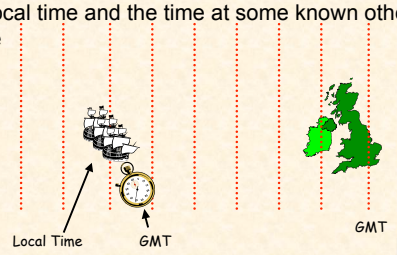
- Harrison, Longitude Act c.1714



- Synchronised clocks used to solve problem of longitude

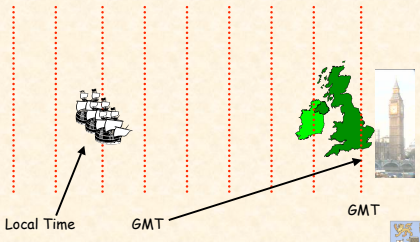
Measuring Longitude

- Position can be calculated provided you know the local time and the time at some known other place



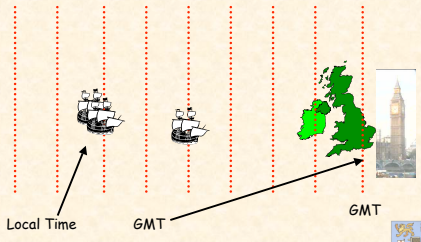
Exercise

- What if there was no portable GMT clock?



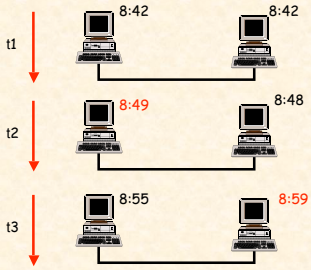
Exercise

- What if there was no portable GMT clock?



Synchronising Computers

-

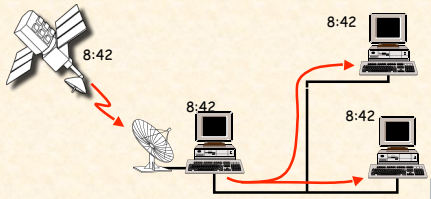


Exercise 1

- How can we agree the time?
  - Clocks "drift"
  - Spend 5 minutes thinking about how you can synchronise your watch to mine
  - We'll spend some time trying to find some solutions, and working out what issues there are to be solved

Time and Co-ordination

- Clock Synchronisation
  - External Synchronisation – refer to external time source



Time and Co-ordination

- Internal Synchronisation – refer to each other

Time and Co-ordination

- Internally Synchronised systems are not necessarily externally synchronised

Synchronising Computers

- Clock Drift
  - Crystal based clocks used in computers are subject to clock drift
  - Count time at different rates
  - Frequency of oscillations varies (e.g. Temperature change)
  - quartz crystal drift: 10^{-6}
    - 1 second in 11.6 days
  - Clocks diverge over time

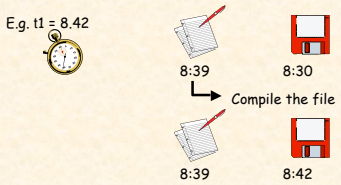
Compensating for clock drift

- If local system time < correct time, the local time can be corrected…
- $$Time_{Local} = Time_{Local} + difference$$
- A few clock ticks will be missed
  12,13,14,20,21,22,…

If local system time > correct time, the local time cannot be corrected immediately because time must always advance!

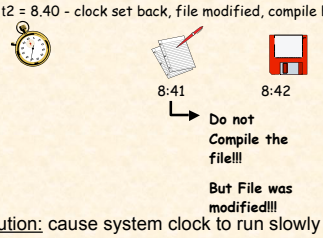
Consider Unix 'make'

- Recompiles only source files that have been modified since the last compilation.
- Compares compiled object file date to source file date.



Consider Unix 'make'

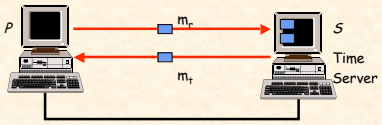
- E.g. t2 = 8.40 - clock set back, file modified, compile begins



Solution: cause system clock to run slowly for a period until it is synchronized with last known correct time

Cristian's method

- Cristian's method for synchronising clocks
  - Central time server process (S) supplies the time according to its own clock upon request.



Cristian's method

- All clients of server S can be synchronised to time

  $T_s \pm$ error

- Note - T_s need not be the correct time.
  - T_s may drift with respect to real time, or it may be synchronised with GPS etc.
- *P* requests time message m_r, and receives time *t* in message m_t.
- Should set clock equal to…

  $t + T_{trans}$

  ...where T_{trans} is time taken to transmit m_t from *S* to *P*
- However, T_{trans} is subject to variation

  i.e. $T_{trans} = t_{min} + t_x$, where $t_x \geq 0$

Cristian's method

- Also, can't measure T_{trans} directly anyway!
- Solution
  - Process *P*, wishing to learn the time from *S*, records the total round-trip time T_{round}.

    $T_{round} = t_2 - t_1$

    t_1 = time m_r is sent
    t_2 = time m_t arrives
  - T_{round} can be measured accurately
    - Round trip time 1- 10 milliseconds on LAN
    - Local clock drift 10^{-5} milliseconds

Cristian's method

- On arrival of m_t at *P*, time *t* in m_t, clock should be set to

  $t + T_{round}/2$

- *Time* $= t + T_{round}/2$ assumes that elapsed time is split equally before and after *S* placed *t* in m_t.
- How accurate is this?

Cristian's method

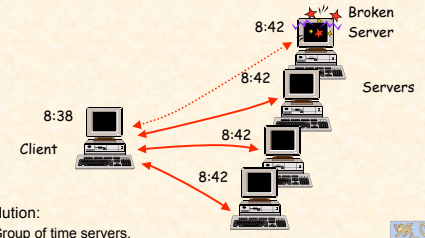
- Determining accuracy of $t + T_{round}/2$
  - Assume t_{min} is known and represents the minimum possible time for message propagation.
  - Earliest point *S* could have placed *t* in m_t was t_{min} after *P* dispatched m_r,
  - Latest point *S* could have placed *t* in m_t was t_{min} before m_t arrived at *P*.
  - By *S*'s clock, the time range for which m_t could arrive at *P* is
    $[t + t_{min}, t + T_{round} - t_{min}]$
  - Width of this range is
    $T_{round} - 2t_{min}$

Cristian's method

- Accuracy of *t* is
  $\pm (T_{round}/2 - t_{min})$
- *P* should set clock to….
  $t + T_{round}/2 \pm (T_{round}/2 - t_{min})$
- To improve accuracy, reduce T_{round}
  - make several requests and take minimum value of T_{round}
- The greater the accuracy required, the smaller the probability of achieving it.

Problems with Cristian's algorithm

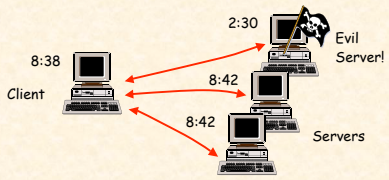
- Single time server is bottleneck
  - If server fails, then cannot synchronise clocks.



- A solution:
  - Group of time servers.
  - Client multicasts requests.

Problems with Cristian's algorithm

- Malfunctioning or impostor time server could wreak havoc!

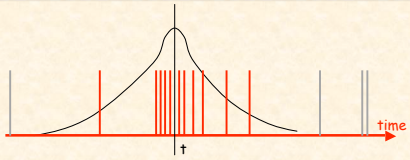


- Solution: Algorithm to detect bad server (E.g. if $t_{bad} < 1/3\ t_{servers}$).
- Solution: Authentication
  - but some security techniques depend on accurate time.

The Berkley Algorithm

- Internal synchronisation algorithm based on Master/Slave model
- Master periodically polls slaves.
- Slaves send back their clock values.
- Master estimates each slaves local clock time by observing round trip times.
  - Similar to Cristian's algorithm.
- Master collects and derives new time *t from* averages values obtained.
  - Includes its own clock.
  - Sets internal clock time to this average.

The Berkley Algorithm



- *t*, on the balance of probabilities, will cancel out the clock drift of each individual clock.
- Unless round trip time of some measurements are very large
  - remove deviant timings
- Fault tolerant average

The Berkley Algorithm

- Master sends back to each slave the amount by which its clock is out of sync with *t*.
  - May be positive or negative value.
- Slaves update their clocks.
  - If slow, update immediately.
  - If fast, slow clock down.
- The clocks of 15 computers typically drift no more than 20 – 25 milliseconds apart.
- System is fault tolerant.
  - If master fails, another can be elected.

Exercise

- Groups of 10
- I am an external time source
- Synchronise to my time
- Take 10-15 minutes to build an approach
- Implement it (i.e. synchronise to my time)