TIME SYNCHRONIZATION, CRISTIAN’S ALGORITHM, BERKELEY ALGORITHM, NTP

George Porter
Feb 10, 2019

ATTRIBUTION
• These slides are released under an Attribution-NonCommercial-ShareAlike 3.0 Unported (CC BY-NC-SA 3.0) Creative Commons license
• These slides incorporate material from:
  • Kyle Jamieson, Princeton University (also under a CC BY-NC-SA 3.0 Creative Commons license)
Outline

1. Time synchronization
   - Cristian's algorithm
   - Berkeley algorithm
   - NTP

A DISTRIBUTED EDIT-COMPILE WORKFLOW

- 2143 < 2144 \(\Rightarrow\) make doesn't call compiler

Lack of time synchronization result – a possible object file mismatch
WHAT MAKES TIME SYNCHRONIZATION HARD?

1. Quartz oscillator sensitive to temperature, age, vibration, radiation
   - Accuracy one part per million (one second of clock drift over 12 days)

2. The internet is:
   - Asynchronous: arbitrary message delays
   - Best-effort: messages don’t always arrive

JUST USE COORDINATED UNIVERSAL TIME?

- UTC is broadcast from radio stations on land and satellite (e.g., the Global Positioning System)
  - Computers with receivers can synchronize their clocks with these timing signals

- Signals from land-based stations are accurate to about 0.1–10 milliseconds

- Signals from GPS are accurate to about one microsecond
  - Why can’t we put GPS receivers on all our computers?
Outline

1. Time synchronization
   - Cristian's algorithm
   - Berkeley algorithm
   - NTP

SYNCHRONIZATION TO A TIME SERVER

- Suppose a server with an accurate clock (e.g., GPS-disciplined crystal oscillator)
- Could simply issue an RPC to obtain the time:

  ![Time Synchronization Diagram]

  - But this doesn’t account for network latency
  - **Message delays** will have **outdated** server’s answer
CRISTIAN’S ALGORITHM: OUTLINE

1. Client sends a **request** packet, timestamped with its local clock $T_1$
2. Server timestamps its receipt of the request $T_2$ with its local clock
3. Server sends a **response** packet with its local clock $T_3$ and $T_2$
4. Client locally timestamps its receipt of the server’s response $T_4$

How the client can use these timestamps to synchronize its local clock to the server’s local clock?

CRISTIAN’S ALGORITHM: OFFSET SAMPLE CALCULATION

Goal: Client sets clock $\leftarrow T_3 + \delta_{\text{resp}}$

- Client samples **round trip time**
  \[ \delta = \delta_{\text{req}} + \delta_{\text{resp}} = (T_4 - T_1) - (T_3 - T_2) \]
- But client knows $\delta$, not $\delta_{\text{resp}}$

Assume: $\delta_{\text{req}} \approx \delta_{\text{resp}}$

Client sets clock $\leftarrow T_3 + \frac{1}{2} \delta$
Outline

1. Time synchronization
   - Cristian’s algorithm
   - Berkeley algorithm
   - NTP

BERKELEY ALGORITHM

- A single time server can fail, blocking timekeeping

- The Berkeley algorithm is a distributed algorithm for timekeeping
  - Assumes all machines have equally-accurate local clocks
  - Obtains average from participating computers and synchronizes clocks to that average
**BERKELEY ALGORITHM**

- **Master machine**: polls $L$ other machines using Cristian’s algorithm $\rightarrow \{ \theta_i \} (i = 1...L)$

---

**Outline**

1. Time synchronization
   - Cristian’s algorithm
   - Berkeley algorithm
   - NTP
THE NETWORK TIME PROTOCOL (NTP)

• Enables clients to be accurately synchronized to UTC despite message delays

• Provides reliable service
  • Survives lengthy losses of connectivity
  • Communicates over redundant network paths

• Provides an accurate service
  • Unlike the Berkeley algorithm, leverages heterogeneous accuracy in clocks

NTP: SYSTEM STRUCTURE

• Servers and time sources are arranged in layers (strata)
  • Stratum 0: High-precision time sources themselves
    • *e.g.*, atomic clocks, shortwave radio time receivers
  • Stratum 1: NTP servers directly connected to Stratum 0
  • Stratum 2: NTP servers that synchronize with Stratum 1
    • Stratum 2 servers are clients of Stratum 1 servers
  • Stratum 3: NTP servers that synchronize with Stratum 2
    • Stratum 3 servers are clients of Stratum 2 servers

• Users’ computers synchronize with Stratum 3 servers
NTP OPERATION: SERVER SELECTION

• Messages between an NTP client and server are exchanged in pairs: request and response
  • Use Cristian’s algorithm
  • For $i^{th}$ message exchange with a particular server, calculate:
    1. **Clock offset** $\theta_i$ from client to server
    2. **Round trip time** $\delta_i$ between client and server
  • Over last eight exchanges with server $k$, the client computes its **dispersion** $\sigma_k = \max_i \delta_i - \min_i \delta_i$
  • Client uses the server with **minimum dispersion**
  • Outliers are discarded

NTP OPERATION: CLOCK OFFSET CALCULATION

• Client tracks minimum round trip time and associated offset over the last eight message exchanges ($\delta_0, \theta_0$)
  • $\theta_0$ is the best estimate of offset: client adjusts its clock by $\theta_0$ to **synchronize to server**

![Offset vs Round trip time](image.png)
NTP OPERATION: HOW TO CHANGE TIME

• Can’t just change time: Don’t want time to run backwards
  • Recall the make example

• Instead, change the update rate for the clock
  • Changes time in a more gradual fashion
  • Prevents inconsistent local timestamps

CLOCK SYNCHRONIZATION: TAKE-AWAY POINTS

• Clocks on different systems will always behave differently
  • Disagreement between machines can result in undesirable behavior
• NTP, Berkeley clock synchronization
  • Rely on timestamps to estimate network delays
  • 100s $\mu$s–ms accuracy
  • Clocks never exactly synchronized
• Often inadequate for distributed systems
  • Often need to reason about the order of events
  • Might need precision on the order of ns

Images used with permission
Outline

1. Time synchronization
   - Cristian's algorithm
   - Berkeley algorithm
   - NTP