End-to-End Arguments in System Design

J. H. Saltzer, D. P. Reed, and D. D. Clark

E₂EA

• A set of best practices when designing a system

• At any given level in a system, only implement functionality which can be effectively utilized by all higher levels in the system
Careful File Transfer (Computer A)

1. App requests file
   - Disk returns Requested data

2. App asks OS or communications program to send data
   - Data is moved from CPU buffers to comm device buffers

3. Data is sent across network to computer B
Careful File Transfer (Computer B)

1. Received communications data buffers sent to CPU
2. Data is sent across network to computer B
3. App writes data to disk
4. OS or dedicated communications program delivers data to app
5. Data is sent across network to computer B
Where can things go wrong?
Where can things go wrong?

- An entire system could crash during the transfer.
- Transient errors in the CPU or RAM subsystems could cause buffers to be corrupted.
- The network subsystem could drop packets or flip bits.
- Hardware faults cause data to be read or written incorrectly.
- Incorrect logic or other flaws in the OS or file transfer software can corrupt data.
Check at the endpoints

The application endpoint is best suited to verify the data. It knows how the data is used, and how to check that the operation succeeded.
Verify data in low level systems

Inspect each packet as it crosses the network
Verifying each packet

- Encapsulate a 20KB file transfer using the XMODEM protocol and transfer at 9,600 bps

**XMODEM Packet Structure:**

<table>
<thead>
<tr>
<th>SOH</th>
<th>Frame #</th>
<th>Frame #</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>...</th>
<th>Byte 128</th>
<th>CRC</th>
<th>CRC</th>
</tr>
</thead>
</table>

Determine total size of data including container:

\[
\begin{align*}
20 \text{ KB} \times \frac{\text{frame}}{128 \text{ B}} &= 160 \text{ frames} \\
160 \text{ frames} \times \frac{5 \text{ B}}{\text{frame}} &= 800 \text{ B} \\
20 \text{ KB} + 800 \text{ B} &= 20.78125 \text{ KB}
\end{align*}
\]

**XMODEM transfer time**

\[
\begin{align*}
20.78125 \text{ KB} \times \frac{8 \text{ b}}{\text{B}} \times \frac{1 \text{ s}}{9,600 \text{ b}} &= 17.738 \text{ s}
\end{align*}
\]

**Raw data transfer time**

\[
\begin{align*}
20 \text{ KB} \times \frac{8 \text{ b}}{\text{B}} \times \frac{1 \text{ s}}{9,600 \text{ b}} &= 17.056 \text{ s}
\end{align*}
\]

**XMODEM Overhead**

\[
\frac{17.738 \text{ s}}{17.06 \text{ s}} - 1 = 0.0390625 \approx 4\%
\]

The overhead imposed by checking each packet seems modest...
Other errors can defeat packet inspections

- Data is corrupted before it gets to the network
- Disk errors cause bits to flip during a write

Inspect each packet as it crosses the network
Checking packets at each hop
Let the application decide what error checking and recovery is needed.

Error checking and recovery is not preferred in every situation.
What do you think?
Multicast

• Unicast requires a dedicated message for each client
• Can be bandwidth intensive, since identical content may be sent across the network to different users

Multicast allows users to register to receive messages from a particular source
• Allows the sender to send one message, which is duplicated at a node with multiple interested clients attached

This task cannot be performed in the ends, but clients are not forced to use multicast

http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=997043
Sorting Libraries

void qsort ( void * base,
           size_t num,
           size_t size,
           int ( * comparator ) ( const void *, const void * ) );

• The endpoint is an application which requires sorted data sets
• The sort function doesn’t have enough information about how to compare items
  – It’s a partial implementation
• The calling application provides the rest of the implementation using a comparator

The sorting library does not restrict callers to use a particular value system, yet provides a library implementation of quicksort
Questions?
Time, Clocks, and the Ordering of Events in a Distributed System
by L. Lamport

CS 5204  Operating Systems
Vladimir Glina
Fall 2005
Overview

- Key Points
- Background
- Partial Ordering
- Extension for Total Ordering
- Further Work
- Key Points Reiteration
- Evaluation
- Discussion
Key Points

1. The “happens before” relation on the system event set
2. The events partial ordering on the base of the relation
3. The distributed algorithm for logical clock synchronization
4. The algorithm extension to the case of total events ordering
5. The algorithm application for physical clock synchronization
Background: Distributed System Features

- Spatially separated processes
- Processes communicate through messages
- Message delays are considerable
- Absence of the single timer leads to synchronization problems
  - Example: totally ordered multicast
Background: Synchronization Approaches

- Physical Clock Adjustment
  - All clocks show the same actual time
  - Problems:
    - **Most important**: backward time flow possible
    - Sophisticated time services (i.e. WWV); or
    - Reliance on a human operator

- Logical Clock Adjustment
  - Consistency is important, not actual time
Partial Ordering: Basics

- A system is a set of processes $P_i$
- A process is a set of events $a, b, \ldots$ with total ordering
- “Happened before” ($\rightarrow$) relation:
  - $(a \in P) \land (b \in P) \land (a \text{ comes before } b) \Rightarrow a \rightarrow b$
  - $(P_1 \text{ sends } a \text{ to } P_2) \land (b \text{ is the receipt of } P_2 \text{ for } a) \Rightarrow a \rightarrow b$
  - $(a \rightarrow b) \land (b \rightarrow c) \Rightarrow a \rightarrow c$
- $(a \rightarrow b) \land \neg(b \rightarrow a) \Rightarrow a \text{ and } b \text{ are concurrent}$
- $(a \rightarrow a) \forall a$, so “happened before” is an irreflexive partial ordering on the set of all the system events
Partial Ordering: Example

P1

a → f
b → s
c → m
d || s

P2

g → i
j → k
l → m

P3

n → o
p → q
r → s

d || s
i || q
k || r
Partial Ordering: Synchronization

- **Logical clock**: $C\langle a \rangle = C_j\langle a \rangle$ if $a \in P_j$
- **Check condition**: for $\forall$ $a, b$
  
  $a \rightarrow b \Rightarrow C\langle a \rangle < C\langle b \rangle$ (not vice versa)
  
  The check condition is satisfied if

  - **C1.** $(a, b \in P_j) \&\& (a$ comes before $b)$
    
    $\Rightarrow C_i\langle a \rangle < C_i\langle b \rangle$

  - **C2.** $(P_i$ sends $a$ to $P_j) \&\& (b$ is the receipt of $P_j$ to $a)$
    
    $\Rightarrow C_i\langle a \rangle < C_j\langle b \rangle$

- **C never decreases!**
Partial Ordering: Implementation Rules

- **IR1.** Each $P_i$ increments $C_i$ between any two successive events.

- **IR2.**
  a) If $a$ is the sending of a message $m$ by $P_i$, then $m$ contains a timestamp $T_m = C_j\langle a\rangle$; and
  b) Upon receiving $m$, $P_i$ sets $C_j$ greater than or equal to its present value and greater than $T_m$.
Partial Ordering: Unregulated Clocks

\[
\begin{array}{c|c|c|c|c}
& A & B & C & D \\
0 & 6 & 8 & 10 & 54 \\
6 & 12 & 16 & 20 & 60 \\
12 & 18 & 24 & 30 & 64 \\
18 & 24 & 32 & 40 & 72 \\
24 & 30 & 40 & 50 & 70 \\
30 & 36 & 48 & 60 & 80 \\
36 & 42 & 56 & 70 & 90 \\
42 & 48 & 64 & 80 & 100 \\
48 & & & & \\
54 & & & & \\
60 & & & & \\
\end{array}
\]
Partial Ordering: Corrected Clocks

0  6  12  18  24  30  36  42  48  54  60  66  70  74  76  78  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100
Total Ordering: Definition

- $\prec$ is an arbitrary total ordering of processes
- “Happen before” for total ordering($\iff$):
  \[(a \in P_i) \land (b \in P_j) \Rightarrow a \iff b \text{ iff}
  \]
  - $C_i\langle a \rangle < C_j\langle b \rangle$, or
  - $P_i \prec P_j$
- The total ordering depends on $C_i$ and is not unique
Total Ordering: Synchronization

1. $P_i$ broadcasts the message $T_m:P_i$ (request resource) and puts it on its request queue.
2. When $P_j$ receives $T_m:P_i$, it puts the message on its request queue and sends the acknowledgment to $P_i$.
3. To release the resource, $P_i$ removes $T_m:P_i$ from its queue, broadcasts a timestamped release message.
4. When $P_j$ receives the release message, it removes $T_m:P_i$ from its queue.
5. $P_i$ is granted the resource when
   1) It has $T_m:P_i$ in its queue ordered before any other request in the queue by the relation $\rightarrow$; and
   2) $P_i$ has received a message from every other process timestamped later than $T_m$.  

9/14/2005
Further Work: Vector Timestamps

- Lamport clock is:
  - Consistent: $a \rightarrow b \Rightarrow C\langle a \rangle < C\langle b \rangle$
  - Not: $C\langle a \rangle < C\langle b \rangle \Leftrightarrow a \rightarrow b$ (not strongly consistent)

- Vector timestamps (VT) are strongly consistent

- VT address potential causality
  - Allow to say if $a$ happened before $b$, but not if $a$ caused $b$

- VT say how many events have occurred so far at all processes

- VT solve the totally-ordered multicasting problem
Lack of Strong Consistency
Vector Clocks (1)

\[
\begin{align*}
\text{P1} & : [1 \ 0 \ 0] \rightarrow [5 \ 5 \ 3] \\
& \quad [2 \ 0 \ 0] \rightarrow [2 \ 3 \ 0] \\
& \quad [3 \ 0 \ 0] \rightarrow [4 \ 3 \ 0] \\
\text{P2} & : [0 \ 1 \ 0] \rightarrow [2 \ 3 \ 0] \\
& \quad [2 \ 2 \ 0] \rightarrow [2 \ 4 \ 3] \\
& \quad [2 \ 3 \ 0] \rightarrow [2 \ 5 \ 3] \\
\text{P3} & : [0 \ 0 \ 1] \rightarrow [3 \ 1 \ 5] \\
& \quad [0 \ 1 \ 2] \rightarrow [3 \ 1 \ 6] \\
& \quad [0 \ 1 \ 3] \rightarrow [3 \ 1 \ 5] \\
\end{align*}
\]
Vector Clocks (2)

P1

a [1 0 0] b [2 0 0] c [3 0 0] d [4 3 0] f [5 5 3]

g [0 1 0] i [2 0 0] j [3 0 0] k [2 3 0] l [2 5 3] m

P2

[0 1 0] [2 2 0] [2 3 0] [2 4 3] [2 5 3] [3 6 5]

P3

[0 0 1] [0 1 2] [0 1 3] [3 1 4] [3 1 5] [3 1 6]

d || s [4 3 0] < [3 1 6]  q || i [3 1 4] < [2 2 0]  k || r [2 4 3] < [3 1 5]
Key Points Reiteration

1. The “happens before” relation on the system event set
2. The events partial ordering on the base of the relation
3. The distributed algorithm for logical clock synchronization
4. The algorithm extension to a case of total events ordering
5. The algorithm application for physical clock synchronization
Evaluation

- The logical clocks idea is very appealing
- Virtually no revision on previous work
- Nice to have more mathematically strict extension on total ordering, if possible
Discussion

Thank you!

Any questions?