Virtual Time

“Virtual Time and Global States of Distributed Systems”
Friedmann Mattern, 1989
Previous Lectures

• Concurrent and distributed programming limitations
  • Amdahl’s vs. Gustafson's law
  • Memory architecture: UMA, NUMA and distributed
  • Memory-optimized programming (latency vs. bandwidth)
  • Cache coherence
  • Cache false sharing

• Problem partitioning/decomposition
  • SIMD vs. MIMD
  • Granularity and load balancing
  • Task-management/synchronization overhead vs. strugglers.
  • Communication cost of different implementation approaches

• Concurrent and distributed libraries
  • OpenMP: shared memory
  • MPI: distributed memory (tutorials)
Rolling shutter effect

Panoramic fail
Virtual Time

• The model: asynchronous distributed system
  • A set of processes having no shared memory, communicating by message transfer.
  • Message delay > 0, but is not known in advance.

• A global observer – sees the global state at certain points in time.
  • It can be said to “take a snapshot” of the global state.

• A local observer – (one of the processes in the system) sees the local state.
  • Because of the asynchrony, a local observer can only gather local views to an approximate global view.

• The lack of a global observer is a hard hazard for many management and control problems:
  • mutual exclusion, deadlock detection, distributed contracts, leader election, load sharing, checkpointing etc.
Panoramic Consistency
Facebook as an Example

• Group discussion
234666 - Introduction to Hell
Closed Group
Art Vandelay
עברית כבר 10 ימים מ哪家好! מה עם הצויים?
Like · Comment · Toda

👍 64 likes this.

Write a comment...
Art Vandelay
עבריתibre 10 ימים מҺמוотор! מה עם היצורים?
Like · Comment · Toda

64 likes this.

Martin Van Nostrand
Canrak ha mashtbritim um hareshov shel fektora
· Like

Write a comment...
עבורה כבר 10 ימים מרمين! מה עם הצוים?

64 likes this.

Martin Van Nostrand
ננהה המ מסתכלים על החשש של הפקטור
• Like

H.E. Pennypacker
לא נראה לי, אם wollen אני פקטור בקוריםذه...
• Like

Write a comment...
Art Vandelay
 عبرו כבר 10 ימים מהמשהו! מה עם הצויפים?
Like · Comment · Toda

64 likes this.

Kel Varnsen
',["image"]

mysho פניה כבר莲צוג סנטור?
· Like

Write a comment...
עברו כבר 10 ימים ממהכתו! מה עם היצורים?

Like · Comment · Today

64 likes this.

מארטין ון נוסרנד
כראתemy
מסתברים עפ"ם החשש של הפקטור
· Like

H.E. Pennypacker
לא וראתי לי, אני בזוז בזוז
...פעמים אני פקטור בקורות הזה
· Like

Write a comment...

מישאו פנה כבר לציוג
سمסר?
· Like
Art Vandelay

עבר 10 ימים ממהќתים! מה עם היזיומך?

Like · Comment · Today

64 likes this.

Martin Van Nostrand

 Conseqa may confuse us the obvious effect on the picture.
· Like

H.E. Pennypacker

 Likewise, isn't anything written both there...
· Like

Kel Varnsen

 Misho Fen he cer llitng stem or?
· Like

Write a comment...
Facebook as an Example - Conclusions

- A post should appear after all the posts that was already visible to the sender at the moment it was sent
- Some posts can appear in any order and the thread will still “make sense”
Events

- An event is a change in the process state.
- An event happens instantly, it does not “take time”.
- A process is a sequence of events
- There are 3 types of events:
  - send event – causes a message to be sent
  - receive event – causes a message to be received
  - local event – only causes an internal change of state

- Events correspond to each other as follows:
  - All events in the same process happen sequentially, one after the other.
  - Each send event has a corresponding receive event
- This allows us to define the happened before relation among events.
The Happened Before Relation

• We say that event $e$ happened before event $e'$ if one of the following properties holds:
  
  • **Processor Order**: $e$ precedes $e'$ in the same process
  • **Send-Receive**: $e$ is a send and $e'$ is the corresponding receive
  • **Transitivity**: exists $e''$ such that $e < e''$ and $e'' < e'$

• Denoted by $e \to e'$ or $e < e'$
Independent/Concurrent Events (1)

• Two events \(e, e'\) are said to be independent or concurrent if not \(e < e'\) and not \(e' < e\).
• Denoted by \(e \parallel e'\)
Independent/Concurrent Events (2)

- Two such diagrams are called **equivalent** when the happened before relation is the same in both.
  - When global time differs for certain events, think of processor execution as if it was a rubber band.
Cuts

- A **cut** $C$ of a set of events $E$ is the union set of the cut events set $\{c_i\}$ and all the events $e \in E$ which precede any of the $c_i$ in program order.
Consistent Cuts (1)

• A cut is said to be **consistent** if also:
  • $e \in C$ and $e' < e$ always imply $e' \in C$.

• If a receipt of a message is inside a consistent cut – so is its sending.
  • The opposite does not necessarily hold.
Consistent Cuts (2)

- Intuitively, a cut is **consistent** if when stretching the execution lines so that the cut events form a vertical line, there is no send-receive arrow which crosses the cut right to left.
Inconsistent Cut (1)

$\hat{c}_i$ are events of the cut
Inconsistent Cut (2)

$c_i$ are events of the cut
Global View and Consistent Cuts

- If a cut is consistent, then there is an execution of the diagram in which all cut events happen at the same moment.
- The view of a global observer at any point in time is necessarily consistent:
  - A message that arrived was necessarily previously sent.
- Thus: A global view determines a consistent cut at any point in time.
Consistent Cuts and Global Snapshots

• Any **consistent cut** may actually happen at a certain point during an execution

• The cut is said to contain the local states of the cut processes and the set of messages that were already sent but not received

• This is a “**snapshot**” of what is happening in the system

• The Global Snapshot problem:
  • Find an efficient protocol to compute consistent cuts. i.e. collect the local process states and the messages which are “in flight” (or, “in the communication buffers”) at the time of the cut.
Virtual Time (Lamport, 1978)

• A **logical clock** is a function \( C: E(\text{vents}) \rightarrow T(\text{ime}) \)

• *E* – set of events

• *C(e)* – timestamp of *e*

• *T* – a partially ordered set such that \( e < e' \implies C(e) < C(e') \)
  • The opposite not necessarily true, e.g. concurrent events.

• Commonly, \( T = \mathbb{N} \), and there is a local clock \( C_i \) for every process \( P_i \)
Logical Clocks Protocol (Lamport, 1978)

The **logical clocks** tick using the following protocol:

1. When a local event $e$ is executed by $P_i$:
   - new $C_i := C_i + d$ \quad (d>0)
   - $C(e) := \text{Timestamp}(e) := \text{new} \ C_i$

2. A message $m$, sent on event $s = \text{send}(m)$, is time-stamped:
   - $t(m) = C(s)$
   - $t(m)$ is sent together with the *message* as a pair $(t(m), m)$

3. When $P_i$ receives $(t(m), m)$:
   - $C_i := \max \{ C_i, t(m) \} + d$ \quad (d >0)

• Usually, $d=1$.
  • However, $d$ may change arbitrarily and dynamically (as long as it is positive)
  • E.g., in an attempt to reflect actual time
Logical Clocks Protocol Example

- **P1**: $C(e_{11})=1$
  - $e_{11}$
  - $C(e_{21})=1$
  - $e_{21}$
  - Send$(m)$, $C(e_{22})=2$
  - $e_{22}$
  - $(t(m)=2,m)$
  - $e_{31}$
  - $C(e_{31})=1$
  - Receive$(m)$
  - $e_{32}$
  - $C(e_{32})=3$
  - $e_{32}$

**Problem?**
Logical Clocks Protocol Example (2)

Problem?
Loss of Execution Information

• Suppose we read execution log where only the timestamps of the events were recorded

• Suppose \( C(e) < C(e') \).

• We cannot distinguish between \( e < e' \) and \( e \parallel e' \).
  • Information whether the events are independent is lost
  • Notice: a global observer would know that

• But, we can infer that \( e' \) cannot have happened (Lamport-sense) before \( e \):
  • “not \((e>e')\)”.

• Question:
  • Can we change the clock mechanism to preserve more info?
A Vector Clock for a Global Observer

Vector of times as seen by a global observer

0 0 0 0
0 1 0 0
0 0 0 0
0 0 0 0
Approximating Global Vector Time

• Idea: let’s compute for every process an approximation of the global observer’s **vector clock**
  • Will have to show approximation is good enough
• Instead of scalar clocks, process $P_i$ will maintain a vector clock $C_i$ [of size $|P|$]
• Timestamp $C(e)$ of an event $e$ at process $P_i$ is the entire vector $\vec{C}_i$ when $e$ happens
Approximating Global Vector Time Protocol

- When executing a local event $e$ in $P_i$:
  - $C_i[i] := C_i[i] + 1$ (ticks)
  - $C(e) := \overrightarrow{C_i}$ (the entire vector)

- A message $m$ sent by event $s = \text{send}(m)$, is time-stamped:
  - $t(m) = C(s)$
  - The message is sent together with the timestamp as a pair $(t(m), m)$

- Upon receipt of a message $m$ by $P_i$:
  - For every $j$: $C_i[j] := \max \{ C_i[j], t(m)[j] \}$
  - $C_i[i] := C_i[i] + 1$ (ticks)

- The time stamp of a received message contains information about other processes states

- On receipt of a message, a process combines this information with what it already knows about the system state, and communicates it elsewhere, piggybacking on messages it sends out
P3 receives ticking information from P2 *albeit* they never communicate *directly*
Actual Order of Independent Events not Important

What is the actual global order?

- Really don’t Care!
- Can tell $e_{2,1}$ and $e_{3,1}$ are concurrent (vectors independent)
- Inconsistent global time is impossible

$e.g. \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}$
An Invariant (*)

• Only $P_i$ can advance his clock: $C_i[i]$ ticks only at $P_i$
• Thus, $P_i$ always knows the up-to-date time of his local time
• Thus, at any point in time, for any $i, j$, it holds that:

\[ C_i[i] \geq C_j[i] \]
Vector Times can be Compared

• $u$ happened before $v$:
  \[ u < v \iff \forall i: u[i] \leq v[i] \text{ and } u \neq v \]

• $u$ and $v$ concurrent:
  \[ u \parallel v \iff \neg(u < v) \text{ and } \neg(v < u) \]

$\Rightarrow$ With vector times, timestamps can keep more information on order of events

• **Need to show**: time stamp vector relations and Lamport’s happens-before relation of events are equivalent.
Vector Times Relations Represent Order of Events

**Claim**: for every $e, e'$

1. $C(e) <_{\text{vectorial}} C(e') \iff e <_{\text{Lamport}} e'$

2. $C(e) \parallel_{\text{vectorial}} C(e') \iff e \parallel_{\text{Lamport}} e'$

3. If $e$ is known to have happened in process $P_i$ then:
   
   for every $e' \neq e$: $C(e)[i] \leq C(e')[i] \iff e < e'$
Vector Times Relations Represent Order of Events

• **Proof for 1 (⇒):** $C(e) <_{\text{vectorial}} C(e') \Rightarrow e <_{\text{Lamport}} e'$

• If $e'$ “knows” the local time of $e$
  • i.e., the vector component corresponding to the process where $e$ happened is bigger in $e'$ than in $e$

• Then there must exist a sequence of time stamps from $e$ to $e'$ which represents the happened before relation
Vector Times Relations Represent Order of Events

• **Proof for 1 (⇐):** $C(e) <_{\text{vectorial}} C(e') \iff e <_{\text{Lamport}} e'$
  
• If $e < e'$ then there exist a sequence of time stamps from $e$ to $e'$

• And these time stamps monotonically grow in each component.

• The growth is strict in at least one component: that corresponding to $e'$'s process

• Proofs of 2,3 are similar
Cuts Revisited

• Let $X$ be a cut, $\{x_i\}$ cut events, a system of $n$ processes.
• We say that the global time of $X$ is (max is taken per entry)
  \[ t_X = \max_{\text{element wise}} \{C(x_1), \ldots, C(x_n)\} \]
• Different cuts have same global time (see example below)

\[ \begin{array}{cccc}
  & 1 & & 2 \\
0 & 1 & 0 & 2 \\
0 & 0 & 2 & 0 \\
0 & 0 & 0 & 0 \\
\end{array} \]

Example:

\[ t_X = t_Y = \begin{pmatrix} 2 \\ 1 \\ 2 \\ 0 \end{pmatrix} \]

• However ...
Different Consistent Cuts Have Different Global Times

• **Claim:** $X$ is a **consistent cut** $\iff t_X = [C(x_1)[1], \ldots, C(x_n)[n]]$

• **Proof ($\Rightarrow$):**
  - If $X$ is consistent, then its cut events $x_i$ can be viewed as if they occurred at the same moment.
  - By the invariant (* $C_i[i] \geq C_j[i]$) at that moment the claim holds.

• **Proof ($\Leftarrow$):**
  - If $X$ is not consistent, then there exists a message that was sent (say) from $P_i$ after $x_i$, and was received by $P_j$ before $x_j$.
  - Let $t$ be the timestamp on the message, then $x_i[i] < t[i] \leq x_j[i]$.
  - Thus $t_x > [C(x_1)[1], \ldots, C(x_n)[n]]$. 