Today’s Theme: Consistency in a Distributed System

Papers:

Why should we care about this theme?

• How can we make a consistent global state of an **asynchronous** system?
  • What even is our synchronous protocol?
  • Who today is asking/using these questions?

• How can we coordinate in a distributed system?
  • If an (airline) reservation database system is replicated and distributed and people can purchase tickets locally, how can we order the purchasing of tickets (e.g. two people want to buy a seat, but only one seat is left)?
Why should we care about these papers?

- Why to care about the set
- 5414?
Initial thoughts?
Today’s Goals

  - Understand:
    - The “happened before” relation
    - How to establish partial/total order and what to do with it
    - Physical, logical, local/global clocks

  - Understand:
    - Global Snapshot
    - Consistent Cut
    - How Global Snapshot helps in finding a Consistent Cut
Time, Clocks, and the Ordering of Events in a Distributed System

Leslie Lamport
Massachusetts Computer Associates, Inc.
Communications of the ACM - July 1978
Leslie Lamport

- BS in Math from MIT, 1960
- MA, Ph.D. in Math from Brandeis, 1972
- Research in industry
  - Massachusetts Computer Associates (1970-77)
  - SRI International (1977-85)
  - DEC/Compaq (1985-2001)
  - Microsoft Research (2001-)
- 2000 PODC Influential Paper award for *Time, Clocks, and the Ordering of Events*
- 2013 Turing Award Winner and initial developer of LaTeX
instructions simultaneously on the various processors. The problem was how to take a Fortran program which does everything sequentially—one instruction at a time—and compile it to do multiple instructions at once. Lamport plunged into the project.

That was the leading edge of computers at that time [1972]. I applied a little bit of simple mathematics to figure out what the algorithms were. I thought it was very straightforward stuff. It involved linear algebra. The reaction at COMPASS was like I was Moses coming down from the mountain with this very obscure sacred text that they would study. They eventually figured out what it meant in terms of an algorithm.

After receiving his Ph.D. degree in 1972, Lamport continued to work on the ILLIAC in COMPASS’s Palo Alto office.

I went off to California by myself, working long distance for COMPASS, but not talking regularly to anyone at COMPASS. I think that my isolation from the academic research community was a key to my work.

I was outside academia so I wasn’t influenced by the current fashion of what one was supposed to do.
Outline

● Partial Ordering of Events
● Logical Clocks
● Total Ordering of Events
● Anomalous Behaviour
● Physical Clocks
Definitions and Perspectives

- A *process* as a set of events with an *a priori* total ordering -- a sequencing.
- System as a collection of processes, each process as a sequence of events
- sending/receiving a message as an event in a process
"Happened before" relation, "→"

**Definition.** The relation "→" on the set of events of a system is the smallest relation satisfying the following three conditions: (1) If \( a \) and \( b \) are events in the same process, and \( a \) comes before \( b \), then \( a \rightarrow b \). (2) If \( a \) is the sending of a message by one process and \( b \) is the receipt of the same message by another process, then \( a \rightarrow b \). (3) If \( a \rightarrow b \) and \( b \rightarrow c \) then \( a \rightarrow c \). Two distinct events \( a \) and \( b \) are said to be concurrent if \( a \leftrightarrow b \) and \( b \leftrightarrow a \).

(Lamport 559)
\[ a \Rightarrow a \]

“an irreflexive partial ordering on the set of all events in the system”
Fig. 1.

Space-time diagrams
I.e., what can we say from $a \rightarrow b$?

$a \rightarrow b$ means $a$ can causally affect $b$

Concurrency?

Two events are concurrent if neither can causally affect the other.
Outline

● Partial Ordering of Events
● Logical Clocks
● Total Ordering of Events
● Anomalous Behaviour
● Physical Clocks
Logical Clocks

• How might we begin thinking about clocks?
• “We begin with an abstract point of view in which a clock is just a way of assigning a number to an event, where the number is thought of as the time at which the event occurred” (Lampson 559)
• a clock $C_i$, for each process $P_i$, as a function assigning a number $C_i(a)$ to any event $a$ in $P_i$
• system of clocks as a function $C$ assigning to any event $b$ the number $C(b)$ such that $C(b) = C_j(b)$ if $b$ is an event in $P_j$
• yet relating numbers $C_i(a)$ to physical time, instead regarding $C_i$ as logical clocks
  ○ Use of counters rather than use of an ‘actual timing mechanism’
Correctness?

- Physical timing of events, order of events
- If an event $a$ happens before another event $b$, $a$ should happen at an earlier time than $b$
  - Time??
    - Clock Condition
      - “For any events $a, b$: if $a \rightarrow b$ then $C(a) < C(b)$
    - Why can we not assume the converse as well?
"Happened before" relation, "→"

Definition. The relation "→" on the set of events of a system is the smallest relation satisfying the following three conditions: (1) If \( a \) and \( b \) are events in the same process, and \( a \) comes before \( b \), then \( a \rightarrow b \). (2) If \( a \) is the sending of a message by one process and \( b \) is the receipt of the same message by another process, then \( a \rightarrow b \). (3) If \( a \rightarrow b \) and \( b \rightarrow c \) then \( a \rightarrow c \). Two distinct events \( a \) and \( b \) are said to be concurrent if \( a \leftrightarrow b \) and \( b \leftrightarrow a \).

(Lamport 559)
Correctness

• Clock Condition
  ○ “For any events $a, b$: if $a \rightarrow b$ then $C(a) < C(b)$”

• 2 conditions to satisfy the Clock Condition:
  ○ 1. If $a$ and $b$ are events in a process $P_i$, and $a$ comes before $b$, then $C_i(a) < C_i(b)$
     ■ There must be a tick line between any two events on a process line
  ○ 2. If $a$ is the sending of a message by process $P_i$ and $b$ is the receipt of that message by process $P_j$, then $C_i(a) < C_j(b)$
     ■ Every message line must cross a tick line

...Tick lines?
Fig. 1.

Space-time diagrams
Space-time diagrams
Problem with Partial Order

- A single resource
- Processes must synchronize to avoid conflicts
- Requests must be granted in order
Problem with Partial Order

- A single resource
- Processes must synchronize to avoid conflicts
- Requests must be granted in order

Partial order is not helpful in solving this problem as partial order cannot resolve concurrent requests

Total ordering required…..
Outline

- Partial Ordering of Events
- Logical Clocks
- **Total Ordering of Events**
- Anomalous Behaviour
- Physical Clocks
Total Ordering of Events

- Total ordering eliminates concurrency
- Identify message with event sending it
Distributed Mutual Exclusion

Every node $i$ has a request queue $q_i$

keeps requests sorted by logical timestamps (total ordering enforced by including process id in the timestamps)

To request critical section:

1) send timestamped REQUEST($T_m : P_i$) to all other nodes

2) put ($T_m : P_i$) in its own queue

On receiving a request ($T_m : P_i$):

1) send timestamped REPLY to the requesting Node $i$

2) put REQUEST ($T_m : P_i$) in the queue
Distributed Mutual Exclusion

Process i enters critical section if:

1) $(T_m, P_i)$ is at top of its own queue, and
2) process i received a message (any message) with timestamp larger than $(T_m, P_i)$ from all other nodes.

To release critical section:

1) Process i removes its request from its own queue and sends a timestamped RELEASE message to all other nodes.
2) On receiving a release message from i, i’s request is removed from the local request queue.
Distributed Mutual Exclusion

Process i enters critical section if:

1) $(T_m : P_i)$ is at top of its own queue, and

2) process i received a message (any message) with timestamp larger than $(T_m : P_i)$ from all other nodes. Why is this condition required?

To release critical section:

1) Process i removes its request from its own queue and sends a timestamped RELEASE message to all other nodes.

2) On receiving a release message from i, i’s request is removed from the local request queue.
Total Ordering of Events

**Step 1:** $P_i$ sends request resource

- $P_i$ sends request $T_m:P_i$ to $P_j$
- $P_i$ puts request $T_m:P_i$ on its request queue
Total Ordering of Events

**Step 1:** $P_i$ sends request resource

- $P_i$ sends request $T_m:P_i$ to $P_j$
- $P_i$ puts request $T_m:P_i$ on its request queue
Total Ordering of Events

Step 2: $P_j$ adds message

$P_j$ puts request $T_m : P_i$ on its request queue

$P_j$ sends acknowledgement $T_m : P_j$ to $P_i$
Total Ordering of Events

**Step 2:** $P_j$ adds message

- $P_j$ puts request $T_m:P_i$ on its request queue
- $P_j$ sends acknowledgement $T_m:P_j$ to $P_i$
Total Ordering of Events

**Step 3:** $P_i$ sends release resource

- $P_i$ removes *request* $T_m : P_i$ from request queue
- $P_i$ sends *release* $T_m : P_i$ to each $P_j$
Total Ordering of Events

**Step 3:** $P_i$ sends release resource

- $P_i$ removes *request* $T_m:P_i$ from request queue
- $P_i$ sends *release* $T_m:P_i$ to each $P_j$
Total Ordering of Events

Step 4: $P_j$ removes message

$P_j$ receives release $T_m : P_i$ from $P_i$

$P_j$ removes request $T_m : P_i$ from request queue
Message Complexity?
Message Complexity?

- $3(n - 1)$ messages per critical section invocation
- Requests are granted in the increasing order of timestamps
Message Complexity?

• $3(n - 1)$ messages per critical section invocation
• Requests are granted in the increasing order of timestamps

Ricart Agarwala – $2(n - 1)$, also doesn’t require FIFO assumption

Roucairol – Carvalho – $(0, 2(n - 1)]$ based on request pattern
Outline

- Partial Ordering of Events
- Logical Clocks
- Total Ordering of Events
- Anomalous Behaviour
- Physical Clocks
Anomalous Behavior
Strong Clock Condition

*Strong Clock Condition.* For any events $a, b$ in $\mathcal{S}$:

$$\text{if } a \rightarrow b \text{ then } C(a) < C(b).$$

This is stronger than the ordinary Clock Condition because $\rightarrow$ is a stronger relation than $\rightarrow$. It is not in general satisfied by our logical clocks.

*Now what?*
Outline

- Partial Ordering of Events
- Logical Clocks
- Total Ordering of Events
- Anomalous Behaviour
- Physical Clocks
Physical Clocks

“One of the mysteries of the universe is that it is possible to construct a system of physical clocks which, running quite independently of one another, will satisfy the Strong Clock Condition” (Lamport 562)
Physical Clocks

- $C_i(t)$ denote reading of clock $C_i$ at physical time $t$
- **Assumptions:**
  
  $C_i(t)$ is a continuous, differentiable function of $t$ except for isolated jump discontinuities where clock is reset
Physical Clocks

• For clock $C_i$ to be a true physical clock it must run at approximately the correct rate. That is we must have $\frac{dC_i}{dt} \approx 1$

• So, we assume the following conditions are satisfied

1) $\exists k$ such that $\forall i: \left| \frac{dC_i}{dt} - 1 \right| < k$

2) $\forall i, j: \left| C_i(t) - C_j(t) \right| < \epsilon$
Physical Clocks

• For preventing anomalous behavior ($\mu < \text{Shortest transmission time}$)
  3) $\forall i,j,t: C_i(t + \mu) - C_j(t) > 0$

Combining 3) with 1) and 2) we get

$$\frac{\epsilon}{1 - k} \leq \mu$$
Fig. 2.
Summary

- Partial Ordering of Events
- Logical Clocks
- Total Ordering of Events
- Anomalous Behaviour
- Physical Clocks
Distributed Snapshots: Determining Global States of Distributed Systems

- Ph.D from MIT in Electrical Engineering
- Was in University of Texas at Austin, CS Department from 1970 – 89
- Simon Ramo Professor at CalTech
- Member of National Academy of Engineering
- IEEE Koji Kobayashi Award(1987)
- A.A. Michelson Award(1985)

- BS in Math from MIT, 1960
- MA, PhD in Math from Brandeis, 1972
- Research in industry
  - Massachusetts Computer Associates (1970-77)
  - SRI International (1977-85)
  - DEC/Compaq (1985-2001)
  - Microsoft Research (2001-)
- 2000 PODC Influential Paper award for Time, Clocks, and the Ordering of Events
- 2013 Turing Award Winner and initial developer of LaTeX
What is a Snapshot?

• Just a collection of local states of processes and channels!
Why do we need Snapshots?
Why do we need Snapshots?

• **Garbage collection**: Removing objects that do not have any references

• **Deadlocks**: Detecting deadlocks in distributed system

• **Checkpointing**: Helps in rolling back in case of failure
Challenges in recording Global State

• Processes do not share clocks or memory (or) there is no universal clock.
• So, no way to ensure that all nodes record state at exactly the same time…
Challenges in recording Global State

• Processes do not share clocks or memory (or) there is no universal clock.

• So, no way to ensure that all nodes record state at exactly the same time.

• Global snapshot is all about finding a **consistent cut**!
  • so what is a consistent cut?
Consistent Cut

Consistent Cut: a cut that obeys causality

• A cut C is a consistent cut if and only if:
  for (each pair of events \( e, f \) in the system)
  • Such that event \( e \) is in the cut \( C \), and if \( f \rightarrow e \) (\( f \) happens-before \( e \))
    • Then: Event \( f \) is also in the cut \( C \)
Example of (In)Consistent Cut

Consistent Cut

Inconsistent Cut

G → D, but only D is in cut
Solution: Chandy-Lamport’s Algorithm

• Send a marker along all channels immediately after recording state

• Upon receipt of a marker along channel $c$:
  • Record process state if not already recorded
  • Record state of $c$ as all messages received between recording process state and receiving marker

• Eventually markers will reach all processes, so all state will be recorded
Example

P1
A B C D E

P2
E F G

P3
H I J

Instruction or Step

Message
P1 is Initiator:
  • Record local state $S_1$,
  • Send out markers
  • Turn on recording on channels $C_{21}, C_{31}$
- First Marker!
- Record own state as S3
- Mark $C_{13}$ state as empty
- Turn on recording on other incoming $C_{23}$
- Send out Markers
S1, Record $C_{21}, C_{31}$

- S3
- $C_{13} = <>$
- Record $C_{23}$
Duplicate Marker!
State of channel $C_{31} = <>$

- S1, Record $C_{21}, C_{31}$
- $C_{13} = <>$
- Record $C_{23}$
- S1, Record $C_{21}, C_{31}$
- $C_{31} = <>$

- S3
- $C_{13} = <>$
- Record $C_{23}$

- First Marker!
- Record own state as S2
- Mark $C_{32}$ state as empty
- Turn on recording on $C_{12}$
- Send out Markers
• S3
• $C_{13} = <>$
• Record $C_{23}$

• S1, Record $C_{21}, \epsilon_{31}$
• $C_{31} = <>$

• S2
• $C_{32} = <>$
• Record $\epsilon_{12}$

• Duplicate!
- Duplicate!
- $C_{21} = \langle \text{message G\rightarrow D} \rangle$
- $C_{31} = \langle \rangle$
- S1, Record $\epsilon_{21}$, $\epsilon_{31}$
- $C_{13} = \langle \rangle$
- S3
- Record $C_{23}$
- $C_{32} = \langle \rangle$
- $C_{12} = \langle \rangle$
Algorithm has Terminated

- \( C_{21} = \langle \text{message } G \rightarrow D \rangle \)
- \( C_{31} = \langle \rangle \)
- \( C_{13} = \langle \rangle \)
- \( C_{32} = \langle \rangle \)
- \( C_{12} = \langle \rangle \)
- \( C_{23} = \langle \rangle \)

\( S_1 \)
Collect the Global Snapshot Pieces

$C_{21} = \langle \text{message } G \rightarrow D \rangle$

$C_{31} = \langle \rangle$

$C_{13} = \langle \rangle$

$C_{32} = \langle \rangle$

$C_{12} = \langle \rangle$

$C_{23} = \langle \rangle$
Consistent Cut Obtained

Consistent Cut captured by our Global Snapshot Example
Summary

- Global Snapshot definition
- Uses of collecting a Global Snapshot
- Consistent Cuts
- Chandy-Lamport’s Algorithm
Parting Thoughts & Questions: Ours

- What does Lamport bring from the first paper into the second?
- Why might we want to cover these papers before MapReduce?
- What assumptions have been made?
  - FIFO - “The assumption that all messages sent from process Pi to Pj are received in the same order as they are sent is also too strong.”
Parting Thoughts & Questions: Yours

- “The idea is very novel in the time when it published as distributed systems were not so much relevant in those days.”
  - “I want to highlight [the first paper’s] elegance. We often take for granted that computing problems didn't come logically into the minds of our predecessors.”

- “The paper also didn’t consider the scenario of process and message delivery failure.”
  - “unreasonable assumptions about reliability, such as nodes cannot fail and messages cannot be dropped and must have predictable delay.”

- “The ideas presented were quite complex...This paper presented the formalisms needed quite well and established a clear language.”

- “The biggest weakness of this paper is that the system cannot inherently capture non-computational events such as phone calls, and also that there's no sense of the physical feasibility of this synchronization approach. How does propogation delay affect the synchronization scheme? How fault-tolerant is the system?”
“The strong point of this paper is a novel algorithm to guarantee synchronization of distributed system that does not rely on a central authority to perform the synchronization. A total ordering of events can be obtained even when only a partial ordering is available from the space-time diagram. However, in order to achieve this result, all processes must be aware of every other processes messages, which assume that all processes are not anomalous. This is an assumption that can be easily violated by a rogue process. The next logical step for this work is to be resilient to rogue processes which could be purposefully providing incorrect timestamps, or even collaborate to fool the synchronization algorithm.”
A Nice Review to Consider Together :)

“Lamport's paper on Time, Clocks and Ordering in a distributed system does an excellent job of defining a problem in distributed systems and proposing several solutions along with their requirements and drawbacks. It also omits a number of details for the sake of brevity and readability; it is outside of my expertise to be able to identify if any/all of these details are essential to his argument. For example, he omits details on ensuring eventual message delivery and message delivery order. Obviously TCP can be used to ensure order between any two processes, but it's not clear to me what algorithm Lamport would use to ensure eventual delivery; is it simply timeout based along with a known assumption of max delivery time? I assume it is since he says elsewhere that failures and fault-tolerance are not considered in this paper; but since he omits this small detail it's hard for me to know for sure. This is a minor point to pick as most of the paper is very clear, especially thanks to his decision to omit agonizingly small details. However, there are a few places where he introduces details for reasons that are unclear to me; he mentions 'unpredictable message delay', denotes a symbol for it and then never mentions it again. It seems like a waste and a bit confusing since it gave me the feeling that that measurement was meaningful and necessary for some part of the proof, even though it was not. Overall, the paper's structure is clear, addresses practical issues in distributed systems with a very practical theoretical approach (e.g. considering the fidelity of modern clocks in his error calculations) and is not overly burdened with complex proofs (but does provide them where necessary as footnotes and in the appendix.)

N.B. I don't have anywhere near the experience reading theoretical papers, so it's very likely that I'm missing some critical angles of assessment and/or giving him some free passes (e.g. I don't expect much more than a motivating example on how one might actually build or use a system that a theoretical paper describes)”
Further Reading Suggested by Prof. Alvisi

Dijkstra’s Take on Chandy-Lamport:

https://www.cs.utexas.edu/users/EWD/transcriptions/EWD08xx/EWD864.html
Acknowledgements

- Previous years’ slides
- CS4410 Lecture Slides
- Professor Hakim Weatherspoon and Professor Lorenzo Alvisi for presentation advice

THANK YOU