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

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:
  - $((a \in P) \&\& (b \in P) \&\& (a \text{ comes before } b) \Rightarrow a \rightarrow b$
  - $(P_i \text{ sends } a \text{ to } P_j) \&\& (b \text{ is the receipt of } P_j \text{ for } a) \Rightarrow a \rightarrow b$
  - $(a \rightarrow b) \&\& (b \rightarrow c) \Rightarrow a \rightarrow c$
  - $!(a \rightarrow b) \&\& !(b \rightarrow a) \Rightarrow a \text{ and } b \text{ are concurrent}$
  - $!(a \rightarrow a) \Rightarrow a$, so "happened before" is an irreflexive partial ordering on the set of all the system events
Partial Ordering: Example

\[ a \rightarrow f \quad b \rightarrow s \quad c \rightarrow m \quad d \parallel s \quad l \parallel q \quad k \parallel r \]

Partial Ordering: Synchronization

- **Logical clock:** \( C(a) = C(j) \) if \( a \in P_j \)
- **Check condition:** for \( \forall a, b \) \( a \rightarrow b \Rightarrow C(a) < C(b) \) (not vice versa)
- The check condition is satisfied if
  - **C1.** \( (a, b \in P_j) \) \& \& \( a \) comes before \( b \)
    \( \Rightarrow C(a) < C(b) \)
  - **C2.** \( (P_j \text{ sends } a \text{ to } P_i) \) \& \& \( (b \text{ is the receipt of } P_i \text{ to } a) \)
    \( \Rightarrow C(a) < C(b) \)
- **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 \( T_m = C_i(a) \) \& and
  b) Upon receiving \( m \), \( P_i \) sets \( C_i \) greater than or equal to its present value and greater than \( T_m \)

Partial Ordering: Unregulated Clocks

Partial Ordering: Corrected Clocks

Total Ordering: Definition

- \( \prec \) is an arbitrary total ordering of processes
- "Happen before" for total ordering (\( \Rightarrow \)):
  \( (a \in P) \text{ \&\& } (b \in P) \Rightarrow a \Rightarrow b \) iff
  - \( C(a) < C_j(b) \), or
  - \( P_i \prec P_j \)
- The total ordering depends on \( C_i \) and is not unique
Total Ordering: Synchronization

1. Pi broadcasts the message Tm:Pi (request resource) and puts it on its request queue.
2. When Pj receives Tm:Pi, it puts the message on its request queue and sends the acknowledgment to Pi.
3. To release the resource, Pj removes Tm:Pi from its queue, broadcasts a timestamped release message.
4. When Pi receives the release message, it removes Tm:Pi from its queue.
5. Pi is granted the resource when
   1) It has Tm:Pi in its queue ordered before any other request in the queue by the relation \( a \preceq b \)
      and
   2) Pi has received a message from every other process timestamped later than \( T_{m} \).

Further Work: Vector Timestamps

- Lamport clock is:
  - Consistent: \( a \rightarrow b \Rightarrow C(a) < C(b) \)
  - Not: \( C(a) < C(b) \nRightarrow a \rightarrow b \) (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)

Vector Clocks (2)

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?