Causality, consistency and logical time in distributed computations

Presented by: Dominik Menzi
Papers by: Prof. Mattern
Mentor: Thomas Locher
Outline

- Synchronous, asynchronous, and causally ordered communication
- Vector time
- Detecting causal relationships in distributed computations
- Conclusion
Part 1: Synchronous, asynchronous, and causally ordered communication

- A formal definition of different types of computations is needed w.r.t. causality
- Model
  - Processes form a distributed system
  - Internal-, send- and receive-events
  - Computation consists of local computations and messages
  - reliable communication
Types of computations

- A-computations
  - send and receive events are asynchronous

- FIFO-computations
  - channels have FIFO-property

- Causally ordered (NEW)

- S-computations
  - send and receive events are synchronous
  - message transmissions appear to be instantaneous
Types of computations (contd.)

- Generally, no computation type is superior to the others
- S-computations can be simulated with A-computations and vice versa
Def: Causality relation

- $\Gamma = \{(s,r) \in C_i \times C_j : s \text{ corresponds to } r\}$
- AS1: If $a \prec_i b$, then $a \prec b$
- AS2: $(s,r) \in \Gamma$, then $a \prec b$
- AS3: If $a \prec b$ and $b \prec c$, then $a \prec c$
**Def: A-computations**

- Processes $P_1 ... P_n$ with a tuple $C=(C_1 ... C_n)$ of local computations
- A set $\Gamma$ of corresponding send and receive events for which the causality relation holds
Def: FIFO-computations

- Additionally, for all \((s,r)\) and \((s',r')\) \(\in \Gamma\)
  \[ s \sim s' \land r \sim r' \land s \prec s' \Rightarrow r \prec r' \]
Def: CO-Computations

- Additionally, for all \((s,r)\) and \((s',r')\) \(\in \Gamma\)
  \[ r \sim r' \land s \prec s' \implies r \prec r' \]
Characterizations of CO-computations

- **Message ordered:**
  \[ s \prec s' \Rightarrow \neg (r' \prec r) \]

- **Empty Interval:**
  For each pair \((s, r) \in \Gamma\) the open interval \(<s, r> = \{x \in C : s \prec x \prec r\}\) is empty
Characterizations of CO-computations (contd.)

- CO-computations: triangle inequality:
  a computation is CO iff no message is bypassed by a chain of other messages

- CO-computations: Vertical message arrow criterion
  A computation C is CO iff for every m in C there exists a space-time diagram for C such that m can be drawn as a vertical message arrow and no arrows go from right to left
Def: RSC-computations

- RSC-computations: Realizable with Synchronous Communication
- A computation is called RSC if there exists a non-separated linear extension of $(C, \prec)$
Characterizations of RSC-computations

- Crowns: A crown is a sequence of pairs of corresponding send and receive events such that

\[ s_1 < r_2, s_2 < r_3, \ldots, s_{k-1} < r_k, s_k < r_1 \]

- A computation is RSC iff it contains no crown
Characterizations of RSC-computations (contd.)

- All message arrows in a diagram can be drawn vertical

- RSC-computations are equivalent to S-computations
Informal view: A-computations and S-computations

- S-computations are often regarded as a special case of A-computations (A-computations with empty channels)
- Proofs of algorithms for A-computations hold with rules for S-computations
- (but algorithms could deadlock in synchronous case)
Hierarchy of computations

- The paper shows a hierarchy of computations with different characteristics: synchronous, asynchronous, FIFO, causally ordered

S-computations $\subset$ CO-computations $\subset$ FIFO-computations $\subset$ A-computations
Hierarchy

S-computation

RSC

No crown

Empty interval

Message ordered

Vertical message arrows criterion

Vertical message arrow criterion

FIFO
Termination detection algorithm revisited

- Processes $P_0 \ldots P_{n-1}$, passive or active
- Send a token along a virtual ring
Termination detection algorithm revisited

- Processes $P_0 \ldots P_{n-1}$, passive or active
- Send a token along a virtual ring
Termination detection algorithm revisited

- Processes $P_0 \ldots P_{n-1}$, passive or active
- Send a token along a virtual ring
Termination detection algorithm revisited

- Processes $P_0...P_{n-1}$, passive or active
- Send a token along a virtual ring
Part 2: Vector time

- Calculation of global state in a system without real-time clocks
- Calculate potential causality between events.

- One can try to simulate a synchronous system on an asynchronous system
  - ... simulate global time
  - ... simulate global state
    and build algorithms on top
Virtual time

- Simulate global time by Lamport:
- Every process stores the “global time“
- Before a send event, a process increases its value of the global time and attaches the new value to the message
- If a process receives a message with a timestamp attached that is greater than its own value, it updates its local clock
Lamport Time

- Insufficient in some cases, it loses information by mapping events to integers:
- Events happening at the same time can get different timestamps...
Cuts

- Subset of events; Graphically, a zigzag line which cuts the diagram into two parts

- Cuts the diagram into past and future
Consistent Cuts

- A Cut is consistent if every message received was sent
- Inconsistent cuts yield „invalid“ space-time diagrams

- Can be seen as an instant in time
- One could use a cut to compute a global state
Vector Time

- Every process has a local clock
- Before a receive- or send-event a process increases its local clock
- Every process saves the most recent values it knows from all processes in a vector $V_i$
- A process attaches its local vector to the message
- If a process receives a message it updates its local vector
Properties of Vector Time

- The lattice of consistent cuts and the lattice of time vectors are isomorphic
- Vector time is able to model concurrency
Minkowski's space-time

- Maybe a better model of time than the „standard“ model
- Event P can only affect event b if b lies in the future light cone of P

- Close analogy to vector time
Snapshot algorithm

- $P_i$ wants to request a global snapshot
- $P_i$ fixes a time $s = V_i + (0,\ldots,0,1,0,\ldots,0)$
- $P_i$ broadcasts $s$ to all other processes and freezes until it knows that all other processes know $s$
- $P_i$ „ticks“ again, takes a local snapshot and broadcasts a dummy message, so all processes advance their clocks to some value $\geq s$
- If a process' local clock becomes $\geq s$, it takes a local snapshot and sends it to $P_i$
Snapshot algorithm (contd.)

- The algorithm can be made much simpler and more efficient
  - External process
  - No need for whole vectors to be sent
Part 3: Detecting Causal Relationships in Distributed Computations

- In Search of The Holy Grail
- Debugging
- Consistent recovery
- Detecting deadlocks
Causal History

- Assign complete history to each event
  - Too expensive
- Can be reduced to vector time
  - Lamport time does not characterize causality
Efficient Vector time

- Attaching vector time to each message is unacceptable
  - Vector timestamps can become large
- Typically, only a few processes communicate directly
Efficient Vector time (contd.)

- Store LS (last sent) and LU (last update)

- FIFO is required
The size of vector time

- Unfortunately, causal order is in general of order N
- Application of vector time is substantially limited
Realizations for Offline Analysis

- Depth-first search algorithm to get complete causal history
  - Each event has at most 2 direct predecessors
- Store *direct* dependencies of each event

- Breadth-first search to get vector time
Concurrency Regions

- Regions of events which share the same causal past and future

- Characterizing causality is reducible to characterizing concurrency
Global predicates

- Important for debugging
- Not all observers of a computation establish the truth for a given predicate
Observers

- Report every event to an external observer
- Use causal delivery protocol
  - Preserves causality relation
- All observations are valid
- One observer may claim that a predicate has been established while another claims that the predicate wasn't satisfied during the computation
Observers (contd.)

First observer’s view of the original computation

Observation 1

Original computation

Observation 2

Second observer’s view of the original computation
Possibly and Definitely

- **Possibly:** There is an instant in an observation at which the predicate holds
- **Definitely:** In every complete observation there is an instant at which the predicate holds

- **Stable predicates:** A predicate which eventually in every observation
Detecting definitely

- Based on vector time
- Compute the set $A_i$ of intersection points of level $i$ for each level; $i \in \{0...l\}$, $l=|E|$
  - All intersection points in $A_{k-1}$ are accessible by a path on which the predicate is never satisfied on lower levels
- If $A_l$ is empty, the predicate definitely holds

- Similar for Possibly

- Costly
More efficient algorithms

- Decomposable predicates are easier to detect
- Establish parts of the global predicate
- Go into one direction until parts of the predicate are satisfied
Computation replay

- Record non-deterministic events
- Replay with recorded decisions
Currently

- Evaluate predicate on the real-time order of the events
  - Must use a powerful observer
  - Intrusive: block at each invalidating event

- Might miss some predicates
Behavioral patterns

- Classes of events, each event belongs to a class
- Combine classes to patterns: A happens between B and C

- What timestamps should be assigned to combined events?
  - (A||B) → C
Conclusion

- Hierarchy of computation types
- Vector time is interesting
  - limited application
- Detection requires much effort
Questions?