Timestamping Messages in Synchronous Computations

Vijay K. Garg
garg@ece.utexas.edu

Chakarat Skawratananond
skawrata@ece.utexas.edu

Parallel and Distributed Systems Laboratory
The University of Texas at Austin
Austin, TX 78712

July 5, 2002

Causality Tracking

- Lamport's *happened-before* relation [Lamport78] captures potential causal relationship in distributed systems.
- Definition: e *happened before* f (denoted by $\rightarrow$)
  - e occurs before f in the same process, or
  - there is a transfer of information from e to f, or
  - there exists an event g such that $e \rightarrow g$ and $g \rightarrow f$.
- Causality tracking requires us to timestamp events such that *happened before* relationship can be determined between events.

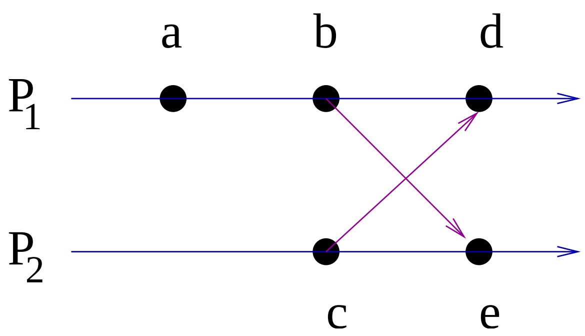
Causality Tracking: Applications

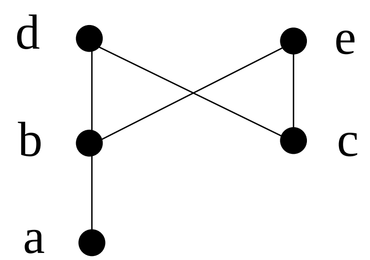
Determining causal relationship is essential in distributed computing.

- Debugging Distributed Systems.
  - Visualization of the computation.
    - POET [Kunz et al.'97]
    - XPVM [Kohl-Geist95]
    - Object-Level Trace [IBM]
  - Predicate Detection.
    - e.g. [Fidge89,Mattern89,Garg-Waldecker94].
- Fault-tolerance.
  - Determining *orphan* processes. e.g. [Strom-Yemini88,Damani-Garg96].

Model of Distributed Computations

Distributed Computation: A partial ordered set (poset) $(X, \rightarrow)$, where X is the set of events, and $\rightarrow$ is Lamport's *happened-before* relation.





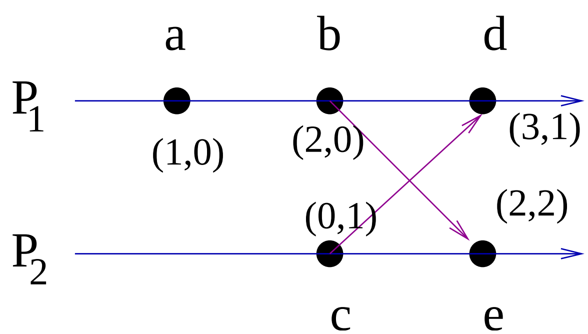
$$X = \{a, b, c, d, e\}$$

Relation $\rightarrow = \{(a,b), (a,d), (a,e), (b,d), (b,e), (c,d), (c,e)\}$

Causality Tracking with Vector Clocks

Fidge and Mattern [Fidge89,Mattern89] introduced vector clocks such that

$$\forall e, f \in X : e \rightarrow f \iff v(e) < v(f)$$



Vector order: componentwise comparison
$(2, 0)$ is less than $(2, 2)$
$(0, 1)$ is incomparable to $(1, 0)$.

Issues of Vector Clocks

Overhead:

- Message overhead: $O(N)$.
- Space overhead: $O(N)$.

where N is the number of processes in the computation.

Characteristics:

- *Online* algorithm.
- requires knowledge about *the number of processes*.
- *asynchronous* messages.

Talk Outline

- **Online Algorithm.**
  - Background: Synchronous Computation.
  - Algorithm.

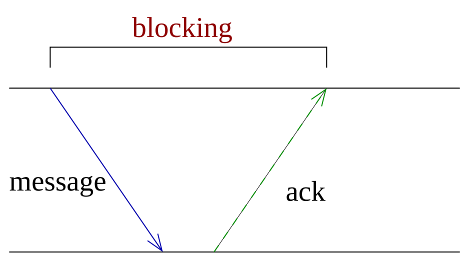
- **Off-line Algorithm.**
  - Background: Dimension Theory.
  - Algorithm.

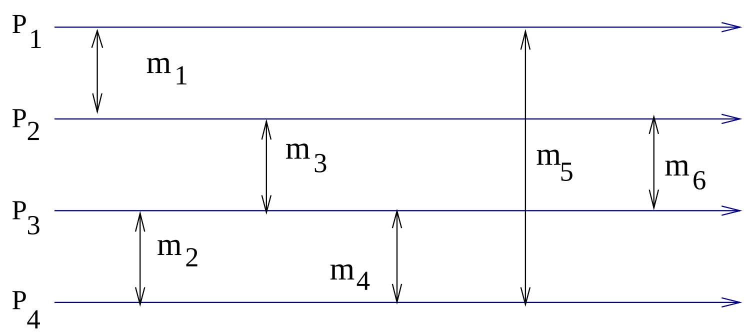
- Summary and Future Work.

Synchronous Computation

Synchronous Computation is the computation that uses only synchronous messages.

A message is called *synchronous* when the sender has to wait for the acknowledgement or a reply from the receiver before executing further.



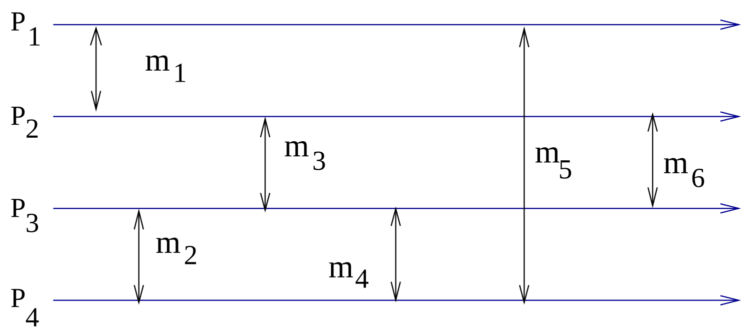


Synchronous communication is widely supported in many programming languages and standards such as CSP, Ada, RPC, and Java RMI.

Message Timestamping

Synchronously-Precede Relation ($\mapsto$): m_1 synchronously precedes m_2 if

- There is a process participating in m_1 and then m_2, or
- If $m_1 \mapsto m_3$ and $m_3 \mapsto m_2$, then $m_1 \mapsto m_2$.



$m_1 \mapsto m_3$, $m_2 \mapsto m_6$, $m_1 || m_2$

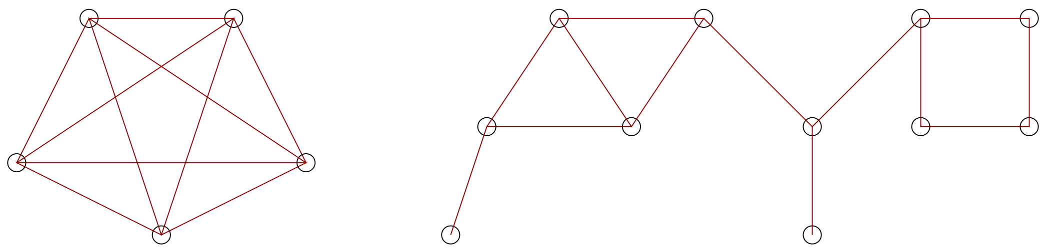
Message Timestamps:

$$m_1 \mapsto m_2 \iff v(m_1) < v(m_2)$$

Problem Statement: Give an algorithm to compute v efficiently.

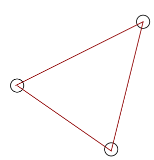
Edge Decomposition

Communication Topology: $(G = (V, E))$

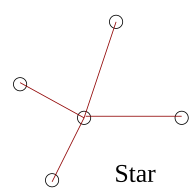


A partition of the edge set, $\{E_1, E_2, \ldots, E_d\}$, is called an **edge decomposition** of G if $E = E_1 \cup E_2 \cup, \ldots, \cup E_d$ such that

- $\forall i, j : E_i \cap E_j = \emptyset$, and
- $\forall i : (V, E_i)$ is either a star or a triangle.

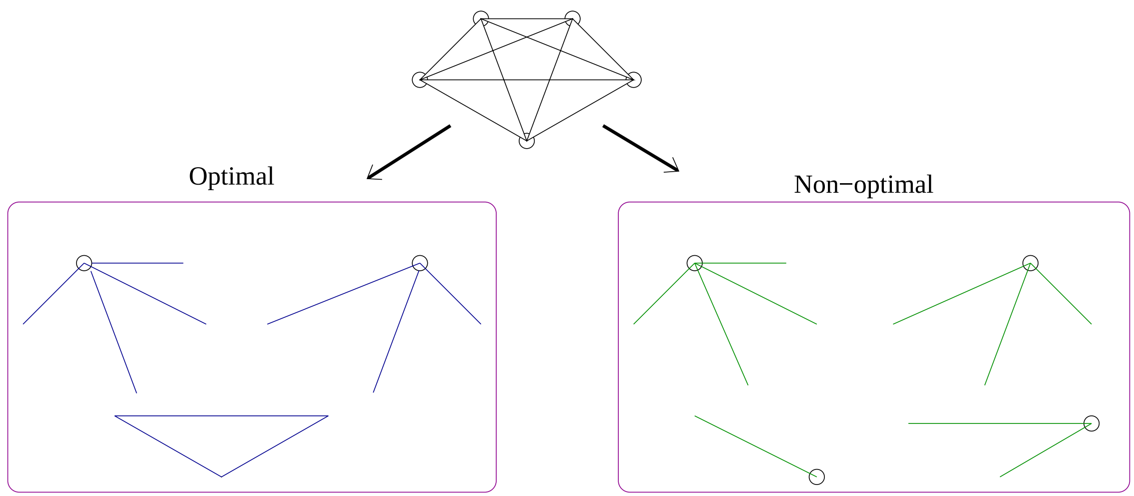


Triangle

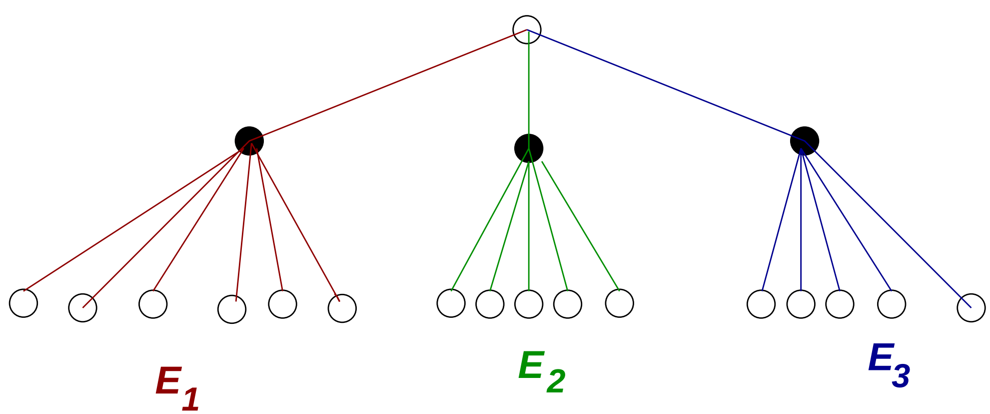


Star

Edge Decomposition: Examples



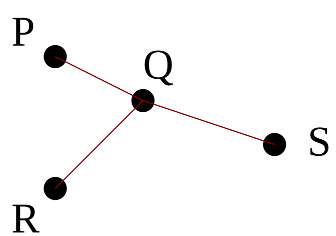
Two of the possible decompositions.

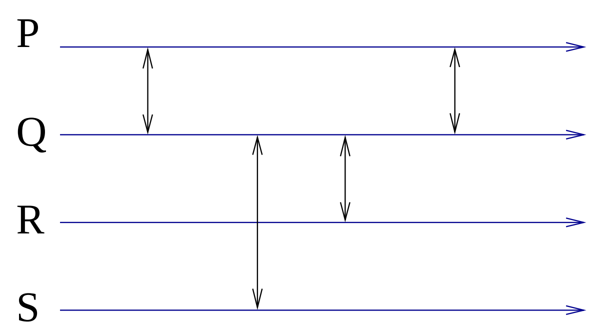


Properties of Star and Triangle

Lemma: Messages in synchronous computations on *star* or *triangle* topologies are always totally ordered.

Concurrent messages must belong to different edge groups



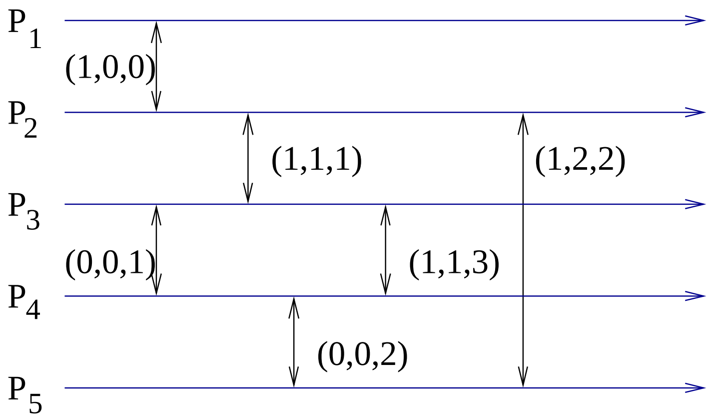


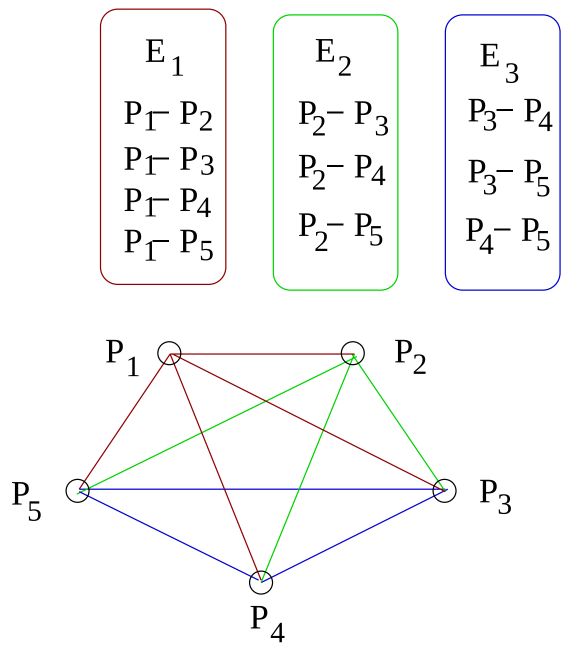
Online Algorithm

Each process maintains a vector of size d (size of edge decomposition) initially 0 vector.

- Sender and Receiver exchange vector clocks.
- Take component-wise maximum.
- Both increment the *component corresponding to the edge group* of the edge connecting the sender and receiver.

Online Algorithm: Example





Determining Optimal Edge Decomposition

A **vertex cover** of an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ such that if (u, v) is an edge of G, then either $u \in V'$ or $v \in V'$ (or both)

If only *stars* are allowed in Edge Decomposition, then the problem becomes Vertex Cover Problem.

Vertex Cover Problem is *NP-Hard* [Garey-Johnson79].

Approximation Algorithm Idea

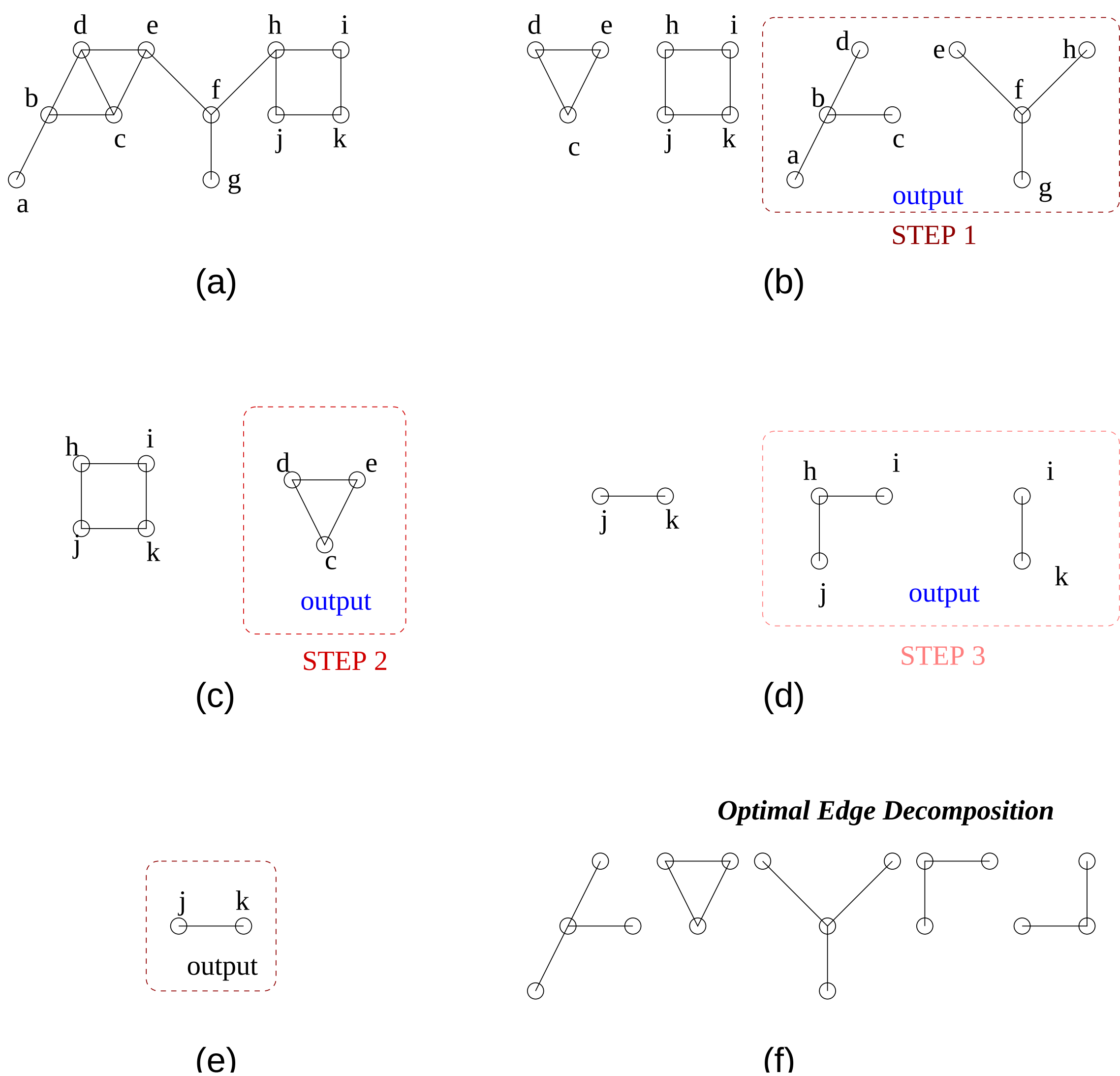
Repeatedly apply the following three steps:

Star Removal 1:
If x is connected only to y then remove star rooted at y.

Triangle Removal:
Remove any triangle (x, y, z) with $degree(x) = degree(y) = 2$

Star Removal 2:
Let (x, y) be the edge of with largest number of edges adjacent to it Remove stars rooted at x and rooted at y.

Approximation Algorithm: Example



Approximation Algorithm

Theorem:
The proposed approximation algorithm produces an edge decomposition which is at most twice the size of the optimal edge decomposition.

Theorem:
If the topology is an acyclic graph, the algorithm produces an optimal edge decomposition.

Time Complexity: $O(|E||V|)$.

Background: Dimension Theory

Extension: A poset (X, Q) is called an *extension* of poset (X, P) iff

$$\forall x, y \in X : \quad (x, y) \in P \Rightarrow (x, y) \in Q$$

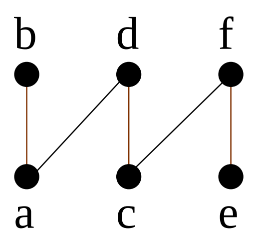
- (X, Q) is called *linear extension* if Q is a total order.

Chain Realizer: A family $\mathcal{R} = \{L_1, L_2, \ldots, L_t\}$ of linear extensions of P is called a *chain realizer* of a poset (X, P) if $P = \cap\mathcal{R}$.

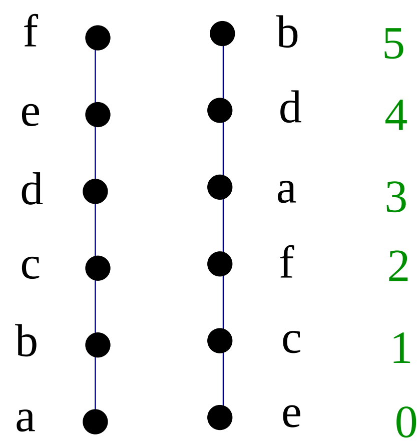
- $x < y \in L_i \cap L_j$ if $x < y$ in both L_i and L_j.

Dimension: the cardinality of the smallest possible chain realizer of (X, P).

Timestamping from Chain Realizer



Poset



Chain Realizer

Each element of the poset can be encoded using a vector of size 2.

$$a = (0,3); b = (1,5); c = (2,1)$$
$$d = (3,4); e = (4,0); f = (5,2)$$

Chain Order:

$$v_x < v_y \iff \forall i : v_x[i] < v_y[i]$$

Off-line Algorithm

$\boldsymbol{width(X, P)}$: the size of the longest *antichain* of (X, P).

Let M be the poset formed by messages in a synchronous computation with N processes. $width(M)$ is at most $\lfloor \frac{N}{2} \rfloor$.

Theorem:
Given a synchronous computation consisting N processes, vector clocks of size at most $\lfloor \frac{N}{2} \rfloor$ can be used to timestamp messages.

Off-line Algorithm (Cont.)

Algorithm:

- construct chain realizer of the poset using Dilworth's algorithm [Dilworth50].
- compute timestamp for each message from the chain realizer.

Note:
$Dimension(M) \leq width(M) \leq N/2$

Note: Calculating dimension of poset is NP-hard [Yannakakis82].

Summary of Online Algorithm

	Fidge and Mattern	**Our Algorithm**
communication	Any	Synchronous
Require knowledge of topology?	No	Yes
Require knowledge of # of processes?	Yes	Not for some topologies
Message/Space Overhead	Number of processes	Size of edge decomp.

For client-server systems, our algorithm requires as many coordinates as the number of servers *independent* of the number of clients.

Future Work

- Complexity of Edge decomposition problem.

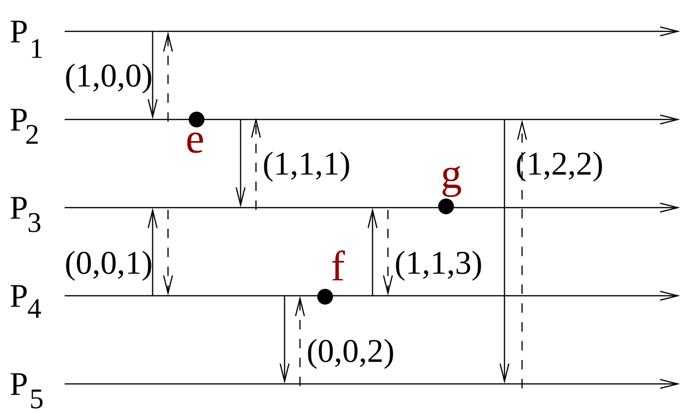
- Efficient algorithm that constructs chain realizer of size less than $width$.

Timestamping Internal Events

Assign to each event e with a tuple $(prec(e), succ(e))$.

- $prec(e)$ is the timestamp of the message immediately prior to e.
- $succ(e)$ is the timestamp of the message immediately after e.

Theorem: $e \rightarrow f \iff succ(e) \leq prev(f)$



e = [(1,0,0),(1,1,1)]

f = [(0,0,2),(1,1,3)]

g = [(1,1,3),(...)]