

Distributed Operating Systems

10th Chapter
Global State

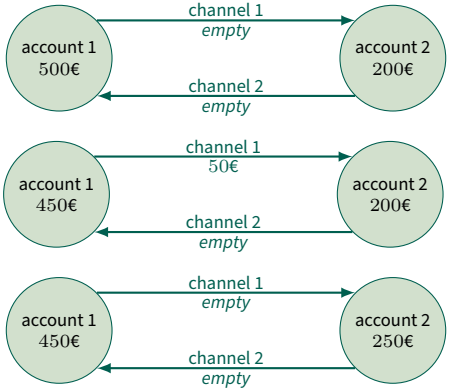
Matthias Werner
Chair Operating Systems

10.1 Motivation

- Sometimes, one needs information on (a part of) the global distributed state of a distributed application
- Problem of **observability**
  - Local states can be observed locally only
  - Communication can be observed by send/receive events, but not directly
- Example: connected bank accounts

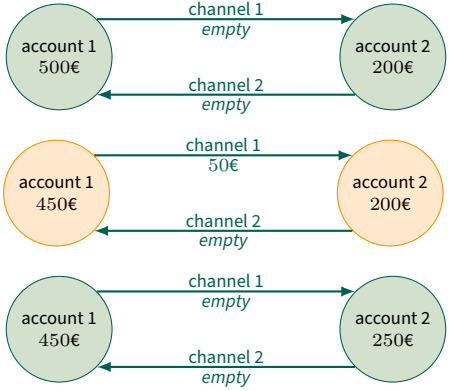
Example: connected bank accounts

- Given: connected accounts where money can always be transferred (between the accounts or from/to an external account)
- What is the **total amount** in all accounts?



Example: connected bank accounts (cont.)

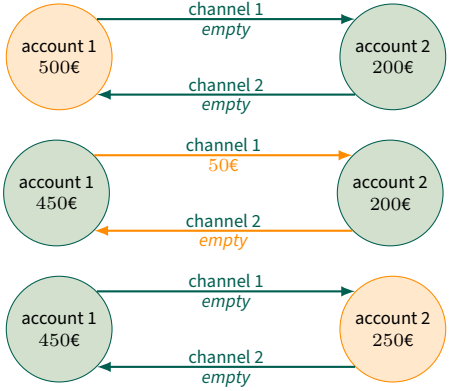
- Case 1:



- **Observation**: $450€ + 200€ = 650€$ ➡ **not correct**
- **Finding**: Concept of channel as object to observe

Example: connected bank accounts (cont.)

- 2. Fall:



- **Observation**: $500€+50€+250€= 800€$ ➡ **not correct**
- **Finding**: consistency is a problem (again 😛)

10.2 Konzepte

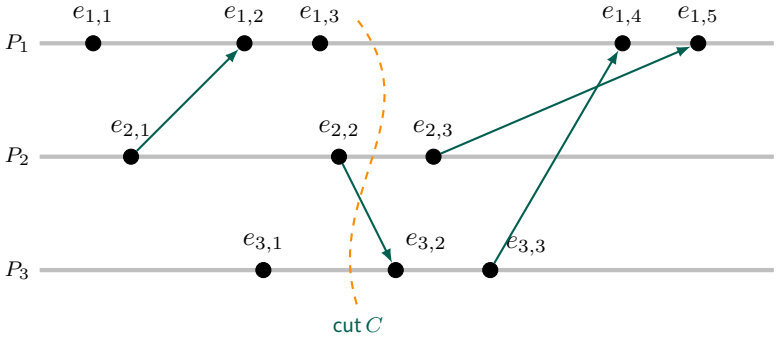
Definitions: History and Cuts

- Given a set of processes $\mathbf{P} = \{P_1, P_2, ..., P_n\}$, where each process runs on another node
- The sequence of **local events** at process P_i is called **history** of P_i: $h_i =< e_{i,1}, e_{i,2}, \dots >$
  - **Note**: Send/receive events are local events too.
- Considering history till a certain point of time k ➡ **prefix** of the history $h_i(k) =< e_{i,1}, e_{i,2}, \dots, e_{i,k} >$
- $s_{i,k}$ is the **state** of process P_i immediately after event $e_{i,k}$
- The **initial state** is called $s_{i,0}$

Definitions: History and Cuts (cont.)

- By union of local histories one gets a **global history** $H = h_1 \cup h_2 \cup \ldots \cup h_n$
- A **cut** C of a global history is the union of local prefixes: $C \stackrel{def}{=} h_1(c_1) \cup h_2(c_2) \cup \ldots \cup h_n(c_n)$
- A cut C corresponds to
  - a **global state** of the system $S(C) = (s_{1,c_1}, s_{2,c_2}, \ldots, s_{n,c_n})$,
  - the (left-side) **boundary** of the cut $E(C) = (e_{1,C_1}, e_{2,C_2}, \ldots, e_{n,C_n})$

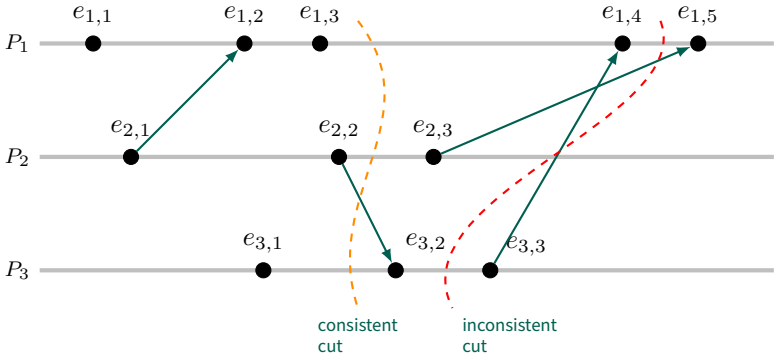
Cut



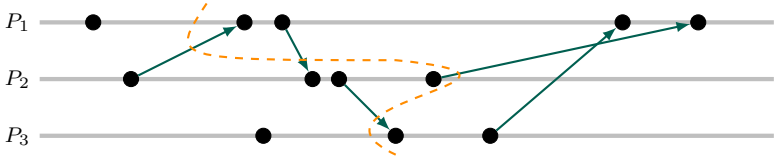
- The cut has the (left-side) boundary $(e_{1,3}, e_{2,2}, e_{3,1})$

Consistent Cut

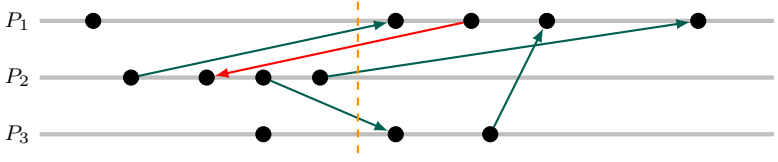
- A cut divides the global history into two parts: the cut C (past) and its complement $H - C$ (future)
- A cut C is **consistent**, if $\forall e \in C, e' \prec e \Rightarrow e' \in C$
- ➡ No causal dependency from the “future”



“Rubber-band Transformation”



- As a test, whether a cut is consistent, the local time axis may be compressed or stretched
- ➡ All arrows have to point into the future (rightwards)
- Example with inconsistent cut



10.3 Termination and Snapshot

- The problem of a consistent global state can be rephrased in terms of the consistent cut

A consistent state is the sum of the partial states at a consistent cut.

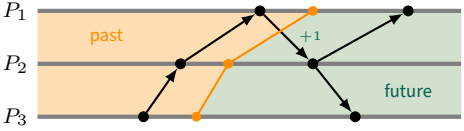
- First, consider “small” state problem: **termination**
- **Termination Problem**
  - Determine, whether all processed are terminated at a consistent cut.
- Based thereon, we will consider the more general **snapshot problem**
- **Snapshot problem**
  - Determination of the local states and the messages on the way at a consistent cut

Termination – Model

- **Activity states**: Distributed components (nodes) of a distributed algorithm may have one of two local states (w.r.t. termination):
  - **active**:
    - Messages may be sent
    - State may change to “terminated” (without an external reason)
  - **terminated**:
    - Can’t send no messages at all
    - A received message triggers a change to the state “active”
  - Following, we use the term “**state**” only
- If the termination detection algortihm (control algorithm) detects at least one active node in the application algorithm, the latter is (obviously) not terminated
  ➡ times of activity can be “condensed” to a single point (**atom model**)
- ➡ Only messages that are still on their way are of interest

Simple Counting

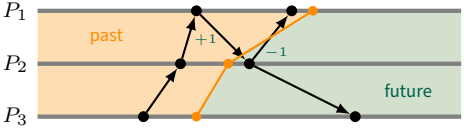
- An “observer” visits consecutively all nodes and sums up all **sent** and **received** messages of the application algorithm (base messages)
- If the sums differ, there are sent messages that are not received yet!
- Thus: if both sums are equal, there is no travelling message any more, and the algorithm is terminated, *isn’t it*?



- observer reports:
  - 3 messages sent
  - 2 messages received
- ➡ $3 \neq 2$, no termination

Simple Counting (cont.)

- Unfortunately, this approach fails, since the conditon is **necessary** but not **sufficient**
- Example: observer reports 3 received messages and 3 sent messages ➡ **phantom termination**
- This is due to messages which are sent in the “future”, but are received in the “past”
  - These messages may compensate the difference



- Beobachter meldet:
  - 3 sent messages
  - 3 received messages
- ➡ $3 = 3$, termination (**not!**)

Solution Approaches

- Avoid messages from the future
  ➡ **freezing** communication in the critical area
- Check the result in a second wave
  ➡ **double counting**
- Distinct Counting
  ➡ **vector method**

Freezing the System

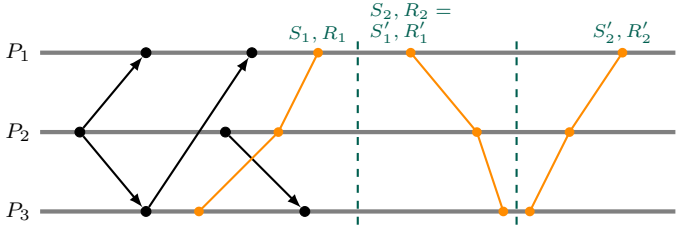
- First wave **freezes** the system
  - No process delivers a (base) message
  - Buffering of messages, considering them “on the way”
  - ➡ No newly send messages in the frozen part of the system
- Second wave **sums up** sent/received messages
  - If both sums are equal, the system has terminated
- The third wave **defrosts** the system

Problem

Approach **strongly** reduces concurrency and steps into the base algorithm!

Double Counting

- The observer visits all nodes **twice** and counts in each round
- If **all four sums** are equal, the base algorithm has terminated: $S_1 = R_1 = S_2 = R_2$
- If no termination is detected: the second round may serve as the first round in the next try

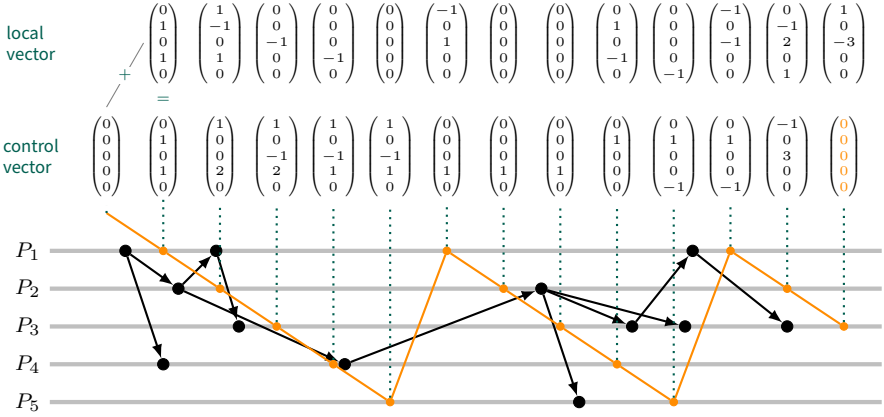


Double Counting: Properties

- Number of control rounds is a priori not bounded by the base algorithms
  - While a very slow base message is on its way, **an arbitrary number** of control rounds may happen
  - Modification: a process that receives a base message without sending a new one, **starts** the next round, **startet** eine neue Runde
- Double counting is **reentrant**
  - Local states are not affected
  - ➡ Multiple initiators may concurrently start termination detections

Vector Method

- Each process P_i maintains a local $\vec{V}_i$ with n components
- $\vec{V}_i$ is initialized with zero vector
- If P_i sends a base message to P_j the component $\vec{V}_i[j]$ is increased by one
- If P_j receives a base message, the componet $\vec{V}_j[j]$ is decreased by one
- The control vector $\vec{C}$ visits consecutively all nodes
  - While visiting a nodes, the local vector $\vec{V}$ is added to $\vec{C}$; then $\vec{V}$ is reset (becomes the zero vector)
  - Termination is detected, if the control vector becomes the **zero vector** wird
  - The counters can't be mocked!


local vector
control vector
+
=
P_1
P_2
P_3
P_4
P_5

Vector Method: Discussion

- **Improvement**
  - If the vector component of the next node is zero, this node may be skipped
  - ➡ Avoid potential useless visits
- **Advantages**
  - Independent of the topology
  - Mostly small number of control messages
  - No change of base message (no counter is appended)
- **Disadvantages**
  - Length of control message $\mathcal{O}(n)$
  - Method is **not** reentrant

Consistent Snapshot

Consistent Snapshot

A consistent snapshot corresponds to the union of states of local nodes **and** channels at a consistent cut..

- Channels can't be observed (directly)
- **But**: in case of termination the channels are empty
- ➡ Any union of local states (without considering the channels) at the time of determined termination is a consistent snapshot
- However, distributed termination algorithms can't be used directly, since
  - snapshots **before termination** may be of interest too
  - some algorithms never terminate
- **But:** Basic ideas are useful!

Freezing the System

- Use of wave algorithm (e.g., Echo)
- First wave freezes the sytem
  - No freezed node sends a message
- Second wave sums up local states
- Third wave defrosts system
- **Problem**: affects base algorithm.



Türken einen Tag unter "Hausarrest" gestellt

ANKARA (dpa). Mehr als 50 Millionen Einwohner der Türkei standen gestern unter "Hausarrest". Weil die Wählerlisten für ein am 6. September anstehendes Referendum überprüft werden sollten, durfte die Bevölkerung den ganzen Tag die Wohnungen nicht verlassen. In der Volksbefragung soll über Fortdauer oder Aufhebung des seit 1980 bestehenden politischen Betätigungs-Verbots für die früheren Ministerpräsidenten Ecevit und Demirel entschieden werden. Während landesweit Zehntausende von Helfern unterwegs waren, um in dem wie ausgestorben wirkenden Land die Eintragungen in die Wählerlisten in den Wohnungen der Wähler zu kontrollieren, attackierten die beiden Oppositionspolitiker die umständliche Methode der Zählung.

Source: F. Mattern, Verteilte Basisalgorithmen

Chandy-Lamport Algorithm (1985)

- **Assumptions**:
  - Directed FIFO channels
  - Connected communication graph (special case: fully connected)
  - Each base message is sent only once (no failure occurs)
  - Communication time and taking of a local snapshot need a bounded time (synchronous)
- **Properties**:
  - Each process may start the algorithm
  - Concurrent activation possible
  - Base algorithm is not affected

CHANDY-LAMPORT Algorithm (1985) (cont.)

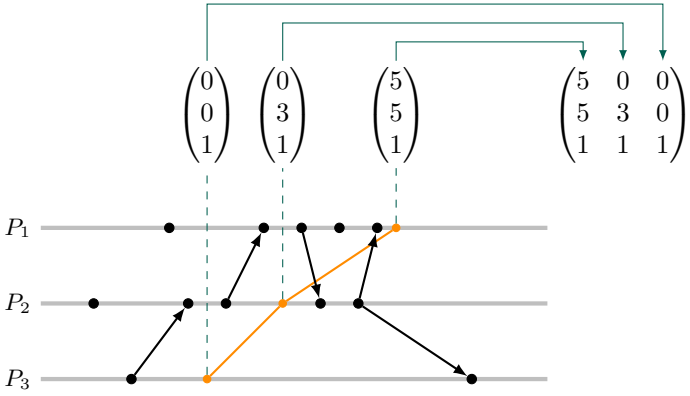
```
I: {NOT marked}:                    // initiator
   S_i := Z_L;                      // take local state
   marked := TRUE;
   DO FOR ALL channel co IN set-of-out-channels
      SEND <Marker> VIA co;
   DONE

 R_i: { message <Marker> is received via channel C}
   IF NOT marked THEN
      marked := TRUE;
      S_i := Z_L;                   // take local state
      S_i,C := {};
      DO FOR ALL channel co IN set-of-out-channels
         SEND <Marker> VIA co;
      DONE
   ELSE
      S_i := S_i + S_i,C
   FI
```

Chandy-Lamport Algorithm (1985) (cont.)

- The markers separate messages **before** and **after** taking the local snapshor
- Algorithm terminates, if all members have received a marker from all input channels
  - Global state: sum of all local states + all input channel's states
  - Collect the local states, e.g., by sending to the initiator
- **Modification**: Local snap shots and receive events are immediately sent to the initiator and the consistent state is constructed at the initiator

Using Vector Time

- Construct a **cut matrix**
- When does a cut matrix represent a consistent cut?



Consistent Cut Matrix

- Let $\mathbf{S}$ be a cut matrix
  - $\mathrm{dia}(\mathbf{S})$ is the **diagonal vector**

    $$\mathrm{dia}(\mathbf{S}) = (s_{1,1}, s_{2,2}, \ldots, s_{n,n})^T$$

    $$\mathbf{S} = \begin{pmatrix} 5 & 0 & 0 \\ 5 & 3 & 0 \\ 1 & 1 & 1 \end{pmatrix} \Rightarrow \mathrm{dia}(\mathbf{S}) = \begin{pmatrix} 5 \\ 3 \\ 1 \end{pmatrix}$$

  - $\sup(\mathbf{S})$ is the **maximum row vector**

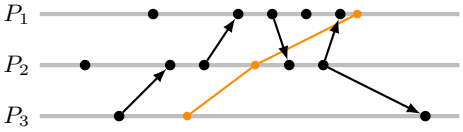
    $$\sup(\mathbf{S}) = (\max(s_{1,1}, \ldots, s_{1,n}), \ldots, \max(s_{n,1}, \ldots, s_{n,n}))^T$$

    $$\mathbf{S} = \begin{pmatrix} 5 & 0 & 0 \\ 5 & 3 & 0 \\ 1 & 1 & 1 \end{pmatrix} \Rightarrow \sup(\mathbf{S}) = \begin{pmatrix} 5 \\ 5 \\ 1 \end{pmatrix}$$

Consistent Cut Matrix (cont.)

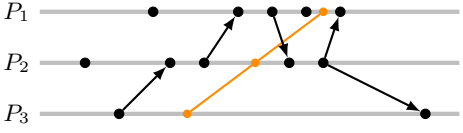
Consistent cut matrix

A cut is consistent if and only if for its cut matrix $\mathbf{S}$ is true that: $\mathrm{dia}(\mathbf{S}) = \sup(\mathbf{S})$.



inconsistent cut

$$\mathrm{dia}(\mathbf{S}) = \begin{pmatrix} 5 \\ 3 \\ 1 \end{pmatrix} \neq \sup(\mathbf{S}) = \begin{pmatrix} 5 \\ 5 \\ 1 \end{pmatrix}$$



consistent cut

$$\mathrm{dia}(\mathbf{S}) = \begin{pmatrix} 4 \\ 3 \\ 1 \end{pmatrix} = \sup(\mathbf{S}) = \begin{pmatrix} 4 \\ 3 \\ 1 \end{pmatrix}$$

From Cut to Snapshot

- How to achieve the consistent cut?
  - **Trivial Approach**:
    Keep requiring vectors and waiting, till $\mathrm{dia}(\mathbf{S}) = \sup(\mathbf{S})$
- What about the messages?
  - **Trivial Approach**:
    Count local send- and receive events. If the sum is equivalent to a consistent cut, then all messages are on their way.
- **Disadvantage**: We may require plenty[1] attempts to get a snapshot

[1] in an extreme case, infinitely many

Matterns Solution (1993)

- Initiator constructs a timestamp s **in the future** and sends it to all nodes
- All nodes remember the timestamp and send an acknowledgement (if local time not smaller then s ➡ negative acknowledgement)
- As soon as the initiator has received a positive acknowledgement from all nodes, it sets its vector time to s and sends a **dummy-message**
- Every node takes local snapshot, immediately before setting its vector time to a value $\geq s$
- The global snapshot is the sum of local snapshots plus all messages that arrive after a local snapshot with a timestamp smaller s

Discussion

- The initiator must not be active till all acknowledgements have arrived
  ➡ base algorithms on initiator are affected
- Termination is not obvious
  ➡ termination test required (e.g. count messages)
- **Optimizations**
  - Initiator as “virtual” process ➡ no freezing
  - If s is sufficiently large, acknowledgements are not necessary

Causal Multicast

- If causal multicast (➡ Chapter 4) is used, we receive consistent steps “automatically”
- Channels need to be observed

- Approach according to Acharya/Badrinath (1992)
  - Every process P_i counts send and receive messages from/to every other process in $\vec{S}_i$ and $\vec{R}_i$, meaning that e.g. a message received by P_j increments $r_{i,j}$
  - Messages with sequence numbers between $r_{j,i} + 1$ and $s_{i,j}$ are transferred in the channel between P_i and P_j

- Approach according to Alagar/Venkatesan (1994)
  - Starting with the local snapshot, “old” messages are logged
  - Initiator sends terminate message to stop logging, as soon as it has received an acknowledgement for the taking of the local snapshots

Literature

[TaSt02] Tanenbaum, A. S. and M. van Steen: *Distributed Systems*. Prentice Hall, 2002 , Abschnitt 6.2

[CDK05] Coulouris, G., J. Dollimore and T. Kindberg: *Distributed Systems: Concepts and Design*. Prentice Hall, 2005, Kapitel 18