Distributed Systems

Distributed State

Prof. Dr. Oliver Hahm

Frankfurt University of Applied Sciences
Faculty 2: Computer Science and Engineering
oliver.hahm@fb2.fra-uas.de
https://teaching.dahahm.de

23.06.2023

Processes

In computer systems two type of processes exist

- stochastic processes[1] and
- deterministic processes

[1] see: https://en.wikipedia.org/wiki/Stochastic_process

Processes

In computer systems two type of processes exist

- stochastic processes[1] and
- deterministic processes

In order to implement programs whose execution results in a deterministic process, the program should be SMART and have the following attributes:

[1] see: https://en.wikipedia.org/wiki/Stochastic_process

Processes

In computer systems two type of processes exist

- stochastic processes[1] and
- deterministic processes

In order to implement programs whose execution results in a deterministic process, the program should be SMART and have the following attributes:

S pecific: The process is defined to fulfill exactly the dedicated case.

[1] see: https://en.wikipedia.org/wiki/Stochastic_process

Processes

In computer systems two type of processes exist

- stochastic processes[1] and
- deterministic processes

In order to implement programs whose execution results in a deterministic process, the program should be SMART and have the following attributes:

S pecific: The process is defined to fulfill exactly the dedicated case.

M easurable: The process provides a well defined impact on its objects.

[1] see: https://en.wikipedia.org/wiki/Stochastic_process

Processes

In computer systems two type of processes exist

- stochastic processes[1] and
- deterministic processes

In order to implement programs whose execution results in a deterministic process, the program should be SMART and have the following attributes:

- S pecific: The process is defined to fulfill exactly the dedicated case.
- M easurable: The process provides a well defined impact on its objects.
- A chievable: The process is able to fulfill its goals given the provided resources.

[1] see: https://en.wikipedia.org/wiki/Stochastic_process

Processes

In computer systems two type of processes exist

- stochastic processes[1] and
- deterministic processes

In order to implement programs whose execution results in a deterministic process, the program should be SMART and have the following attributes:

- S pecific: The process is defined to fulfill exactly the dedicated case.
- M easurable: The process provides a well defined impact on its objects.
- A chievable: The process is able to fulfill its goals given the provided resources.
- R epeatable: The process can be invoked multiple times with the same input and produce the same output.

↪ In the literature instead of Repeatable, you will also find Responsible or even Relevant

[1] see: https://en.wikipedia.org/wiki/Stochastic_process

Processes

In computer systems two type of processes exist

- stochastic processes[1] and
- deterministic processes

In order to implement programs whose execution results in a deterministic process, the program should be SMART and have the following attributes:

- S pecific: The process is defined to fulfill exactly the dedicated case.
- M easurable: The process provides a well defined impact on its objects.
- A chievable: The process is able to fulfill its goals given the provided resources.
- R epeatable: The process can be invoked multiple times with the same input and produce the same output.
- T erminated: Given the same resources the process produces the same results in a determined time frame.

↪ In the literature instead of Repeatable, you will also find Responsible or even Relevant

[1] see: https://en.wikipedia.org/wiki/Stochastic_process

Agenda

- Coordination
- Global State
- Mutual Exclusion

Agenda

- Coordination
- Global State
- Mutual Exclusion

Coordination in the Distributed System

Problem statement:

- Distributed systems consist of objects and dynamic interrelationship between these objects: processes
- Each individual object has a set of attributes and the processes have a state
- Objects and processes are distributed in the system and may be independent from each other or require some kind of coordination.

Coordination and Synchronization

Coordination in the distributed systems allows to make the behavior of the system predictable and interactions causal by **ordering** them. The letter requires the introduction of a 'time line' in the system, which is known as **clock synchronization** among the nodes.

Global states in a Distributed System

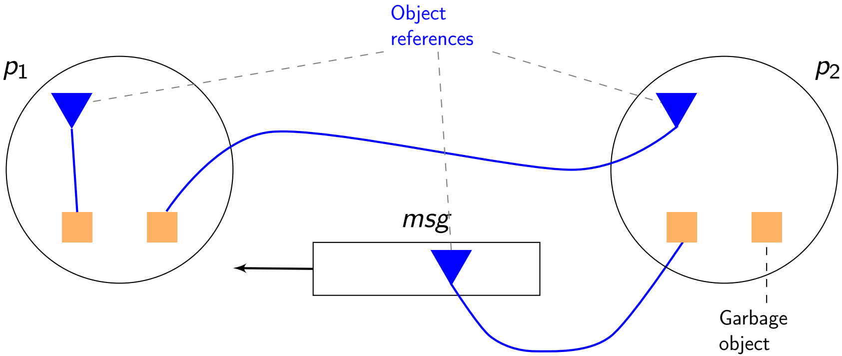
Processes in a distributed systems require synchronization and coordination

- in case a process is accessing shared resources
- the process needs interruption during its operation (*triggered events*).

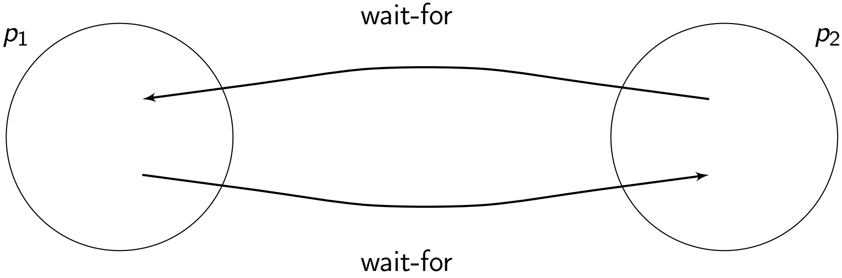
- Different nodes have individual clocks

$\hookrightarrow$ Without a clock and time synchronization processes in a distributed systems may behave erratically and coordination becomes difficult or even infeasible

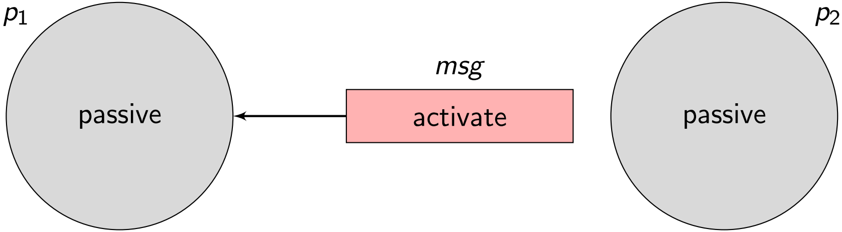
Global Properties: Garbage Collection



Global Properties: Deadlock



Global Properties: Termination



Agenda

- Coordination
- Global State
- Mutual Exclusion

Happened-Before Relation

Problem statement

Is it possible to maintain a global view on the state of system's behavior wrt the **happened-before** relationship?

Happened-Before Relation

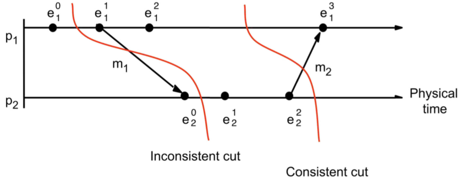
Problem statement

Is it possible to maintain a global view on the state of system's behavior wrt the **happened-before** relationship?

In other words:
If we introduce a cut C (a snapshot), can we guarantee

$$\forall e \in C : f \rightarrow e \Longrightarrow f \in C\ ?$$

$\Rightarrow$ Catching one particular event e, we catch all events **happened-before** f



Source: Coulouris et al, Distributed Systems, Pearson

Consistent Cuts

- A consistent cut requires a consistent global state of the distributed system
- The history of a process i is defined as $h(i) =< e_i^1, e_i^2, e_i^3, \ldots >$
- The global history H is the union of all histories of the involved processes
- A cut is a union of prefixes of process histories
- A run is a total ordering of all the events in a global history that is consistent with each local history's ordering
- A consistent run orders (serializes) the events in the global history H; to be consistent with the happened-before relation ($\rightarrow$) on H.

Global States

Within a distributed system a **Global State** implies the following consistency conditions:

- Assigning a **Global State Predicate** to a distributed system is equivalent of providing a function, that maps the set of Global States to $\{true; false\}$.
- A **Global State** is stable: Once it has reached condition $\{true\}$ and it remains in that state for all states connected to that state.
- **Safety** is an assertion once an undesired state predicate evaluates to $\{false\}$ all other states S reachable from the starting state S_0 are `false` also.
- **Liveness** is an assertion to a desired state predicate to $\{true\}$ all other states reachable from S_o are `true` as well.

Agenda

- Coordination
- Global State
- Mutual Exclusion

Exclusive Resources for a Process

Problem statement:
For a process it might be necessary to have exclusive access to a resource. How this can be accomplished in a distributed system?

Examples:

- A process P wants to write to a file (storage) and has to make sure no other process is reading to that file yielding inconsistencies.
- A database is required to update a cell in a table (exclusive lock).
- A process P wants to remove by means of `rm -r d` the directory d recursively while guaranteeing that no other (remote) process P_j accesses any other file in the underlying directory structure.

We know this problem from the Operating Systems as entering a **critical section**:

Critical Sections

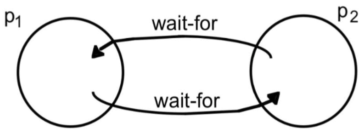
`enter()`	enter critical section – set up blocking
`accessResource()`	access shared resource in critical section
`exit()`	leave critical sections – free resource

Mutual Exclusion: Requirements

A distributed system has to conform to some essential requirements in order to provide **Mutual Exclusive** capabilities:

1. Safety: At most one process p may execute a critical section in a given time interval δt.
2. Liveness: A process p requests to enter the critical section and eventually succeeds.
3. Ordering: Request from processes p_i to enter the critical section follow the **happened-before** relationship.

$\hookrightarrow$ A distributed system not conforming to these requirements will experience deadlocks in process handling and eventually stalling of execution.



Source: Coulouris et al, Distributed Systems, Pearson

Mutual Exclusion: Solutions

Some possible architectures have been developed to cope with these requirements:

1. We provide a central service (*coordinator*) for resource allocation.
2. Nodes operate entirely decentralized on a peer-to-peer bases; thus not transitive dependencies exist.
3. Nodes operate entirely independent and distributed, without considering any topology dependencies; thus the intrinsic architecture has to guarantee for this.
4. Operations take place in or ordered manner; typically a logical ring; thus access rights are ordered in time (and by node).

Mutual Exclusion: Caveats

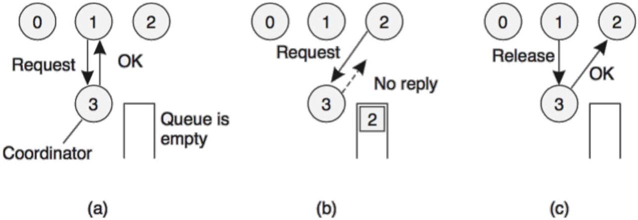
Due to the message (= information) transfer in the distributed system to synchronize activities, **mutual exclusion** is not free of costs:

- Message transfer consumes bandwidth and require processing for `entry()` and `exit()` operations in addition to operating with the resource.
- Operations at the client side to access the resource are significantly delayed.
- Access rates is limited given he concurrent access by clients entering the critical section.
- Throughput is limited by synchronization delay between two processes exiting an entering the critical section.

↪ A good system design require as little mutual exclusions as possible

Solution 1: Central locking

One dedicated node in the distributed system is assigned a coordinator tracking all unsatisfied and pending processes requests P_k in a **Queue**:



Source: Tanenbaum, Van Steen, Distributed Systems, Pearson

Let process 3 be the coordinator. Access to a resource is permitted only in case 3 has provided an Ok message.

(a) Process 1 requests access to a resource. Since no other process wants to access the same resource, coordinator 3 immediately permits this.

(b) Process 2 is asking for the same resource. 3 puts the request for process 2 in the queue, 2 is blocked.

(c) Once process 1 has released the resource and notified 3, 2 is informed about its permissive use.

Solution 2: Decentralized/local locking

In this scenario,

- all resources in the distributed system needs to be replicated n times having its own (local) coordinator,
- access permissions are given via a majority vote $m > n/2$ of local coordinators while
- responses from the local coordinator are given immediately.

Solution 2: Decentralized/local locking

In this scenario,

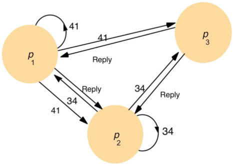
- all resources in the distributed system needs to be replicated n times having its own (local) coordinator,
- access permissions are given via a majority vote $m > n/2$ of local coordinators while
- responses from the local coordinator are given immediately.

Consequences

- Amnesia of a coordinator: If a coordinator crashes it has lost all reported states. Even if the bookkeeping is done persistently, time sync operations are required; thus better scratch the entire state tables.
- Robustness of the distributed system: In order for the system to work, just a little over 50% of the coordinators need to vote – or are available. Assuming the availability of a coordinator processes being 99.9% the probability of a dysfunctional distributed system is extremely small

Solution 3: Mutual exclusion according to Ricart & Agrawala

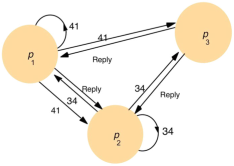
- We consider processes $p_1, p_2, ..., p_n$ providing mutual exclusion by means of
  - unique process identifiers (PID)
  - inter-process communication (IPC) between processes, and
  - Lamport clocks attached to each message.



Source: Coulouris et al, Distributed Systems, Pearson

Solution 3: Mutual exclusion according to Ricart & Agrawala

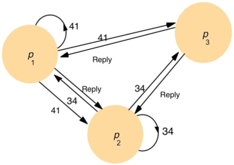
- We consider processes $p_1, p_2, ..., p_n$ providing mutual exclusion by means of
  - unique process identifiers (PID)
  - inter-process communication (IPC) between processes, and
  - Lamport clocks attached to each message.
- A process states can be:
  - `released()`: outside the critical section
  - `wanted()`: trying to enter the critical section
  - `accessed()`: process is within the critical section



Source: Coulouris et al, Distributed Systems, Pearson

Solution 3: Mutual exclusion according to Ricart & Agrawala

- We consider processes $p_1, p_2, ..., p_n$ providing mutual exclusion by means of
  - unique process identifiers (PID)
  - inter-process communication (IPC) between processes, and
  - Lamport clocks attached to each message.



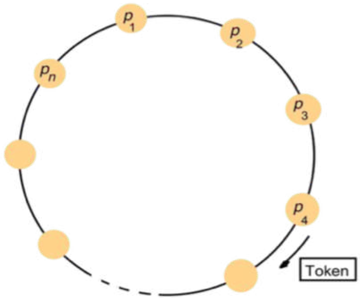
Source: Coulouris et al, Distributed Systems, Pearson

- A process states can be:
  - `released()`: outside the critical section
  - `wanted()`: trying to enter the critical section
  - `accessed()`: process is within the critical section
- A process in state `released()` immediately answers requests
- A process in state `accessed()` is blocked and does not reply to messages
- If more than one process is in state `wanted()`, the first one collecting $n-1$ replies is allowed to `accessed()`.

Solution 4: Token Ring based means

Exclusive access to a resource can be provided by possessing a particular message a **Token**:

- Processes needs be be logical ordered in a ring – irrespective of real network.
- A **Token** is passed around, permitting access to a critical section.
- Conditions Safety and Liveness are fulfilled.
- Ordering in time is not achieved and substituted by the logical process order.
- Significant consumption of bandwidth due to **Token** passing for every critical resource.
- Access delay of resources depend on the topology (= number of nodes) for the **Token** passing.



Source: Coulouris et al, Distributed Systems, Pearson

Comparison of Solutions

Solution	**Algorithm**	`#msgs per entry/exit`	Delay entry (in `msg times`)	Caveats
1	centralized	3	2	coordinator crash
2	decentralized	$2mk + m$ $k = 1, 2, \ldots$	$2mk$	Starvation, low efficiency
3	distributed	$2 * (n - 1)$	$2 * (n - 1)$	Crash of any process
4	token ring	1 to ∞	0 *to* $n - 1$	Lost token, process crash

Table: Comparison of solutions for mutual exclusions in Distributed Systems

Important takeaway messages of this chapter

- Coordination in distributed systems is not trivial
- The happened-before relationship is crucial to assess the global state of a distributed system
- Different ways for mutual exclusion in distributed systems exist – each with its individual benefits and drawbacks