Distributed Systems

Global State and Synchronization

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

Agenda

1. Coordination
2. Global State
3. Mutual Exclusion

Agenda

1. Coordination
2. Global State
3. 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 an processes are distributed in the system and may be independent from each other or require some kind of co-ordination.

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.

Processes

In computer systems two type of processes exist

- stochastic processes[1] and
- SMART processes

SMART processes can be realized as program having 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 it's objects.
- A chievable: The process is able to fulfill it's goals given the provided resources.
- R epeatable: The process can be used/invoked more often.
- 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`

Global states in a Distributed System

Hence, in distributed systems consist of distributed processes which require to be synchronized and coordinated

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

- In distributed systems nodes have individual clocks

→ There is no trivial common understanding what time means and how to express this

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

Agenda

1 Coordination

2 Global State

3 Mutual Exclusion

Happened-Before Relation

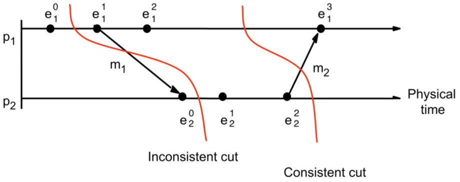
Problem statement

Is it possible to maintain a global view on the state of system's behavior thus we have consistency what did happened-before?

Thus, if we introduce a cut C (a snapshot), can we guarantee

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

which means: Catching one particular event e, we catch all events happened-before f[2]



---

[2] $\rightarrow$ is the happened-before operator; $\rightarrow f$

Consistent Cuts

- A consistent cut requires a consistent global state of the distributed system
- Ordering all events in a global history ($\rightarrow i_e \forall_{i=1,\ldots,n}$) can be considered as run
- 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

1 Coordination

2 Global State

3 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

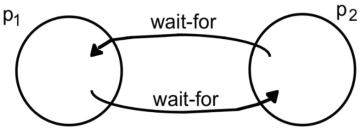
`enter()`	enter critical section – set up blocking
`accessResource()`	access shared resource in critical section
`exit()`	leave critical sections – free resource

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.



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**:

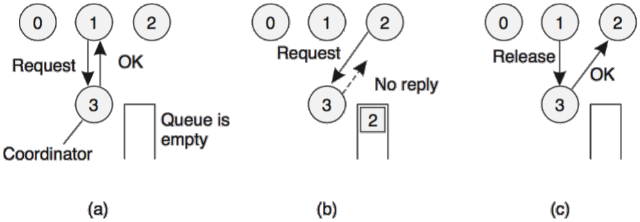


Figure: Centralized locking of resources [Tanenbaum]

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 resource. Since no other process is asking for permission, coordinator 3 immediately permits this.

(b) Process 2 is asking for the same resource. Rather for sending a 'disallow' the coordinator 3 put the request for process 2 in the queue.

(c) Once process 1 has released the resource and notified 3, 2 is informed about it's 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.

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 (perhaps out-of-band) to each other,
  - attaching **Lamport** clocks 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

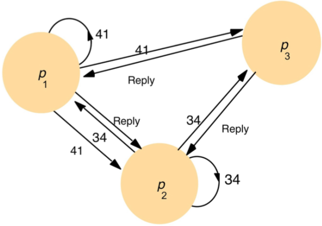


Figure: Mutual exclusion using Ricart & Agrawala algorithm with Lamport's clock [Coulouris]

- 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.

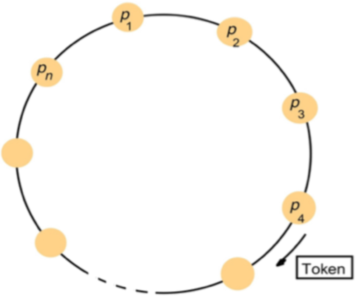


Figure: Mutual exclusion using Token passing [Coulouris]

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