Distributed Algorithms

▼ Leader Election Algorithms (motivation, examples for logical ring and tree).

- Short intro https://youtu.be/TzwiGTbUSHg?si=wL4qgs_kUSk2tjKn
- To explore // Ring leader election in detail https://youtu.be/w_-SX4vR53M?si=YHq6lLtFvq171TFV // https://acrobat.adobe.com/id/urn:aaid:sc:AP:60aad5c1-74c1-47c6-9474-8b93fe0b74e2

Leader election algorithms are crucial in distributed systems where multiple nodes need to coordinate and make decisions collectively. The primary motivation behind leader election is to establish a single node as the leader or coordinator among a group of nodes. This leader node can then take charge of tasks such as coordinating distributed computations, managing resources, or serving as a centralized point of control.

Here are examples of leader election algorithms using logical ring and tree topologies:

1. **Logical Ring Topology**:
    - In a logical ring topology, nodes are arranged in a circular manner, where each node has knowledge of its neighbors.
    - A common leader election algorithm for a logical ring is the **Bully Algorithm**:
        - When a node detects that the current leader has failed or is unavailable, it initiates an election by sending an "election" message to all nodes with higher IDs.
        - If no higher-ID nodes respond, the initiating node declares itself as the leader.
        - Otherwise, the node with the highest ID among the responders becomes the leader, and the election process continues.
        - The algorithm ensures that eventually, the node with the highest ID becomes the leader.

Example:
Imagine a set of distributed nodes with unique IDs arranged in a logical ring. Node A detects that the current leader (Node L) has failed. Node A starts an election by sending "election" messages to nodes with higher IDs (Nodes B, C, D). Node D, with the highest ID among the responders, becomes the new leader after all lower-ID nodes acknowledge its leadership.

2. **Tree Topology**:
    - In a tree topology, nodes are organized hierarchically, with a root node and child nodes branching out.
    - An example of a leader election algorithm for a tree topology is the **Depth-First Search (DFS) Algorithm**:
        - Each node initiates a DFS traversal of the tree, visiting its children recursively.
        - When a node completes its traversal, it compares its ID with the IDs of nodes encountered during the traversal.
        - The node with the highest ID among those encountered becomes the leader.
        - This information is propagated upward toward the root node, ensuring that the highest-ID node is established as the leader.

    Example:
    Consider a distributed system organized in a tree structure with Node R as the root. Each node initiates a DFS traversal, visiting its children and passing along the highest-ID node encountered. Eventually, the information reaches the root node, and Node D, with the highest ID, is elected as the leader. This decision propagates down the tree, ensuring that all nodes recognize Node D as the leader.

▼ Time in distributed systems, problem definition (physical vs logical time), Lamport’s clock. Christians and Berkeley algorithm.

▼ Time in distributed systems, problem definition (physical vs logical time)

In distributed systems, time management is crucial for coordinating actions, maintaining consistency, and ensuring correctness across multiple nodes or processes. The problem of time in distributed systems can be defined in terms of both physical time and logical time:

1. **Physical Time**:
    - Physical time refers to the real-world time as measured by clocks in the system.
    - In a distributed system, each node typically has its own local clock, which may not be perfectly synchronized with clocks on other nodes due to network delays, clock drift, or other factors.
    - The problem with physical time in distributed systems is ensuring consistency and coordination among nodes despite differences in clock readings.
    - Achieving perfect synchronization of clocks across distributed nodes is practically impossible due to the limitations of network communication and clock accuracy.
2. **Logical Time**:
    - Logical time provides a means of ordering events within a distributed system independently of physical clocks.
    - Instead of relying on real-world time, logical time assigns a logical timestamp or ordering to events based on causality relationships.
    - Events are ordered in such a way that if event A causally precedes event B, the logical timestamp of A is less than that of B.
    - Logical time allows for reasoning about the order of events even when physical clocks are not perfectly synchronized.
    - Various algorithms, such as Lamport timestamps and vector clocks, are used to implement logical time in distributed systems.

Problem Definition:
The problem of time in distributed systems involves managing time-related issues to ensure correct and consistent behavior across multiple nodes or

processes. This problem can be framed in the context of both physical time and logical time:

1. Physical Time Problem:
    - The challenge is to maintain consistency and coordination among distributed nodes despite differences in their local clock readings.
    - Ensuring accurate timestamps for events and transactions across nodes while accounting for clock drift and network delays.
    - Dealing with clock synchronization issues and determining the order of events when clocks are not perfectly synchronized.
2. Logical Time Problem:
    - The goal is to establish a consistent and meaningful ordering of events based on causality relationships rather than physical time.
    - Ensuring that events are correctly ordered according to their causal dependencies to maintain system correctness.
    - Implementing algorithms for logical time management, such as Lamport timestamps or vector clocks, and handling concurrency and distributed communication effectively.

In summary, the problem of time in distributed systems revolves around managing time-related issues, whether in the form of physical clock synchronization or logical event ordering, to ensure correct and consistent behavior across distributed nodes or processes.

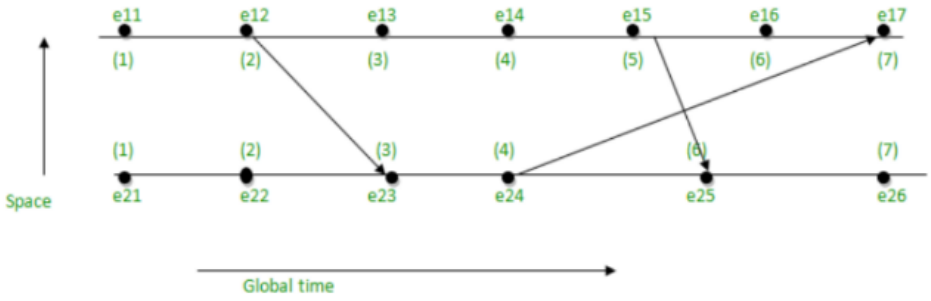
▼ Lamport’s clock

Used to order events in a distributed system
Assigns a logical timestamp to each event, which is used to order the events in
a consistent and meaningful way.

- **Process:** P_i
- **Event:** E_{ij}, where i is the process in number and j: j^{th} event in the i^{th} **process**.
- t_m: vector time span for message m.
- C_i vector clock associated with process P_i, the j^{th} element is **Ci[j]** and contains P_i**'s** latest value for the current time in process P_j.
- **d:** drift time, generally d is 1.

between C_j and $T_m + d$.

- e17 = max(7, 5) = 7, [e16 + d = 6 + 1 = 7, e24 + d = 4 + 1 = 5, maximum among 7 and 5 is 7]
- e23 = max(3, 3) = 3, [e22 + d = 2 + 1 = 3, e12 + d = 2 + 1 = 3, maximum among 3 and 3 is 3]
- e25 = max(5, 6) = 6, [e24 + 1 = 4 + 1 = 5, e15 + d = 5 + 1 = 6, maximum among 5 and 6 is 6]



▼ Christians

https://www.geeksforgeeks.org/cristians-algorithm/

▼ Berkeley

https://www.geeksforgeeks.org/berkeleys-algorithm/?ref=lbp

▼ Mutual exclusion algorithms, problem definition, Algorithms Lamport, Ricart-Agrawala.

▼ Mutual exclusion

It refers to the property of ensuring that only one process or thread can access a particular resource or critical section of code at any given time. The purpose of enforcing mutual exclusion is to prevent concurrent access to shared resources that could lead to data inconsistency or race conditions.

The problem arises when multiple processes or threads need to access a shared resource simultaneously. Without proper synchronization mechanisms in place, such as mutual exclusion, these processes or threads may interfere with each other's operations, leading to unpredictable behavior and potential data corruption.

▼ Lamport

Lamport's bakery algorithm is a mutual exclusion algorithm that uses a shared
variable, called a "ticket," to assign a unique number to each process that wants to enter the critical section. The process with the lowest ticket number
is allowed to enter the critical section, while the others must wait.

▼ Ricart Agrawala

https://www.geeksforgeeks.org/ricart-agrawala-algorithm-in-mutual-exclusion-in-distributed-system/?ref=gcse

▼ Deadlock detection in distributed systems, problem definition (based on resource allocation graph), Lomet algorithms, Chandy-Misra-Haas algorithm.

▼ Deadlock

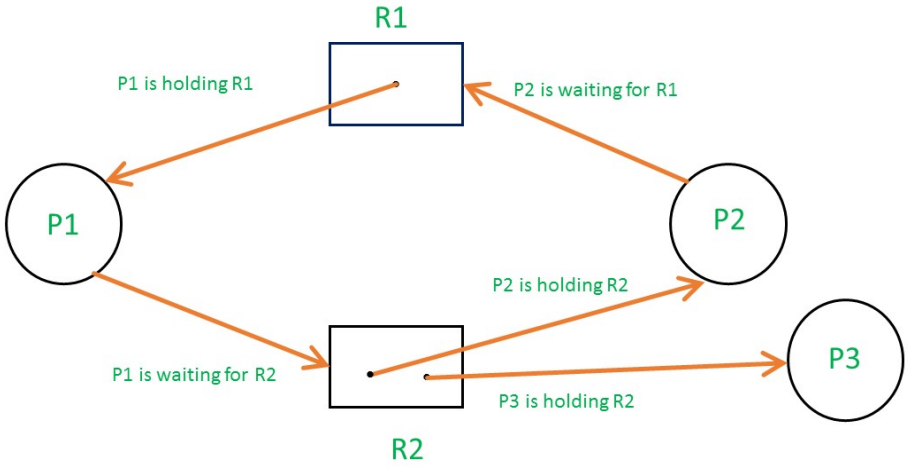
Deadlock detection in distributed systems involves identifying and resolving situations where multiple processes or nodes are waiting indefinitely for resources held by each other, resulting in a deadlock state.

▼ Problem definition

In a distributed system consisting of multiple processes or nodes and a set of shared resources, the occurrence of a deadlock can disrupt system operations and prevent progress. A deadlock arises when each process or node in the system is holding resources while simultaneously waiting for additional resources held by other processes or nodes. This circular waiting pattern creates a state where no process or node can proceed, leading to a deadlock.

To represent resource allocation and requests in the distributed system, we use a resource allocation graph. In this graph:

- Nodes represent processes or nodes in the distributed system.
- Resource nodes represent the shared resources available in the system.
- Directed edges from a process or node to a resource node indicate that the process or node is currently holding that resource.
- Directed edges from a resource node to a process or node indicate that the process or node has requested that resource.
-



MULTI INSTANCES WITHOUT DEADLOCK

▼ Lomet(priori)

1. Create a wait-for-graph with vertices representing processes and edges
   representing resource dependencies.
2. Initialize all vertices as white (unvisited).
3. For each vertex in the graph, perform a depth-first search (DFS) starting
   from that vertex.
4. If a DFS cycle is found, mark all vertices in the cycle as black (deadlocked)

Solution - Rollback transaction

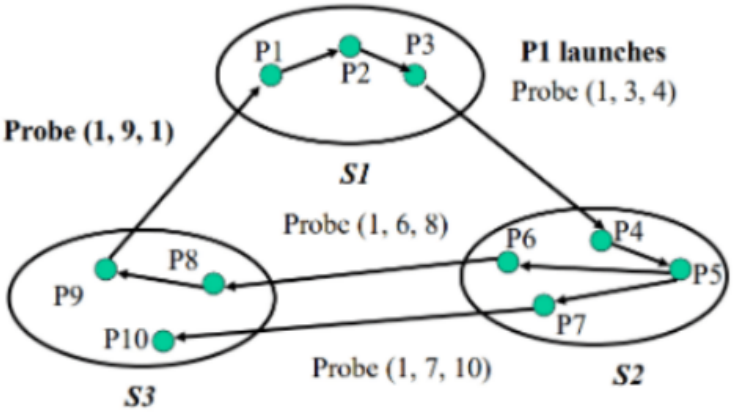
▼ Chandy-Misra-Haas algorithm.(posteriori)

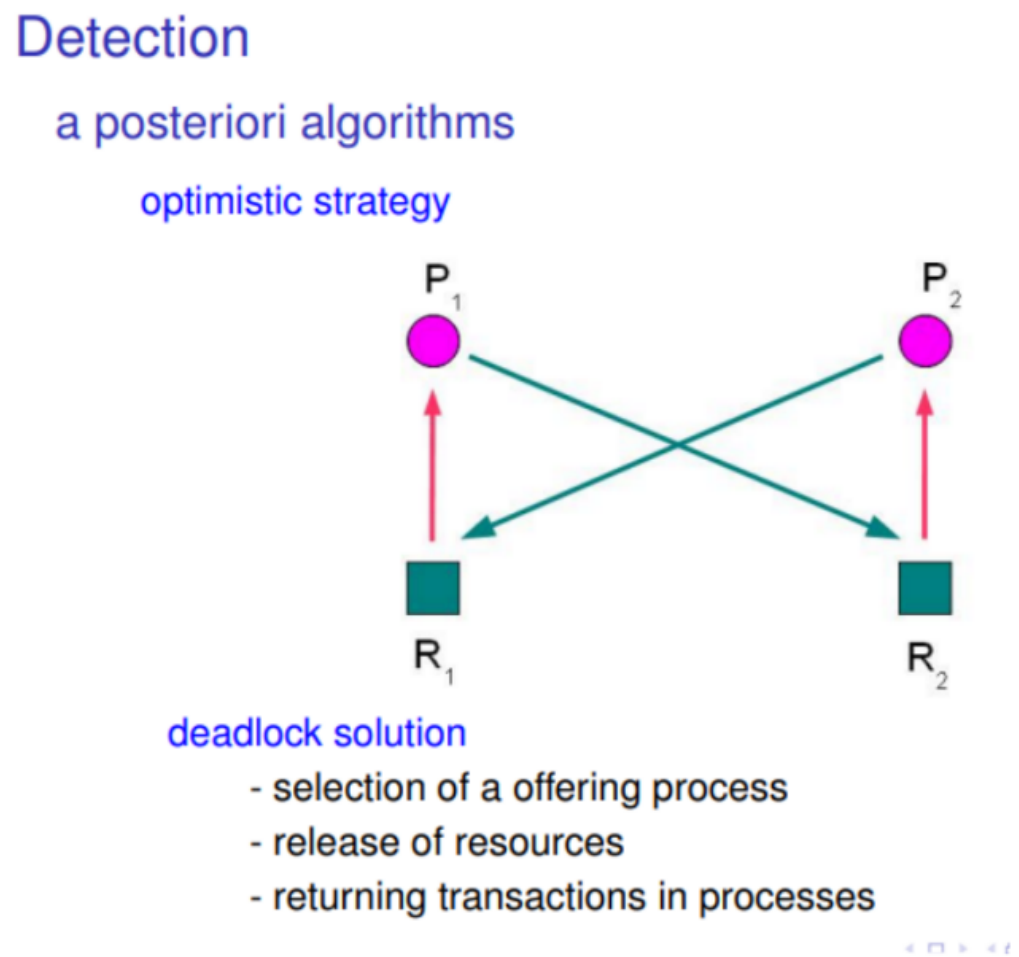
Detecting and resolving deadlocks in communication systems

The algorithm works as follows:

1. When a process detects a deadlock, it sends a "Marker" message to all of
   its neighbors.
2. When a process receives a "Marker" message, it marks all of its outgoing
   channels as "blocked" and sends a "Marker" message to all of its neighbors.
3. The process then takes a snapshot of its current state and sends a "Record" message to the process that sent the "Marker" message.
4. When all "Record" messages have been received, the process that initiated the "Marker" message can use the snapshot to identify the deadlock and resolve it.
   The main disadvantage of the CMH algorithm is that it requires a significant
   amount of communication between the processes, which can cause a significant increase in network traffic. Additionally, it can be computationally
   expensive and time-consuming to analyze the snapshot and identify the
   deadlock.
   It uses the information gathered in the snapshot to determine which process(es) should release their resources to break the deadlock.
   A knot in CMH algorithm refers to a group of processes that are involved in a
   deadlock.

- Probe (start, curr, next) - if start = next then cycle
- sol - resource release, returning transaction , selection of offering
- sends snapshot to parent

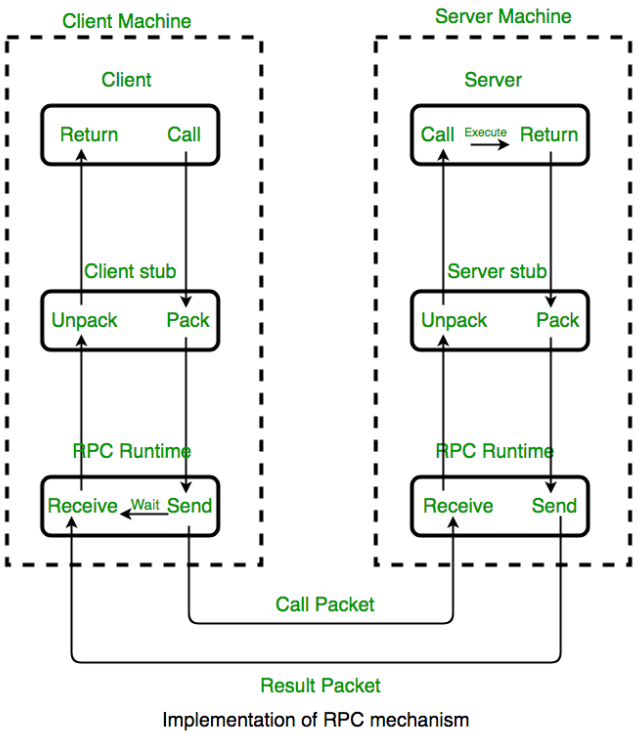




▼ Remote Procedure Calls, motivation, principles. Google RPC, motivation, **types of services, message and methods specification base dom Protocol Buffers.**

▼ RPC

1. Technique for constructing distributed, client server-based applications.
2. Based on extending conventional local procedure calling so that called procedure need not exist in the same address space as the calling procedure.
3. Processes may be on the same or different system with a network connecting them.



Implementation of RPC mechanism

Diagram explanation:

1. Client invokes client stub (CS) procedure. Client stub resides within the client's own address.
2. CS marshalls parameters into a message. Convert params into standard format and copy into message.
3. CS pass message to transport layer, which sends it to remote server machine.
4. On server, transport layer pass message to server stub, demarshalls and calls the server routine using regular procedure call mechanism.
5. when server procedure completes, it returns to server stub, marshalls. Server stub hands the message to the transport layer.
6. TL sends the result message back to client TL, which hands it back to CS.
7. CS demarshalls and returns to the caller.

ADVANTAGES

1. Support process oriented and thread oriented models.
2. Internal message passing is hidden from the user.
3. Effort to re-write and re-develop code is min.
4. Used in distributes and local env.
5. Protocol layers are omitted by RPC to improve performance.

DISADVANTAGES

1. No standard implementation format.
2. No flexibility for hardware architecture. Interaction based.
3. Increase in cost

▼ GRPC

Framework for implementing RPC via http2.

Advantages:

1. Allows to define request and response for rpc and handle everything else for you
2. Modern, fast, efficient
3. Built on http2, low latency
4. Supports streaming and is language independent
5. Easy to plugin, authorization, load balancing, logging, monitoring
6. Open source framework by google
7. Abstraction is easy.

Features

1. Use of http2 instead of http1.1
2. Protocol buffers instead of xml and json in rest.

HTTP1 Limitations

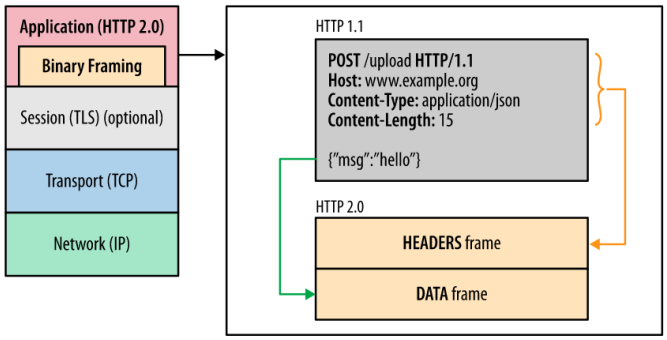
1. Request response protocol
    a. Each connection supports pipelining , but not parallelism (in order only)

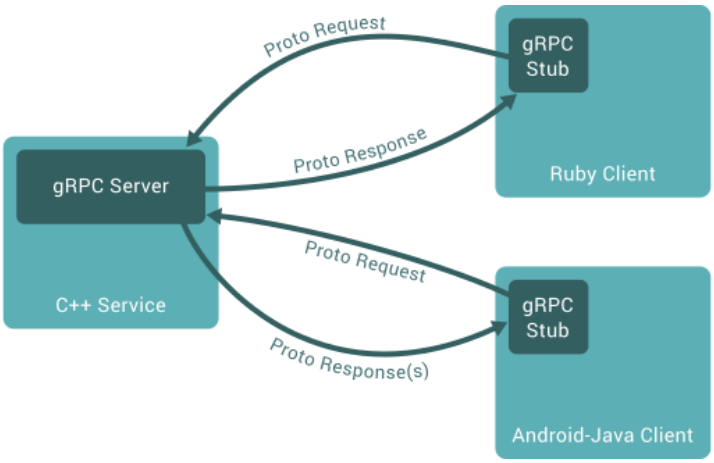
b. Need multiple connections per client server pair to avoid in order stalls across multiple requests → multiple CPU intensive TLS handshakes , higher memory footprint

2. Content may be compressed
    a. headers are in text format
3. Naturally supports single direction streaming

HTTP2 main features

1. One TCP connection for each client server pair Request → Stream
    a. Streams are multiplexed using framing
2. Compact binary framing layer
    a. Prioritisation
    b. Flow control
    c. Server push
3. Header compression
4. Directly supports bidirectional streaming





ARCHITECTURE:

1. Proto files are distributed among all the clients and server.
2. Whenever a client needs some data, grpc stub of the client send a proto request to the grpc server.
3. Grpc server receives the proto request, demarshalls it and sends back the proto response