Distributed Coordination

Hui Chen [a]

[a]CUNY Brooklyn College

November 19, 2025

Need for Coordination

- In distributed systems, multiple processes from multiple hosts often need to coordinate their actions to achieve a common goal.
- Coordination ensures consistency, reliability, and efficiency in distributed applications.
- Examples of coordination tasks include leader election, mutual exclusion, and consensus.

Need for Coordination: Example

Consider a distributed key-value store where Raft consensus algorithm to ensure data consistency across replicas.

- What if we add another layer of coordination to manage configuration changes (e.g., adding/removing nodes)?

Challenges of Distributed Coordination

In a distributed system:

- No fixed coordinator
- No shared memory
- Failure of nodes and networks

Major Role of Coordinator

Coordination is often the problem of:

- deciding who is to decide
- knowing who is alive

Coordination Models

- Interaction model
  - asynchronous or synchronous
- Failure model:
  - Will nodes crash?
  - Will crash nodes return to life?
  - Is crashing the only failure?

Three Aspects/Subproblems

- Mutual exclusion: who is to enter a critical section
- Leader election: who is to be the new leader
- Atomic multicast:
  - which messages
  - which order

Distributed Mutual Exclusion

- Safety - At most, one process may execute in the critical section at a time
- Liveness - A process requesting entry to a critical section is eventually granted it (so long as any process executing in the critical section eventually leaves it)
  - starvation free
  - deadlock free
- Ordering:
  - Entry to the critical section should be granted in "happened-before" order

Distributed Mutual Exclusion Models and Their Evaluation

Algorithm Models:

- Centralized algorithm (dedicated coordination service)
- Ring-based algorithm (token ring)
- Distributed algorithm (distributed coordination service - Coordinator chosen when needed via election from all processes)

Evaluation Criteria:

- Number of messages needed.
- Client delay: time to enter critical section
- Synchronization delay: time between exit and enter

Central Server Algorithm

Server has "token"

- To enter a critical section, a process sends a request message to the server and awaits a reply from it.
  - Reply with token for access
- At time of request, if no other client has access token, server replies immediately, granting the token.
- If token, at time of request, is held by another process, server does not reply. Instead, it queues the request.
- When client (who has token) exits the critical section it sends a message to the server, release the token.

If request queue (queue of waiting processes) is not empty, server chooses the oldest entry in request queue, removes it from queue and replies to the corresponding process with token.

Disadvantages of Central Server Algorithm

Is there any advantage? What are the Disadvantages?

Disadvantages of Central Server Algorithm

Is there any advantage? What are the Disadvantages?

- Server might become a bottleneck
  - Server is critical point of failure
  - Must provide means to select new server.
  - This server must be unique, requires a multicast
- Client might also fail. Failure of client process holding token will cause other clients to be locked out. Will a timeout solve the problem?

Ring-based Algorithm

A token-ring algorithm

- Arrange n processes in a logical ring
- Exclusion is conferred by obtaining a token in the form of a message passed from process to process in one direction around the ring
- Each process must know the identity of the next process in the ring
- On receiving token, if a process does not need the token, it passes it on immediately
- A process that requires the token waits until the token is passed to it, and then retains it.
- To exit the critical section, the process simply passes the token on to its neighbor in the ring.

Ring-based Algorithm: Evaluation

- can take from 1 to $n - 1$ messages to obtain the token;
- token is passed around ring even when nobody needs it; and
- no progress possible if process fails.

Distributed Algorithm

- Implements mutual exclusion based on distributed agreement – instead of using a central server, using Logical Clocks
- Processes that require access to critical section must multicast a request message and can only enter critical section when all other processes have replied – only have token when all other processes reply
- Algorithm assumes:
  - all processes know each other
  - all messages are eventually delivered
  - each process keeps a logical clock
  - Messages form: (T, P) where T is senders timestamp and sender identifier

Distributed Algorithm: Design

- Each process is in one of three states
  - RELEASED - does not have or want the token
  - WANTED - wants but does not have the token
  - HELD - currently has the token
- On initialization
  - state = RELEASED
- To obtain token
  - state = WANTED
  - Multicast request to all processes
  - T = request's timestamp
  - wait until received reply from all other processes
  - then state = HELD

Distributed Algorithm: Evaluation

- Obtaining the token takes 2(n-1) messages.
  - $n-1$ to multicast the request
  - $n-1$ replies
- If hardware support for multicast is available, the obtaining the token requires n messages
- It is fully distributed, however, a very expensive algorithm.
  - Failure of any node makes progress impossible
  - All processes receive and process requests
  - No performance gain over central algorithm

Summary

- Distributed coordination is essential for ensuring consistency and reliability in distributed systems.
- Various models and mechanisms exist to facilitate coordination, each with its own advantages and challenges.
- Understanding the trade-offs between different coordination approaches is crucial for designing effective distributed systems.