Design Distributed System: Processes, Threads, and RPC

Hui Chen [a]

[a]CUNY Brooklyn College

September 10, 2025

Design Space: Process, Threads, and RPC

To design a library like MapReduce, we need to decide how to implement it

- Processes: heavy-weight, but isolated, no shared memory
- Threads: light-weight, share memory, but hard to program
- RPC: remote procedure call, hide network communication

Central Problem

- A client sends requests to N servers in parallel and waits for replies.
- A server processes many simultaneous client requests.

What are the challenges?

Threads

A thread is a “lightweight process” that has its own stack, registers, program counter and shares memory with other threads in same process at a single host

- CPU utilization:

  ```
  A thread may be blocked (request reading disk)
    when serving a request from Client A
  While waiting for client A,
    process a request from client B.
  ```

- Parallel Processing: execute code in parallel on multicore system

Challenges of Threading

With threads, sharing data is convenient, but hard to do safely.

- Race condition
- Coordination between threads
- Deadlock
- Solutions: mutexes, condition variables, channels

Remote Procedure Call (RPC)

Threads are great for concurrency on a single machine; however, execute code on multiple machines?

- A client calls a procedure on a remote server
- The client is blocked until the server returns
- The client and server may be on different machines
- The client and server may be written in different languages

RPC: A key piece of distributed system mechanism – ease client/server communication

- Hide details of network protocols
- Convert data (strings, arrays, maps, etc.) to "wire format"
- Portability / interoperability

Summary

To design a library like MapReduce, we need to decide how to implement it

- Processes: heavy-weight, but isolated, no shared memory
- Threads: light-weight, share memory, but hard to program
- RPC: remote procedure call, hide network communication