Informe sobre el diseño y organización de una Agenda Distribuida

Daniel Angel Arró Moreno, Abel LLerena Domingo
4to Año de Ciencias de la Computación
Universidad de La Habana

January 13, 2025

Introducción

Este informe explora el diseño y la organización de una agenda distribuida, destacando los aspectos clave de su arquitectura, organización, roles, distribución de servicios, comunicación, coordinación, consistencia, tolerancia a fallos y seguridad.

1 Arquitectura del Sistema

El sistema está diseñado con una arquitectura distribuida que permite que el servidor y el cliente operen de manera independiente. Esto significa que ambos pueden evolucionar sin interferir el uno con el otro. Cada nodo en la red tiene la responsabilidad de gestionar una parte del espacio de claves, lo que facilita la distribución y localización eficiente de los datos. Para gestionar la DHT, sugerimos implementar el algoritmo Chord, que es conocido por su eficiencia en la localización de recursos en sistemas distribuidos. Este enfoque garantiza que las operaciones de inserción, búsqueda y eliminación de datos se realicen en tiempo logarítmico, mejorando así la escalabilidad y la tolerancia a fallos.

2 Organización del Sistema Distribuido

El servidor y el cliente están diseñados para funcionar de manera independiente, comunicándose a través de APIs bien definidas. Esto elimina las dependencias estrechas entre ambas partes. Para la organización y gestión de la red, recomendamos el uso del protocolo Kademlia. Este protocolo utiliza métricas XOR para medir la distancia entre nodos, lo que permite una localización y recuperación de datos rápida y eficiente. Su diseño descentralizado se adapta perfectamente al desacoplamiento entre cliente y servidor.

3 Distribución de Servicios en Redes Docker

Los servicios del servidor, como la autenticación y la gestión de datos, se despliegan de manera modular en contenedores Docker. Los clientes interactúan con estos servicios a través de APIs REST desacopladas. Para gestionar las tareas y la distribución de carga entre servicios, recomendamos el uso de CTMQ (Clustered Task Message Queue), que permite al servidor manejar solicitudes de clientes de manera eficiente y en paralelo, distribuyendo la carga entre múltiples instancias.

4 Tipos de Procesos y Organización

Los procesos del servidor están desacoplados y organizados en servicios especializados, mientras que los clientes manejan operaciones locales y sincronizan asíncronamente con el servidor. Sugerimos el uso de colas de mensajes distribuidas y patrones asíncronos para reducir la dependencia temporal entre cliente y servidor, permitiendo operaciones optimizadas incluso en redes con latencia elevada.

5 Comunicación en el Sistema

La comunicación entre cliente y servidor se realiza mediante APIs REST para llamadas sin estado, mientras que entre procesos del servidor se utilizan colas de mensajes asíncronas. Recomendamos el uso de gRPC para operaciones entre módulos del servidor y RabbitMQ para colas de mensajes distribuidas, asegurando así la interoperabilidad entre cliente y servidor, manteniendo independencia y un rendimiento eficiente.

6 Coordinación y Sincronización

El cliente sincroniza los cambios registrados localmente con el servidor utilizando un esquema de sincronización eventual. Para mantener la coherencia en actualizaciones críticas, sugerimos aplicar el algoritmo de exclusión mutua distribuida Ricart-Agrawala y el uso de relojes lógicos para ordenar eventos. Estos algoritmos aseguran que el cliente y el servidor permanezcan consistentes incluso en escenarios de concurrencia elevada.

7 Consistencia y Replicación

Los clientes manejan consistencia eventual con sincronización periódica al servidor, mientras que el servidor utiliza replicación fuerte en su backend. Recomendamos la implementación de replicación basada en quorum y el uso del algoritmo Paxos para garantizar coherencia en la escritura de datos replicados. Este

diseño separa los requerimientos de consistencia en cliente y servidor, logrando un balance entre disponibilidad y rendimiento.

8 Tolerancia a Fallos

El sistema cliente está diseñado para soportar un modo offline, permitiendo que las operaciones continúen sin una conexión constante al servidor. En el servidor, sugerimos el uso de Phi Accrual Failure Detector y estrategias de reintentos exponenciales para la sincronización desde el cliente. Además, proponemos un mecanismo de líder dinámico con IP flotante, utilizando algoritmos como Raft o Zookeeper para gestionar la IP flotante entre nodos. Este mecanismo asegura que siempre haya un nodo líder disponible para gestionar las operaciones críticas, y la IP flotante permite que los clientes se conecten al líder actual sin interrupciones, mejorando la tolerancia a fallos.

9 Seguridad

Las credenciales y los tokens de sesión se generan y gestionan en el servidor, mientras que los clientes operan con autorización basada en tokens. Recomendamos el uso de verificación de firmas JWT (JSON Web Tokens) para proteger la comunicación entre cliente y servidor, evitando la necesidad de almacenar credenciales sensibles en los clientes.