Sistemas Distribuidos

Monica Scheduler

Laura Victoria Riera Pérez
Marié del Valle Reyes

Cuarto año. Ciencias de la Computación.
Facultad de Matemática y Computación, Universidad de La Habana, Cuba

18 de julio de 2023

Repositorio del proyecto

https://github.com/computer-science-crows/monica-scheduler

Implementación de una Agenda Electrónica como Sistema Distribuido con DHT Kademlia

I. Introducción

El tiempo es un recurso invaluable y su gestión eficiente es esencial para la productividad y el bienestar personal. Una estrategia comúnmente utilizada para la gestión del tiempo es el uso de una agenda. Sin embargo, en muchas ocasiones, es necesario coordinar dicha agenda con otras personas para llevar a cabo actividades conjuntas. Este proceso implica la identificación de horarios compartidos y la detección de intervalos de tiempo libres. Además, estas planificaciones pueden verse alteradas por eventos imprevistos que requieren asistencia, lo que conlleva la necesidad de modificar la agenda nuevamente.

Para abordar estos desafíos, este proyecto propone la creación de una agenda electrónica distribuida como herramienta de gestión del tiempo para eventos personales o grupales. El sistema se diseñó e implementó como un sistema distribuido, utilizado la Tabla Hash Distribuida (DHT) de Kademlia para la gestión de datos.

II. Requerimientos

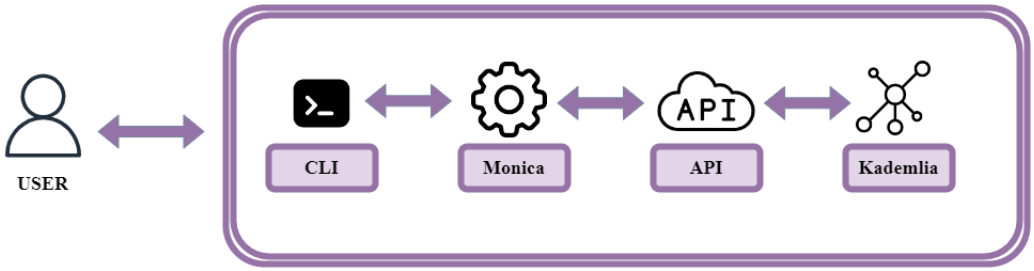
Este proyecto consiste en crear una agenda distribuida como herramienta de gestión del tiempo para eventos personales o grupales. Los requisitos clave para este sistema son:

- **Arquitectura Distribuida**: El sistema debe ser diseñado e implementado como un sistema distribuido. Esto significa que el sistema debe ser capaz de funcionar en múltiples máquinas mientras se presenta a los usuarios como un sistema coherente único.
- **Mecanismo de Autenticación e Identificación**: El sistema debe ser capaz de autenticar e identificar a cada usuario. Esto asegura que solo los usuarios autorizados puedan acceder al sistema y realizar ciertas acciones.
- **Formación de Grupos**: El sistema debe permitir la creación de grupos, ya sean jerárquicos o no jerárquicos. Esto significa que los usuarios deben poder crear y gestionar grupos dentro del sistema de manera flexible.

- **Citas Grupales**: El sistema debe admitir la creación de citas grupales. Si se utiliza un grupo jerárquico, una cita creada por un superior debe aparecer automáticamente en las agendas de todos los miembros del grupo. Para grupos no jerárquicos, todos los miembros deben aceptar la cita para que se confirme.
- **Actualizaciones Automáticas de la Agenda**: Cuando se crea, modifica o elimina una cita, los cambios deben reflejarse automáticamente en las agendas de los usuarios relevantes.
- **Identificación de Conflictos**: El sistema debe ser capaz de identificar conflictos en las agendas locales. Por ejemplo, si un usuario tiene programadas dos citas al mismo tiempo, el sistema debe señalar esto como un conflicto.

III. Arquitectura del Sistema

El sistema se compone de cuatro partes fundamentales: CLI, capa de negocio, capa que conecta la lógica con la red y la red con los datos.



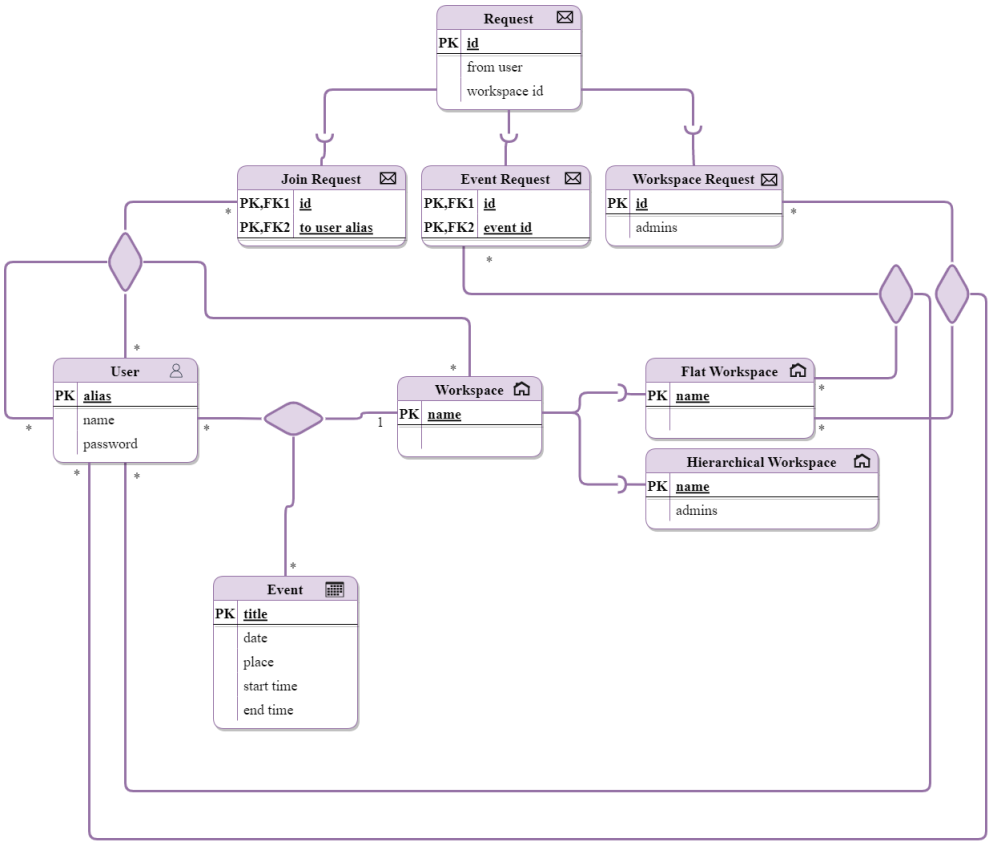
I. CLI

Para la presentación y prueba del sistema, se desarrolló una interfaz de línea de comandos (CLI). Los usuarios pueden interactuar con el sistema a través de la CLI, lo que permite una interacción flexible y eficiente con el sistema. La lista de comandos y su descripción se pueden observar al ejecutar el sistema e insertar el comando `-h` o `-help`.

II. Lógica de negocio

En la capa de negocio se implementaron las clases y métodos necesarios para la representación de las entidades y relaciones que el problema se evidencian. Entre dichas entidades se encuantran:

- **User**: Representa a un usuario de la agenda distribuida.
- **Workspace**: Representa un grupo en la agenda distribuida. Los grupos pueden ser jerárquicos o no.
- **Event**: Representa un evento de un workspace en la agenda distribuida.
- **Request**: Representa una petición o invitación que realizan los usuarios y se muestran como notificaciones. Se utilizan para invitar a usuarios a un grupo o aceptar la creación de un evento en grupos no jerárquicos.



III. Capa intermedia

Esta capa se encarga de conectar la lógica de negocio con la red de datos. Para ello se auxilia de métodos `get()` y `set()` para pedir y guardar datos en la red.

IV. Red de Kademlia

Kademlia [1] es un protocolo de la capa de aplicación diseñado para redes P2P descentralizadas. En una de este tipo los nodos se distribuyen de forma descentralizada utilizando una estructura de árbol binario.

El proceso de ingreso de un nodo a la red Kademlia se realiza mediante un mecanismo de broadcast. Cuando un nuevo nodo se quiere unir a la red, envía un mensaje de broadcast a todos los nodos existentes. Si hay algun servidor escuchando este devolvera su ip, lo cual permitira al nodo poder conectarse a la red. Los nodos existentes actualizan sus tablas de enrutamiento para incluir al nuevo nodo.

Para realizar búsquedas en la red, se utiliza un algoritmo basado en DHT (Distributed Hash Table). El algoritmo busca los k nodos más cercanos (en términos de distancia XOR) a una clave específica. Comienza seleccionando α contactos de los k cubetas que contienen nodos más cercanos a la clave buscada. Luego, envía mensajes asincrónicos a estos contactos para obtener información sobre la clave. Cada contacto activo devuelve información que ayuda a acercarse al nodo objetivo. Este proceso se repite hasta que se encuentra el nodo deseado.

Por otra parte, para el almacenamiento de los datos se utilizó la librería `dictdatabase`, la cual proporciona métodos para obtener y guardar datos en archivos *.json*.

En cuanto a la tolerancia a fallos, Kademlia utiliza una estructura descentralizada que proporciona una fuerte defensa contra ataques de denegación de servicio. Debido a la distribución de los nodos y la redundancia de la información en la red, es difícil para un atacante afectar el funcionamiento de toda la red. Además, Kademlia permite que los nodos se unan y abandonen la red de forma independiente, lo que proporciona flexibilidad y adaptabilidad a cambios en la disponibilidad de nodos.

IV. Docker

Para simular distintos servidores en un sistema distribuido, utilizamos Docker. Cada contenedor Docker puede considerarse como un nodo en el sistema distribuido, capaz de ejecutar su propio sistema operativo, tener su propia configuración de red y ejecutar las aplicaciones como si estuviera en una máquina separada. Esto permite simular un sistema distribuido en una sola máquina física, lo que facilita el desarrollo y las pruebas del sistema.

Referencias

[1] Maymounkov, P., Mazières, D. (2002). *Kademlia: A Peer-to-Peer Information System Based on the XOR Metric.* In: Druschel, P., Kaashoek, F., Rowstron, A. (eds) Peer-to-Peer Systems. IPTPS 2002. Lecture Notes in Computer Science, vol 2429. Springer, Berlin, Heidelberg.