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

19 de julio de 2023

Repositorio del proyecto

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

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.

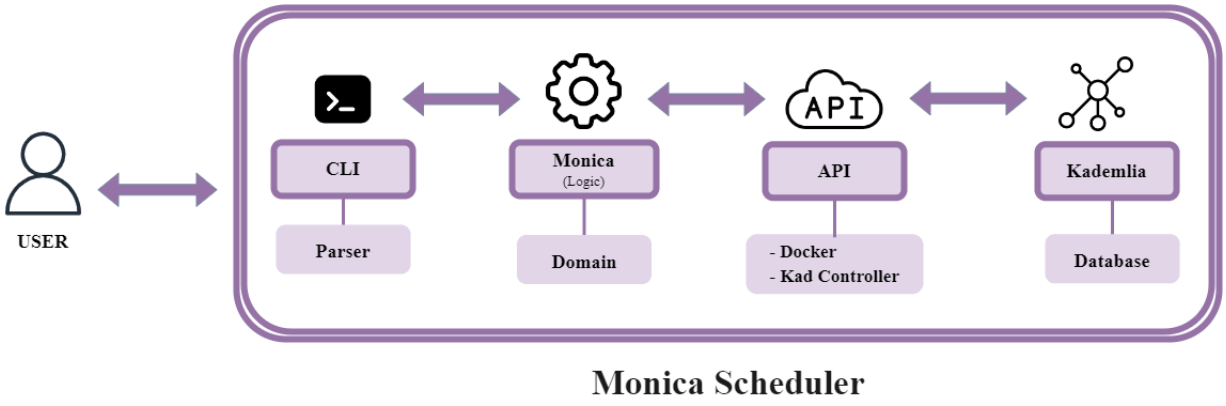

Figura 1: *Esquema de flujo del sistema.*

I. CLI

Para facilitar la presentación y prueba del sistema, se implementó una interfaz de línea de comandos (CLI, por sus siglas en inglés) que permite a los usuarios interactuar con el sistema de manera flexible y eficiente. En su desarrollo se utilizó el módulo ***argparse*** de Python para analizar y procesar cada línea de comando ingresada por los usuarios.

La CLI proporciona una variedad de comandos que los usuarios pueden ejecutar para realizar diferentes acciones en el sistema. Al ejecutar el sistema e ingresar el comando `-h` o `--help`, se muestra una lista completa de los comandos disponibles junto con su descripción correspondiente. Esta funcionalidad brinda a los usuarios una referencia rápida y fácil de los comandos admitidos, lo que facilita la interacción con el sistema.

Gracias al uso del módulo argparse, se implementa una lógica robusta para analizar y manejar los argumentos de la línea de comandos. Esto incluye la validación de los argumentos ingresados y la ejecución de las acciones correspondientes en función de los comandos proporcionados. Los usuarios pueden proporcionar los argumentos requeridos y opcionales de manera ordenada y estructurada, lo que mejora la experiencia de uso y evita posibles errores de entrada.

II. Monica

La capa de lógica de negocio desempeña un papel fundamental en la implementación y gestión de las reglas y procesos específicos del sistema. Dentro de esta capa, se llevaron a cabo diversas actividades clave para garantizar el correcto funcionamiento y cumplimiento de los requisitos del negocio.

Una de las actividades esenciales fue la validación y verificación de datos. Esto implicó asegurarse de que los datos recibidos fueran válidos y cumplieran con las reglas establecidas. Se realizaron comprobaciones exhaustivas para garantizar la integridad y calidad de los datos antes de procesarlos, evitando posibles inconsistencias o errores en el sistema. Para la coordinación de procesos del sistema se tuvieron que orquestar y sincronizar diferentes operaciones y componentes para garantizar la correcta secuencia y flujo de las tareas, asegurando que cada paso se realizara en el momento adecuado y en conformidad con las dependencias y restricciones establecidas. Además, se manejaron adecuadamente las excepciones, capturándolas, registrándolas y tomando las medidas necesarias para mitigar su impacto, así como se proporcionaron mensajes de error claros y significativos para informar al usuario sobre las incidencias y facilitar su resolución.

ii.1. Dominio

El dominio de nuestro sistema se centra en la gestión colaborativa de eventos y actividades entre múltiples usuarios. En él se identifican cuatro entidades principales: User (Usuario), Request (Petición), Event (Evento) y Workspace.

- **User:** Representa a los usuarios de la aplicación de agenda distribuida. Cada usuario puede tener atributos como nombre, alias y contraseña, los cuales se utilizan para identificar y autenticar a los usuarios en el sistema.
- **Request:** Representa las peticiones realizadas dentro de la aplicación. Estas peticiones pueden ser de distintos tipos y se envían entre usuarios para realizar acciones específicas. Los atributos y relaciones de la entidad Request se detallan en la Figura [2].
- **Event:** Representa los eventos o actividades programadas en la agenda distribuida. Cada evento puede tener atributos como título, fecha, lugar, hora de inicio y hora de finalización, que permiten identificar y organizar las diferentes actividades en el sistema.
- **Workspace:** Representa los espacios de trabajo colaborativo en los cuales se comparte la agenda distribuida. Cada workspace puede tener un nombre asociado y permite agrupar a los usuarios y eventos relacionados.

Las relaciones y atributos específicos de las entidades se encuentran detallados en la Figura [2], proporcionando una representación visual que muestra cómo se conectan y se relacionan estas entidades dentro del dominio de la aplicación de agenda distribuida.

III. API

Esta capa se encarga de conectar la lógica de negocio con la red de datos. La clase API, primeramente se encarga de construir la imagen de docker y levantar la red de Kademlia con una cantidad de nodos igual a dos, creando un container por cada uno. Tiene cinco métodos fundamentales: build_image (construye la imagen de docker), create_servers(crea contenedores y por cada uno levanta un servidor de la red), remove_servers (para un número random de contenedores), get_value (dada una llave levanta un contenedor temporal que se conecta a la red

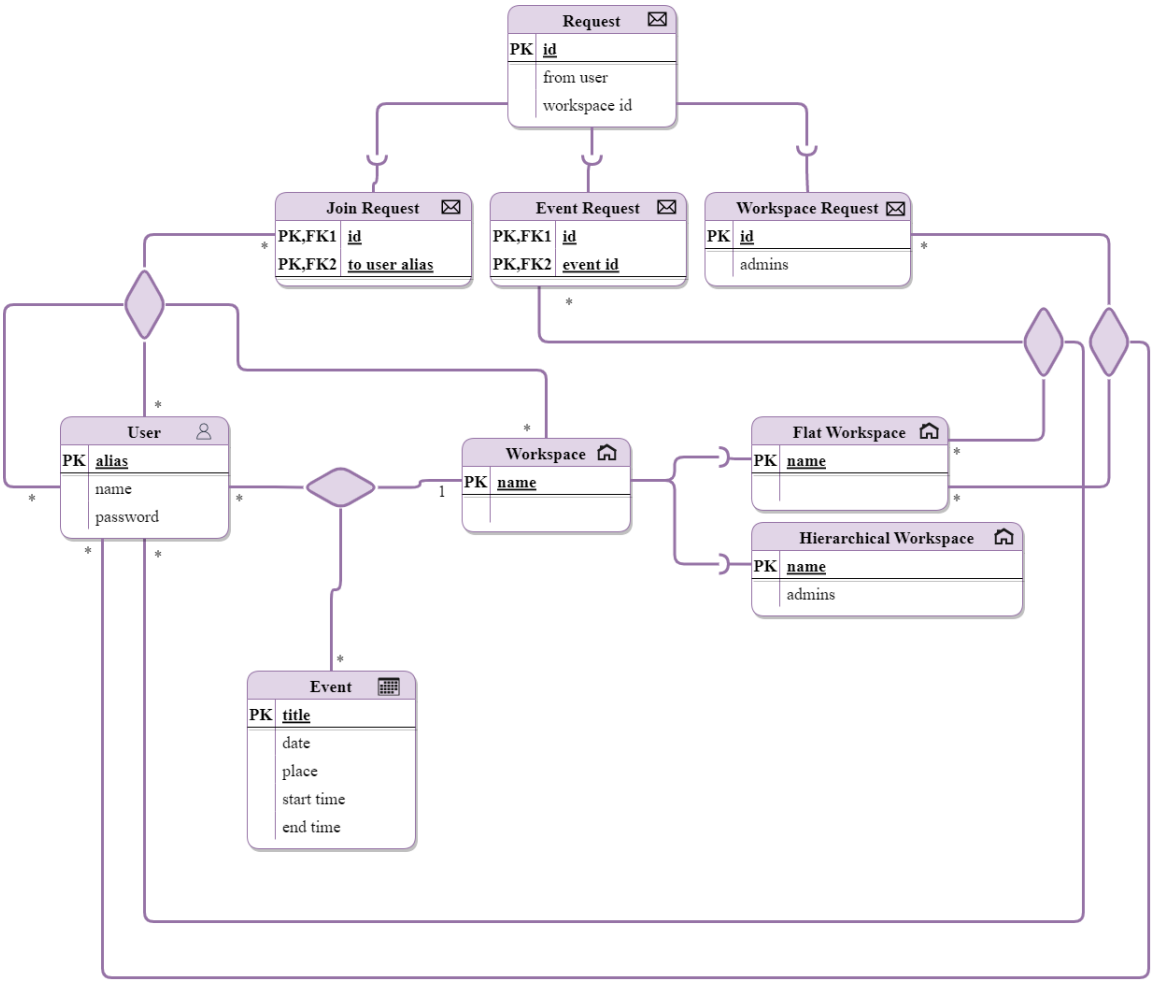

Figura 2: *Diagrama entidad-relación del problema agenda distribuida.*

para obtener el valor correspondiente y al obtenerlo lo para) y set_value (dada una llave y un valor levanta un contenedor temporal que se conecta a la red para obtener el establecer el valor correspondiente en la llave especificada, y lugo para el container).

IV. Red de Kademlia

Kademlia [1] es un protocolo descentralizado utilizado en redes peer-to-peer (P2P) para el almacenamiento y recuperación eficiente de información. Los nodos en la red se representa mediante una identificación única llamada ID de nodo que se deriva del valor hash de una clave pública criptográfica. El protocolo utiliza una estructura de árbol binario conocida como ”árbol de cubetas” para organizar los nodos en la red. Cada nodo mantiene una tabla de enrutamiento que contiene información sobre otros nodos en la red y su cercanía en el espacio de direcciones. Esta tabla se divide en ”cubetas” que agrupan nodos con direcciones similares. Cuando un nodo necesita buscar o almacenar información, realiza consultas a través de la red utilizando la distancia de XOR entre los ID de nodo como criterio de enrutamiento. Esto permite un enrutamiento eficiente hacia los nodos más cercanos al ID de destino.

El protocolo es conocido por su resistencia a fallos y su capacidad para adaptarse a cambios en la red, como nodos que se unen o abandonan. También es escalable, lo que significa que puede funcionar eficientemente en redes P2P grandes con millones de nodos.

iv.1. Implementación

En la implementación de la red Kademlia, se utiliza un algoritmo basado en DHT (Distributed Hash Table) para realizar búsquedas en la red. Este algoritmo busca los k nodos más cercanos a una clave específica en términos de distancia XOR. El proceso comienza seleccionando α contactos de las k cubetas que contienen los nodos más cercanos a la clave buscada. Luego, se envían 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 encontrar el nodo deseado.

Cuando un nuevo nodo desea unirse a la red de Kademlia, se utiliza un mecanismo de *broadcast*. El nuevo nodo envía un mensaje de broadcast y espera activamente a que algún nodo responda a su solicitud. Los nodos que reciben el mensaje pueden decidir responder al nuevo nodo proporcionando su dirección IP. Una vez que se recibe la respuesta, el nuevo nodo puede utilizar la información proporcionada para establecer una conexión con la red y actualizar su tabla de enrutamiento para incluir a los nodos con los que ha establecido comunicación. A su vez, los nodos existentes actualizan sus tablas de enrutamiento para incluir al nuevo nodo.

Para el almacenamiento de datos, se utilizó la biblioteca `dictdatabase`, la cual proporciona métodos para obtener y guardar datos en archivos con formato JSON. La clave de los valores se genera a partir del SHA1 del identificador del objeto. Cada vez que se modifica el objeto, se actualiza el valor asociado a dicha llave en la base de datos.

Se consideró otra opción que consistía en aplicar el algoritmo SHA1 a toda la información que encapsula el objeto, lo que generaría una nueva clave cada vez que se modifica el objeto. Sin embargo, esta opción aumentaría la cantidad de datos almacenados y sería difícil mantener la consistencia de la mayoría de los datos, ya que se requeriría cambiar la clave antigua por la nueva en cada objeto que la almacene. Por lo tanto, se decidió utilizar el enfoque anteriormente mencionado para asegurar una gestión eficiente y coherente de los datos almacenados.

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.