Índice general

Proyecto Final de Distribuido

Belsai Arango Hernández
Daniela Rodríguez Cepero
Eduardo García Maleta

Universidad de La Habana,
San Lázaro y L. Plaza de la Revolución, La Habana, Cuba
https://github.com/bel5599/Whatsapp_Distribuido
http://www.uh.cu

Abstract. Messaging has an important role in our days. Getting started with emails even the most complete messaging applications. It would also be desired that the courier service does not depend on a centralized entity that regulates the profiles and potentially **reads** our conversations. Instead of this, it is desired to provide a decentralized and secure architecture, in which anyone can receive and send messages with the guarantee that only the involved in the conversation.

Keywords: client, connection, network, server.

.

Resumen La mensajería tiene un papel importante en nuestros días. Empezando con los correos electrónicos hasta las aplicaciones de mensajes más completas. Se quisiera además que el servicio de mensajería no dependiera de un ente centralizado que regula los perfiles y potencialmente **lee** nuestras conversaciones. En lugar de esto se desea proveer de una arquitectura descentralizada y segura, en la que, cualquiera pueda recibir y mandar mensajes con la garantía de que solo lo podrán ver los involucrados en la conversación.

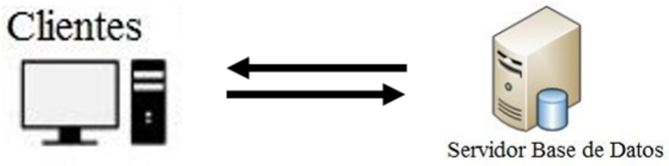
Palabras claves: cliente, conexión, servidor, red.

1. Requerimientos

- **SQLAlchemy:** Es el kit de herramientas SQL de Python y el Mapeador relacional de objetos que ofrece a los desarrolladores de aplicaciones la máxima potencia y flexibilidad de SQL. Esta proporciona un conjunto completo de patrones de persistencia conocidos a nivel empresarial, diseñados para un acceso a bases de datos eficiente y de alto rendimiento, adaptados a un lenguaje de dominio simple y Pythonic.
- **Typer:** Es una biblioteca para crear potentes aplicaciones de interfaz de línea de comandos de la forma más sencilla. Es más fácil de leer y la forma más sencilla de crear una aplicación de línea de comandos en lugar de usar la biblioteca estándar de Python arguparse, que es complicada de usar. Se basa en las sugerencias de tipo de Python 3.6+ y se basa en Click (Typer hereda la mayoría de las funciones y beneficios de Click), que es un paquete de Python para crear una interfaz de línea de comandos. El módulo también proporciona ayuda automática y finalización automática para todos los shells. Además, es corto de escribir y fácil de usar.
- **FastApi:** Es un marco web moderno y rápido (de alto rendimiento) para crear API con Python 3.6+ basado en sugerencias de tipo Python estándar. Constituye un conjunto de herramientas que permite a los desarrolladores utilizar una interfaz REST para llamar a funciones de uso común para implementar aplicaciones. Se accede a travées de una API REST para llamar a bloques de creación comunes para una aplicación.
- **Uvicorn:** Es una implementación de servidor web ASGI para Python. Además, es el elemento de enlace que maneja las conexiones web desde el navegador o el cliente de API y luego permite que FastAPI sirva la solicitud real. Uvicorn escucha en un socket, recibe la conexión, procesa un bit y entrega la solicitud a FastAPI, de acuerdo con la interfaz ASGI.
- **Requests:** Es el estándar de facto para realizar solicitudes HTTP en Python. Abstrae las complejidades de hacer solicitudes detrás de una API simple para que pueda concentrarse en interactuar con los servicios y consumir datos en su aplicación.

2. Arquitectura

El sistema está diseñado con dos estructuras fundamentales: un servicio en función a un cliente quien es el sistema en sí y otro capacitado como un servidor, que son los gestores de identidad encargados de manejar la información de los usuarios.



Este modelo de diseño de software permite que las tareas se distribuyan entre los proveedores de servicios (servidor) y los demandantes (cliente). El cliente solicita varios servicios al servidor, y este se encarga de dar la respuesta demandada por el cliente.

3. Cliente

El cliente es la entidad encargada de mandar y recibir mensajes así como almacenar la información de las conversaciones. La primera vez que un usuario se añade a la red, debe identificarse para luego adicionar contactos y crear chats.

La estructura del cliente dispone de los siguientes servicios:

El cliente puede registrarse a través de cualquier dirección válida en el sistema, ingresando sus datos personales: id(nickname)y contraseña a partir del cual iniciará el proceso de registro o logueo. Este servicio se efectúa mediante la función Register.

Para obtener acceso a los recursos de mensajería proporcionados en el entorno, el usuario debe iniciar sesión en el sistema mediante sus credenciales. Si dicho acceso es permitido, entonces puede hacer uso de las funcionalidades que facilita el sistema. Dicho preceso se efectúa mediante la función Login.

El envío de mensajes se puede efectuar entre cualquier par de usuarios, sin requerir que el destinatario esté registrado en la lista de contactos del remitente siempre y cuando conozca el nickname del destinatario. La funcionalidad de envío de mensajes se efectúa mediante SendMessenge y la obtención de estos mediante Messenges.

El usuario mediante el cliente también puede visualizar, guardar y eliminar contactos, mediante las funcionalidades GetContacts, AddContacts y DeleteContacts. Así como también puede visualizar los chat que tiene.

La aplicación del cliente cuenta con un proceso de verificación que mantiene una lista de servidores con los cuales comunicarse con el sistema para realizar las peticiones.

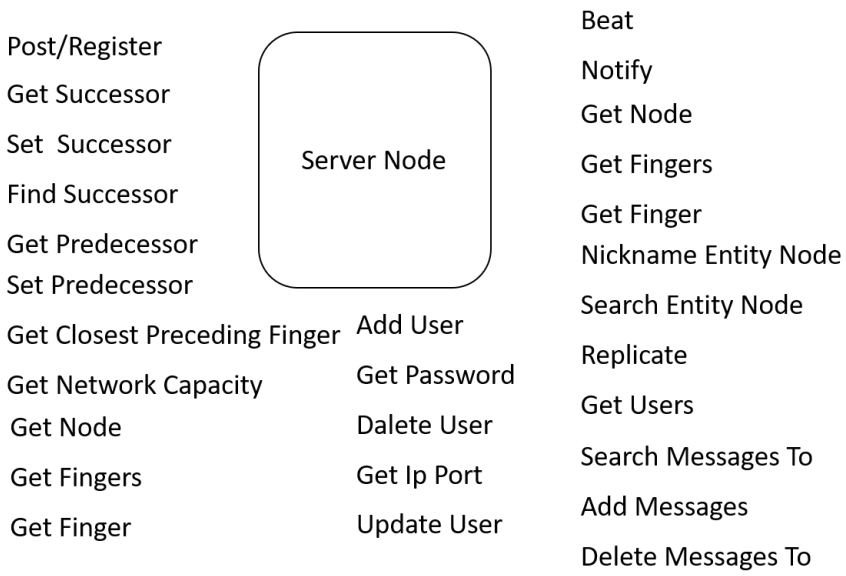
4. Servidores

El servidor está compuesto por nodos entitys conocidos como gestores de identidades, cada nodo es un servicio en el que los usuarios crean sus cuentas para guardar información que luego puede ser consumida por otras aplicaciones(cliente). Esta información solo puede ser consultada por el propio usuario, o por lo aplicación previa consulta al mismo. Este elemento facilita que los usuarios finales puedan utilizar un servicio como este sin preocuparse por las interioridades del sistema.

Cada nodo dispone de un proceso de verificación que mantiene la conectividad enlazada en la red. Estos nodos tienen la funcionalidad de nodos chord.

- **Chord:**Chord es un protocolo y algoritmo para la implementación de tablas de hash distribuidas. Es un sistema sencillo, escalable y esta diseñado para funcionar en redes descentralizadas. Este proporciona soporte para una sola operación: dado una clave, asigna la clave a un node. La ubicación de los datos puede ser fácilmente implementada sobre Chord asociando una clave para cada elemento, y almacenar el par de clave elemento en cada nodo. Se adapta de manera eficiente a medida que los nodos se unen y dejan el sistema, y puede responder consultas incluso si el sistema esta continuamente cambiando.

A continuación se muestran los distintos recursos o servicios que brinda un servidor propio del sistema.



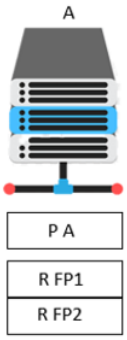
4.1. Estructura

Un servidor se compone de: un id único hasheado mediante su dirección ip, puerto y la capacidad de la red a la que pertenece y una tabla basada en una estructura de hash distribuida de Chord, conocida por las siglas DHT Chordhash (del inglés, Distributed Hash Tables). Ademas cada nodo está compuesto por un sucesor y un predecesor, donde este último se encuentra como una propiedad a la cual se puede acceder y en el caso del sucesor se puede obtener en la posición 0 de la finger table.

La tabla que contiene se llama tabla de fingers o finger table la cual se utiliza para optimizar la búsqueda de los nodos en la red de chord, es decir evita, que se tengan que recorrer todos los nodos, para encontrar uno en específico, de forma lineal. Cada finger tiene un id que se utiliza para buscar el succesor de ese id y guardarlo en la tabla. Los fingers en nuestra implementación son útiles para encontrar un nuevo succesor a la hora que el succesor que estaba se desconectó.

Si solo existe un servidor activo en la red, en la tabla de finger se guarda este nodo como su succesor y los nodos de su finger se contendrán eventualmente como succesor a él.

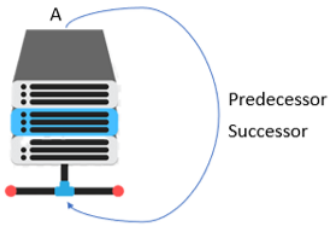
En cuanto a la información de los usuarios y los mensajes temporales que se alojan en el servidor, como consecuencia de la desconexión del usuario al que se le desea enviar el mensaje; cada nodo actúa como almacén de datos mediante la inclusión de tres bases de datos en su estructura. En la primera almacena su información propia y en las otras dos bases de datos, la información correspondiente de los datos heredados, como parte de la concepción del modelo de sistema distribuido diseñado.



4.2. Conexión

Para el proceso de conexión se utilizan diferentes comandos.

Creación de la red: Para crear una red de servidores se utiliza el comando python *server_app.py up*, donde el nodo que la inicia levanta 2 servicios, el servicio de la red de chord y un servicio de disponibilidad para ser descubierto por los futuros nodos. Además define la capacidad de la red.



Inserción: La inserción de un nodo al sistema se efectúa a través de una petición a cualquier servidor que se encuentre en la red: python *server_app.py join*.

Para obtener un servidor activo se ejecuta un broadcast en la subred para obtener su información e iniciar el proceso de conexión a través de el.

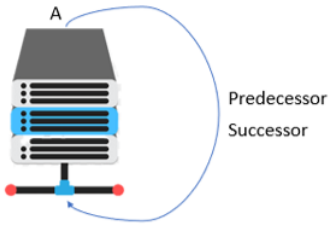
De esta misma forma se conecta un cliente a la red lo que a diferencia del anterior, se utliza un nuevo comando, en este caso python *client_app.py*.

Una vez que un servidor se conecta inicia sus bases de datos, vacías en el caso del servidor de la red y con la información anterior en el caso del servidor del cliente, si este había efectuado una conexión anteriormente.

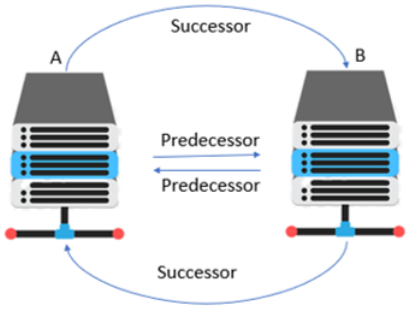
Al entrar un nodo a la red ocurre el siguiente procedimiento:

BEFORE ⇓

Al estar un solo nodo en la red, ocurre que este nodo es su sucesor y su antecesor a la vez.



AFTER ⇓



Luego que se inserta un nuevo nodo en dependencia de su id pueden ocurrir 2 situaciones:

- El nuevo nodo está antes del nodo ya existente, por tanto el nuevo nodo es predecesor del nodo existente y el nodo existente sucesor de él.

- El nuevo nodo está después del nodo que ya esta en la red por lo que ocurre lo contrario al caso de arriba, es decir el nuevo nodo es el sucesor del nodo existente y el nodo existente es el predecesor de él.

Mantenimiento de la red

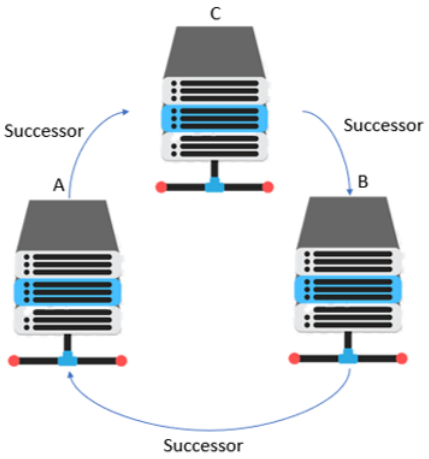
Para mantener la red actualizada se inicia un proceso de estabilización, donde los nodos que forman parte, desde un intervalo configurable ejecutan un conjunto de tareas para mantener la salud de la red.

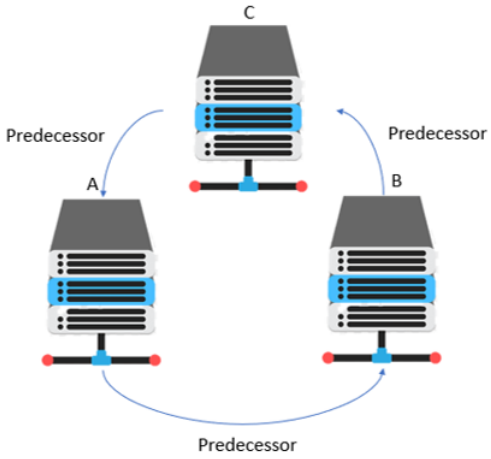
- La primera tarea es tener siempre actualizado el succesor de cada nodo, porque de esa manera se garantiza que se pueda conectar nuevos nodos a la red y se actualicen los predecesores y las tablas finger table.
- La segunda tarea se encarga de verificar que el predecesor guardado es el predecesor actual.
- La tercera tarea es que cada nodo actualice los valores de algunos fingers aleatorios de su tabla.

Desconexión de un nodo

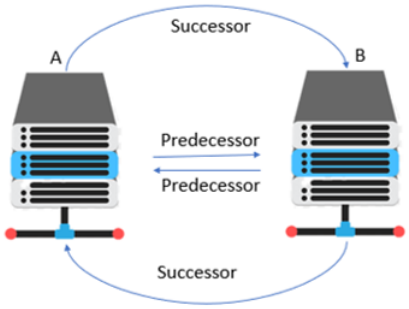
Ante la desconexión de un nodo de la red, digamos C, se actualizan los predecesores y sucesores de los nodos activos cercanos. A continuación se muestra un ejemplo de una red de 3 nodos(A, B y C) y el resultado luego de la desconexión del nodo c.

BEFORE ⇓





AFTER ⇓

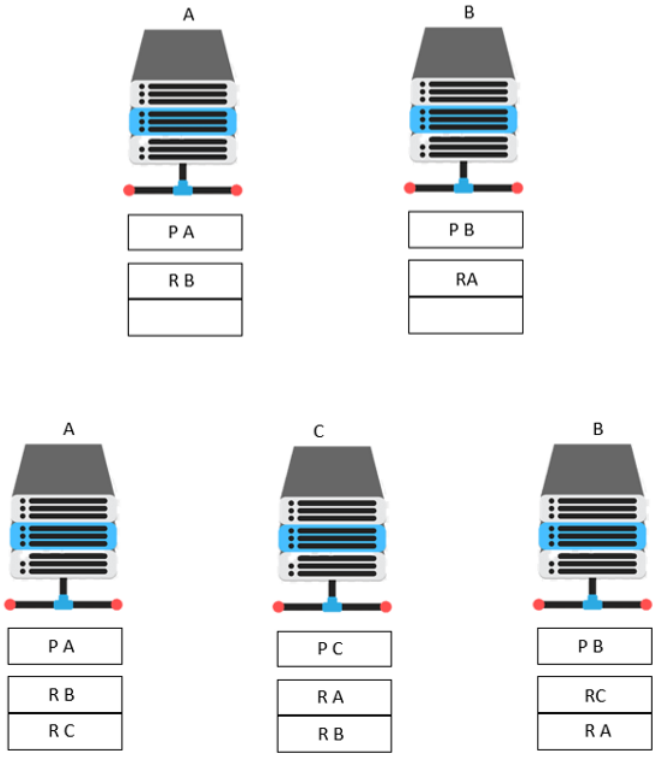


5. Replicación

Con el objetivo de mantener una consistencia en los datos y evitar la pérdida de la información, la replicación resulta una de las tareas más importante en el desarrollo de la aplicación.

5.1. Adición

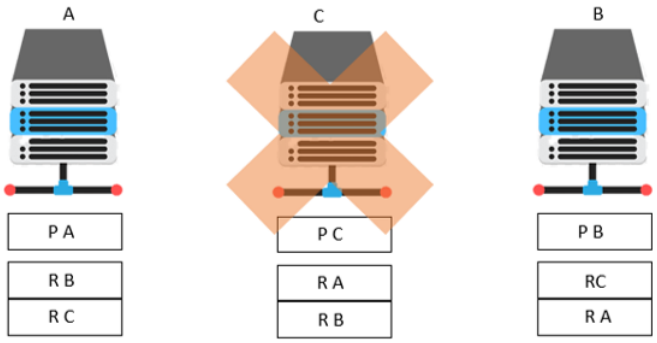
Ante la adición de un nodo a la red, existe dos nodos(sus dos sucesores) los cuales van a sufrir un cambio en sus antecesores. por tanto estos actualizarán las bases de datos que replican.



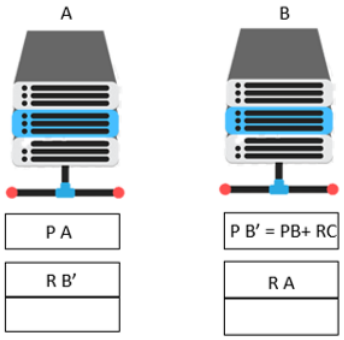
5.2. Eliminación

Ante la eliminación de un nodo su sucesor es el encargado de salvar su información. Para eso el sucesor verifica que este conectado su predecesor y en caso contrario se salva la información en la base datos propia de este y se replica.

A continuación un ejemplo:



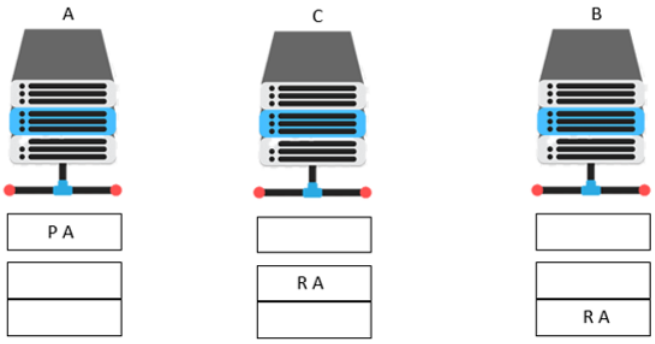
Se desconectó el nodo C, por tanto su información principal pasa como información nueva agregada a la base de datos de su sucesor, en este caso B.



5.3. Mantenimiento

Para mantener la consistencia en los datos que se almacenan en los nodos de la red, una vez que se inserta información en la base dato propia de un nodo, se replica en la base de datos secundaria de sus sucesores, en la que referencia al nodo.

A continuación se muestra un ejemplo de el resultado al insertar información en la base de datos propia de un nodo A.



PA - Información propia de A
RA - Información replicada de A

5.4. Recomendaciones

Para una mayor seguridad en la red se recomienda un sistema de incriptación para los datos que se almacenan en los servidores.

Una interface visual amigable que puedan usar los usuarios, similar al de la aplicación WhatsApp.

En nuestro sistema, la comunicación se realiza a través del protocolo HTTP, con la intención de chequear el flujo de datos en un entorno de desarrollo. Para el caso de un entorno de producción, se requiere de certificados que avalen la seguridad en la transmisión y obtención de datos, para emplear un protocolo HTTPS.