Informe del Diseño del Proyecto de Sistemas Distribuidos: WhatsApp P2P

Enero, 2025

Equipo:

- **Leonardo D. Artiles Montero (C412)**
- **Daniel Machado Pérez (C411)**

1. Arquitectura o el problema de cómo diseñar el sistema

Organización del sistema distribuido:

- El sistema distribuido estará basado en la arquitectura **Chord**, donde cada nodo gestor conoce su sucesor, predecesor y su Finger Table.
- Cada nodo gestor contiene pares de datos en la forma `<USUARIO (KEY), <IP, PASSWORD, LAST_UPDATE>>` , donde `LAST_UPDATE` corresponde a la última fecha y hora de actualización de los datos.

Roles del sistema:

- **Nodos Clientes**: Localizados en la red de clientes, permiten enviar mensajes, almacenar datos localmente y mantener comunicación con los gestores.
- **Nodos Gestores**: Localizados en la red de servidores, gestionan las operaciones de registro, consulta y actualización de usuarios.
    - Un subconjunto de gestores (denominado **Gestores Iniciales**) servirá como punto de inicio para las consultas de los clientes. Esta lista se mantiene actualizada en todos los gestores y se envía a los clientes al realizar una consulta.

2. Procesos o el problema de cuántos programas o servicios posee el sistema

Tipos de procesos:

- **En los clientes**:
  - `send_alive_signal` : Envía señales de vida a los gestores.
  - `receive_msg` : Escucha y procesa mensajes entrantes.
  - `send_pending_messages` : Gestiona el envío de mensajes pendientes cuando los usuarios destinatarios se conectan.
- **En los gestores**:
  - `alive_signal` : Escucha señales de vida de los clientes y responde si el gestor está activo.
  - `handle_client` : Procesa solicitudes de clientes (actualizaciones de IP, login, registro, etc.).
  - `ring_update` : Gestiona cambios en el anillo de Chord y actualizaciones en la finger table.
  - `initial_gestors_update` : Actualiza la tabla de Gestores Iniciales local.
  - `find_if_duplicates_are_alive` : Verifica si los nodos duplicados están activos.
  - `receive_duplicate` : Recibe actualizaciones o copias completas de datos de otro nodo.
  - `replicate_originals_if_dead` : Maneja la replicación de datos en caso de que un nodo original falle.

Cada nodo gestor ejecuta múltiples hilos dentro de un único programa.

3. Comunicación o el problema de cómo enviar información mediante la red

Tipo de comunicación:

Actualmente se utiliza `sockets` , pero se está valorando utilizar **ZeroMQ** como el middleware para facilitar patrones de mensajería eficientes y robustos en esta segunda etapa.

Comunicaciones específicas:

- **Cliente-Servidor**:
  - **PUB-SUB**: Para las señales de vida enviadas por los clientes.
  - **REQ-REP**: Para consultas como obtener la IP de otros usuarios.
- **Servidor-Servidor**:

- **PUB-SUB**: Para actualizar la tabla de Gestores Iniciales y verificar si los nodos duplicados están activos.
- **REQ-REP**: Para la transmisión de datos a duplicar entre gestores.

4. Coordinación o el problema de poner todos los servicios de acuerdo

- Cada recurso tiene una marca de tiempo (`LAST_UPDATE`) que permite resolver conflictos al seleccionar siempre la versión más reciente de los datos, por ejemplo, en caso de existir duplicados o modificaciones.
- Los gestores deciden de forma autónoma su estrategia de replicación, eligiendo aleatoriamente a los nodos donde duplicar sus datos.
- La tabla de **Gestores Iniciales** es el único dato replicado en todos los nodos gestores de forma sincronizada, y su última fecha de modificación evita inconsistencias o condiciones de carrera.

5. Nombrado y Localización o el problema de dónde se encuentra un recurso y cómo llegar al mismo

La **localización de recursos** es gestionada por Chord, que organiza los nodos y datos en un anillo lógico. Los nombres de los recursos están definidos mediante la clave del usuario (`USERNAME`), y se utiliza un hash polinomial para distribuirlos uniformemente en el espacio de claves. Los datos de cada gestor están almacenados en una base de datos SQLite para garantizar persistencia y eficiencia.

6. Consistencia y Replicación o el problema de solucionar los problemas que surgen por tener varias copias de un mismo dato

- Cada nodo gestor mantiene una copia de sus datos en un mínimo de otros x nodos donde x es el mínimo entre la cantidad de nodos existentes y 3. Cada vez que se actualiza un dato se produce una replicación de este en los backups. Como esto solo ocurre cuando un usuario cambia su IP o

se desconecta, y eso se espera que no sea tan frecuente, dicha replicación no deberá sobrecargar los servidores.

7. Tolerancia a fallas o el problema de, para que pasar tanto trabajo distribuyendo datos y servicios si al fallar una componente del sistema todo se viene abajo

- Como regularmente se verifica si las copias de cada gestor están "vivas", a menos que todas las copias fallen a la vez, la replicación garantiza que no se pierdan datos.
- Como se duplica la información de cada nodo en al menos otros 3 nodos (en caso de existir más de 3 nodos) el nivel de tolerancia a fallos es 2.
- Si un nodo se cae temporalmente, el predecesor activo más cercano toma sus rango de datos. Sin embargo, si este nodo se vuelve a conectar, funcionará como un nodo nuevo en el anillo de Chord.

8. Seguridad o el problema de qué tan vulnerable es el diseño

La aplicación utiliza una arquitectura **peer-to-peer** donde los mensajes nunca pasan ni se almacenan en un gestor centralizado. Esto protege la privacidad y asegura que solo el usuario pueda acceder a su historial de mensajes.

Las sesiones están protegidas mediante un nombre de usuario único y una contraseña cifrada almacenada por los gestores, garantizando autenticación y autorización seguras.

Algunos preguntas que pudieran ser importantes en cuanto a la seguridad son:

- ¿Qué pasa si un nodo externo maligno se intenta unir al anillo de Chord? Para esto se planea implementar algun patrón de verificación, tal que si un nodo se conecta es porque es confiable.
- ¿Si un ataque se apodera de un nodo, es posible sacarlo del anillo? En este casose podría crear un mecanismo para que lo notifique como Nodo eliminado, lo cual haría que cambiaran las Finger Tables, y además que su contenido se replicara automáticamente en otro nodo.