Universidad Técnica Federico Santa María
Sistemas Distribuidos
Enero 2020

Tarea 2

Arquitecturas distribuidas y microservicios

Informe Comparativo

Integrantes: Gabriel Arjona - Francisco Vásquez

1. Comparación aspectos generales

El sistema implementado usando gRPC cumple con los requisitos solicitados sin problema, tanto del lado del servidor como del lado del cliente. Desde la perspectiva de diseño, se obtuvo un resultado de complejidad moderada en cuanto a las líneas de código necesarias para construir el sistema (aproximadamente 400) y a la dificultad de lectura que tiene el código, pues a pesar de estar constituído por llamado a funciones y uso de métodos específicos de gRPC, no deja de ser comprensible.

Mientras que el sistema implementado usando RabbitMQ también cumple con los requisitos solicitados tanto de parte del servidor como del lado del cliente con la diferencia de que el código se hace visualmente más fácil que el sistema anterior, tanto en la complejidad del código como en la cantidad de líneas implementadas.

2. Comparación Arquitectura

2.1 Lado del servidor

2.1.1 Lado del servidor usando gRPC

El código desarrollado se concentró en tres archivos: *server.py*, *chat.py* y *id_mapper.py*.

En *server.py* se construyen dos servicios: un servicio para el envío y recepción de mensajes y uno para manejar (por parte del servidor) las ID de los clientes. Ambos servicios tienen los métodos necesarios para cumplir con lo solicitado, sin representar mayor complejidad en su constitución. Por último en este archivo se levantan los servicios y el servidor de manera sencilla.

En el archivo *id_mapper.py* se defiene la forma en que el servidor opera las IDs de los clientes, lo cual corresponde a un contador y una función que retorna el valor actual de este. Por su parte, en *chat.py* se debió construir un sistema de colas para desarrollar el envío y recepción de mensajes en el orden correcto. Además, incluye la implementación necesaria para que se puedan extraer todos los mensajes de un cliente.

En cuanto a la *dockerización* del servidor, se debió incluir en el *dockerfile* la instrucción para instalar gRPC en un container de docker, además de incluir los *requirements* de esta tecnología.

2.1.2 Lado del servidor usando RabbitMQ

El código desarrollado se concentró en solo un archivo *server.py*.

Gracias a la implementación de RabbitMQ no se necesitó un archivo de manejo y administración de colas porque RabbitMQ ya lo tiene implementado y solo fue necesario el archivo *server.py* para recibir las solicitudes de username de los clientes y registrarlos con una ID. La segunda parte importante de *server.py* es la implementación de recibir mensajes de los clientes y enviarlo a los destinatarios correspondientes mediante un canal que tiene como *routing_key* el username del cliente. De esta forma RabbitMQ se encargaría de administrar las colas de mensajes para cada username.

Del archivo *requirements.txt* solo es necesaria la librería *pika* para realizar las conexiones y se utilizó *threading* para hacer asíncrona la función de registrar usuarios y la función de recibir y enviar mensajes a los clientes.

2.2 Lado del cliente

2.2.1 Lado del cliente usando gRPC

En el caso del cliente el código desarrollado se concentra en el archivo *client.py*. Se distinguen dos componentes principales, que son: 1) Un thread para la recepción de mensajes, creado para recibirlos de manera asíncrona y 2) El código principal de un cliente. En el código principal se instancia un channel al server, los stub de *chat* e *id_mapper* con sus respectivos servicios, se obtiene una ID y se abre el hilo para poder recibir mensajes de forma asíncrona.

Luego, mediante los servicios implementados en el servidor se construye el procedimiento necesario para cumplir con cada requisito de la tarea. La mayor parte de esta implementación se hace “limpia” en el sentido de que no se necesita programar el funcionamiento de la cola de mensajes, a excepción de que es necesario enviar los mensajes (por debajo, sumado a lo que ingresa el cliente del sistema) con una estructura separable en lado del servidor, para poder separar el contenido del mensaje de su identificador y su timestamp.

2.2.2 Lado del cliente usando RabbitMQ

Por el lado del cliente en RabbitMQ se destacan dos funciones principales que se implementaron de forma asíncrona: La función de escuchar mensajes de parte del servidor y la función de enviar mensajes al servidor para que este lo pueda mandar al cliente destinatario.

Al iniciar el cliente tiene que mandar un username donde el servidor lo registra asociándolo a una ID.

3. Principales aspectos a comparar

- gRPC tiene entre sus principales aplicaciones conectar aplicaciones con clientes o entre ellos mismos, por lo que ha de ser común el paso de mensajes en las implementaciones de esta tecnología. Sin embargo, lo que no es común es utilizarlo como sistema de colas, hay otras tecnologías que se enfocan en ello. Se tiene entonces que no tiene el propósito principal de cumplir como tecnología para componer un sistema de cola de mensajes. Por su parte, en RabbitMQ se definen colas que van a almacenar los mensajes que envían los “productores” hasta que las aplicaciones obtienen el mensaje y lo procesan.
- En gRPC, para poder cumplir con la coordinación de los mensajes enviados y recibidos fue necesario implementar el sistema de colas en el chat construido con esta tecnología mientras que RabbitMQ ya lo tiene implementado.
- Para lograr el asincronismo en la recepción de mensajes se debió implementar un sistema de hilos en gRPC, puesto que esta tecnología no provee una forma para manejar este caso en alto nivel, en cambio RabbitMQ sí lo prove.
- El sistema gRPC no provee almacenamiento persistente de los mensajes, es decir, una vez cerrado el programa se pierden los mensajes a menos que se haya implementado una estrategia para conservarlos (como su registro en un archivo). RabbitMQ en cambio tiene configuraciones para establecer persistencia.
- gRPC permite generar una solución considerablemente personalizada, puesto que la implementación se debe hacer desde un nivel bajo, pudiendo definir las reglas en que operará el sistema deseado. Mientras que RabbitMQ tiene funciones ya implementadas donde finalmente es una caja negra lo que ocurre en las colas.
- RabbitMQ dentro de sus plugins tiene una interfaz de usuario donde es más simple entender el funcionamiento de las colas y conexiones como también saber el rendimiento. También a través de plugins provee soporte para múltiples protocolos y lenguajes.

4. Recomendación técnica

En base a los aspectos considerados en la sección 3, este equipo recomienda utilizar RabbitMQ como la mejor tecnología, entre las estudiadas en este trabajo, para resolver este problema.