Universidad Técnica Federico Santa María
Campus Santiago San Joaquín
INF-343
Segundo Semestre 2019

Informe Comparativo Tarea 2 SD

Christopher Gilbert, 201573597-2 christopher.gilbert@sansano.usm.cl
Jean Gonzalez Acevedo, 201573517-4 jean.gonzaleza@sansano.usm.cl

1. Implementación gRPC

Para la implementación de un servidor gRPC es necesario comenzar por la definición de los servicios que este contendrá, para ello es necesario crear el archivo proto que contendrá las declaraciones de los servicios y los detalles de los mensajes que estos reciben y envían (tipo de dato de los atributos y nombre con el cual identificarlos).

Una vez listo el archivo proto es necesario compilarlo para generar los archivos correspondientes a las clases de request y response, y el archivo de las clases servidor y cliente.

Finalmente corresponde implementar los servicios declarados en el archivo proto tanto en el lado del servidor para que puedan ser llamados por el client stub en el lado del cliente.

Para la arquitectura propuesta en la tarea se opto por implementar los siguientes funciones en el lado del cliente:

- Handshake

  3-way handshake donde el último mensaje que recibe el cliente corresponde al id que le asigna el servidor

- waitMsg

  Constantemente ejecutándose en un thread aparte para consultar si hay mensajes sin leer para el cliente

- SendMessage

  Envía mensaje con un destinatario definido al servidor para que este luego lo entregue debidamente

- Select_Menu

  Envía opción seleccionada por el usuario para que el servidor la procese y responda con la acción adecuada

Todas las funciones mencionadas utilizan el *client stub* para llamar al servicio implementado en el servidor.

En el lado del servidor los servicios implementados, y la respectiva función en el cliente que los llama, son:

- Handshake

  Servicio encargado de manejar conexiones y asignar id a clientes. Llamado en la función de **Handshake** en el cliente

- Send

  Servicio encargado de recibir los mensajes que envía el cliente para un destinatario y almacenarlos en el diccionario. Servicio llamado por **SendMessage** en el cliente

- Receive

  Servicio que recibe las peticiones del cliente cuando solicita sus mensajes sin leer y se los entrega. Este servicio es llamado por la función **waitMsg** en el cliente

- Menu

  Servicio encargado de procesar la acción que desea realizar el cliente mediante la opción que este envía. Servicio llamada en **Select_Menu**

Para el manejo de mensajes del servicio de chat desarrollado se utilizó un diccionario que almacenará todos los mensajes entrantes organizándolos en los destinatarios (id) que serán las llaves del diccionario. El cliente constantemente consultará al servidor si posee mensajes por leer mediante un thread que ejecuta el servicio *waitMsg* constantemente.

2. Implementación RabbitMQ

Para el correcto funcionamiento de RabbitMQ, lo primero que se debe hacer es levantar un *Message Broker* de rabbit, el cual permite el manejo de las colas, es decir, creación, modificación, eliminación, etc. Tras esto, tanto los Clientes como el Servidor deberán conectarse a este *Broker* para poder enviar y manejar los mensajes respectivamente.

Se crea una cola por cada Cliente, una *Global Queue* la cual mantiene todos los mensajes que no han sido redirigidos, y una *HandShakeQueue* donde se produce un handshake entre Cliente y Servidor, donde el cliente recibe su respectivo ID .

2.1. Servidor

En el servidor se crean dos Threads, uno es el *Producer* (Productor), utilizando el método *send* cumple dos objetivos principales. El primero es enviar los Id's de los clientes nuevos en la red. El segundo objetivo es redirigir los mensajes que están en la cola general a la cola de sus respectivos destinos.

El segundo Thread corresponde al *Consumer* (Consumidor), el cual tiene tres métodos, los cuales serán descritos a continuación:

- **run:** Inicializa el thread encargado de leer los mensajes de la *Global Queue*.
  - Conecta el servidor al *Message Broker* de rabbit, si falla lo vuelve a intentar cada 5 segundos.
  - Inicializa el Thread encargado de redirigir los mensajes y las colas *Global Queue* y *HandShakeQueue*
- **callback:** Función que se dispara cuando se lee un mensaje de la *GlobalQueue*
  - Se encarga de declarar las colas de cada cliente y redirigir los mensajes a las colas correspondientes. Para ello utiliza el método *send* del Thread de Producer.
  - Se encarga de enviar la ID a los clientes nuevos mediante el HandShake.
  - Escribe los logs de los mensajes.
- **receive:** Lee constantemente los mensajes de la *Global Queue* y llama a la función callback cuando recibe alguno.

2.2. Cliente

En el Cliente se crean tres Threads los cuales deben conectarse al Broker de manera individual, es decir hay una conexion a rabbit por cada Thread, cada uno de los threads serán descritos a continuación:

- **HandShake:** Es un thread temporal, el cual se encarga de realizar el handshake de tres pasos y recibir la ID que le corresponde al cliente cuando este se une a la red de mensajes. Tras recibir la id el thread es terminado.
- **Consumer:** Se encarga de declarar la cola particular del cliente y leer los mensajes que llegan a esta.
- **Producer:** Se encarga de enviar los mensajes del cliente a la *GlobalQueue* con headers que permiten identificar emisor y receptor, para que el servidor pueda redirigir el mensaje correctamente. Además cuando el cliente no tiene ID se encarga de solicitarla mediante el método *handshake*.

3. Comparación entre implementación

La utilización de gRPC permite una rápida definición de los servicios de la aplicación a desarrollar y es gracias a esto también que resulta muy cómodo el escalamiento de la aplicación en el futuro. Lamentablemente la dependencia de un cliente con el servidor no deja paso a ninguna posible caída de este ultimo, ya que de ocurrir el cliente no tendría posibilidad alguna de solicitar servicios.

Por otro lado, una de las grandes ventajas que posee RabbitMQ es la posibilidad de desarrollar un cliente independiente del servidor ya que este puede enviar mensajes a las colas sin ningún problema aun si el servidor se ha caído, pero lamentablemente no es capaz de recibir mensajes debido a que es necesario que el servidor procese los mensajes de la GlobalQueue para poder moverlo a la cola del cliente destinatario.

4. Conclusiones

Tanto RabbitMQ como gRPC son herramientas aptas para el desarrollo de un chat. A la hora de decidir cual utilizar no hay una que sea mejor que otra completamente ya que su uso va a estar condicionado a los requerimientos que existan, ya que si se desea que el servicio de mensajería continué funcionando sin importar la presencia del servidor, la mejor opción a utilizar es RabbitMQ debido a que provee el Broker que permite el funcionamiento de la apliacion solo que limitado.

Luego, si es necesario que la aplicación sea dependiente del servidor, la opción que se debería utilizar para el desarrollo debería ser indudablemente gRPC debido a su facilidad y rápido uso mediante las definiciones de servicios que a su vez permite realizar un fácil escalamiento en caso que se requiera extender las funciones de la aplicación desarrollada.