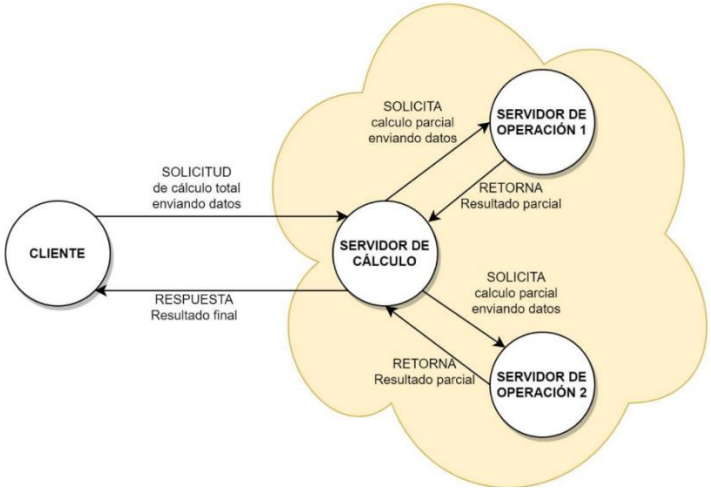
Taller 2

Sistemas Distribuidos

Profesor: Osberth Cristhian Luef De Castro Cuevas
Estudiantes:
Julián Andres Arana Guiza,
Angie Valentina León González,
Isabella Blanco Chaparro,
Juan David Barajas Urrea.

Explicación del desarrollo del taller 2:



Grafica 1: Topología de la conexión

Se realiza la reimplementación de la aplicación de cálculo distribuido utilizando gRPC en Python, nos basamos en la aplicación desarrollada en el taller anterior utilizando sockets.
Para probar la ejecución primero se hace de forma individual con los servidores 1 y 2, después el servidor principal y por último se ejecuta el cliente quien es el que envía la lista de números para que así finalmente se obtenga el resultado esperado del cálculo.

Archivo PROTO:

- Se crea un archivo de definición de servicios y mensajes en formato protobuf que en nuestro caso es array.proto, es el que define el servicio ArrayService con un método SendArray que toma un mensaje Array como entrada y devuelve un mensaje Response.

ARRAY_PB2_GRPC: Facilita la comunicación entre el cliente y el servidor mediante el protocolo gRPC.

- Por medio del ´ArrayServiceStub´(Cliente) permite que los clientes lamen al método `SendArray` en el servidor manejando la comunicación de red y la serialización de datos automáticamente.
- En el `ArrayServiceServicer` (Servidor) define la lógica del método `SendArray`, especificando cómo procesar las llamadas entrantes y los datos.
- add_ArrayServiceServicer_to_server hace un registro del servicio en el servidor gRPC, permitiendo que el servidor maneje las llamadas a `SendArray`.

- En la clase `ArrayService` se proporciona un método alternativo para realizar llamadas de servicio, principalmente para uso experimental.

ARRAY_PB2.PY: Define las estructuras de datos y servicios que se especificó en el archivo `array.proto`, base sobre la cual Python interactúa con el sistema de tipos de Protocol Buffers y gRPC.

- En el `Array` se define una lista de valores enteros `values`, usando la estructura `repeated`, lo que puede contener múltiples valores enteros.
- `Response` es el que contiene un solo campo entero (`message`) destinado a ser utilizado como respuesta del servicio.
- En el servicio `ArrayService` se define un servicio con un método `SendArray`, acepta un mensaje que es el `Array` como solicitud y devuelve un mensaje de `Response` como respuesta.
- En el protocolo se tiene una variable `DESCRIPTOR` que contiene la descripción completa del archivo `.proto` en un formato que Protocol Buffers y gRPC. Se utiliza internamente para registrar los tipos y servicios, permitiendo la serialización/deserialización de mensajes y la invocación de métodos de servicio.

ARRAY_PB2.PYI: Proporciona las definiciones de Python para las clases `Array` y `Response`, basadas en las especificaciones de `.proto` para la aplicación en gRPC lo que permite trabajar con mensajes `Array` y `Response`.

- El `Array` es para manejar listas de números enteros y se puede inicializar esta clase con una lista de enteros.
- En `Response` es para crear los mensajes de respuesta con un único valor entero.

Son esenciales para enviar y recibir datos estructurados a través del servicio gRPC, utilizando las capacidades del Protocol Buffers para la serialización y deserialización de mensajes.

CLIENTE:

- Se establece una conexión al servidor utilizando grpc.insecure_channel con el valor 192.168.214.8:5000 que representa la dirección IP y puerto del servidor principal al que se conectará el cliente.
- Se crea una instancia del stub del servicio ArrayService utilizando la clase ArrayServiceStub del archivo array_pb2_grpc y se está inicializando con un canal (channel).
- Se declara un array arreglo de enteros con algunos valores del 1 al 13.
- Se llama al método SendArray del stub enviando el arreglo como parámetro, esto para enviar el arreglo al servidor anterior y recibir la respuesta.
- Se imprime la respuesta recibida del servidor, que sería la suma de los valores del arreglo.

SERVER:

- Se crea la función SendArray, la cual toma dos argumentos request, que es un mensaje de tipo Array enviado por el cliente, y context, que proporciona información sobre el contexto de la llamada RPC.

- Se crea una variable longitud para la longitud del arreglo y luego dividirla en dos.
    - Con request.values[:mitad] y request.values[mitad:] se divide el arreglo recibido desde el cliente en dos mitades una en primeraMitad y otra en segundaMitad.
    - Envía cada mitad del arreglo a dos servidores secundarios (servidor1 y servidor2) diferentes utilizando la función enviarYRecibirMensaje(), en el cual se envía por parámetro la dirección IP con el puerto de cada servidor y la mitad del arreglo correspondiente. En este caso uno de nuestros computadores tenía la dirección IP 192.168.214.163 el cual era el servidor1 con el puerto 5002 y el otro computador con dirección IP 192.168.214.91 siento el servidor2 con el puerto 8001.
    - Se espera las respuestas de los servidores secundarios.
    - Se suma las respuestas y devuelve la suma como respuesta al cliente.
- Se crea la función serve
    - Se crea e inicializa un servidor grpc utilizando grpc.server().
    - Se agrega el servicio ArrayService al servidor utilizando array_pb2_grpc.add_ArrayServiceServicer_to_server(ArrayService(), server).
    - Se inicia el servidor en el puerto 5000 utilizando server.add_insecure_port('[::]:5000').
    - Inicia el servidor con server.start() y espera a que termine con server.wait_for_termination().
- Se crea la función enviarYRecibirMensaje, la cual enviará un mensaje a un servidor y recibirá la respuesta.
    - Se crea un canal de comunicación con el servidor secundario utilizando la dirección host.
    - Se instancia un stub del servicio ArrayService en el servidor secundario.
    - Se crea un Array con el contenido mensaje.
    - Se envía el mensaje al servidor secundario utilizando stub.SendArray(arreglo).
    - Se devuelve el mensaje de respuesta recibido del servidor secundario.

SERVIDOR1 y SERVIDOR2:

- Para estos archivos el código es igual, lo único que cambia es que como se dijo anteriormente para el servidor1 corresponde el puerto 5002 y para el servidor2 el puerto 8001.
- Se crea la función SendArray, suma los valores del arreglo recibido y devuelve la suma como respuesta.
- Se crea la función serve.
    - Se crea un servidor grpc utilizando grpc.server().
    - Se agrega el servicio ArrayService al servidor utilizando array_pb2_grpc.add_ArrayServiceServicer_to_server(ArrayService(), server).
    - Se asigna un puerto para escuchar las conexiones entrantes (5002 para servidor1 y 8001 para servidor2) utilizando server.add_insecure_port('[::]:puerto').
    - Inicia el servidor con server.start() y espera a que termine con server.wait_for_termination().

DIAGRAMA SECUENCIA

Pontificia Universidad JAVERIANA Bogotá