AlGlobo - Pagos

Grupo D*
Federico del Mazo - 100029
Javier Di Santo - 101696
Camila Dvorkin - 101109

Facultad de Ingeniería, Universidad de Buenos Aires
[71.59] Técnicas de Programación Concurrente I

14 de diciembre de 2021

Informe

Este informe puede ser leido tanto en PDF (gracias a `pandoc`) como en HTML (gracias a `rustdoc`)

Para documentación especifica del código fuente que excede a este informe se puede consultar la documentación de la aplicación (en inglés).

Trabajo Práctico

Introducción

En este trabajo práctico, se desarrolló la continuación de Alglobo.com desarrollada en la primera étapa de la materia. En este caso, se debe resolver un sistema de procesamiento de pagos con la complejidad de que se trata de un sistema distribuido, es decir que los pagos no solo se deberan ejecutar concurrentemente pero ademas se deben enviar via Sokets. Por otro lado, se suma la complejidad de que tanto los agentes a los que se les envia el dinero como la plataforma de alglobo puede fallar y el programa debe poder continuar sin problema.

Ejecución

Por un lado se debe levantar el sistema de alglobo que sera el encargado de resolver todo el procesamiento de pagos y enviarselo a cada uno de los agentes en cuestión. Para levantarlo: `cargo run --bin alglobo` Por otro lado, se debe levantar el sistema de agentes(Banco, Aereolinea y Hotel) que se encargaran de recibir y procesar el pago. Para levantarlo: `cargo run --bin agents` TOdo que se puede agregar un archivo

Supuestos

- Al tratar las transacciones con un two-phase lockig, se asume que si un nodo de alglobo falla:
  - Antes de finalizar la primera fase (no termina el PREPARE), el siguiente nodo de alglobo comienza desde el principio.
  - Despues de finalizar la primera fase (imprime el PREPARE pero no el COMMIT/ABORT), el siguiente nodo de alglobo ABORTA esa transacción.
  - Durante la segunda fase(no logra imprimir COMMIT/ABORT), el siguiente nodo de alglobo ABORTA esa transacción.
  - Tras finalizar la segunda fase, significa que se completo la transacción y el siguiente nodo podra seguir con la siguiente transacción.
- Una vez que finalizan las lineas del archivo, se cierran ambos sistemas.

Implementación

Alglobo

Coordinador El sistema de AlGlobo.com es de misión crítica y por lo tanto debe mantener varias réplicas en línea listas para continuar el proceso, aunque solo una de ellas se encuentra activa al mismo tiempo. Una vez que se enciende el sistema, la terminal se queda a la espera de que el usuario ingrese un número, el identificador de la réplica, para poder simular la salida de servicio de la misma, mostrando que el sistema en su conjunto sigue funcionando. Esto se resuelve con el **algoritmo Ring**. Para resolver con este algormitmo, fue importante que cada réplica conozca el identificador de la siguiente réplica y el identificador de la réplica lider(la replica que se mantiene activa y resuelve el procesamiento de pagos). Al conocer el identificador, el mismo va a poder conocer la dirección a la cual enviar los mensajes ya que las direcciones se obtienen a partir de la siguiente función:

```
pub fn id_to_ctrladdr(id: usize) -> SocketAddr {
    let port = (1100 + id) as u16;
    SocketAddr::from(([127, 0, 0, 1], port))
}
```

Todas estas direcciones se van a utilizar para enviar y recibir mensajes via Socket UDP en torno a resolver el problema de disponibilidad del servicio: 1. Cuando un proceso nota que el coordinador fallo, tras un TIMEOUT especifico que le indica que no recibe información del lider, este arma un mensaje ELECTION que contiene su identificador y envía este mensaje al siguiente nodo de la red. Este primer mensaje se puede ver en el método `find_new()` en `src/leader_election.rs`. 2. El proceso que recibe el mensaje, agrega su identificador al final del mensaje y lo envía al siguiente nodo de la red. Esta recursión se puede ver en el método `safe_send_next` en `src/leader_election.rs`. 3. Cuando el proceso original que noto que el lider fallo recibe el mensaje que originalmente inicio él, busca entre los diferentes ids encolados aquel con identificador más alto, para así decidir quien es el lider. Arma un nuevo mensaje, pero esta vez del tipo COORDINATOR y lo envia a su sucesor. 4. Cuando el mensaje finaliza la circulación, se elimina el anillo.

Procesamiento de pagos El sistema de alglobo debe encargarse de resolver todo el procesamiento de pagos y enviarselo a cada uno de los agentes en cuestión. Para ello se abre una conexión UDP para cada uno de los procesos. En el caso de ser lider, el proceso se encargara de leer una linea a la vez del archivo pasado por párametro o el default `src/prices.csv`. Cada línea del archivo va a contener 3 números, siendo el primero el precio a cobrar al Banco, el segundo el de la Aereolinea y el tercero del Hotel. De manera concurrente y via TCP se les envia a los tres agentes el precio a cobrar. Este envio se va a resolver con **commit en dos fases**: - Fase 1: El coordinador escribe el mensaje de PREPARE y lo broadcastea a los tres agentes. Luego broadcastea al resto de los procesos que se encuentra en la fase PREPARE para la transacción corresponde. - Fase 2: A partir de las respuestas obtenidas por los agentes, el coordinador envia el mensaje de COMMIT o ABORT (de haber al menos un agente que responda ABORT, el proceso entero se debera abortar) a los tres agentes y luego lo broadcastea a todos los procesos. De esta forma garantizamos que las transacciones sean serializables, por lo que si se cae el coordinador, la replica que tome su lugar va a tener la información necesaria para terminar su trabajo y continuarlo sin notar cambios en el funcionamiento del sistema.

Agentes Cosas a decir: - la terminal los escucha pa morir - commit/abort/prepare - TCP

Conclusión

TODO: @jdisan