# 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 etapa 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 deberán ejecutar concurrentemente a los diferentes agentes pero además se deben enviar vía Sockets.

Por otro lado, se suma la complejidad de que tanto los agentes a los que se les envía 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 será el encargado de resolver todo el procesamiento de pagos y enviárselo a cada uno de los agentes en cuestión. De forma opcional, se puede pasar por parámetro un archivo txt con los diferentes precios a cobrar, de no agregar este parámetro, se utilizará el archivo default `src/prices.csv`. Para levantarlo: `cargo run --bin alglobo <archivo>`

Por otro lado, se debe levantar el sistema de agentes (Banco, Aerolínea y Hotel) que se encargaran de recibir y procesar el pago. Para levantarlo: `cargo run --bin agents`

### Supuestos

- Al tratar las transacciones con el método de commit de dos fases, 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.
  - Después 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 completó la transacción y el siguiente nodo podrá seguir con la siguiente transacción.
- Una vez que finalizan las líneas del archivo, se cierran ambos sistemas.
- Las transacciones resultarán en ABORT si alguno de los agentes se encuentra caído.

**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. Cada réplica estará representada por la estructura **AlgloboNode**.

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 algoritmo, fue importante que cada réplica conozca el identificador de la siguiente réplica y el identificador de la réplica líder (la réplica 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 mismas dependen de este. Cada réplica cuenta con dos sockets para recibir información, una para control y todo los relacionado al algoritmo de elección por los que va a enviar y recibir mensajes del tipo ELECTION, COORDINATOR, ACK (por tratarse de UDP) y otro de data por el cuál el lider enviará información de los pagos procesados a las réplicas para que estén actualizadas.

Todas estas direcciones se van a utilizar para enviar y recibir mensajes vía Socket UDP en torno a resolver el problema de disponibilidad del servicio: 1. Cuando una réplica nota que el coordinador fallo por llegar a un TIMEOUT sin obtener nueva información del líder, este arma un mensaje ELECTION que contiene su identificador y envía este mensaje a la siguiente réplica de la red. Este primer mensaje se puede ver en el método `find_new()` en `src/alglobo_nodo.rs`. 2. El proceso que recibe el mensaje, agrega su identificador al final del mensaje y lo envía a la siguiente réplica de la red. En el caso que la siguiente réplica no responda con un ACK, se intenta con la siguiente. De esta forma se asegura que el mensaje pase por todas las réplicas activas. Esta recursión se puede ver en el método `safe_send_next` en `src/alglobo_nodo.rs`. 3. Cuando el proceso original que noto que el líder fallo recibe el mensaje que originalmente inició él, busca entre los diferentes ids encolados aquel con identificador más alto, y este será el nuevo líder. Arma un nuevo mensaje, pero esta vez del tipo COORDINATOR y lo envía a su sucesor. 4. Cuando el mensaje COORDINATOR finaliza la circulación, todas las réplicas estarán al tanto del nuevo líder.

**Procesamiento de pagos** El sistema de alglobo debe encargarse de resolver todo el procesamiento de pagos y enviárselo 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 líder, el proceso se encargará de leer una línea a la vez del archivo pasado por parámetro 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 Aerolínea y el tercero del Hotel. De manera concurrente y vía TCP se les envía a los tres agentes el precio a cobrar. Este envío se va a resolver con **commit en dos fases**: - Fase 1: El coordinador escribe el mensaje de PREPARE y lo envía a los tres agentes (con TCP). Luego emite al resto de las réplicas que se encuentra en la fase PREPARE para la transacción corresponde (con UDP por la dirección de data mencionada anteriormente). - Fase 2: A partir de las respuestas obtenidas por los agentes, el coordinador envía el mensaje de COMMIT o ABORT (este último sucede si hay al menos un ABORT o hubo un error de conexión con algún agente) a los tres agentes y luego emite dicho estado a las otras réplicas.

De esta forma garantizamos que las transacciones sean serializables, por lo que si se cae el coordinador, la réplica 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** Al igual que del lado de alglobo, tras levantar el servicio de agentes la terminal se queda a la espera de que el usuario ingrese un número, el identificador del agente, para poder simular la salida de su servicio, mostrando nuevamente que el sistema en su conjunto sigue funcionando. Sin embargo, a diferencia de alglobo, los agentes no cuentan con réplicas, por lo tanto una vez que se cae uno de ellos, siempre se va a devolver ABORT.

La estructura **Agent** maneja la lógica básica de las transacciones, realizando COMMIT o ABORT de forma acorde, mientras que en `agents.rs` se levantan los servicios correspondientes donde cada uno tendrá

una estructura **Agent** asociada.

Como se meciono anteriormente, las transacciones se resuelven con commit en dos fases, por lo que cada agente va a tener que recibir dos mensajes: - Primero resuelve el PREPARE, en donde simplemente a partir del `success_rate` del agente específico devuelve si se trata de un COMMIT o ABORT. El mensaje se envía a alglobo vía TCP - Luego, tras recibir el mensaje de la segunda fase de alglobo, loguea COMMIT o ABORT según corresponda.