FACULTAD
DE INGENIERIA
Universidad de Buenos Aires

75.59 Técnicas de Programación Concurrente I

Trabajo Practico: AlGlobo.com

1er Cuatrimestre 2022

GRUPO N° 5	
APELLIDO, Nombres	**N° PADRÓN**
Giampieri Mutti, Leonardo	102358
Hojman de la Rosa, Joaquín Guido	102264
Vazquez Fernandez, Francisco Manuel	104128

Objetivo

Se deberá implementar un conjunto de aplicaciones en Rust que modele el sistema de procesamiento de pagos de AlGlobo.com.

Se debe implementar un proceso para cada una de las entidades intervinientes y estas se comunicarán entre sí por sockets.

Se debe poder simular la salida de servicio de cualquiera de los procesos y réplicas de forma aleatoria o voluntaria, mostrando que el sistema en su conjunto sigue funcionando.

Introducción

AlGlobo.com es un nuevo sitio de venta de pasajes online. Gracias al fin de la pandemia ha empezado a crecer fuertemente y necesitan escalar su sistema de procesamiento de pagos.

Para ello desean reemplazar una implementación monolítica actual con un microservicio en Rust que se encargue específicamente de este proceso.

En el presente Trabajo Práctico se implementarán los objetivos enunciados anteriormente y en la consigna utilizando las herramientas de concurrencia vistas durante la materia y propias del lenguaje.

Se detalla la implementación del trabajo, dando una visión general de cómo funciona el mismo y como se comportan los actores que intervienen en el sistema. Se presentarán diagramas que ayudan a entender cómo se relacionan los mismos y se comentarán las herramientas utilizadas.

Repositorio

Se adjunta el repositorio público de Github que contiene el código fuente del proyecto:

https://github.com/joaquinhojman/AlGlobo.com

Implementación

A continuación se dará un vistazo general de la aplicación y se comentará cómo se comporta cada actor. Vale aclarar que la explicación siguiente corresponde a una replica de AlGlobo, en particular la única que está ejecutándose. Más adelante detallaremos en otra sección el Algoritmo de Elección de Líder.

AlGlobo

Una vez que alguno de los procesos llama a Main lo primero que debe hacer es levantar los actores que tomarán parte en la ejecución del programa. Estos actores son los siguientes y tienen, a grandes rasgos, la siguiente tarea:

- Logger: Se encarga de loggear las acciones del sistema en un archivo de log.
- File reader: lee el archivo de transacciones.
- Transaction Dispatcher: prepara una línea del archivo para ser enviada a las entidades.
- Entity Sender: Envía información a las entidades.
- Entity Receiver: Recibe información de las entidades.
- Coordinator: Coordina las transacciones que se reciben y se envían. Lleva la cuenta de que transacciones se commitearon y cuales se abortaron.
- Statistics Handler: Imprime las estadísticas del sistema en cuanto a transacciones.
- FIleWriter: escribe un archivo con las transacciones fallidas y commiteadas.

A continuación se detalla el flujo del programa, vale destacar que la aplicación de AlGlobo.com está basada internamente en el modelo de Actores.

Una vez inicializados todos los actores, el primero en comenzar a ejecutarse es el actor File Reader, quien toma el archivo de transacciones que se le pasó al programa por parámetro y lee línea por línea hasta el final del mismo. Cada línea que lee se la envió en un string record al actor Transaction Dispatcher.

El actor Transaction Dispatcher procede a deserializar la línea, leyéndola y convirtiéndola en un Transaction Request, un objeto que tiene el id de la transacción, y el costo para cada entidad. Luego el Dispatcher le envía este objeto al actor Entity Sender.

Lo primero que hace el actor Entity Sender es avisarle al Coordinator qué va a enviarle a las entidades una transacción. Aquí comienza la primera parte del algoritmo de transaccionalidad, la fase del PREPARE. Lo que hace el Coordinator entonces es setear esa transacción como Wait, es decir que está esperando que las entidades la procesen y den su respuesta. El Coordinator tiene un cierto tiempo donde espera la respuesta de las entidades, y en caso de no obtenerla, se marca la transacción como abortada y se le notifica dicha información a cada entidad. Una vez que el Entity Sender le aviso al Coordinator de la nueva transacción, procede a enviarle la misma a cada entidad.

Una vez que las entidades procesan la transacción y responden la misma, esa información es recibida por el Entity Receiver, actor cuya función es castear esa información a un objeto Transaction Reponse, el cual además del id de la transacción posee el estado que respondió cada entidad para la misma, y pasarle la data al Coordinator, el cual recordemos que estaba esperando estas respuestas.

Una vez que el Coordinator recibió la respuesta de una transacción para cada una de las 3 entidades del programa, comienza la segunda fase del algoritmo de

transaccionalidad, la fase donde la transacción queda en estado de COMMIT o en estado de ABORT. El coordinador lee la respuesta de las tres entidades, si las tres fueron COMMIT el estado de la transacción será COMMIT, en caso de que al menos una entidad haya respondido con un ABORT, la transacción debe abortarse, esto se debe a que si alguna falla, se debe mantener la transaccionalidad y por lo tanto revertir o cancelar apropiadamente.

Sea cual sea el caso, hay que avisarle a las 3 entidades de esto, por ende se le envía al Entity Sender un objeto Broadcast Transaction con el id y estado de la transacción. El sender se encarga de hacerle llegar esta información a cada entidad. Si bien las entidades nuevamente responderán algo, el algoritmo de transaccionalidad está indefinido para el caso donde falla un Commit, por lo que cuando el Coordinator le envió el Broadcast Transaction al Entity Sender, se desentendió de esta transacción y ya la considera finalizada. Finalizadas estas acciones ya quedó completo el procesamiento de una transacción.

Para el caso particular donde una transacción falló, se le notifica al actor FileWriter cual fue la línea que terminó abortandose y dicho actor la escribe en un archivo para su posterior procesamiento manual. En caso de que la transacción se haya Commiteado, se escribe en un archivo de transacciones Commiteadas.

De esta manera se procesan las transacciones de cada línea del archivo. Durante todo el procesamiento el logger fue escribiendo las acciones más importantes del sistema que permiten dar cuenta de que fue pasando durante dicho procesamiento. También el Statistic Handler fue recibiendo la información de las transacciones y periódicamente fue imprimiendo por consola las estadísticas de las mismas.

Entidades

Por otra parte, a continuación se detalla el funcionamiento de las entidades, las cuales tienen exactamente el mismo comportamiento cada una de ellas, con la diferencia de que poseen diferentes address. Vale destacar que las entidades también escriben su propio archivo de log que da cuenta del desarrollo de las transacciones que reciben y responden.

Cada entidad funciona en un loop donde se dispone a escuchar transacciones que le lleguen. Una vez que recibe algo lo castea a un objeto Entity Payload, que posee el estado de la transacción, el costo para esa entidad y el id. Lo que debe verificar ahora es si esa transacción que le llegó es una transacción nueva, que aún no proceso, o es un commit o abort (diciéndole que confirme una transacción anterior o la aborté, respectivamente). Para esto la entidad utiliza un diccionario con el id de la transacción como clave y el estado que le asignó anteriormente (si lo hizo) como valor.

Revisa entonces el estado de la transacción que llegó, la cual puede ser un PREPARE (transacción nueva), un COMMIT (confirmación de una transacción vieja) o un ABORT (debe abortarse una transacción vieja). Veamos cada caso:

En caso de que la transacción recibida sea un Prepare, unicamente podriamos tener en el diccionario de la entidad un Commit (o Accept) o un Abort en caso de que esten mandando una transacción que ya se proceso y finalizo en el pasado, por ejemplo porque alglobo se cayó y volvió a levantarse sin tener en cuenta el estado de las transacciones anterior. Si se diera ese caso la respuesta tendrá el mismo estado que figuraba en el diccionario. El caso “normal” es que si la transacción tiene estado Prepare, aún no haya nada en el diccionario con ese id. En ese caso es donde la entidad pueda aceptar o abortar la transacción, las posibilidades de que la transacción “falle” son del 10% y en ese caso se guarda un Abort en el diccionario y se marca ese mismo estado en la respuesta. En el caso de que la transacción no falle se guarda un Accept en el diccionario (que indica que esa entidad aceptó la transacción pero aun no le confirmaron que el resto de las entidades también lo hicieron) y responde un Commit. Cabe aclarar que el estado Accept no se envía por socket, es solo para uso interno de la entidad para el uso antes declarado.

En caso de que el estado de la transacción sea Commit, es decir las 3 entidades procesaron la transacción correctamente, quiere decir que la misma quedó confirmada. Si en nuestro diccionario teníamos un Accept lo reemplazamos por un Commit, que simboliza justamente que la transacción terminó, y respondemos con un Commit. En caso de que ya tuviéramos un Commit, no escribimos nada en el diccionario y respondemos Commit. No es posible el caso que tuviéramos un Abort en el diccionario, ya que si la transacción hubiera fallado para esa entidad no podrían mandarnos un Commit para la misma.

Por último, puede llegarnos un Abort, que indica que alguna de las entidades rechazó la transacción y por ende si alguna la había aceptado debe revertir dicha transacción. Por ende si en el diccionario había un Accept (es decir que no fue esa entidad quien fallo) se reemplaza por un Abort y se responde también un Abort. Si había un Abort en el diccionario (quien falló fue esa entidad) no se cambia nada y se responde un Abort. No es posible que tuviéramos un Commit ya que eso simboliza que la transacción finalizó en el pasado y actualmente estamos recibiendo una finalización errónea.

En cada caso se enviará por socket la respuesta correspondiente para que la reciba el Entity Receiver del lado de AlGlobo.

Resiliencia

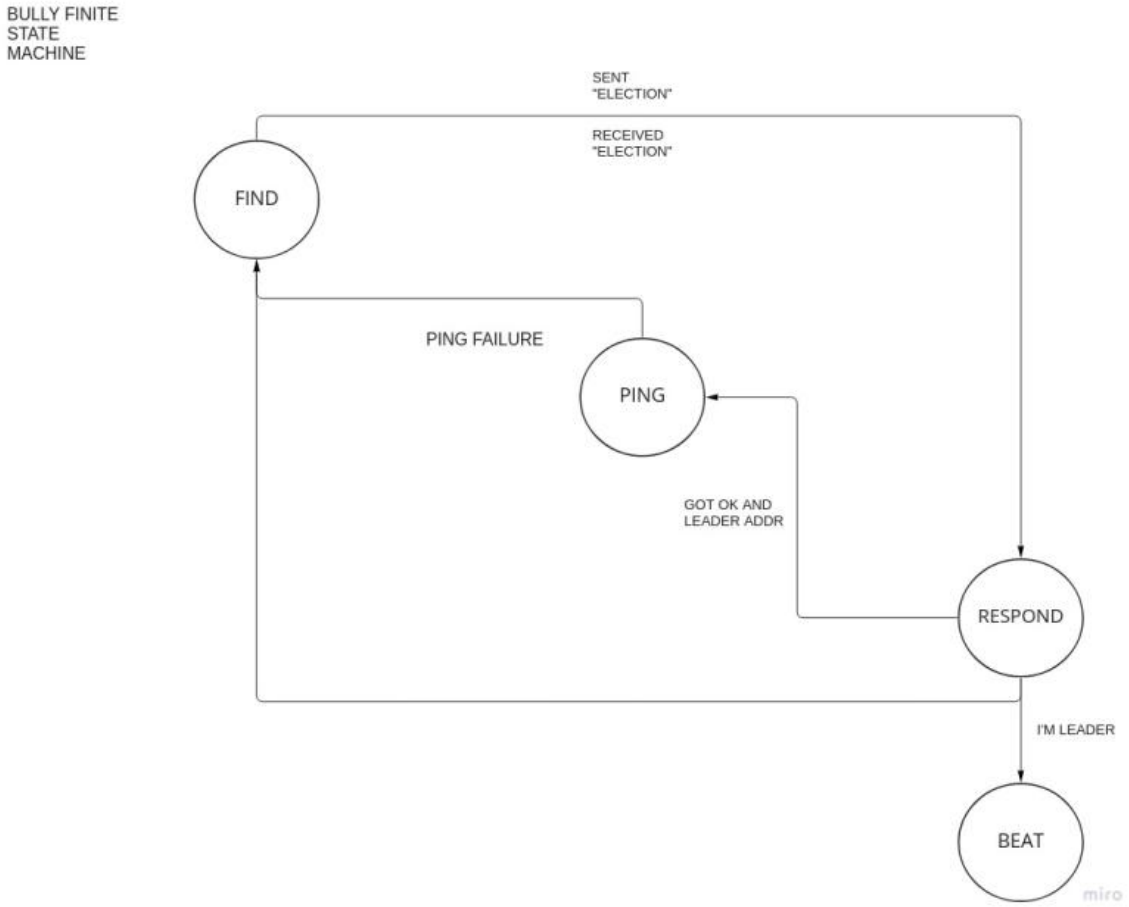
El sistema de AlGlobo.com es de misión crítica y por lo tanto debe mantener varias réplicas en línea listas para continuar el proceso, aunque solo una de ellas se encuentra activa al mismo tiempo. Para ello utiliza un algoritmo de elección de líder y mantiene sincronizado entre las réplicas la información.

Para implementar la resiliencia de la aplicación decidimos utilizar el Algoritmo Bully visto en la clase teórica y en la clase práctica. El mismo funciona de la siguiente manera:

Cuando un proceso P nota que el coordinador no responde, inicia el proceso de elección:

1. P envía el mensaje ELECTION a todos los procesos que tengan número mayor.
2. Si nadie responde, P gana la elección y es el nuevo coordinador.
3. Si contesta algún proceso con número mayor, éste continúa con el proceso y P finaliza.
4. El nuevo coordinador se anuncia con un mensaje COORDINATOR.

Siempre gana el proceso con mayor número de PID.



El proceso que gane la elección, es decir el coordinador, será el que pueda ejecutar la aplicación de AlGlobo, y lo hará hasta que por algún motivo se caiga, momento en que se elegirá un nuevo líder y se seguirán procesando transacciones.

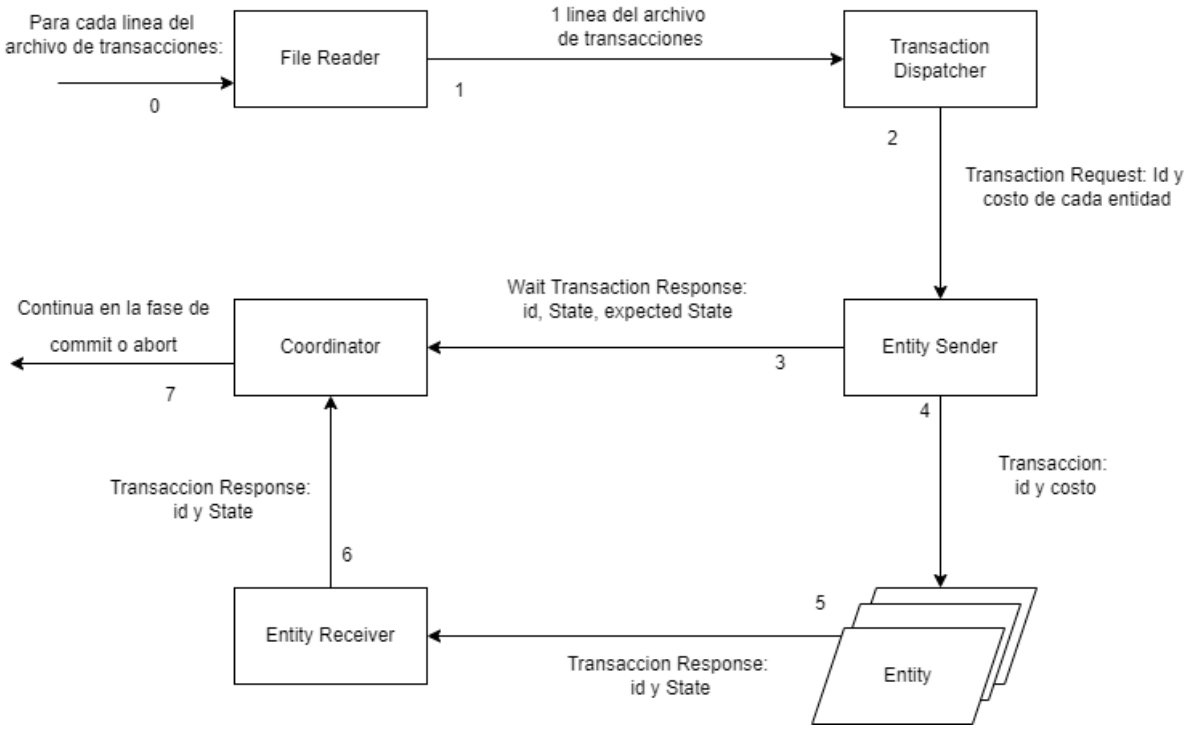
Una vez que una nueva replica de AlGlobo comienza a ejecutar el programa, empieza a leer las transacciones que AÚN no fueron hechas, basándose para eso en los archivos de transacciones fallidas y realizadas.

Diagramas de Entidades

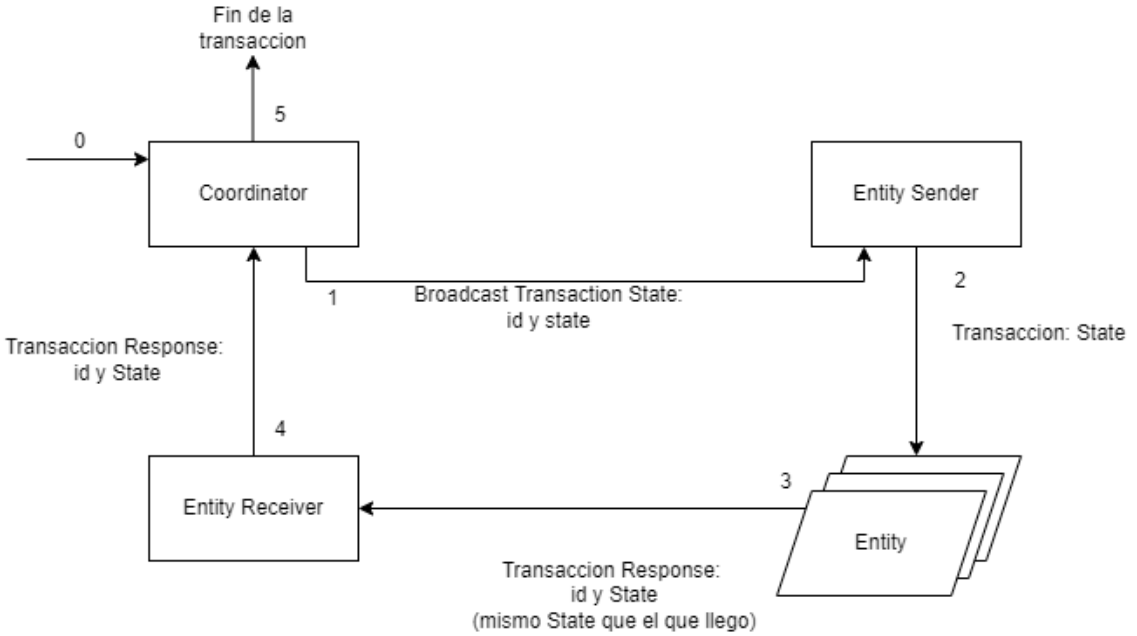
A continuación se detallan dos diagramas que muestran la interacción entre actores durante la ejecución del programa. Para mayor entendimiento se divide en dos

diagramas que muestran, respectivamente, la interacción de los mismos durante la fase de Prepare del algoritmo de transaccionalidad y la interacción de los mismos durante la fase de Commit o Abort.

FASE DE PREPARE

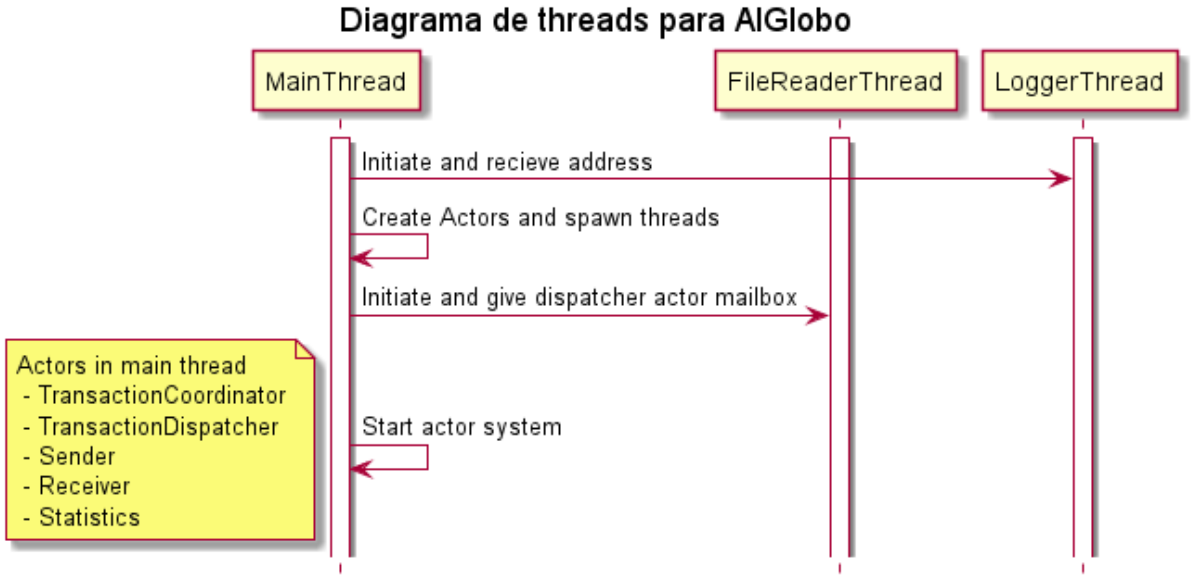


FASE DE COMMIT O ABORT



Diagramas de Threads

Se muestra ahora un diagrama de Threads del sistema en ejecución. Cabe destacar que únicamente hay 3 hilos ejecutándose concurrentemente. El motivo de que el Logger y el File Reader se ejecuten en un thread tiene que ver con que interactúan con archivos que escriben y leen, respectivamente. Luego, todos los demás actores se ejecutan en un único hilo.



Herramientas utilizadas

El código está escrito en Rust utilizando los crates externos de actix y tokio para manejar los actores y eventos asincrónicos. También se utilizan los crates ‘csv’ y serde para parsear el archivo csv de las transacciones. Con la intención de reutilizar el código que más de una entidad utiliza se nuclearon varias estructuras en una carpeta de commons.