Reporte del trabajo Distributed E-Tournaments

Integrantes

- Arian Pazo Valido C-411
- Jorge Alberto Aspiolea C-411

Requerimientos

Implementar un sistema distribuido que permita la organización de torneos de un juego determinado donde se pueda utilizar la capacidad de cómputo de varios equipos.

El sistema debe permitir definir desde un cliente específico los jugadores virtuales y el tipo de torneo a efectuar; así como la consulta y visualización de todos los resultados y estadísticas del torneo.

Se debe preveer de un mecanismo de tolerancia a fallas de forma que el desarrollo del torneo no sea interrumpido por el fallo de un equipo donde se estén efectuando un conjunto de enfrentamientos.

Se debe preveer que se puedan desarrollar varios torneos a la vez.

• Arquitectura distribuida. • Tolerancia a fallas. • Migración de código de los jugadores virtuales al nodo donde se efectúen los enfrentamientos. • Capacidad para definir varias modalidades de torneos (todos contra todos, eliminación directa, por grupos...etc) • Acceso desde el cliente a la información del desarrollo del torneo y las estadísticas del mismo. • Capacidad para desarrollar varios torneos simultáneamente.

Implementación general

No se utiliza ninguna biblioteca que maneje sistemas distribuidos. Todos los procesos de comunicación son a través de sockets . La asincronía y la tolerancia a fallas son con funciones propias que utilizan multiprocessing.

En el diseño de el sistema distribuido fueron creado 5 tipos de nodos: dns, database, client, server, minion. Separando así las responsabilidades de forma independiente.

El nodo de tipo **dns** es único en el aspecto de que solo un nodo será creado, su función es cumplir con el comportamiento regular de un dns registrar la dirección ip y el puerto de los nodos que se comunican con él, revisar periodicamente si las direcciones almacenadas aún son parte de la red, así como proporcionar las direcciones disponibles para un dominio determinado. El nodo **dns** siempre se crea en el puerto `5353` y con ip fijo: `172.18.0.250`; replicando el comportamiento natural de un dns donde toda la red conoce la dirección de este.

Los nodos de tipo **database** se encargan de controlar todo lo relacionado con el acceso a datos. Gestiona todas las peticiones de insertar, editar y consultar datos de los torneos. Los nodos de tipo **database** utilizan el puerto `8040` y el rango de ips: `172.18.0.120-172.18.0.169`.

Los nodos de tipo **server** gestionan el desarrollo de los torneos, la actualización de estos en la base de datos, así como las consultas sobre el estado de los torneos por parte de los clientes. Son los que ordenan la ejecución de nuevas partidas. Además de poder crear un torneo desde 0 también pueden cargar un torneo incompleto y terminar su ejecución (sería necesario si el server que se encargaba de ese torneo se desconectó de la red). Los nodos de tipo **server** utilizan el puerto `8080` y el rango de ips: `172.18.0.20-172.18.0.69`.

Los nodos de tipo **minion** se encargan de ejecutar partidas individuales según los nodos server lo necesitan. Para esto reciben el identificador de los jugadores a partir del cual se obtiene de la base datos las funciones necesarias para ejecutarlos. Los nodos de tipo **minion** utilizan el puerto `8020` y el rango de ips: `172.18.0.70-172.18.0.119`.

Los nodos de tipo **client** tienen como función realizar los pedidos de crear nuevos torneos, así como obtener el estado de un torneo, dado su identificador único, a los nodos de tipo servidor. Los nodos de tipo **client** utilizan el puerto `5000` y el rango de ips: `172.18.0.2-172.18.0.19`.

Instalación y ejecución

Solo es necesario tener instalado docker.

Se debe crrear una red de docker con el comando:

```
docker network create --driver bridge --subnet=172.18.0.0/24 tournament-net
```

Para crear un nuevo nodo es necesario estar en la carpeta del proyecto y ejecutar, de los comandos siguientes, el que corresponde al nodo a crear.

dns container:

```
cd dns
docker build -t dns .
docker run -it --name dns --net tournament-net --ip 172.18.0.250 -p 5353:5353 dns
```

client container:

```
cd client
docker build -t client .
docker run -it --name <container_name> --net tournament-net --ip <ip_address in
172.18.0.2-172.18.0.19> -p 5000:5000 client
```

server container:

```
cd server
docker build -t server .
docker run -it --name <container_name> --net tournament-net --ip <ip_address in
172.18.0.20-172.18.0.69> -p 8080:8080 server
```

minion container:

```

```

```
cd minion
docker build -t minion .
docker run -it --name <container_name> --net tournament-net --ip <ip_address in
172.18.0.70-172.18.0.119> -p 8020:8020 minion
```

database container:

```
cd database
docker build -t database .
docker run -it --name <container_name> --net tournament-net --ip <ip_address in
172.18.0.120-172.18.0.169> -p 8040:8040 database
```

Para crear nodos de un tipo que ya al menos uno exista:

- no puede tener el mismo --name o --ip.
- no es necesario hacer `docker build`, con solo el `docker run` es suficiente.

Explicación detallada de los tipos de nodo del sistema distribuido

dns: permite resolver dinámicamente como encontrar a otros nodos. Atiende dos peticiones: `POST` (que añade una nueva dirección) y `GET` (que se encarga de la resolución de nombre); además de mantener un hilo donde comprueba cada un tiempo determinado si las direcciones almacenadas siguen activas.

Cada registro del **dns** contiene: `domain` (indica el tipo de nodo), `ttl` (indica por cuántos segundos almacenar el registro), `data` (tupla (ip,port) almacenando la dirección) y `added_at` (con el tiempo en el que fue añadido el registro).

La resolución de nombre permite obtener todas las direcciones IP de los nodos que sean del dominio especificado.

Para mantener la consistencia de los registros, en un proceso en un hilo, cada 5 segundos se comprueba si el `ttl` de algún registro ha expirado. En ese caso se hace un ping al nodo en cuentión, si sigue activo se actualiza `added_at` con el tiempo actual.

Todos los nodos, excepto el client, al unirse a la red contactan al dns para añadir su dirección. Además cuando un nodo detecta que se desconectó ejecuta nuevamente este protocolo.

---

database: utilizando SQLite como alternativa sencilla y ligera se almacenan los torneos (con el tipo y si está finalizado), por cada tipo de torneo las partidas (con los jugadores, si fue jecutada y el ganador), además de almacenar los players de cada torneo. Para asegurar el correcto desarrollo de los torneos, en un proceso en un hilo, cada un tiempo determinado se comprueba si en la base de datos existe algun torneo que no ha sido actualizado en un tiempo relativamente grande, se asume que el servidor que estaba gestionando dicho torneo esta caido, o que por otras razones esta procesando muy lento. En ese caso se hace una peticion a un nodo de tipo servidor para que continue con la ejecucion de dicho torneo.

Atiende 9 peticiones fundamentales:

- 'save_match': dado el tipo de torneo, el id del torneo, los id de los payers y cualquier información extra necesaria se crea una nueva partida. Si se recibe el id de partida en lugar de crear una nueva se actualiza en base de datos.
- 'get_match': dado el id del torneo y el id de la partida se lee de la base de datos y se envía la información de la partida en cuestión.
- 'add_players': dado el id del torneo y una lista con la información de cada jugador se insertan en la tabla de `participants`.
- 'get_player': dada una lista de ids de jugadores se leen los records de la tabla `participants` y se envía una lista con la información de cada uno de los jugadores.
- 'insert_tournament': crea un nuevo torneo a partir del tipo de torneo y la lista de jugadores al final envía el id del torneo creado y de los jugadores.
- 'get_tournament': dado el id de un torneo se obtiene el estado y los jugadores del torneo
- 'save_tournament': dado el id de un torneo actualiza el estado. Esto se utiliza generalmente para señalizar que el torneo se acabó.
- 'get_tournament_matches': dado el id de un torneo y el tipo de torneo se obtiene la lista de todas las partidas; tanto las que han sido ejecutadas como las que no.
- 'get_tournament_status': dado el id de un torneo se envía el estado, escencialmente si ha sido completado o no.

---

server: maneja las peticiones de los clientes de comenzar nuevos torneos o consultar el estado actual de un torneo determinado. Gestiona la ejecución de torneos, utilizando nodos minion para jugar las partidas en un orden apropiado.

Este nodo puede iniciar la ejecución de un torneo o continuar uno. Dado que el algoritmo consiste en encontrar una partida que se pueda jugar actualmente (o sea, si todas las partidas que se requieren para jugar esta están terminadas). Y es independiente de si este nodo fue el que inició el torneo o no.

Atiende 3 peticiones fundamentales:

- 'new_tournament': dado el tipo de torneo y una lista de jugadores, se crea un nuevo torneo, se almacena en base de datos y se comienza la ejecución del torneo.
- 'continue_tournament': dado el id de un torneo se carga de la base de datos la información de las partidas y se continúa la ejecución del torneo.
- 'tournament_status': dado el id de un torneo se carga de la base de datos el estado actual del torneo y sus partidas. Se utiliza para enviarle al cliente dicha información.

---

minion: ejecuta una partida de TicTacToe a partir del id de los jugadores que deben jugarla. Se utiliza la funcion extraida de la base de datos para la ejecucion de cada jugador. Al final se envía al server que ordenó la partida el id del ganador.

---

client: aplicación de consola que dado el tipo de torneo de entrada y la lista de jugadores envía una petición a un nodo server para ejecutar la partida. Además consulta la información de del estado de un torneo y sabe como mostrar esa información de acuerdo al tipo de torneo.

Tolerancia a fallas

Fueron implementadas varias medidas para la tolerancia a fallas:

- se garantiza el correcto funcionamiento del sistema siempre que haya al menos un nodo de cada tipo en la red.
- existen mecanismos en el envío y la recepción de datos para hacer reintentos si ocurre un error de conexión, haciendo transparente para el usuario las desconexiones por cortos períodos de tiempo; donde el nodo puede reconectarse y completar la tarea.
- en general se detectan los errores, tanto internos del funcionamiento de los nodos, como de conexión. Se tratan adecuadamente para que el cliente no vea fallos catastróficos.
- el dns envía una lista con todos los nodos del dominio pedido, permitiendo transparentemente realizar reintentos con otro nodo del mismo tipo para la operación que se requería, esto es en caso de que las medidas anteriores no sean suficientes.
- al enviar información por la red, si ocurre una desconexión, el nodo que envía toma un tiempo y reintenta la operación desde el inicio. Permitiendo que el sistema se repare solo y continue el funcionamiento normal de la red.
- al utilizar socket tcp se añade los mecanismo de tolerancia a fallas de este protocolo.

Migración de código

Se implemento un sistema de migración de código tipo REV (Remote Evaluation). El cliente tiene el codigo de los jugadores virtuales y le envia al servidor al crear un nuevo torneo dicho codigo por cada jugador. Luego el nodo de ejecutar las partidas (Minion) ejecuta el codigo provisto por el cliente, que mientras tanto esta almacenado en la base de datos.

Para que el codigo sea funcional debe cumplir con dos requerimientos:

- Debe recibir una matriz de 3x3 (la representacion del tablero actual) y un valor que representa a quien le toca jugar
- Debe retornar 3 valores que representan, la fila y la columna en la que jugará y que simbolo corresponde

Base de datos con transacciones

Los nodos database utilizan operaciones de transacción al acceder y modificar la base de datos. Por ejemplo para introducir nuevas filas solo se hace commit cuando el conjunto de todas las filas a insertar estén correctamente en la base de datos, en caso de error en el procedimiento no se hace commit, anulando la operación.

Replicación y consistencia de los datos

Balance de cargas

Cuando hay más de un nodo de tipo database, server o minion entran en acción ideas de balance de cargas, comenzando porque al elegir a cuál de los nodos realizar la petición, se escoge aleatoriamente con distribución uniforme. Obteniendo así el mejor resultado posible si no se cuenta con información sobre que carga tiene cada nodo.

Como el nodo servidor procesa los torneos leyendo de la base de datos el estado de las partidas, y decide cual es la siguiente partida a jugar entre las que cumplan todos los requerimientos para ser ejecutadas.

Nuestro sistema permite ejecutar el mismo torneo en mas de un nodo servidor simultaneamente, ayudando asi como el rendimiento general del sistema

Seguridad

Un cliente solo puede consultar información sobre los torneos de los que tiene identificador, o sea, los que ha ordenado a crear él.

Protocolos de comunicación con códigos privados para que cualquiera no pueda unirse, leer o hablar con los nodos de la red.