E-Tournaments

Jorge Soler González, Ernesto Alfonso Hernández y Abraham González Rivero

Facultad de Matemática y Computación, Universidad de La Habana, La Habana, Cuba

1. Arquitectura

El sistema está diseñado con el enfoque de cliente-servidor, ademas cuenta con un middleware que su función es hacer de intermediario entre el cliente y los servidores de la red. Este modelo de diseño de software permite que las tareas se distribuyan entre los proveedores de servicios (servidor) y los demandantes (clientes). El cliente solicita varios servicios al servidor, y este se encarga de dar la respuesta demandada por el cliente.

2. Servidor

Cada servidor dispone de un proceso de verificación que mantiene la conectividad enlazada en la red. La forma de realizarse es consultar si su predecesor inmediato se encuentra activo en el sistema. En caso de verificarse como falso, actualiza como su sucesor el segundo predecesor que contiene en su tabla distribuida de conexiones y asegura la consistencia para este y para su previo.

A continuación se enumeran los distintos recursos o servicios que brinda un servidor propio del sistema:

- AddTourServer
- SetTableConnection
- GetNodeConnection
- Insert
- IsConnect
- GetServerData
- UpdateData
- Ping
- Active
- FindTournament
- FindAvailableServer
- create-tournament
- Available
- SetEnv
- finish-tour
- execute

Cada servidor de torneo tiene la posibilidad una vez que comience un torneo de añadir 2 servidores de partidas a la red, en los cuales se ejecutan las partidas de dicho torneo.

Mediante el servicio AddTourServer es que se efectúa la inserción de un servidor nuevo a la red, especificando la dirección del servidor al que se desea conectar.

2.1. Estructura

Un servidor se compone de una tabla basada en una estructura de hash distribuida, conocida por las siglas DHT (del inglés, Distributed Hash Tables), donde se registra la dirección para acceder a los dos servidores inmediatos que le preceden(next1 y next2) y el que le antecede(previous). Si solo existe un servidor activo en la red, entonces dicha tabla contiene valores nulos en estos campos.

Cada servidor actúa como depósito de datos mediante la inclusión de dos bases de datos en su estructura. En la primera almacena su información propia, que hace referencia a los torneos efectuados en él, y en la segunda base de datos, la información correspondiente de los datos heredados, como parte de la concepción del modelo de sistema distribuido diseñado. Además se puede mandar a crear un torneo desde cualquier servidor de la red, incluso si el servidor al que se le hace la petición ya tiene un torneo en ejecución, ya que cada servidor tiene la responsabilidad de buscar si hay algún servidor disponible para ejecutar el torneo en la red.

2.2. Inserción

La inserción de un nodo o servidor al sistema se efectúa a través de una petición a cualquier servidor que se encuentre en la red del sistema. Veamos como se realiza la inserción de un nuevo servidor a la red. Supongamos que tenemos un servidor A en la red y un servidor B hace una petición de conexión a la red. Como resultado de esto $previous(A) = B$, $next1(A) = B$, $next2(A) = none$, entonces B actualiza su tabla de distribución y $previous(B) = A$, $next1(B) = A$, $next2(B) = none$.

Si un servidor nuevo, llamémosle C realiza una petición de conexión al servidor B, entonces la nueva red de conexión del sistema resulta:

$previous(A) = C$, $next1(A) = B$, $next2(A) = C$

$previous(B) = A$, $next1(B) = C$, $next2(B) = A$

$previous(C) = B$, $next1(C) = A$, $next2(C) = B$

Luego al añadir más servidores la red se mantiene consistente mediante esta estructura.

2.3. Eliminación

Si se desconecta del sistema uno de los servidores, se debe lograr mantener la consistencia de este. Por ejemplo, si se desconecta el servidor B de la red anterior: el nodo A actualiza al nodo C como su nuevo sucesor, y su segundo predecesor sería el sucesor inmediato de C.

2.4. Torneo

Cada servidor tiene la posibilidad de crear un torneo y ejecutarlo, cada vez que se crea un torneo, en principio se crea un entorno para este, que se basa en añadir 2 servidores de partidas. A la hora de ejecutar el torneo se reparten las partidas para que se ejecuten en esos servidores de partidas con las siguientes políticas:

- En caso de demorarse mucho una partida asumimos que el servidor en cuestión falló por lo que lo sacamos de la red, e iniciamos otro
- Cada vez que falle una partida esta se repite.
- Se lleva constancia en todo momento de las partidas que faltan por ser ejecutadas, para en caso de que falle el servidor, el torneo pueda continuar en otro

Al tener varios servidores de torneo y por como está estructurada la red, se permite la ejecución de varios torneos a la vez, así como dentro de cada torneo se ejecutan las partidas en paralelo.

3. Replicación

3.1. Inserción

Cuando un nodo ingresa a la red, adquiere dos bases de datos: una que referencia a los datos heredados de la información propia de su antecesor y otra que representa su información propia.

Durante el proceso de conexión del nodo B al nodo A este recibe como datos heredados la información propia del nodo A, y a su vez este recibe los del nodo B, que en primera instancia de la inserción son nulos.

Luego, si el nodo C se conecta a la red a través del nodo A, entonces se procede con el mismo método de replicación de información, transfiriendo los datos propios del nodo A hacia la correspondiente base de datos heredados de su sucesor inmediato C, y en el nodo B se almacenaría como datos heredados la información propia de C.

3.2. Eliminación

Si se efectúa el proceso de eliminación del servidor C, se debe lograr replicar la información de este hacia su antecesor y sucesor inmediato. La información propia del nodo A se le adiciona a los datos heredados del servidor B, y la información heredada que este contenía previamente del nodo C se le adiciona a los datos propios del nodo A, proporcionándole así al sistema un mayor rendimiento, disponibilidad y escabilidad mediante dicho mantenimiento. Además una vez que se desconecte un servidor, la red debe ser capaz de continuar con el torneo que se estaba efectuando en dicho servidor en caso que lo requiera. Todo esto se realiza a nivel de servidores por lo que es transparente al cliente.

4. Middleware

Esta capa se encarga de comunicarse con el cliente(interfaz de usuario) y gestionar los servidores presentes en la red. Añadir, eliminar o actualizar los servidores en caso de ser necesario, Otra de las funciones es crear el entorno del sistema que se basa en los contenedores de docker que simulan la red, cada contenedor actúa como servidor, esto se realiza a través del API de docker para python. La comunicación se realiza mediante el protocolo HTTP, cada servidor funciona como un API-RestFull que provee los recursos y servicios para quien lo necesite, esto con el objetivo de un mejor chequeo del flujo de los datos y el funcionamiento de la red.

5. Requerimientos

- Docker: para la simulación de la red se necesita docker con las imágenes correspondientes(tournament-server, match-server)