Sistemas Distribuidos

E-Tournaments

Andry Rosquet Rodríguez
Kevin Talavera Díaz
Sergio Pérez Pantoja

Cuarto año. Ciencias de la Computación. Facultad de Matemática y Computación, Universidad de La Habana, Cuba

Resumen

El proyecto desarrollado implementa un simulador de torneos de juegos de dos contra dos donde sus jugadores son virtuales, los cuales se ejecutan sobre un sistema distribuido.
Palabras claves: sistema, distribuido, torneo

1. Introducción

Un sistema distribuido es un conjunto de equipos independientes que actúan de forma transparente formando un único equipo. Su objetivo es descentralizar tanto el almacenamiento de la información como el procesamiento. Este proyecto, de manera general, implementa una arquitectura DHT Chord con algunas modificaciones, un algoritmo de replicación de información, y otro de balanceo de carga para la correcta y eficiente ejecución de diferentes tipos de torneos creados por múltiples clientes.

2. Torneos

Fueron implementados dos tipos de torneos: clasificación y !!!!!!!!(Rellena con el segundo tipo de torneo)!!!!!!!!!, pero esta implementación es extensible a la incorporación de nuevos tipos de torneos sin modificar el funcionamiento del sistema. Para crear un nuevo tipo de torneo basta con seguir los siguientes pasos:

1. En *tournament.py* crear una nueva clase con el torneo que se desee implementar y heredar de la clase abstracta tournament e implementar el método *create_matching*().

2. En *client.py* añadir la opción de crear el nuevo torneo añadido.

3. Juegos

Fueron implementados dos juegos, el Nim y el Tic-Tac-Toe, al igual que con los torneos, esta implementación es totalmente extensible a la incorporación de cualquier otro juego deseado sin modificar el sistema. Para crear un nuevo juego basta con seguir los siguientes pasos:

1. El nuevo juego que se desee implementar debe heredar de la clase abstracta *game* que se encuentra en *game.py* y tanto las jugadas óptima y aleatoria deben ser hechas en *play.py* en las clases *optimal* y *random*.

2. En *client.py* añadir la opción de crear el nuevo juego añadido.

4. Chord

Es una Tabla Hash Distribuida utilizada para la implementación de redes peer-to peer estructuradas. Chord especifica la información que cada nodo de la red debe almacenar, referente a otros nodos, así como, la información referente a los ficheros que es compartida entre nodos. Además, regula la comunicación y el intercambio de mensajes entre estos.

4.1. Asignación de IDs a los nodos

Los nodos y los ID de los ficheros compartidos tienen asociado un identificador binario de m bits. El valor de m es un parámetro configurable en la creación de la instancia de los servidores en server.py. La asignación se realiza mediante la función hash SHA-1. Los identificadores de los nodos se representan en un anillo de identificadores módulo 2m. El rango de los valores de los identificadores de los nodos y claves oscila desde 0 hasta 2m - 1.

4.2. Lista de sucesores y predecesores

Para aumentar la robustez de la Tabla Hash Distribuida, cada nodo mantiene una lista de sucesores y predecesores de tamaño k donde k≤n y n es la cantidad total de nodos de la red. En este proyecto el valor tomado de k es k=n. Más adelante serán explicadas las ventajas y desventajas de tomar esta decisión.

4.3. Elección de =l coordinador

A diferencia de la implementación clásica de Chord, en este proyecto el nodo coordinador siempre será el nodo con mayor ID en la red, por tanto, como cada nodo tiene una lista de sucesores que contiene a todos los nodos, estos siempre estarán de acuerdo. Más adelante serán explicadas ventajas y desventajas.

4.4. Finger Table

!!!!!!!!!!!!Explica bien las entradas de la tabla, que ni yo las entiendo jajaj!!!!!!!!!!!!!!
Un nodo Chord sólo necesita almacenar información sobre unos pocos nodos de la red para localizar la clave que busca. Esta información se almacena en una tabla llamada finger table. Esta tiene m entradas, donde m es la longitud de bits de los identificadores del anillo. Cada entrada 2^i , donde 0≤i≤m-1, de la tabla finger de un nodo con identificador n está formada por el identificador del

nodo sucesor de (n+2^i) y su dirección IP.

4.5. Inserción de un nodo en la red

Este proceso debe mantener siempre tres invariantes:

1. La estructura de anillo donde el ID de cada nodo es menor que el de su sucesor, exceptuando caso donde el nodo con ID más alto se conecta al de ID más pequeño para cerrar el anillo.

2. La finger table actualizada ya que esta es de vital importancia para llevar a cabo los algoritmos de búsqueda.

3. La tabla de sucesores actualizada, debido a que en este proyecto es utilizada para mantener la consistencia del anillo en caso de falla de algún servidor, este algoritmo será explicado en detalle más adelante.

A diferencia del algoritmo de Chord clásico donde el nodo coordinador se encarga de manejar solicitudes de nuevos nodos que quieran ingresar a la red, en este proyecto la inserción de nodos ocurre haciendo uso del Multicast. Cuando un nodo quiere entrar en la red, este envía un mensaje multicast, en este punto pueden ocurrir tres casos:

1. Nadie escucha el mensaje (no hay más servidores), por tanto, este es el primero y ejecuta un método para recibir mensajes multicast de nuevos servidores.

2. Lo escucha un solo servidor que ya pertenece a la red, y en dependencia del ID del nuevo nodo, el servidor de la red crea la conexión manteniendo la primera invariante, además le envía la lista de sucesores de la red para que el nuevo nodo se actualice. Luego, en ese momento, ambos actualizan sus respectivas finger tables, de esta forma se cumplen las otras dos invariantes.

3. En la red hay más de un nodo, en este caso todos escuchan el mensaje, pero solo le responden los nodos entre los que tiene que insertarse el nuevo servidor. Estos son los que le envían la lista de sucesores y finger table al igual que en el apartado anterior. Los servidores que no tienen que ver con la inserción simplemente actualizan.

4.6. Eliminación de un nodo de la red

Al igual que en la inserción, se deben mantener las tres invariantes. Cuando un nodo sale de la red, el encargado de mantener la consistencia del algoritmo es el nodo predecesor, ya que este puede detectar cuando un nodo sale de la red casi instantáneamente. Al ocurrir esto, dicho nodo utiliza la tabla de sucesores para intentar conectarse al nodo sucesor del caído. En caso que este tampoco responda, lo intentará con el siguiente, así sucesivamente. Realizando esto pueden ocurrir dos casos:

1. No existe ningún nodo en la lista de sucesores que pueda aceptar conexión, por lo que el nodo se queda solo en la red.

2. Existe al menos un nodo al cual puede conectarse. Luego de hacerlo, le informa sobre todos los nodos que se detectaron caídos para que actualice la lista de sucesores y finger table. Finalmente, este último propaga el mensaje a los demás servidores, de forma tal que todos los nodos en la red actualicen el cambio y se pueda mantener la estabilidad de la red.

4.7. Estabilización

Por la forma de implementar los algoritmos de inserción y eliminación, no es completamente necesario un algoritmo de estabilización como el que realiza Chord. Aun así, es un método extra para mantener la consistencia de la red en caso de fallas excepcionales. Este se encuentra implementado de la siguiente manera: Usando el algoritmo de Multicast, anteriormente descrito, todos los servidores luego de un tiempo cierran su multicast durante un breve momento (no podrán recibir nuevas conexiones). Aunque no perciban ningún problema, revisarán el correcto estado de su lista de sucesores, finger table y luego vuelven a abrir la función multicast para seguir recibiendo nuevas conexiones, en caso de que ocurran.

4.8. Algoritmo de búsqueda

Un nodo n busca la información de un ID k. Inicialmente comprueba si el ID que busca está entre su identificador y su predecesor. Si k está entre su identificador y su predecesor, entonces, el nodo n es el responsable de k. Consultará su tabla de claves y se finaliza la búsqueda. Si k no está entre su identificador y su predecesor, entonces, el nodo n pasa a buscar el nodo responsable de k en su finger table siguiendo los pasos siguientes:

1. Encuentra el nodo responsable de k en su finger table. El nodo n envía la petición de búsqueda directamente a ese nodo y se finaliza la búsqueda.

2. No encuentra el nodo responsable de k en su finger table. El nodo n busca en su finger table un nodo j cuyo ID sea más cercano a k, que él mismo, ya que ese nodo podría conocer más sobre la región del anillo en la que se encuentra k. El nodo n manda la petición de búsqueda de k a ese nodo j.

Repitiendo este proceso, los nodos van descubriendo otros nodos cuyos ID están cada vez más cerca de k.
En este proyecto este algoritmo de búsqueda es utilizado para enviarle eficientemente al nodo coordinador las notificaciones de cada jugada hechas por cada servidor para que luego este se las muestre al cliente.

5. Réplica

5.1. En juegos

Cada nodo n de la red guarda una réplica del estado de los juegos que se está ejecutando en cada uno de los nodos que pertenece a su finger table. De esta forma, si n cae el primer nodo al que le llegue el mensaje avisando sobre esto, si guardaba una réplica suya, este continúa la ejecución de cada uno de los juegos por donde los había dejado n.

5.2. En torneos

!!!!!!! Rellenar !!!!!!!

5.3. Balanceo de cargas

!!!!!!! Completar !!!!!!!!

6. Análisis de las características de la implementación

6.1. Tabla de sucesores con tamaño k=n

Ventajas:

1. Mejora el proceso de eliminación, debido que, mientras más grande sea el tamaño de la tabla más difícil será romper la red.
2. Elimina la necesidad de realizar un algoritmo de elección de líder ya que todos siempre estarán de acuerdo con que el líder sea el servidor con ID más alto de la lista de sucesores.
3. Al no tener que implementar algoritmos de elección de líder se evitan sus respectivas desventajas.
4. Funciona bien en redes de servidores no muy grandes.

Desventajas:

1. Difícil de mantener para cantidades lo suficientemente grandes de servidores.
2. Al ser indispensable, tanto en la eliminación como en la elección de líder, una falla podría desestabilizar la consistencia del anillo; desembocando en un mal funcionamiento del mismo.

6.2. Uso de multicast para crear conexiones

Ventajas:

1. Elimina la necesidad de que el nodo coordinador tenga que centrarse en la inserción. Por lo tanto, disminuye su cantidad de tarea, que si bien no son muchas, estas si requieren de mucho tráfico de la red.

2. Facilita la actualización de la lista de sucesores y finger table de cada servidor. Gracias a que todos reciben el mensaje a la misma vez.

3. Disminuye el tráfico en la red orientado a enviar mensajes de un servidor a otro para saber donde puede insertarse un nuevo nodo; como se realiza en el algoritmo de Chord clásico.

Desventajas:

1. Si el receptor de multicast de un nodo que debería estar implicado en la inserción de otro falla, esto puede retrasar un poco el funcionamiento de la red, aunque luego se solucione el problema.

2. Para cantidades de servidores lo suficientemente grandes es posible que el mensaje no les llegue a todos, lo cual conlleva a la primera falla mencionada.

6.3. Replicar en la finger table

Ventajas:

1. Como la finger table siempre se encuentra correctamente actualizada, las réplicas siempre van a estar perfectamente localizadas.

2. Siempre que los servidores fallen en intervalos de al menos 2 segundos, siempre se va a poder preservar la réplica, incluso aunque solamente quede un servidor en todo el sistema.

Desventajas:

1. Si dos o más servidores fallan simultáneamente en un tiempo menor a 2 segundos, es altamente probable que se pierda alguna información. (Esta falla es controlada realizando una verificación cada cierto tiempo, y en caso de exceder ese tiempo, esa información se da por perdida, se le notifica al cliente y continúa la ejecución del programa).

Referencias