DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

TP2 - Blockchain

Sistemas Operativos

Integrante	LU	Correo electrónico
Lucas Puterman	830/13	Lucasputerman@gmail.com
Ivan Vercinsky	141/15	ivan9074@gmail.com
Alonso Tomás	396/16	tomasalonso96@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2160 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (54 11) 4576-3359

http://www.fcen.uba.ar

1. Introducción

Fue desarrollado un protocolo para la comunicación de nodos que interactúan entre sí trabajando en una blockchain. Cada nodo se encarga en paralelo de minar bloques mediante una función de proof of work, enviar los bloques conseguidos al resto de la red y escuchar mensajes de los nodos de esta. En nuestra implementación del protocolo, hay cuatro threads distintos que se ejecutan, uno general que lanza los otros y es el encargado de escuchar mensajes, uno encargado de minar, uno de agregar nuevos bloques y uno de enviar cadenas de bloques a otros nodos. Para evitar condiciones de carrera entre los threads corriendo en paralelo, nuestra implementación cuenta con un mutex general que y un mutex para el migrado de cadena cuyos funcionamientos serán explicados en el desarrollo.
Hablaremos del protocolo implementado diferenciando el mismo en cuatro aspectos: minado, recepción de mensajes, migración de cadena y finalización.

2. Minado

El minado consiste en la función de proof of work que se ejecuta en un thread que se lanza dentro de la función node al correr el programa de la blockchain. Este genera hashes modificando el campo nonce del bloque que se tiene hasta que el hash tenga un cantidad de ceros en su comienzo correspondiente a la dificultad de la red definida en la constante DEFAULT_DIFFICULTY. Este se ejecuta hasta que el último nodo aceptado tenga el índice definido en MAX_BLOCKS.

Cuando la función proof of work consigue un hash que soluciona el problema, lockea el mutex general para tener acceso exclusivo al puntero donde se guarda el último bloque de la cadena y poder enviar un broadcast del nodo minado sin procesar nuevos nodos que llegan a la red.

Si el índice del nuevo nodo es mayor al del último bloque de la cadena, este se actualiza y se envía al resto de la red de forma no bloqueante el nodo descubierto.Una vez enviado, se libera el mutex general y se vuelve a minar un nuevo nodo.

3. Recepción de Mensajes

Los nodos están siempre escuchando para recibir los mensajes de los otros nodos en la función node. Existen distintos tipos de mensajes que puede recibir un nodo, estos son: nuevo nodo, pedido de cadena, recepción de cadena y finalización.

3.1. Nuevo Nodo

Cuando un nodo (de ahora en adelante Alice) recibe el mensaje de que otro nodo (Bob) minó un bloque este es sometido a una validación que se lanza en un nuevo thread. Antes de lanzar la verificación se lockea el mutex general que es liberado al terminar la misma. En la verificación pueden suceder las siguientes cosas:

- Si el bloque no es válido (es viejo o tiene un mal hash) este se omite.
- Si Alice no tiene ningún bloque y el que recibe tiene índice 1 entonces lo acepta como primer bloque y seguirá minando el bloque siguiente al recibido.
- Si el índice del bloque recibido es el siguiente a al último bloque de Alice y su bloque último bloque también es el anterior del recibido entonces se acepta el bloque como nuevo último.

- Si el índice del bloque recibido es el siguiente al bloque actual pero el anterior del recibido no es el último bloque actual entonces hay una cadena más larga y Alice realiza un pedido de cadena a Bob para migrar a esa (*).
- Si el recibido tiene el mismo índice que mi bloque actual entonces se descarta el recibido.
- Si el índice del recibido está más de una posición adelante del último bloque en la cadena entonces Alice intenta pasarse a esa cadena más larga realizando un pedido de cadena a Bob (*).

(*)Cuando se realiza un pedido de cadena para migrar a la misma mediante la función verificar y migrar cadena se lockea un mutex de pedido que es liberado cuando se recibe el mensaje con la cadena para continuar con la ejecución.

3.2. Pedido de cadena

Cuando Alice recibe un mensaje de pedido de cadena, se lanza un thread que genera la misa y la envía. Para generar la misma, se busca el bloque pedido y se envía este junto con la cantidad de bloques anteriores definida en VALIDATION_BLOCKS.

3.3. Recepción de Cadena

Cuando un nodo (Alice) necesita migrar a otra cadena debido a que otro nodo (Bob) le envió un bloque más adelantado este hace lo siguiente: El nodo alice le envía un mensaje a bob para que este nos envíe los bloques anteriores al recibido y poder completar la cadena. Este mensaje contendrá el hash del bloque a partir del cual necesitamos los bloques anteriores. Mientras tanto alice se bloquea esperando una respuesta Cuando alice reciba el mensaje de bob con la cadena procederá a verificar que:

- El primer bloque de la lista contenga el hash pedido y el mismo índice que el bloque original.
- El hash del bloque recibido es igual al calculado por la block_to_hash.
- Cada bloque siguiente de la lista, contiene el hash definido en previous_block_hash del actual elemento.
- Cada bloque siguiente de la lista, contiene el índice anterior al actual elemento.

Si dentro de los bloques recibidos por Alice alguno ya estaba dentro de node_blocks (los bloques conocidos por Alice) o el último tiene índice 1, entonces ya puedo reconstruir la cadena. Alice agrega todos los bloques recibidos a node_blocks y marca el primero como el nuevo último bloque de la cadena. De lo contrario, se descarta la cadena y los nuevos bloques por seguridad.

El funcionamiento de verificar y migrar cadena arroja algunas conclusiones sobre el funcionamiento del protocolo en general. En primer lugar, si por algún motivo dos grupos de nodos se distancian entre sí en una cantidad de bloques (distintos) mayor a VALIDATION_BLOCKS entonces estas dos cadenas no convergerán en ningún momento ya que nunca pasarán la verificación al querer migrar de cadena. Esto puede suceder, por ejemplo, si dos nodos minan en simultáneo VALIDATION_BLOCKS bloques consecutivos y estas dos cadenas no coinciden, entonces en el siguiente bloque minado por algún nodo, el otro nodo no va a poder migrar a su cadena.

Si VALIDATION_BLOCKS no fuese una condición necesaria (yo puedo copiar toda la cadena) se vuelve casi imposible que diverjan dos cadenas, pero sin embargo existe la probabilidad de que dos grupos de nodos vayan exactamente a la par en índice pero sobre dos cadenas distintas y que no converjan.

El hecho de que un nodo se quede esperando cuando pide una cadena a la recepción de la misma es problemático también. Por ejemplo, en el caso en el que un nodo decida migrar a otra cadena, y se pierda el mensaje o se pierde su respuesta, el nodo en cuestión se quedará esperando una respuesta sin procesar nuevos nodos escuchados ni verificar nodos minados y no hay forma de retomar su ejecución normal. Esto podría ser solucionado mediante, por ejemplo, un time out que al ser agotado, continúe la ejecución rechazando la cadena o pidiéndola nuevamente.
Pudimos ver también distintos comportamientos cuando variamos la dificultad de proof of work que se pueden resumir a 3 casos:

- Cuando la dificultad es muy baja se pudo ver que rapidamente (primer bloque) un nodo pasó a ser el único minando y los otros solamente pasaban el tiempo agregando los bloques minados a sus cadenas. Creemos que esto se debe a que minar es tan fácil que el trabajo de recibir un bloque y agregarlo a nuestra cadena consume un tiempo considerable, esto causaría que el nodo que minó el bloque ya tuvo tiempo suficiente de minar otro bloque más mientras el resto lo copia a su cadena, entonces todos los otros nodos nunca llegan a estar a su altura, siempre hay una sola cadena, por lo que nunca divergen.
- Cuando la dificultad es media (entre 5 y 15) los nodos trabajan como se pretendía, minando y migrando entre sus cadenas equitativamente. Así y todo, es raro ver que dos cadenas diverjan ya que es improbable que dos cadenas distintas esten parejas por 5 bloques consecutivos como para que estas se separen sin posibilidades de volver a unirse.
- Cuando la dificultad es muy alta un bloque es minado muy esporádicamente, entonces, desde el principio, cuando un nodo mina un bloque todos rápidamente se pasan a esa cadena, es casi imposible que dos nodos minen un bloque al mismo tiempo entonces todos siempre están a la par, intentando minar y a la espera de que un nodo adelante un bloque, estas cadenas prácticamente nunca se separan y por lo tanto es raro que diverjan.

3.4. Finalización

Cuando un nodo o bien mina el bloque con índice MAX_BLOCKS, o bien acepta un nodo que lo contenga como último, se considera que ha finalizado y deberá avisarle al resto de los nodos. Para eso, lanza un broadcast al resto de la red avisando su finalización y además activará un flag de terminación que es el que controla la guarda por la cual se mina, generando así la terminación de la minería en su nodo.
A su vez, los nodos guardan una variable la cantidad de nodos que han terminado, y al recibir un mensaje de terminación la incrementan. Mientras la cantidad de nodos que terminaron sea menor que la cantidad total de nodos, se seguirán escuchando mensajes ya que es posible que reciba un nodo terminado un pedido de cadena.
Una vez que todos los nodos hayan terminado, se dejará de escuchar mensajes y se terminaran los procesos.
Aquí podemos ver otra vez como la pérdida de mensajes puede afectar la finalización ya que es posible que un nodo se quede escuchando indeterminadamente. Una forma de mitigar este problema es que en el mensaje de finalización se envíe la cantidad de nodos que sabe ese nodo que terminaron, así en caso de que un haya perdido un mensaje, lo pueda actualizar en el futuro