DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

# Trabajo Práctico 2

## Algoritmos en sistemas distribuidos

Sistemas Operativos
Primer Cuatrimestre de 2018


| Integrante | LU | Correo electrónico |
|---|---|---|
| Berríos Verboven, Nicolás | 046/12 | nbverboven@gmail.com |
| Lasso, Andrés | 714/14 | lassoandres2@gmail.com |





**Facultad de Ciencias Exactas y Naturales**
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2610 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (++54 +11) 4576-3300
http://www.exactas.uba.ar

# Índice

# 1. Introducción

En el siguiente trabajo trata de implementar los conocimientos aprendidos sobre la comunicación de sistemas distribuidos mediante el envío de mensajes. Para la resolución del mismo utilizamos el lenguaje c++ con la interfaz provista por la cátedra MPI.

El objetivo del trabajo es implementar una blockchain, la cual utiliza nodos distribuidos para proveer bloques a la misma. Estos bloques son minados por un hilo de ejecución que depende de cada nodo. Luego de ser minado, el nodo utiliza la interfaz que provee MPI para poder comunicarse con otros nodos y divulgar el bloque.

Para el desarrollo de la comunicación entre los nodos, el trabajo consiste en 4 ejercicios prácticos mas un ejercicio de análisis del protocolo.

# 2. Desarrollo

## 2.1. Inicialización

En el momento en que se inicializa un nodo, se separa la ejecución en dos threads: uno será el encargado de minar bloques utilizando la función *proof_of_work* mientras que el otro estará a la espera de mensajes provenientes de otros procesos. Para esto último se utilizó una función no bloqueante (*MPI_Irecv*), agregando a continuación una verificación de si efectivamente se se recibió alguno. De ser afirmativo, se procedió como se indica en el algoritmo 1. La decisión sobre qué respuesta adoptar se basó en el tag que acompaña al mensaje (*TAG_NEW_BLOCK* para bloques nuevos y *TAG_CHAIN_HASH* para pedidos de cadena).

**Algoritmo 1** Protocolo para la recepción de mensajes por parte de un nodo

```
si el mensaje recibido corresponde a un nuevo bloque entonces
    utilizando la función validate_block_for_chain , validar y agregar, si correspondiese, el nuevo bloque a la cadena
si no si el mensaje recibido corresponde a un pedido de cadena entonces
    reservar memoria para la cadena a enviar
    buscar la cadena y copiarla al espacio reservado
    comunicar la cadena con la función MPI_Send
    liberar el espacio utilizado
fin si
```

## 2.2. Minado de bloques

El thread encargado de minar lo hará según se describe en el algoritmo 2. Como se verá más adelante, se utiliza un mutex para evitar que no se modifique el último bloque de la cadena si llegase un mensaje de otro nodo con nuevo bloque en el trancurso de agregar el nuevo bloque.

Este thread finaliza en el momento en que detecta que ya fue agregada la máxima cantidad posible de bloques a la cadena.

**Algoritmo 2** Protocolo para minar nuevos bloques (*proof of work*)

```
genero un nuevo bloque
si el hash del nuevo bloque minado es válido entonces
    bloquear el mutex que se comparte con la función encargada de validar los bloques recibidos
    si el último bloque de la cadena no cambió mientras calculaba entonces
        agregar el bloque minado como último de la cadena
        agregar el bloque minado al diccionario < hash, bloque >
        levantar el flag para avisar al otro thread que deje de recibir mensajes
        enviar el bloque minado al resto de los nodos
        bajar el flag para avisar al otro thread que vuelva a atender mensajes
    fin si
    desbloquear el mutex
fin si
```

## 2.3. Broadcast

En el momento en que un nodo completa el minado de un bloque, y este resulta ser válido, comunica este resultado a todos los otros procesos del sistema mediante la función *broadcast_block*. En el momento en que estos reciben el mensaje, deciden si agregan o no dicho bloque a la cadena,...

El procedimiento utilizado fue:

- Se envía el bloque minado a todos los nodos (exceptuando al que envía el mansaje) con la instrucción *MPI_ISend* con la etiqueta *TAG_NEW_BLOCK*
- Se espera a que todos los nodos a los que les fue enviado el mensaje lo reciban utilizando la instrucción *MPI_Waitall*

Si bien la función *MPI_ISend* no es bloqueante, se tomó la decisión de utilizarla basándose en que, en primer lugar, permite enviar rápidamente el mensaje a todos los nodos sin necesidad de esperar por alguno en particular; en segundo lugar, la prioridad es comunicar el bloque lo más pronto posible y la función *MPI_Waitall* esperará a recibir la confirmación de todos los receptores antes de continuar. Dadas estas condiciones, puede permitirse la comunicación del bloque minado de una forma no bloqueante.

## 2.4. Validación de bloques recibidos

Al recibir un mensaje de otro nodo con un nuevo bloque (etiquetado con *TAG_NEW_BLOCK*) el receptor decide agregarlo o no a la cadena.

Debido a que -para minar un nuevo bloque- es necesario invertir cómputo en un proceso que depende de la información del último bloque de la cadena, es del interés de cada nodo aceptar en su *blockchain* a los bloques que vayan a ser aceptados por los demás nodos. A esta necesidad de apostar por la cadena de consenso, se lo conoce como *Consenso de Nakamoto*.

Al recibir un mensaje de que un nuevo bloque ha sido minado, se implementó la función *validate_block_for_chain* de acuerdo con el siguiente protocolo:

- Si el nodo no es válido (su tiempo de creación fue previo a 5 minutos antes de ser recibido o el hash es inválido), entonces se omite.
- Si el índice del bloque recibido es 1 y el último bloque de la cadena del nodo que recibe el mensaje tiene índice 0, entonces este lo agrega como nuevo último (y primero válido, puesto que no poseía ninguno).
- Si el índice del bloque recibido es el siguiente al del último bloque de la cadena del nodo que recibe el mensaje, se lo agrega como nuevo último.
- Si el índice del bloque recibido es igual al índice al del último bloque de la cadena del nodo que recibe el mensaje, entonces hay dos posibles forks de la blockchain, pero se ignora el bloque.
- Si el índice del bloque recibido es el siguiente al del último bloque de la cadena del nodo que recibe el mensaje, pero el bloque anterior apuntado por el bloque recibido no es igual al del último bloque de la cadena de este último nodo, entonces hay una blockchain más larga que la del receptor. Se piden los bloques faltantes al nodo que comunicó en nuevo bloque y se agregan a la cadena de acuerdo a lo que se menciona en la siguiente sección.
- Si el índice del bloque recibido está más de una posición adelantada a la del último bloque de la cadena del nodo que recibe el mensaje, entonces este abandona su blockchain y procede como se indica en el item anterior.

Para garantizar que no se produzca una condición de carrera con el thread encargado de minar nuevos bloques, se agregó un mutex que se bloquea en el momento en que se comprueba que el bloque recibido es válido. De esta forma, el último bloque de la cadena se actualiza correctamente según las reglas del concenso.

### 2.5. Migración de cadena

El nodo que debe migrar su cadena envía el hash del bloque recibido mediante la instrucción *MPI_Send* (bloqueante) con el tag TAG_CHAIN_HASH. El receptor envía el bloque con el hash recibido y los anteriores a ese hasta alcanzar la cantidad predeterminada VALIDATION_BLOCKS o hasta que llegue al primer bloque de su cadena. Luego de esto, envía dicha cadena con el tag TAG_CHAIN_RESPONSE. El primer nodo, al recibir un mensaje etiquetado de esa forma por parte del nodo al cuál se le realizó el pedido, procede a verificar que:

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

Nótese que la cantidad de bloques enviados por **bob** podría haber sido menor que VALIDATION_BLOCKS. Este caso se resolvió colocando en el parámetro *count* de la función *MPI_Send* el número de bloques enviados y mediante la función *MPI_Get_count* del lado receptor para acceder a dicho valor.

De cumplirse las condiciones arriba mencionadas, se realiza la migración. En caso contrario, se descarta la cadena recibida.

## 3. Análisis del protocolo

- ¿Puede este protocolo producir dos o más blockchains que nunca converjan?

  Sí, podría pasar dado que al momento de migrar cadena no controlamos desde qué momento la cadena se bifurca y en vez de enviarle la nueva cadena desde dicha bifurcación le mandamos los últimos 5, los cuales podrían no ser suficientes para migrar la cadena.

  Una situación podría darse cuando tenemos una cantidad grande de nodos y que al propagarse el bloque nuevo, existan nodos que ya estén minando sobre ese bloque antes de terminar de propagarlo por lo que agranda la brecha entre la longitud de la cadena de cada nodo hasta que finalmente llega el punto en que esa brecha supera los últimos 5 nodos y no logran propagarse todos para migrar la cadena completamente.

- ¿Cómo afecta la demora o la pérdida en la entrega de paquetes al protocolo?

  La demora en la entrega o pérdida de paquetes afecta la comunicación entre los nodos. Esto produce que tanto el nodo que esta enviando como el que esta recibiendo el mensaje se 'atrasen' en la cadena dado que si la dificultad del minado es baja, podrían minar mas nodos que los requeridos para migrar la cadena y así quedarse con una cadena que nunca converja.

- ¿Cómo afecta el aumento o la disminución de la dificultad del Proof-of-Work a los conflictos entre nodos y a la convergencia?

  Los afecta en cuanto a que reduce la frecuencia de minar un bloque y hace que el tiempo de la comunicación entre nodos para la propagación de los nuevos bloques empiece a ser 'despreciable' en comparación con el tiempo de minarlo.

  Esto produce que la probabilidad de que minen 2 o mas bloques a la vez sea menor y, por ende, tener dos cadenas diferentes también.