Universidad Nacional Autónoma de México

Facultad de Ciencias

Computación Distribuida - 7176

P R O Y E C T O

Equipo:

Ángeles Sánchez Aldo Javier - 320286144

Sánchez Victoria Leslie Paola - 320170513

Sosa Romo Juan Mario - 320051926

Fecha de entrega:
29 de Noviembre de 2024

Profesor:
Mat. Salvador López Mendoza

Ayudante:
Santiago Arroyo Lozano

Contents

Chapter 1

Planteamiento del problema

Este documento presenta la solución final al proyecto de la materia de Computación Distribuida. El objetivo de este trabajo es desarrollar una blockchain para un sistema de criptomonedas, implementado en el lenguaje de programación *Elixir*.

Durante la ejecución del proyecto, se simulará el sistema de criptomonedas gestionando los distintos procesos distribuidos que representan a los usuarios. Estos usuarios realizarán transacciones entre ellos, enviando mensajes para alcanzar un consenso y detectando y eliminando intentos de alteración de la cadena de bloques. La blockchain, aunque simula un entorno, debe operar de forma descentralizada y garantizar la integridad de la información, asegurando que los nodos honestos mantengan una copia consistente de las transacciones para que estas se validen e inserten correctamente.

Para modelar las conexiones de la red, se utiliza una implementación basada en el modelo *Watts y Strogatz*, que garantiza un coeficiente de agrupamiento alto y una estructura que favorece la propagación eficiente de la información. Se asegura que este coeficiente sea mayor a 0.4 para comenzar el proceso de consenso y transmisión de transacciones.

Una de las características importantes del proyecto es la introducción de **procesos bizantinos**, los cuales representan nodos maliciosos en la red. Estos procesos se encargan de generar bloques basura o incorrectos con la intención de alterar la blockchain y comprometer su integridad. Sin embargo, el sistema debe ser capaz de detectar y evitar que estos bloques maliciosos sean insertados en la cadena. Para garantizar la seguridad y la correcta operación de la red, se asume que, si hay f nodos bizantinos y n nodos en total, se cumple la condición $n > 3f$, lo que asegura que la mayoría de los nodos honestos puedan mantener la coherencia de la blockchain.

Consideraciones extra

Se pueden utilizar cuantas funciones auxiliares sean necesarias, siempre que se logre el objetivo deseado. No es necesario implementar la persistencia del estado o la validación de la cartera; una vez que el programa termina, la blockchain deja de existir, y no se almacenará nada de forma permanente. No se realizará ninguna verificación sobre la validez de las transacciones en cuanto a saldo disponible. Se proporcionará un módulo inicial llamado *Crypto* encargado de realizar los hasheos de los bloques. Finalmente, la red debe ser capaz de evitar la inserción de bloques generados por procesos bizantinos, manteniendo así la integridad del sistema.

Chapter 2

Solución propuesta

2.1 Flujo general de la solución

1. **Inicialización de procesos:**
   - La función `Main.run(10, 1)` inicializa:
     - Una red de 10 nodos, representados como procesos de Elixir.
     - La blockchain, que comienza con un bloque génesis común a todos los nodos.
2. **Construcción de la red:**
   - Los procesos se conectan entre sí utilizando el modelo Watts y Strogatz, garantizando un coeficiente de agrupación mayor a 0.4.
   - Cada nodo guarda una copia inicial de la blockchain, comenzando con el bloque génesis.
3. **Propuesta de un nuevo bloque:**
   - Un nodo selecciona o recibe la tarea de proponer un bloque nuevo.
   - El nodo genera el bloque con la siguiente información:
     - **data:** El contenido del bloque, por ahora un string.
     - **timestamp:** La marca de tiempo de la creación del bloque.
     - **previous_hash:** El hash del bloque anterior en la blockchain.
     - **hash:** El hash del bloque actual.
   - El bloque es transmitido a todos los nodos de la red, esto es, se envia el mensaje a los vecinos, valida el bloque y si es correcto lo propaga a sus vecinos. El mensaje se transmite incluyendo el identificador del nodo que lo propuso para que pueda recopilar los votos.
4. **Proceso de votación:**
   - Cada nodo recibe el bloque propuesto y realiza las siguientes verificaciones:
     - ¿El data del bloque es un string?
     - ¿El hash del bloque anterior coincide con su propia blockchain?
     - ¿El hash del bloque actual es válido?
     - ¿El timestamp del bloque es posterior al último bloque de la blockchain?
   - Con base en las verificaciones, el nodo responde con un voto:

- `ACEPTAR` si el bloque es válido.
- `RECHAZAR` si el bloque es inválido.

5. **Decisión de consenso:**

   - El nodo proponente recolecta los votos de todos los nodos; para esto, los nodos checan el identificador del nodo que propuso el bloque, si es vecino inmediato se envía el voto, si no se reenvía a los vecinos, estos recopilan los votos y los envían al nodo proponente, vamos a usar TTL para eviar que los mensajes se propaguen indefinidamente.
   - Si más del 50% de los votos son `ACEPTAR`, el bloque es considerado válido:
     - El bloque se agrega a la blockchain del nodo proponente.
     - El bloque es propagado a todos los nodos para que lo agreguen a sus respectivas blockchains, se usa un mensaje diferente al de proponer.
   - Si no se alcanza el consenso, el bloque es descartado.

6. **Interacción desde el evaluador:**

   - El evaluador puede utilizar la terminal interactiva de Elixir (`iex`) para:
     - Insertar bloques nuevos.
     - Verificar que las blockchains de todos los nodos son consistentes y reflejan los mismos bloques.-

2.2 Módulos y sus funciones

Chapter 3

Resultados

3.1 Implementación

3.2 Pruebas

3.3 Notas de los resultados

Chapter 4

Conclusiones

Referencias