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

Primer Idea

Proceso de Inserción de un Bloque en la Blockchain

Supongamos que inicialmente la Blockchain solo está compuesta por al menos el bloque *“Genesis Block”* y que existe un nodo A que desea insertar un bloque a la Blockchain.

Si el nodo A crea un bloque, este debe ser válido por sí solo, es decir, al menos debe contar con la siguiente información:

- `data`: La información de transacción.
- `timestamp`: Fecha y hora de la creación del bloque.
- `prev_hash`: Último hash de la Blockchain.
- `hash`: Hash del bloque actual.

Nota: La creación de este bloque implica el uso del módulo `Crypto`.

Si el bloque no cumple con los requisitos mínimos, este es desechado automáticamente.

Si el bloque es válido, este se difunde al resto de nodos en la red.

Para cada nodo que ha recibido el bloque nuevo, se debe validar que este tiene el hash correcto del bloque anterior con respecto a la versión de la Blockchain que dispone cada uno. Además, se debe verificar que la fecha del nuevo bloque sea mayor que la del bloque previo a este.

Si el bloque es válido, este se añade a la Blockchain. En otro caso, se desecha.

Después de la inserción, se verifica cuál es la Blockchain correcta (la más larga) para compartir esta versión a los nodos que se quieran unir a la red.

Para ello, sea *bc* una estructura auxiliar que guardará la Blockchain que, por el momento, tiene la mayoría y sea *mayoría* una variable auxiliar que cuenta cuántos nodos tienen la misma Blockchain que *bc*.

Inicialmente, se toma un nodo de la red y se supone que su Blockchain es la correcta; se copia su información a *bc* y se establece *mayoría = 0*. Tomamos el siguiente nodo para revisar su versión de la Blockchain comparándolo con *bc*.

- Si las dos Blockchains son iguales, se incrementa en uno *mayoría*.
- Si son diferentes, se toma como la verdadera Blockchain la que se está revisando y se actualiza *bc*.

Para las siguientes rondas, si *mayoría* > 0 y la Blockchain que se está revisando es distinta a *bc*, disminuimos en uno *mayoría*. Cuando *mayoría* sea igual a cero, actualizamos *bc*.

El algoritmo termina cuando se hayan revisado todas las Blockchains.

Segunda Idea

Flujo de Alto Nivel del Programa

1. Inicialización del Sistema:

- **Entrada:** El programa recibe dos parámetros principales:
  - n: Número de nodos en la red.
  - f: Número de nodos bizantinos.
- **Configuración de la Red:**
  - *Modelo Watts y Strogatz:* Se configura la topología de la red con base en este modelo, asegurando que el coeficiente de agrupamiento sea mayor a 0.4 antes de comenzar el proceso. Este modelo garantiza que los nodos estén interconectados de manera eficiente para propagar la información.
  - **Creación de Nodos:** Se crean n nodos, de los cuales f son bizantinos, generando un total de $n - f$ nodos honestos, asegurando la desigualdad $n > 3f$ para mantener la integridad de la red.

2. Creación de Procesos y Asignación de Roles:

- **Spawning de Procesos:** Los nodos se representan como procesos concurrentes en Elixir, cada uno responsable de gestionar la lógica de su respectivo nodo en la blockchain.
- **Asignación de Roles:**
  - **Nodos honestos:** Serán responsables de validar y propagar transacciones y bloques correctos.

- **Nodos bizantinos:** Estos nodos generarán bloques maliciosos o basura, intentado alterar la blockchain y perturbar el consenso.

3. Generación de Bloques:

- **Proceso de Bloques:**
  - Cada nodo (honesto o bizantino) genera bloques de acuerdo con los datos que quiere insertar en la blockchain.
  - Un bloque incluye:
    - `data`: El contenido de la transacción o mensaje.
    - `timestamp`: La hora en la que se generó el bloque.
    - `prev_hash`: El hash del bloque anterior.
    - `hash`: El hash del bloque actual generado a partir de los datos.
- **Validación de Bloques:**
  - Los nodos validan los bloques según la regla de que cada bloque debe contener un hash válido del bloque anterior.
  - Los bloques generados por los nodos bizantinos se detectan como inválidos y no se insertan en la blockchain.

4. Proceso de Consenso:

- **Comunicación entre Nodos:**
  - Los nodos honestos intercambian información entre sí para llegar a un consenso sobre la validez de los bloques.
  - Se utiliza un protocolo de consenso **CAMBIAR ESTO** donde los nodos honestos verifican si los bloques propuestos cumplen con las reglas del sistema.
- **Bloques Maliciosos:**
  - Los nodos bizantinos intentan insertar bloques maliciosos en la blockchain. Sin embargo, debido a la validación y el consenso de la red, estos bloques no se aceptan, y se descartan.

5. Validación de la Blockchain:

- **Validación Continua:**
  - En todo momento, la blockchain debe validarse para asegurarse de que no haya bloques comprometidos. Esta validación se realiza mediante la comparación de los bloques, verificando que el hash de un bloque coincida con el `prev_hash` del siguiente bloque.
- **Redefinición de la Blockchain:**
  - Si se detecta una alteración significativa o corrupción en la blockchain (debido a bloques maliciosos de nodos bizantinos), la blockchain puede ser descartada y se puede iniciar una nueva cadena.

6. Inserción de Nuevos Bloques:

- **Proceso de Inserción:**
  - Cuando un bloque es validado correctamente, se inserta en la blockchain.
  - La cadena se mantiene actualizada y es replicada en los nodos honestos.

7. Finalización:

- **Detención del Programa:**
  - Una vez que el proceso de simulación ha terminado, el programa termina, y la blockchain se elimina de la memoria. No se realiza persistencia de datos.
  - Se puede consultar el estado final de la blockchain en la terminal para verificar que las transacciones se han propagado correctamente y que los bloques maliciosos fueron descartados.

Resumen Visual del Flujo:

1. **Inicialización:** Parámetros (n, f) → Topología de Red (Modelo Watts y Strogatz) → Creación de Nodos (honestos y bizantinos)
2. **Generación de Bloques:** Nodos crean bloques (datos, timestamp, prev_hash, hash)
3. **Validación de Bloques:** Nodos validan bloques, descartan bloques maliciosos de nodos bizantinos.
4. **Proceso de Consenso:** Nodos honestos alcanzan consenso, bloques correctos son insertados.
5. **Validación Continua de la Blockchain:** Se valida la cadena para asegurar que no haya corrupción de datos.
6. **Inserción de Nuevos Bloques:** Bloques válidos son insertados y replicados entre nodos honestos.
7. **Finalización:** Programa termina, blockchain se elimina (sin persistencia).

2.2 Consenso

2.3 Módulos y sus funciones

Chapter 3

Resultados

3.1 Implementación

3.2 Pruebas

3.3 Notas de los resultados

Chapter 4

Conclusiones

Referencias