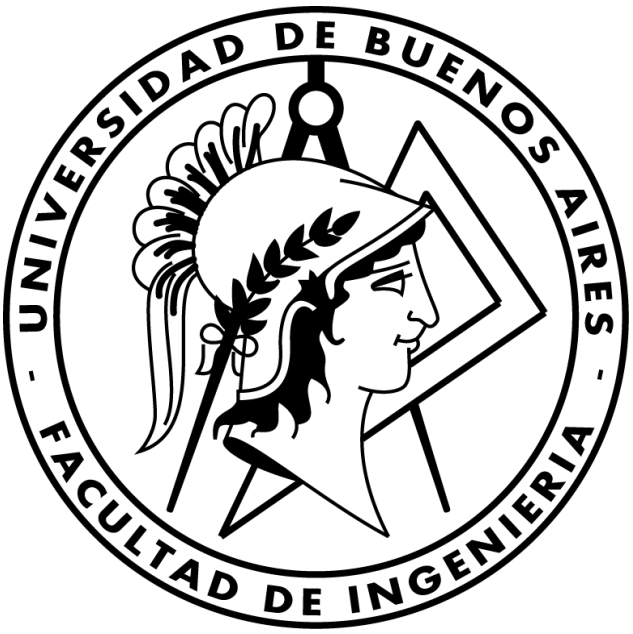


Trabajo Práctico 1

Concurrencia y Comunicaciones

Sistemas Distribuidos I 75.74

Primer Cuatrimestre 2021

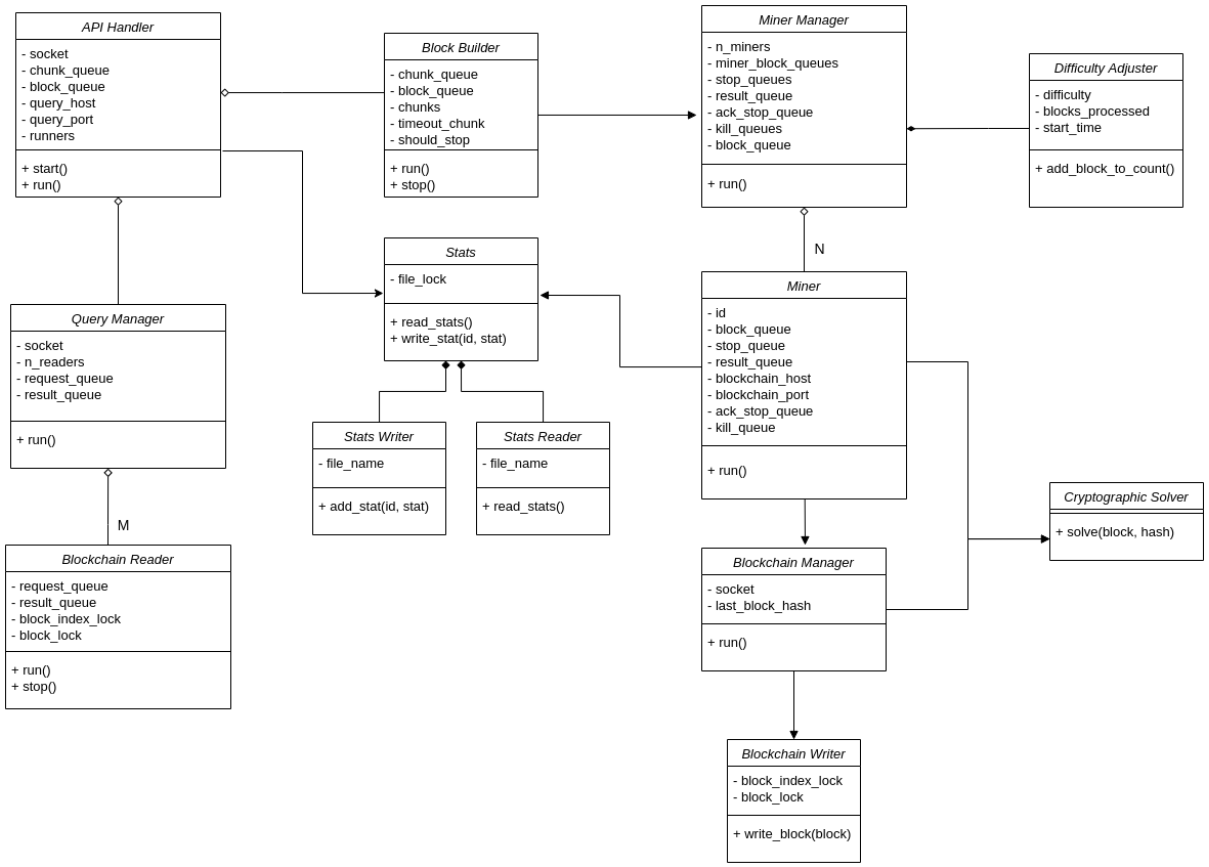
Cecilia Hortas 100687

Introducción

Se propone como objetivo implementar un sistema distribuido que simule el funcionamiento de una blockchain. Para dicho propósito los usuarios del sistema se comunican con una interfaz mediante la cual pueden agregar un chunk de datos a la blockchain, obtener la data de un bloque dado su hash, obtener los bloques de datos procesados en un intervalo de un minuto y obtener métricas de la cantidad de bloques que fueron procesados exitosamente y que fallaron en la blockchain.

En el presente informe se expondrán una serie de diagramas que permiten ilustrar el sistema implementado y así mismo exponer las relaciones y responsabilidades de los distintos componentes del sistema.

Diagrama de clases



Se implementaron las siguientes clases:

1. API Handler

Se encarga de recibir las querys de los usuarios y comunicarse con las entidades que correspondan a cada una de ellas. En particular se comunica con BlockBuilder en el caso que el usuario quiera agregar un chunk de datos, con StatsReader para obtener las stats del sistema y con QueryManager para redirigir las request que corresponden a obtener un bloque dado un hash y obtener los bloques dado un intervalo de un minuto.

2. Stats Reader

Se encarga de leer el archivo donde están guardadas las stats y responder mediante una cola la respuesta al APIHandler para que pueda mandarla al usuario.

3. Block Builder

Se encarga de construir un bloque de datos a partir de un chunk. La lógica realizada para ello consiste en crear un bloque si se llega al máximo de chunks permitido (256) o si se supera cierto umbral de tiempo configurable.

4. Query Manager

Se encarga de recibir las querys que necesitan conocer el estado de la blockchain y se comunica con los múltiples readers (ya que se sabe que las lecturas serían masivas) para poder obtener los resultados y mandarlos al usuario.

5. Blockchain Reader

Se encarga de leer los archivos donde se guarda la información pertinente a los bloques de la blockchain para resolver consultas como por ejemplo obtener un bloque dado un hash u obtener los bloques que se construyeron en un intervalo dado.

6. Miner Manager

Se encarga de coordinar los distintos mineros para minar un bloque. De esta manera envía el bloque a procesar por una cola, recibe el resultado por otra y se tiene otra cola para comunicar a los otros mineros en caso de que un minado haya resultado exitoso. Así mismo se comunica con el DifficultyAdjuster para ajustar la dificultad cada cierta cantidad de bloques.

7. Difficulty Adjuster

Se encarga de calcular la dificultad de un bloque dado cierta cantidad de tiempo de minado. La dificultad se calcula en base a la dificultad previa, la cantidad de bloques procesados y el tiempo transcurrido entre ajustes de dificultad consecutivos.

8. Miner

Es la clase encargada de minar un bloque. Para ello se comunica con el CryptographicSolver a fin de determinar si un bloque puede ser minado o no. Esto se calcula en base a la dificultad del bloque y su hash. Se comunica con el BlockchainManager para saber si un bloque fue minado exitosamente o no, para lo cual se emplea sockets ya que es un nodo externo y con el StatsWriter para comunicarle si el bloque fue minado con éxito o no.

9. Stats Writer

Se encarga de almacenar las stats del sistema.

10. Cryptographic Solver

Es la clase que se encarga de determinar si un bloque cumple con la condición de minado.

11. BlockchainManager

Se encarga de almacenar un bloque en la blockchain. Para ello se comunica con el Miner que le envía el bloque minado y lo guarda en caso de que el prev_hash coincida con el del bloque y que resuelva el algoritmo criptográfico. En el caso en el que el bloque haya sido minado se comunica con el Blockchain Writer para que almacene el contenido del bloque.

12. Blockchain Writer

Se encarga de guardar el bloque en la blockchain. Para ello se tiene un archivo por bloque donde se guarda la información completa y un archivo por día donde se guarda el hash y el timestamp de los bloques minados en tal día.

13. Stats

Se encarga de manejar el acceso concurrente al Stats Reader y al Stat sWriter.

Luego, algunas clases no fueron representadas en el diagrama ya que se consideró accesorias al diseño general. Se procede a describir la responsabilidad de las mismas:

14. Block

Guarda la implementación del bloque con los campos designados al mismo y los distintos métodos para serializarlo y deserializarlo.

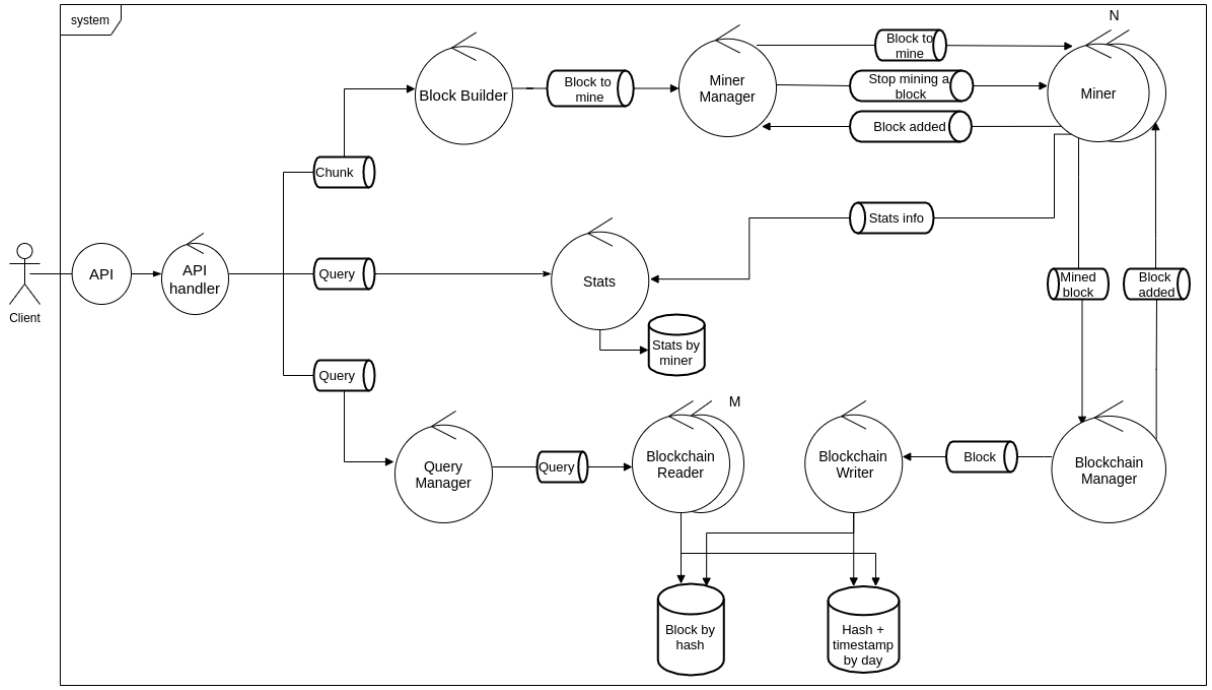
15. Socket

Tiene la implementación de un socket TCP junto con los métodos para poder operarlo.

16. GracefulStopper

Es la clase encargada de manejar las señales de finalización del programa.

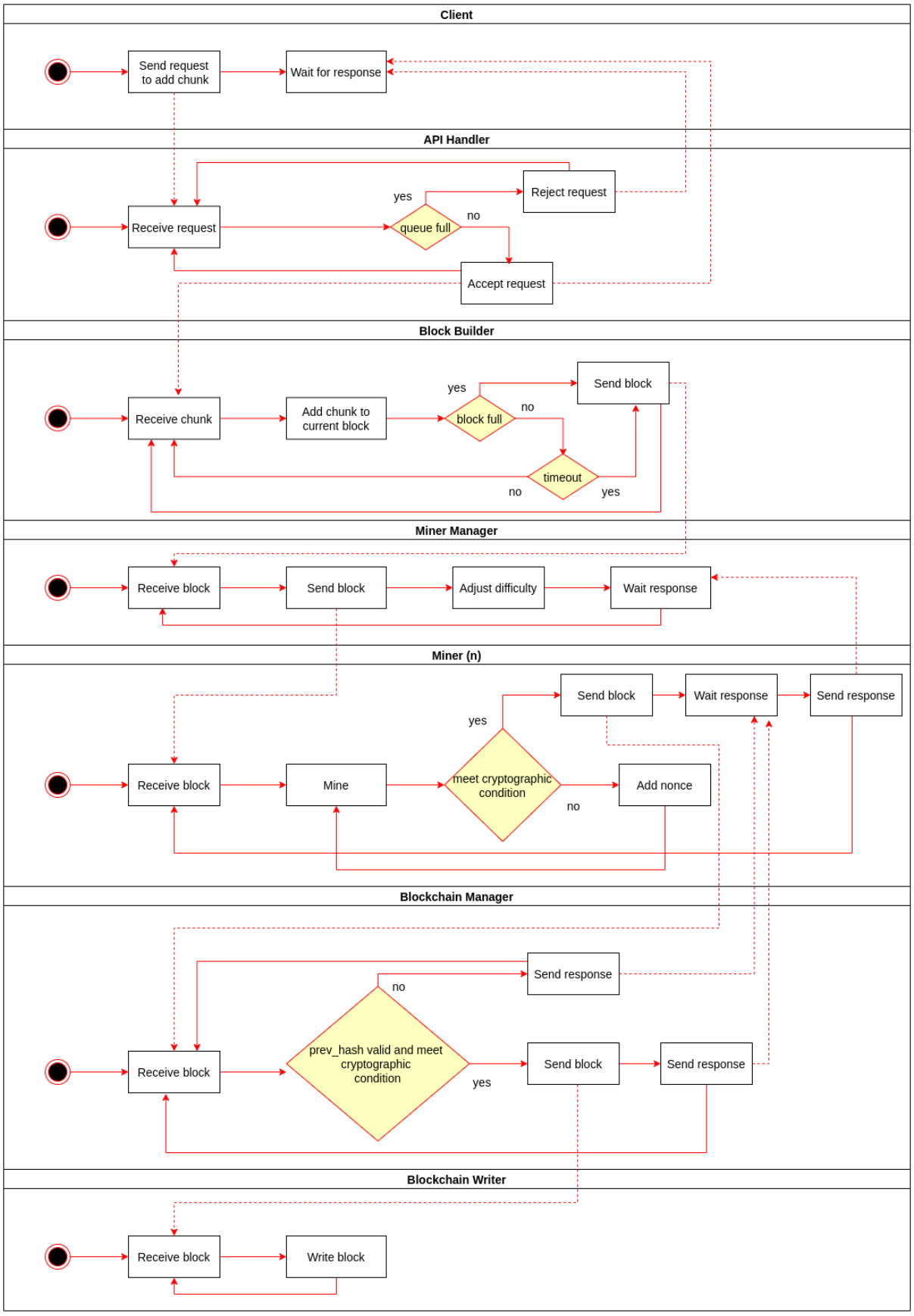
Diagrama de robustez



Se expone en el diagrama los distintos procesos involucrados en la implementación de la blockchain. Se ignoran algunas entidades que no suman valor al diagrama pero que participan de la comunicación como el CryptographicSolver y el DifficultyAdjuster.

El componente BlockchainManager es un nodo aislado del sistema por lo que se comunica con el mismo a través de sockets. Lo mismo aplica para el cliente que se comunica con la interfaz. Luego, entre los distintos miners (modelados como threads) y los componentes internos al sistema se decidió manejar la comunicación con colas bloqueantes (Queue en python).

Diagrama de actividad



Se presentan las principales interacciones entre los distintos componentes a la hora de agregar un chunk a la blockchain. Se propone así ver el camino de un bloque hasta ser persistido en un archivo y los diversos mensajes que se envían entre sí para cumplir tal propósito.

Es pertinente aclarar que no se colocan en el diagrama algunas entidades de procesamiento como el CryptographicSolver o el DifficultyAdjuster ya que se consideró que no aportan valor al diagrama.