Universidad de La Habana
Facultad de Matemática y Computación

Análisis de rendimiento en redes de Hyperledger Fabric.

Autor:
Bryan Machín García

Tutores:

M.Sc. Camilo Denis González
Dr.Sc. Carlos Miguel Legón

Trabajo de Diploma
presentado en opción al título de
Licenciado en Ciencia de la Computación

28 de octubre de 2022

github.com/BryanMachin/Optimal-parameters-for-blockchain-networks-by-Hyperledger-Fabric

Dedicación

Agradecimientos

Agradecimientos

Opinión del tutor

Opiniones de los tutores

Resumen

Resumen en español

Abstract

Resumen en inglés

Índice general

Índice de figuras

Introducción

En 2009 Satoshi Nakamoto introduce Bitcoin [16], la primera criptomoneda descentralizada. Las criptomonedas fueron las primeras aplicaciones que emplearon la tecnología blockchain. Con el tiempo, su aplicación se ha ramificado a distintas esferas, tales como: salud, cadenas de suministro, sistemas electorales, entre otras [21]. Cuando se trata de almacenar y compartir datos entre diferentes entidades, la base de datos centralizada tiene algunas limitaciones de notoria importancia para mantener la integridad de sus datos. Una de ellas es su único punto de falla; si hay un ataque, todo el sistema puede fallar. Además, por motivos de privacidad, pudiera no ser aceptable almacenar los datos en un tercero [25]. Una posible solución es elegir una entidad de confianza para almacenar los datos. Sin embargo, dado que estas entidades tienen políticas diferentes, sería de mayor complejidad lograr un acuerdo sobre la entidad que almacenará los datos. Por su naturaleza distribuida, blockchain supera estas limitaciones al no existir una autoridad centralizada; cada entidad puede tener una copia de los datos, y todas las entidades deben acordar las transacciones antes de su escritura en la blockchain. Cada bloque de transacciones se refiere al bloque anterior por su *hash*, lo que garantiza la integridad de los datos. Si un atacante intenta modificar cualquier bloque, el cambio se propagará a través de la cadena y será reconocido.

En la literatura, existen principalmente dos tipos de blockchains: permisionadas y no permisionadas. El objetivo principal de esta última es proporcionar accesibilidad pública y transacciones transparentes, por lo tanto, elimina la confidencialidad. La blockchain permisionada surgió para solucionar el problema de almacenar datos confidenciales, por ejemplo, para aplicaciones médicas. Permite compartir datos y acceder a entidades/usuarios de confianza específicos [25]. Sin embargo, estas entidades tienen que obtener un consenso entre ellas para identificar cualquier manipulación no autorizada de datos. En Bitcoin, se utiliza un esquema de consenso *PoW*, prueba de trabajo, en el que los mineros compiten para resolver un rompecabezas computacionalmente intensivo y una vez que un minero lo resuelve, transmite el nuevo bloque. Una de sus limitaciones es la vulnerabilidad al ataque del 51 % que permite tomar el control de toda la red [17], lo que sucede si una sola entidad posee más del 51 % del poder computacional de la blockchain. En el caso de Peercoin [14] emplea *PoS*, prueba

de participación, para disminuir la sobrecarga computacional de *PoW*. La prueba de participación se basa en la cantidad de moneda reservada y el tiempo de participación en la red, pero pueden definirse otros criterios; que una vez establecidos, se inicia el proceso de selección de nodos de forma aleatoria para validar transacciones o crear nuevos bloques. A diferencia de *PoW*, este enfoque no consume gran cantidad de recursos. Además, no es vulnerable al ataque del 51 %, ya que el atacante necesita poseer más monedas que el resto de la red; causando un aumento en el precio de la moneda, lo que hace que los ataques sean muy costosos. La prueba de trabajo y la prueba de participación son dos de las técnicas de consenso que garantizan confianza más comunes en las blockchains no permisionadas, sin importar que el proceso de minería consuma mucho tiempo. Por el contrario, las blockchains permisionadas emplean protocolos más rápidos para lograr el consenso. Entre las plataformas más comunes están Ethereum [2] y Hyperledger Fabric. Ethereum se estableció en 2015 y finalmente se convirtió en uno de los marcos de blockchains programables más populares. Si bien Ethereum es más liviano y más fácil de usar, no es altamente personalizable. A diferencia, Hyperledger Fabric ha dado pasos sustanciales en virtud de lograr un sistema lo más adaptable posible que garantice un mejor rendimiento para los distintos casos de uso [22], principalmente en ecosistemas empresariales.

0.1. Situación problémica

Cuando se trata de configuraciones, Hyperledger Fabric brinda un alto grado de libertad a los operadores de red, existen parámetros para configurar el rendimiento y latencia de las transacciones que se pueden ajustar para escenarios donde se ejecuten una gran cantidad de transacciones por segundos (TPS) o donde ocurra todo lo contrario, y es en dependencia de la configuración de estos parámetros que se puede mejorar el rendimiento de redes blockchain usando Hyperledger Fabric. Por lo que el problema consiste en definir un escenario de bajo número de transacciones por segundos y otro que cuente con un elevado número de transacciones por segundos; y determinar una configuración en el canal de comunicación donde se desarrollan las transacciones, para cada escenario, que posibilite un elevado rendimiento.

0.2. Motivación

La tecnología Blockchain ha brindado un nuevo paradigma para generar confianza en los datos, dentro de un ambiente no necesariamente confiable. Los mecanismos para lograrlo, expuestos hoy en día, consumen diversos recursos, que en escenarios de gran trasiego de información pueden fracturar el correcto funcionamiento de la tecnología. Esto nos motiva a realizar un estudio que posibilite minimizar el uso de los recursos

siempre y cuando se logre un desempeño óptimo. En el caso de Hyperledger Fabric ha salido a la vanguardia en cuanto a la variedad de opciones configurables que ofrece, por esto amerita el centro de nuestra investigación.

0.3. Objetivos

0.3.1. Objetivo General

Determinar configuraciones óptimas para canales de redes blockchain de Hyperledger Fabric en escenarios de mayor o menor volumen de transacciones.

0.3.2. Objetivos Específicos

- Configurar y Desplegar una red blockchain de Hyperledger Fabric para cada escenario expuesto.
- Medir el rendimiento de las redes desplegadas.
- Analizar los reportes de rendimiento.
- Determinar los parámetros de configuración óptimos para cada escenario basado en el análisis de los reportes de rendimento.

Capítulo 1

Estado del Arte

Capítulo 2

Marco Teórico

Hyperledger Fabric [11] es una de las plataformas blockchain más populares administrada por Linux Foundation. Constituye una plataforma permisionada de nodos pares, nodos ordenadores y clientes, que conforman organizaciones. Cada uno de estos elementos posee una identidad criptográfica en la red. Todas las entidades de la red tienen visibilidad a las identidades de todas las organizaciones y pueden verificarlas. Difiere de las plataformas blockchain públicas que posibilitan la unión de cualquier usuario a la red. Además, Fabric presenta una arquitectura de ejecución, orden y validación que supera los límites de la arquitectura de orden y ejecución anterior [1]. Esto mejora sustancialmente la escalabilidad de rendimiento en redes blockchain con un número elevado de nodos pares, lo que permite a Fabric ser competente en *Global Trade Digitalization* [5], *SecureKey* [20] y *Everledger* [6].Constituye la primera plataforma blockchain que admite contratos inteligentes creados en lenguajes de programación de uso general como Java, Golang y Node.js; siendo factible para la mayoría de las empresas en el desarrollo de los contratos inteligentes, sin necesidad de capacidad adicional para aprender lenguajes específicos de dominio restringidos, conocidos por sus siglas en inglés *DSL*.

2.1. Principales conceptos de Hyperledger Fabric

2.1.1. Chaincode

El chaincode [3] es un software que define uno o más activos junto a las instrucciones de la transacción para su modificación. Es el encargado de exponer la lógica de negocio. Hace cumplir las reglas para leer o modificar los pares *llave-valor* u otra información de la base de datos de estado. Las funciones de un chaincode se ejecutan a partir de la base de datos del estado actual del libro mayor y se inician mediante una propuesta de transacción dando como resultado un conjunto de escrituras *llave-valor*

que puede ser enviado a la red y aplicado al libro mayor en todos los nodos pares. Se pueden implementar en varios lenguajes de programación. Actualmente, se admiten Go, Node.js y Java.

2.1.2. Contrato inteligente

Los contratos inteligentes de Hyperledger Fabric constituyen la lógica de negocio. Están escritos en chaincode, que son invocados por una aplicación externa a la blockchain cuando necesita interactuar con el libro mayor.

2.1.3. Libro mayor

El libro mayor o *ledger* es un registro secuencial, a prueba de manipulaciones, de todas las transiciones de estado en Hyperledger Fabric. Las transiciones de estado son el resultado de los llamados del chaincode *(transacciones)* presentadas por los miembros participantes. Cada transacción da lugar a un conjunto de pares *llave-valor* de activos que se registran en el libro mayor en forma de creación, actualización o eliminación.

El libro mayor está compuesto por una cadena de bloques para almacenar el registro inmutable y secuenciado en bloques, así como una base de datos de estado para mantener el estado actual de Fabric. Existe un libro mayor por cada canal. Cada nodo par mantiene una copia del libro mayor para cada canal del que es miembro.

2.1.4. Política de aprobación

Los chaincodes están escritos en lenguajes de propósito general que se ejecutan en nodos pares no confiables en la red. Por tanto, surgen múltiples problemas, uno de carácter no determinista, ejecución, y el otro de confiar en los resultados de cualquier nodo par. La política de aprobación aborda estas dos preocupaciones especificando como parte de una política de aprobación, el conjunto de nodos pares que necesitan simular la transacción y aprobar o firmar digitalmente los resultados de la ejecución.

2.1.5. Protocolo de consenso

En muchas plataformas Blockchain, no permisionadas, como Ethereum y Bitcoin, cualquier nodo puede participar en el proceso de consenso. Estos sistemas se basan en algoritmos de consenso probabilísticos que eventualmente garantizan la consistencia del libro mayor en un alto grado de probabilidad, pero que aún son vulnerables a ledgers divergentes, conocidos como *bifurcación* de ledgers), donde diferentes participantes en la red no comparten la misma visión del orden de transacciones aceptado.

Hyperledger Fabric funciona de manera diferente. Cuenta con un nodo llamado ordenador que realiza la ordenación de transacciones, junto con otros nodos ordenadores conformando un servicio de ordenación. Se basa en un diseño de consenso determinista, que garantiza para cualquier bloque validado por un nodo par sea final y correcto. Entre los protocolos de consenso con que cuenta Fabric en la actualidad están:

- Raft: Se introdujo a partir de v1.4.1. Es un servicio de ordenación tolerante a fallas *CFT* basado en una implementación del protocolo Raft en *etcd*[1]. Sigue un modelo de "líder y seguidor", donde se elige un nodo líder (por canal) y sus decisiones son replicadas por los seguidores. Su diseño permite que diferentes organizaciones contribuyan con nodos a un servicio de ordenación distribuido.
- Kafka: Similar a Raft, Apache Kafka es una implementación de *CFT* que utiliza un nodo de "líder y seguidor". Utiliza un conjunto *Zookeeper* [13] con fines de gestión. El consenso basado en Kafka ha estado disponible desde Fabric v1.0, pero muchos usuarios pueden encontrar que administrar un clúster de Kafka es poco deseable.
- Solo: La implementación del servicio de ordenación de *Solo* está destinada solo a prueba, y consiste en un único nodo ordenador.

2.1.6. Sistema de chaincode

Un sistema de chaincode tiene el mismo modelo de programación que los chaincodes de usuarios, pero está integrado en el ejecutable del nodo par. Fabric implementa varios sistemas de chaincodes:

- Sistema de chaincode de ciclo de vida *(LSCC)*: Permite instalar/crear instancias/actualizar chaincodes.
- Sistema de chaincode de respaldo *(ESCC)*: Permite aprobar una transacción firmando digitalmente la respuesta.
- Sistema chaincode de validación *(VSCC)*: Posibilita evaluar la aprobación en la transacción contra la política de aprobación especificada para el chaincode. Si la política de aprobación no se satisface, entonces la transacción se toma como inválida.

[1] almacena un conjunto de *llave-valor* distribuido y consistente que proporciona una forma confiable de almacenar datos. Maneja con eficacia las elecciones de líder durante las particiones de la red y puede tolerar fallas, incluso en el nodo líder.

- Control de Multiversión de concurrencia [19] *MVCC*: Garantiza que las versiones de las llaves leídas por una transacción durante la fase de aprobación son igual que su estado actual en el libro mayor local en la fase de confirmación. Se asemeja a una verificación de conflicto de lectura y escritura realizada para el control de concurrencia, y se realiza secuencialmente en todas las transacciones válidas en el bloque. Si las versiones del conjunto de lectura no coinciden, denota que anteriormente la transacción modificó los datos leídos y fue desde su aprobación confirmada con éxito, la transacción es designada como inválida. Para garantizar que no se produzcan lecturas fantasma la consulta se vuelve a ejecutar y se comparan los *hashes* de los resultados, que también se almacena como parte del conjunto de lectura, capturado durante la aprobación.
- Sistema de configuración chaincode *(CSCC)*: Permite administrar las configuraciones de los canales.

2.1.7. Canal

Hyperledger Fabric introduce un concepto llamado *canal* como una subred privada de comunicación entre dos o más nodos pares para proporcionar un nivel de aislamiento. Las transacciones en un canal solo son vistos por sus nodos pares y participantes. El ledger y los chaincodes son definidos por canal. Además, el consenso es aplicable por canal, es decir, no hay un orden definido para la transacción a través de los canales.

2.1.8. Servicio de membresía

El servicios de membresía o proveedor de servicios de membresía *MSP* [15] es un conjunto de carpetas que se agregan a la configuración de la red y se emplea para definir una organización, tanto interna, como externamente. Mientras que las autoridades de certificación (*CA*) generan los certificados que representan a cada identidad, el *MSP* contiene una lista de identidades autorizadas.

El *MSP* identifica la autoridad certificadora raíz, y las intermedias se aceptan para definir los miembros de un dominio de confianza enumerando las identidades de sus miembros o identificando qué *CA* están autorizadas para emitir identidades válidas para sus miembros.

El poder de un *MSP* va más allá de enumerar quién es un participante de la red o miembro de un canal. Convierte una identidad en un rol al identificar los privilegios específicos que tiene un actor en un nodo o canal. Cuando un usuario está registrado con *Fabric CA*, se debe asociar con el usuario una función de administrador, nodo par, cliente, nodo ordenador o miembro.

Los *MSP* coexisten en dos dominios dentro de la blockchain:

- Servicio de membresía local: Se define para clientes y nodos (pares y ordenadores). Los *MSP* locales definen los permisos para un nodo. Los *MSP* locales de los clientes permiten al usuario autenticarse en sus transacciones como miembro de un canal, por ejemplo en las transacciones, o como propietario de un rol específico en el sistema.

- Servicio de membresía del canal: Define los derechos administrativos y de participación a nivel de canal. Los nodos pares y los nodos ordenadores en un canal de aplicación comparten la misma vista de los *MSP* del canal y, por lo tanto, pueden autenticar correctamente a los participantes del canal. Esto significa que si una organización desea unirse al canal, se debe incluir en la configuración del canal un *MSP* que incorpore la cadena de confianza para los miembros de la organización. De lo contrario, se rechazarán las transacciones que se originen a partir de las identidades de esta organización. Mientras que los *MSP* locales se representan como una estructura de carpetas en el sistema de archivos, los *MSP* de canal se describen en una configuración de canal.

2.1.9. Autoridad certificadora

Un nodo puede participar en la red blockchain, a través de una identidad digital emitida por una autoridad de confianza del sistema. Las identidades digitales constituyen certificados digitales validados criptográficamente que cumplen con el estándar X.509 y son emitidos por una Autoridad de Certificación(*CA*) [3]. Las *CA* tienen un certificado, que ponen a disposición de todos para que los consumidores de identidades emitidas por una determinada *CA* verifiquen comprobando que el certificado sólo pudo haber sido generado por el titular de la clave privada correspondiente.

Fabric CA

Fabric proporciona un componente de *CA* que permite crear identidades en la red blockchain. Es un proveedor de *CA* raíz privado capaz de administrar las identidades digitales de los participantes de Fabric que tienen la forma de certificados X.509. Debido a que *Fabric CA* es una *CA* personalizada dirigida a las necesidades de *CA* raíz de Fabric, no es capaz de proporcionar certificados *SSL* para uso general en navegadores. Sin embargo, debido a que se debe usar alguna *CA* para administrar la identidad (incluso en un entorno de prueba), *Fabric CA* se puede usar para proporcionar y administrar certificados. También es posible, utilizar una *CA* raíz pública/comercial o intermedia para proporcionar identificación.

2.1.10. Nodo Par

Un nodo par ejecuta el chaincode, que implementa un contrato inteligente del usuario y mantiene el ledger en un sistema de archivos. El chaincode tiene acceso permitido al estado compartido definido por la *API*. Entre los nodos pares se distinguen los que mantienen la lógica del chaincode y lo ejecuta para aprobar una transacción. Sin tener en consideración esta diferenciación, todos mantienen el libro mayor y el último estado *StateDB* en un registro de *llave-valor* tal que se puede consultar o modificar el estado.

2.1.11. Nodo Ordenador

Los nodo ordenador participa en el protocolo de consenso y conforma el bloque de transacciones que se entrega a los nodos pares mediante un protocolo de comunicación *gossip*[2]. Juntos constituyen el servicio de ordenación, *OSN* por sus siglas en inglés, que posee un carácter modular y admite un mecanismo de consenso *conectable*. Se construye un bloque para ser entregado a los nodos pares, cuando se llega a un número máximo de nuevas transacciones desde el último bloque conformado, o se cumple al tiempo de espera configurado desde la última transacción producida.

2.1.12. Cliente

Los clientes son responsables de llevar a cabo una propuesta de transacción a uno o más nodos pares simultáneamente para recopilar respuestas a propuestas y satisfacer la política de aprobación. Posteriormente transmiten la transacción al servicio de ordenación para ser incluida en un bloque y entregado a todos los pares para su validación y confirmación.

2.2. Flujo de transacciones en Hyperledger Fabric

A diferencia de otras redes Blockchain que emplean un modelo de *ordenación-ejecución* [23] de transacciones, Fabric emplea un modelo de simulación, validación y confirmación de transacciones.

La figura 2.1 muestra el flujo de la transacción que consta de 3 fases:

1 Fase de aprobación: Simula la transacción en nodos pares selectivos y recopila los cambios de estado.

[2]El protocolo *Gossip* es un protocolo que permite diseñar sistemas de comunicaciones distribuidos (*P2P*) altamente eficientes, seguros y de baja latencia.

2 Fase de ordenación: Ordena las transacciones a través de un protocolo de consenso.

3 Fase de validación: Valida y confirma en el libro mayor.

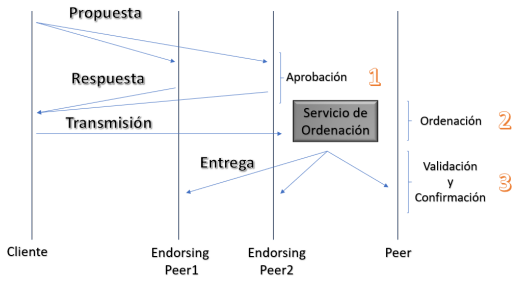

Figura 2.1: Flujo de transacciones.

Antes de que las transacciones sean enviadas, la red debe comenzar con las organizaciones participantes, sus *MSP* e identidades de los nodos pares. Primero, se crea un canal en la red con los respectivos *MSP* de las organizaciones. En segundo lugar, los nodos pares de cada organización se unen al canal y se inicializa el libro mayor. Finalmente, los chaincodes requeridos se instalan en el canal.

2.2.1. Fase de aprobación

Una aplicación cliente que utiliza Fabric *SDK* [7], [18], [12], construye una propuesta de transacción para invocar un chaincode que a su vez realizará operaciones en el estado del libro mayor. La propuesta está firmada con las credenciales del cliente y el cliente lo envía a uno o más nodos pares simultáneamente. La política de aprobación del chaincode dicta los nodos pares de la organización que el cliente necesita para enviar la propuesta a la simulación.
Primero, cada nodo par que aprueba, verifica que el remitente es autorizado para invocar transacciones en el canal. En segundo lugar, el nodo ejecuta el chaincode, que puede acceder al estado actual del libro mayor en el nodo par. Los resultados de la transacción incluyen el valor de respuesta, conjunto de lectura y conjunto de escritura. En tercer lugar, el par que aprueba llama a un sistema chaincode llamado *ESCC* que firma esta respuesta de transacción con la identidad del nodo par y responde

al cliente. Finalmente, el cliente inspecciona la respuesta de la propuesta a verificar que lleva la firma del nodo par. El cliente recoge las respuestas de diferentes nodos pares y verifica que sea la misma. Dado que cada nodo par podía haber ejecutado la transacción en diferentes momentos en la Blockchain, es posible que la respuesta de la propuesta difiera. En tales casos, el cliente tiene que volver a enviar la propuesta a otros pares, para obtener suficientes respuestas coincidentes.

2.2.2. Fase de ordenación

El cliente transmite un mensaje de transacción bien formado al servicio de ordenación. La transacción tendrá los conjuntos de lectura y escritura, las firmas de los nodos pares que aprueban y la identificación del canal. El servicio de ordenación no necesita inspeccionar el contenido de la transacción para realizar su operación. Recibe transacciones de diferentes clientes por varios canales y los pone en cola por canal. Crea bloques de transacciones por canal, firma el bloque con su identidad y los entrega a los nodos pares usando el protocolo de mensajería *gossip*.

2.2.3. Fase de validación

Todos los nodos pares en un canal reciben bloques de la red. Cada nodo par primero verifica la firma del nodo ordenador en el bloque. Cada bloque válido se decodifica y todas las transacciones en un bloque pasan primero por una validación de *VSCC* antes de realizar la validación de *MVCC*.

2.2.4. Fase de actualización en el libro mayor

Como último paso del procesamiento de transacciones, el libro mayor se actualiza agregando el bloque al libro mayor local *StateDB*, que contiene el estado actual de todas las llaves. Se actualiza con los conjuntos de escritura de transacciones válidas. Estas actualizaciones de *StateDB* se realizan para un bloque de transacciones y aplican las actualizaciones para llevar *StateDB* al estado después de que se hayan procesado todas las transacciones en el bloque.

2.3. Parámetros configurables

Nuestro objetivo es estudiar el rendimiento de Fabric en diversas condiciones para comprender cómo las elecciones de las diferentes facetas del sistema afectan el rendimiento. Sin embargo, el espacio de parámetros es amplio y limitamos nuestras opciones para cubrir de manera integral algunos componentes y observar ampliamente

otros aspectos del sistema para que podamos identificar la interacción de las opciones a nivel de componentes.

2.3.1. Tamaño de bloque

Las transacciones se procesan por lotes en el nodo ordenador y se entregan como un bloque a los nodos pares usando un protocolo *gossip*. Cada nodo par procesa un bloque a la vez. La variación del tamaño de bloque también trae consigo la compensación de rendimiento frente a latencia y, para obtener una mejor concepción, lo estudiamos en conjunto con la tasa de llegada de transacciones.

2.3.2. Política de aprobación

Una política de aprobación dicta cuántas ejecuciones de una transacción y firma deben ocurrir antes de que se pueda enviar una solicitud de transacción al servicio de ordenación para que la transacción pueda pasar la fase de validación *VSCC* en los pares. La validación *VSCC* de los aprobados de una transacción requiere la evaluación de la expresión de la política de aprobación frente a los aprobados recopilados y la verificación de la satisfacibilidad [9], que es *NP-Completo*. Adicionalmente, incluye verificar la identidad y su firma. La complejidad de la política de aprobación afectará los recursos y el tiempo necesario para recopilar y evaluar transacciones.

2.3.3. Canal

Los canales aíslan las transacciones. Las transacciones enviadas a diferentes canales se ordenan, entregan y procesan de forma independiente, aunque en el mismo nodo par. Los canales aportan un paralelismo inherente a varios aspectos de procesamiento de transacciones en Hyperledger Fabric. Mientras que el número de canales a emplear, y en qué canales realizar transacciones está determinado por la aplicación y la combinatoria de los participantes, tiene importantes implicaciones en el rendimiento y la escalabilidad de la plataforma.

2.3.4. Recursos

Los nodos pares ejecutan la firma y verificación como parte de un sistema chaincode con empleo intensivo de la *CPU*.

2.3.5. Base de datos del libro mayor

Fabric admite dos alternativas para almacenar *llave-valor*, *CouchDB* [4] y *GoLevelDB* [8] para mantener el estado actual. Ambos almacenan *llave-valor*, mientras que

GoLevelDB es una base de datos incrustada, *CouchDB* usa un modelo *cliente-servidor* (al que se accede mediante la *API REST* a través de un *HTTPS*) y es compatible con documentos *JSON*.

2.4. Hyperledger Fabric v2.x

La primera versión de notoria importancia de Hyperledger Fabric luego de su lanzamiento con la versión 1.0 fue Fabric v2.0. Ofrece nuevas características y cambios representativos para usuarios y operadores [24]. Incluye la compatibilidad con nuevos patrones de aplicaciones y privacidad, gobernanza mejorada en torno a contratos inteligentes y nuevas opciones para nodos operativos.

Fabric v2.0 presenta un gobierno descentralizado para contratos inteligentes, con un nuevo proceso para instalar un chaincode en sus nodos pares e instanciarlos en un canal. El nuevo ciclo de vida del chaincode de Fabric permite que varias organizaciones lleguen a un acuerdo sobre los parámetros del chaincode, como la política de aprobación del contrato, antes de que pueda usarse para interactuar con el ledger. El nuevo modelo ofrece varias mejoras con respecto al ciclo de vida anterior [10]:

- **Múltiples organizaciones deben aceptar los parámetros de un chaincode**. En las versiones de lanzamiento 1.x, una organización tenía la capacidad de establecer parámetros de un chaincode, como la política de aprobación, para todos los miembros del canal, que solo tenían el poder de negarse a instalar el chaincode, por lo tanto, no participar en transacciones que lo invoquen. El nuevo ciclo de vida del chaincode de Fabric es más flexible, admite tanto el modelo de ciclo de vida anterior, como modelos descentralizados que requieren suficiente número de organizaciones para acordar una política de aprobación y otros detalles antes de que el chaincode se convierta en activo en un canal.

- **Proceso de actualización de chaincode más deliberado**. Anteriormente la transacción de actualización podía ser emitida por una sola organización, creando un riesgo para un miembro del canal que aún no había instalado el nuevo chaincode. El nuevo modelo permite que el chaincode se actualice solo después de que un número suficiente de organizaciones han aprobado la actualización.

- **Actualizaciones más sencillas de políticas de respaldo y recopilación de datos privados**. Se permite cambiar una política de respaldo o una configuración de recopilación de datos privados sin tener que volver a empaquetar o instalar el chaincode. Los usuarios también pueden aprovechar una nueva política de aprobación predeterminada que requiere la aprobación de una mayoría de

organizaciones en el canal. Esta política se actualiza automáticamente cuando se agregan o eliminan organizaciones del canal.

- **Paquetes de chaincodes inspeccionables**. El chaincode se empaqueta en archivos **.tar** fáciles de leer, que hace más factible su inspección y coordinación de la instalación en múltiples organizaciones.

- **Iniciar múltiples chaincodes en un canal usando un paquete**. El ciclo de vida anterior definía cada chaincode en el canal usando un nombre y una versión que se especificó en su instalación. Ahora existe la posibilidad de usar un solo paquete de chaincodes y desplegarlo varias veces con diferentes nombres en el mismo canal o en diferentes canales.

- **Los paquetes de chaincode no necesitan ser iguales entre los miembros del canal**. Las organizaciones pueden extender un chaincode para su propio caso de uso, por ejemplo, para realizar diferentes validaciones en interés de su organización. Siempre que el número requerido de organizaciones respalde las transacciones del chaincode con resultados coincidentes, la transacción se validará y se confirmará en el libro mayor. Esto también permite a las organizaciones implementar arreglos menores individualmente.

Los mismos métodos descentralizados para llegar a un acuerdo que sustentan la nueva gestión del ciclo de vida del chaincode pueden usarse también en su propia aplicación para garantizar que las organizaciones den su consentimiento a las transacciones de datos antes de que se realicen escrituras en el libro mayor.

- **Comprobaciones automatizadas**. Las organizaciones pueden agregar comprobaciones automáticas a las funciones del chaincode para validar información adicional antes de aprobar una propuesta de transacción.

- **Acuerdo descentralizado**. Las decisiones humanas se pueden modelar en un proceso de chaincode que abarca múltiples transacciones. El chaincode puede requerir que los actores de varias organizaciones indiquen sus términos y condiciones de acuerdo en una transacción en el libro mayor. Luego, una propuesta final de chaincode puede verificar que las condiciones de todas las transacciones individuales se cumplen y establecer la transacción comercial con carácter definitivo en todos los miembros del canal.

Fabric v2.0 también permite nuevos patrones para trabajar y compartir datos privados, sin el requisito de crear recopilaciones de datos privados para todas las combinaciones de miembros del canal que deseen realizar transacciones. Específicamente,

en lugar de compartir datos privados dentro de una colección de varios miembros. Es posible compartir datos privados entre colecciones, donde cada colección puede incluir una sola organización, o tal vez una sola organización junto con un regulador o auditor.

- **Compartir y verificar datos privados**. Cuando los datos privados se comparten con un miembro del canal que no es miembro de una colección, o compartida con otra colección de datos privados que contiene uno o más miembros del canal (escribiendo una llave para esa colección), las partes receptoras pueden utilizar la *API* del chaincode *GetPrivateDataHash()* para verificar que los datos privados coincidan con los *hash* en la cadena que se crearon a partir de datos privados en transacciones anteriores.
- **Políticas de aprobación a nivel de colección**. Las colecciones de datos privados ahora se pueden definir opcionalmente con una política de aprobación que anula la política de aprobación a nivel de chaincode para las llaves dentro de la colección. Se puede usar para restringir qué organizaciones pueden escribir datos en una colección. Se puede concebir un chaincode que requiere el respaldo de la mayoría de las organizaciones, pero para cualquier transacción determinada, puede necesitar dos organizaciones transaccionales para respaldar individualmente su acuerdo en sus propias colecciones de datos privados.
- **Recopilaciones implícitas por organización**. Si se desea utilizar patrones de datos privados por organización, no se necesita definir las colecciones al implementar chaincode. Las colecciones se pueden usar sin ninguna definición inicial.

La función de lanzador de chaincode externo permite a los operadores crear y lanzar chaincode con la tecnología de su elección. No se requiere el uso de constructores y lanzadores externos ya que el comportamiento predeterminado construye y ejecuta el chaincode de la misma manera que las versiones anteriores con la *API* de Docker.

- **Elimina la dependencia del demonio de Docker**. Las versiones anteriores de Fabric requerían que los nodos pares tuvieran acceso a un *Docker daemon* para construir y lanzar chaincode, algo que puede no ser deseable en entornos de producción.
- **Alternativas a los contenedores**. Ya no es necesario ejecutar chaincode en contenedores Docker, y puede ejecutarse en el entorno elegido por el operador (incluidos los contenedores).

- **Chaincode como un servicio externo**. Tradicionalmente, los chaincodes son lanzados por el nodo par y luego se conectan de nuevo a él. Ahora es posible ejecutar chaincode como un servicio externo, por ejemplo, en un *pod* de Kubernetes, que un nodo par puede conectarse y utilizar para la ejecución de chaincode.

Cuando se utiliza una base de datos de estado externa *CouchDB*, los retrasos de lectura durante las fases de aprobación y validación han representado un cuello de botella en el rendimiento. Con Fabric v2.0, una nueva caché reemplaza muchas de estas costosas búsquedas con lecturas rápidas de caché local. El tamaño de caché se configura mediante la propiedad *cacheSize* en *core.yaml*.

Hyperledger Fabric v2.3 presenta dos nuevas características para mejorar las operaciones entre los nodos pares y ordenadores:

- Gestión de canales de ordenación sin un canal de sistema. Para simplificar el proceso de creación de canales y mejorar la privacidad y escalabilidad de los canales, es posible crear canales de aplicación sin crear primero un *canal de sistema* administrado por el servicio de ordenación. Este proceso permite ordenar a los nodos que se unan (o abandonen) cualquier cantidad de canales según sea necesario.

- **Copia del libro mayor**. Ahora es posible tomar una copia de la información del canal en un par, incluida su base de datos de estado, y unir nuevos pares (en la misma organización o en diferentes organizaciones) al canal en función de la copia.

Con las últimas versiones de Fabric v2.4 se introduce *Fabric Gateway* que elimina gran parte del envío de transacciones y la lógica de consulta de la aplicación del cliente y la cambia a una puerta de enlace común que se ejecuta dentro de los nodos pares, lo que permite que cada uno de los *SDK* del cliente sean más ligeros y requieran menos mantenimiento. Las aplicaciones interactúan con un nodo par de confianza (por ejemplo, en su organización) que coordina la aprobación de otros nodos pares y el envío al servicio de ordenación. También simplifica la sobrecarga administrativa de ejecutar una red Fabric porque las aplicaciones cliente pueden conectarse y enviar transacciones a través de un solo puerto de red en su organización en lugar de abrir puertos desde una aplicación cliente a múltiples nodos pares.

También se agrega el comando conocido como *peer unjoin* que permite a un administrador eliminar un nodo par de un canal. El nodo par debe detenerse cuando se ejecuta el comando para que se puedan limpiar los artefactos propios del canal, como: el ledger del canal y la base de datos de estado. Una vez que se reinicia el nodo par, no recibirá nuevos bloques para el canal.

A partir de Fabric v2.4 se puede determinar el ID del paquete de un chaincode sin instalarlo en los nodos pares mediante el nuevo comando de ciclo de vida del nodo par *chaincodecalculatepackageid*. Entre otras virtudes, posibilita verificar si un paquete de chaincode específico está instalado o no sin necesidad de instalarlo.

Capítulo 3

Propuesta

Capítulo 4

Detalles de Implementación y Experimentos

Conclusiones

Conclusiones

Anexos

anexos

Recomendaciones

Recomendaciones

Bibliografía

[1] Elli Androulaki y col. «Hyperledger fabric: a distributed operating system for permissioned blockchains». En: *Proceedings of the thirteenth EuroSys conference*. 2018, págs. 1-15 (vid. pág. 5).

[2] Andreas M Antonopoulos y Gavin Wood. *Mastering ethereum: building smart contracts and dapps*. O'reilly Media, 2018 (vid. pág. 2).

[3] *Certificate Authorities*. URL: `https://hyperledger-fabric.readthedocs.io/en/latest/identity/identity.html` (visitado 26-10-2022) (vid. pág. 9).

[4] *CouchDB*. URL: `http://couchdb.apache.org/` (visitado 12-10-2022) (vid. pág. 13).

[5] *Digitizing Global Trade with Maersk and IBM. 2018*. URL: `https://www.ibm.com/blogs/blockchain/2018/01/digitizing-global-trade-maersk-ibm/` (visitado 12-10-2022) (vid. pág. 5).

[6] *Everledger—Tech for Good Blockchain Solutions*. URL: `https://www.everledger.io/` (visitado 12-10-2022) (vid. pág. 5).

[7] *Go SDK for Fabric Client/Application*. URL: `https://github.com/hyperledger/fabric-sdk-go` (visitado 12-10-2022) (vid. pág. 11).

[8] *GoLevelDB*. URL: `https://github.com/syndtr/goleveldb` (visitado 12-10-2022) (vid. pág. 13).

[9] Jens Gottlieb, Elena Marchiori y Claudio Rossi. «Evolutionary algorithms for the satisfiability problem». En: *Evolutionary computation* 10.1 (2002), págs. 35-50 (vid. pág. 13).

[10] Fabric Hyperledger. *Hyperledger fabricdocs documentation*. 2018 (vid. pág. 14).

[11] *Hyperledger Fabric*. URL: `https://github.com/hyperledger/fabric` (visitado 12-10-2022) (vid. pág. 5).

[12] *Java SDK for Fabric Client/Application*. URL: `https://github.com/hyperledger/fabric-sdk-java` (visitado 12-10-2022) (vid. pág. 11).

[13] Flavio Junqueira y Benjamin Reed. *ZooKeeper: distributed process coordination*. .ºReilly Media, Inc.", 2013 (vid. pág. 7).

[14] Sunny King y Scott Nadal. «Ppcoin: Peer-to-peer crypto-currency with proof-of-stake». En: *self-published paper, August* 19.1 (2012) (vid. pág. 1).

[15] *Membership Service Providers (MSP)*. 2022. URL: `https://hyperledger-fabric.readthedocs.io/en/latest/membership/membership.html` (visitado 26-10-2022) (vid. pág. 8).

[16] Satoshi Nakamoto. «Bitcoin: A peer-to-peer electronic cash system». En: *Decentralized Business Review* (2008), pág. 21260 (vid. pág. 1).

[17] Arvind Narayanan y col. *Bitcoin and cryptocurrency technologies: a comprehensive introduction*. Princeton University Press, 2016 (vid. pág. 1).

[18] *Node SDK for Fabric Client/Application*. URL: `https://github.com/hyperledger/fabric-sdk-node` (visitado 12-10-2022) (vid. pág. 11).

[19] Christos Papadimitriou y Paris Kanellakis. «On concurrency control by multiple versions». En: *ACM Transactions on Database Systems (TODS)* 9.1 (1984), págs. 89-99 (vid. pág. 8).

[20] *SecureKey: Building Trusted Identity Networks*. URL: `https://securekey.com/` (visitado 12-10-2022) (vid. pág. 5).

[21] Bayu Adhi Tama y col. «A critical review of blockchain and its current applications». En: *2017 International Conference on Electrical Engineering and Computer Science (ICECOS)*. IEEE. 2017, págs. 109-113 (vid. pág. 1).

[22] Martin Valenta y Philipp Sandner. «Comparison of ethereum, hyperledger fabric and corda». En: *Frankfurt School Blockchain Center* 8 (2017), págs. 1-8 (vid. pág. 2).

[23] Marko Vukolić. «Rethinking permissioned blockchains». En: *Proceedings of the ACM workshop on blockchain, cryptocurrencies and contracts*. 2017, págs. 3-7 (vid. pág. 10).

[24] *What's new in Hyperledger Fabric v2.x*. URL: `https://hyperledger-fabric.readthedocs.io/en/latest/whatsnew.html` (visitado 26-10-2022) (vid. pág. 14).

[25] Xiwei Xu y col. «A taxonomy of blockchain-based systems for architecture design». En: *2017 IEEE international conference on software architecture (ICSA)*. IEEE. 2017, págs. 243-252 (vid. pág. 1).