Universidad de La Habana
Facultad de Matemática y Computación

Análisis de rendimiento en Hyperledger Fabric

Autor:
Bryan Machín García

Tutores:

MsC. Camilo Denis González
DrC. Carlos Miguel Legón

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

3 de diciembre de 2022

https://github.com/BryanMachin/Performance-Analysis-in-Hyperledger-Fabric

Este trabajo se lo dedicado en especial a mis padres Alberto Machín Santos y Marisela García Pérez, por su apoyo incondicional y la Fe inquebrantable que siempre han depositado en mí. Son una parte importante de mis logros y una motivación para seguir esforzándome. Desde pequeño me encaminaron hacia una formación de excelencia.

Agradecimientos

En primer lugar, quiero agradecer a mis tutores Camilo Denis y Miguel Legón, por su extraordinaria disposición y apoyo. Agradezco las valoraciones y recomendaciones ofrecidas por los profesores del grupo de criptografía, en especial, a los profesores Daniel y Miguel Katrib. También quiero agradecer a mi novia Daniela Rodríguez por su apoyo en la investigación, a mis padres y a todos aquellos que de una manera u otra han contribuido.

Opinión del tutor

El trabajo de diploma *Análisis de rendimiento en Hyperledger Fabric* del estudiante Bryan Machín García para optar por el título de Licenciado en Ciencia de la Computación, es un tema de investigación de suma importancia para el Instituto de Criptografía porque permite mejorar la eficiencia de los servicios basados en redes *blockchain* de Hyperledger Fabric.

El diplomante ha mostrado interés por la investigación, ha sido receptivo a las sugerencias, críticas y opiniones de ambos tutores; ganando en conocimiento, dominio del tema, y habilidades para reajustar sus experimentos y presentar resultados.

Puedo afirmar que ha mostrado disciplina y dedicación en la realización de las tareas, tanto en la redacción del trabajo de diploma, como en la organización e investigación del tema, lo cual se ve reflejado en los resultados entregados por el diplomante. Para ello, comenzó con la asimilación y estudio de las tecnologías indicadas por los tutores, mostrando además buenas capacidades de asimilación e independencia.

En consecuencia, se define que la tesis cumple con el rigor metodológico, científico y está en función de los requisitos definidos, partiendo además del estudio de fuentes y publicaciones recientes relacionadas al tema de investigación.

Por tanto, hago constar que la tesis reúne los estándares metodológicos exigidos por la Facultad de Matemática y Computación de la Universidad de la Habana, para ser presentada y sometida a evaluación en su ejercicio de defensa.

Felicito al diplomante por haber respondido con responsabilidad al desafío del estudio y haber finalizado exitosamente su trabajo de diploma.

MsC. Camilo Denis González

Resumen

Hyperledger Fabric es actualmente una de las plataformas *Blockchain* comerciales más populares. Con la posibilidad de ejecutar contratos inteligentes desarrollados en lenguajes de programación de propósito general, Fabric se ha convertido en una de las plataformas más populares en el área empresarial. Entre sus bondades también se tiene un conjunto de parámetros configurables, que en dependencia del volumen de transacciones por segundo a la que opere la red, pueden mejorar o no, su rendimiento. Para un desarrollador potencialmente interesado, no es trivial decidir si una determinada configuración de red de Fabric cumplirá con las expectativas requeridas con respecto al rendimiento. Por lo tanto, este trabajo muestra un análisis de rendimiento para Hyperledger Fabric v2.4 y se centra en ofrecer una configuración para dos posibles escenarios de trabajo de: bajos y altos volúmenes de transacciones por segundo. Las métricas de latencia, junto con la escalabilidad en los nodos de la plataforma y el tamaño en los bloques del canal de comunicación son las variables de estudio analizadas. Los resultados muestran, de acuerdo a los parámetros ofrecidos como candidatos, para cada uno de los escenarios planteados, una configuración que optimiza el rendimiento o minimiza la latencia media, en dependencia de las necesidades del operador de la red.

Abstract

Hyperledger Fabric is currently one of the most popular commercial Blockchain platforms. With the ability to run smart contracts developed in general-purpose programming languages, Fabric has become one of the most popular platforms in the enterprise arena. Among its benefits there is also a set of configurable parameters, which depending on the volume of transactions per second at which the network operates, may or may not improve its performance. For a potentially interested developer, it is not trivial to decide whether a given fabric configuration will meet the required performance expectations. Therefore, this work shows a performance analysis for Hyperledger Fabric v2.4 and focuses on offering a configuration for two possible work scenarios: low and high volumes of transactions per second. The latency metrics, along with the scalability in the platform nodes and the size of the communication channel blocks are the study variables analyzed. The results show, according to the parameters offered as candidates, for each of the proposed scenarios, a configuration that optimizes performance or minimizes average latency, depending on the needs of the network operator.

Índice general

Índice de figuras

Índice de tablas

Introducción

En 2009 Satoshi Nakamoto introduce Bitcoin [21], 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 [28].

Cuando se trata de almacenar y compartir datos entre diferentes entidades, la base de datos centralizada tiene algunas desventajas para garantizar la integridad de sus datos. Puede estar vulnerable si ocurre algún ataque o falla y no se encuentra *clusterizada*, o no se tiene copia de seguridad. Además, por motivos de privacidad, pudiera no ser aceptable almacenar los datos en un tercero [32]. 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 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 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.

Se puede afirmar que una red *Blockchain* es un conjunto de nodos, conectados entre sí usando un protocolo común, con el objetivo de validar y almacenar la información en una red *P2P*[1]. La información se registra en una base de datos conocida como *ledger* (registro o libro mayor), de ahí el acrónimo *DLT* (*Distributed Ledger Technology*) asociado a este tipo de tecnologías. En la *blockchain* el *DLT* registra todas las transacciones entre nodos que han ocurrido desde la creación de la red garantizando su inmutabilidad.

En la literatura, es posible agrupar redes *blockchain* como *"permisionadas"*[2] y *"no*

[1]Modelo de comunicación descentralizado, donde cada parte actúa por igual y pueden tener la función de servidor o de cliente.

[2]Los participantes deben obtener un permiso y suele ser gestionado por alguna CA (*Certificate Authority*).

permisionadas". Gracias a las características de las *"no permisiondas"* es posible definir redes seudónimas de accesibilidad pública con transacciones transparentes.

Las redes *blockchain "permisionadas"* se emplean, en mayor medida, para entornos empresariales donde es necesario tener en cuenta aspectos como la privacidad y confidencialidad de los datos. Permite compartir datos y acceder a entidades/usuarios de confianza específicos [32]. 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 [22], lo que sucede si una sola entidad posee más del 51 % del poder computacional de la *blockchain*. En el caso de Peercoin [18] 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 confianzas 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 [29], principalmente en ecosistemas empresariales.

0.1. Problema a resolver

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 ejecu-

ten 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 a resolver consiste en definir una configuración para casos de uso con un bajo número de transacciones por segundo y otro para los casos de un elevado número de transacciones por segundos; de tal modo que se pueda configurar el canal de comunicación donde se desarrollan las transacciones, para cada escenario, que propicie el mejor rendimiento posible.

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.

0.2. Objetivos

0.2.1. Objetivo General

Determinar como pueden ser las configuraciones óptimas para canales de redes *blockchain* de Hyperledger Fabric, que según los escenarios de uso, son de mayor o menor volumen de transacciones.

0.2.2. Objetivos Específicos

- Configurar y Desplegar una red *blockchain* de Hyperledger Fabric para los escenarios de 10 y 100 transacciones por segundo respectivamente.
- Medir la latencia (mínima, máxima y promedio) y el rendimiento de las redes desplegadas.
- Analizar los valores de las métricas estudiadas.
- Determinar un conjunto de parámetros de configuración óptimos para cada escenario, basado en la latencia promedio y el rendimiento de la red.

Capítulo 1

Antecedentes en el análisis de rendimiento en Hyperledger Fabric

Los estudios a las versiones más recientes de Hyperledger Fabric se remontan a noviembre del 2020, cuando el laboratorio de tecnología RF y comunicaciones móviles de la Universidad de Ciencias Aplicadas de Osnabrück en Alemania realizó un estudio al rendimiento de la plataforma en su versión 2.0, estrenada en febrero de ese mismo año [8].

El propósito de ese trabajo fue obtener una evaluación definitiva de cómo la variación de los diferentes participantes de la red influyen en su rendimiento. En particular, el número de nodos pares, organizaciones, nodos ordenadores y el tamaño de los bloques. Julian Dreyer desarrolló una herramienta [10] para Fabric 2.0 que genera automáticamente la red requerida con el número deseado de componentes e instala el contrato inteligente de prueba. Las métricas de rendimiento no se miden directamente, sino emplearon fórmulas para obtenerlas:

$$Rendimiento = \frac{1s}{\frac{\sum_i^N t\_tx_i}{N}} \text{ [TPS]}$$

$$Tasa\ de\ Error = \frac{F}{N} \cdot 100\,\%$$

$$Latencia = t\_tx_i + t\_b_i \text{ [ms]}$$

donde:

- t_tx_i es el tiempo de la i-ésima transacción.
- t_b_i es el tiempo de confirmación del i-ésimo bloque.

- F denota el número de transacciones fallidas.
- N el total de transacciones efectuadas.

El estudio se realizó en un sistema operativo Ubuntu 16.04 con dos CPU Intel Xeon E5-2690 y 128GB 2133MHz RAM. Se utilizó Docker 19.03.8 para las imágenes de Fabric 2.0.

Las configuraciones empleadas para la red de Fabric se muestran en la tabla 4.1 y siguen un esquema abreviado: `<organizaciones>-<pares>-<ordenadores>-<nodos-kafka>`. Para los esquemas de pruebas individuales en cada tasa de transacción se utiliza la notación: `bs<tamaño de los bloques>-<transacciones por segundos>tps`.

En los escenarios de prueba, todos los pares configurados están asignados para cumplir con la política de aprobación.

Primero fijaron el número de nodos ordenadores, nodos kafka y organizaciones en uno, para garantizar un entorno de referencia y escalar los nodos pares comenzando con solo dos hasta llegar a dieciséis por organización. La Figura 1.1 muestra gráficamente los tiempos de ejecución para cada configuración. El eje x también incluye las variaciones en el tamaño de los bloques y las *TPS*.

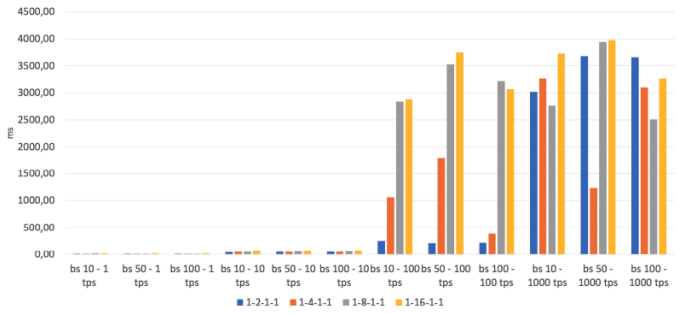


Figura 1.1: Tiempo de ejecución en una organización. *Fuente: Performance analysis of hyperledger fabric 2.0 blockchain platform.*

Al aumentar la cantidad de nodos pares dentro de una organización, se percibe un aumento significativo en el tiempo de ejecución de la transacción. La introducción de más nodos pares conduce a una mayor sobrecarga de comunicación y sincronización dentro de la red. Al analizar el mismo escenario con dos organizaciones, la tendencia general con respecto al impacto en el rendimiento continúa. En el caso de los tiempos de ejecución, la configuración con dos organizaciones, produce tiempos

en las transacciones más bajos en promedio. Sin embargo, esta tendencia particular no continúa cuando se considera la misma configuración con cuatro organizaciones. Los resultados tienen una mayor similitud a los de la figura 1.1, lo que sugiere una optimización de los tiempos de transacción para dos organizaciones. Nasir en [23] había observado igual comportamiento para versiones anteriores de Fabric.

El impacto negativo en el aumento del número de pares también influye en los tiempos de confirmación de los bloques. La Figura 1.2 muestra los tiempos de confirmación de los bloques para cada caso de prueba en una organización.

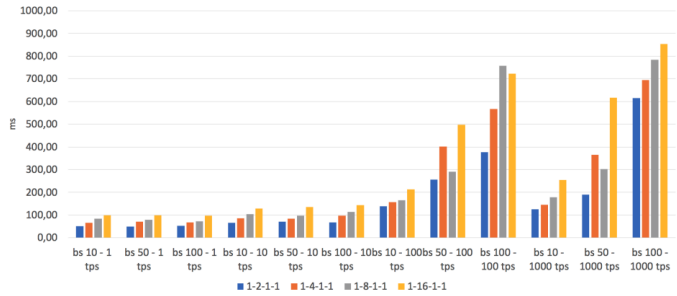


Figura 1.2: Tiempo de confirmación de bloques en una organización. *Fuente: Performance analysis of hyperledger fabric 2.0 blockchain platform.*

Con respecto a la tasa de error, se muestra generalmente creciente con TPS más elevadas como se observa en la figura 1.3. Sin embargo, un tamaño de bloque mayor tiene un efecto positivo, en promedio, en la tasa de error. Causando menos errores en tiempo de ejecución.

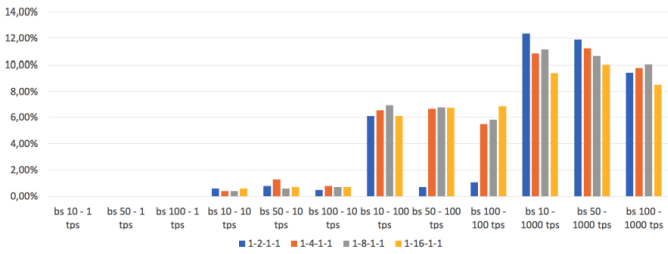


Figura 1.3: Tasa de error en una organización. *Fuente: Performance analysis of hyperledger fabric 2.0 blockchain platform.*

El servicio de ordenación no ejecuta transacciones, por tanto, es de esperar que al escalar los nodos ordenadores apenas afecten los tiempos en las transacciones. Aunque, si deben influir en los tiempos de confirmación de los bloques. Un aumento en el

número de nodos ordenadores requiere una mayor sincronización entre ellos, que se refleja en el rendimiento de la red.

Las pruebas realizadas para comprobar el impacto de la variación del número de nodos ordenadores no arrojaron un resultado generalizable, como en el caso de los nodos pares. Se puede apreciar en las figuras 1.4 y 1.5 .

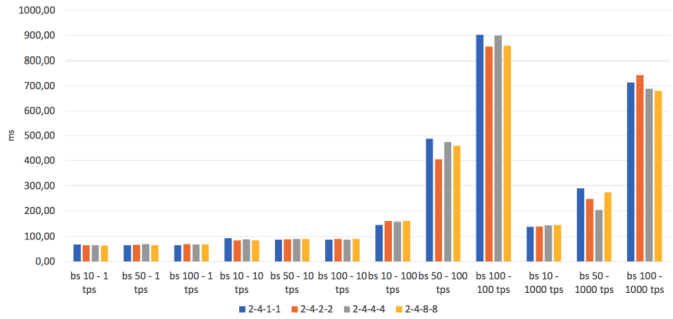


Figura 1.4: Tiempo de confirmación de bloques en dos organización. *Fuente: Performance analysis of hyperledger fabric 2.0 blockchain platform.*

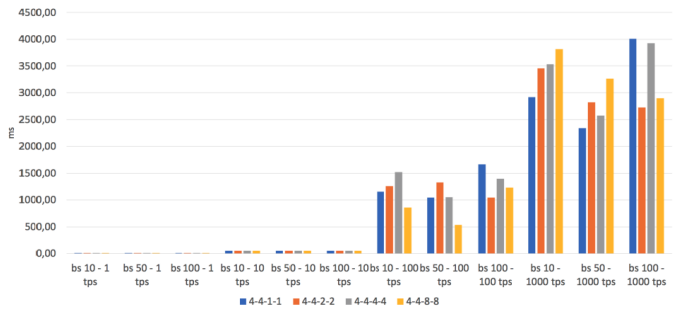


Figura 1.5: Tiempo de ejecución en cuatro organización. *Fuente: Performance analysis of hyperledger fabric 2.0 blockchain platform.*

Se confirma que a mayor tamaño de bloques, mayor será su tiempo de confirmación en el *ledger*. Debido a que se incluyen más transacciones y, por lo tanto, requiere más tiempo para validarlas. Esto conduce a que el tamaño del bloque está en correlación directa con la latencia global del sistema.

En resumen, el rendimiento depende en gran medida de la cantidad de nodos pares. Más nodos pares de respaldo activos dentro de la red ocasiona menor rendimiento. En el caso de la latencia depende, tanto del tamaño del bloque, como de la cantidad de

nodos pares de aprobación. Cuando el tamaño de los bloques es pequeño, generalmente permite una latencia menor. Sin embargo, debe seleccionarse en función de la tasa de *TPS* esperada porque un llenado rápido de los bloques puede resultar en un proceso de validación frecuente. Se afirma además que solo el tamaño de los bloques y la tasa de *TPS* tienen una influencia marcada en la tasa de error. Un tamaño de bloque más grande producirá menos errores que uno más pequeño.

Capítulo 2

Fundamentos Teóricos

Hyperledger Fabric [15] 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* [7], *SecureKey* [26] y *Everledger* [9]. 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. Activo

Los activos pueden ir desde lo tangible (bienes raíces y *hardware*) hasta lo intangible (contratos y propiedad intelectual). Son representados dentro de la *blockchain* como una colección de pares *llave-valor*.

2.1.2. Base de datos de estado

Fabric admite dos alternativas para almacenar el último estado de los activos: *CouchDB* [6] y *GoLevelDB* [12]. Ambas almacenan un conjunto de *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*.

2.1.3. Chaincode

El *chaincode* [5] 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. Se pueden implementar en varios lenguajes de programación de propósito general como son Go, Javascript, Typescript y Java. Son invocados por una aplicación externa a la *blockchain* cuando necesita interactuar con el registro o base de datos de estado.

2.1.4. Contrato inteligente

Los contratos inteligentes constituyen la lógica de negocio, es decir, es un programa que ofrece la lógica de transacción que controla el ciclo de vida de un objeto dentro de la base de datos de estado. Están escritos dentro de un *chaincode*, que puede contener más de un contrato inteligente [27].

2.1.5. Libro mayor

El libro mayor o *ledger* mantiene un registro secuencial, a prueba de manipulaciones, de todas las transiciones de estado en Hyperledger Fabric. Las transiciones de estado son el resultado de la invocación de funciones (*transacciones*) del *chaincode* 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. Existe un libro mayor por cada canal.

El *libro mayor* está compuesto por una *cadena de bloques* para almacenar el registro inmutable y secuenciado en bloques, así como *la base de datos de estado*. Véase en la figura 2.1.

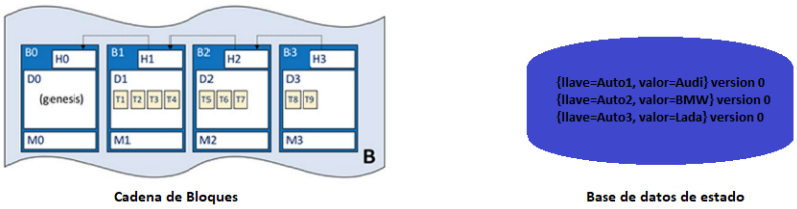


Figura 2.1: Libro mayor.

2.1.6. Nodo Par

Los nodos pares son los encargados de ejecutar los *chaincodes* y mantener el *ledger* en un sistema de archivos. Entre los nodos pares, se distinguen los que mantienen la lógica del *chaincode* y ejecutan mediante transacciones. Sin tener en consideración esta diferenciación, cada nodo par mantiene una copia del libro mayor para cada canal del que es miembro.

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 el canal de comunicación solo pueden ser procesadas por los nodos miembros. El *ledger* y los *chaincodes* son definidos por canal.

2.1.8. 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 se basa en un diseño de consenso determinista para garantizar que cualquier bloque validado por un nodo par, sea final y correcto. Además, el consenso es aplicable por canal, es decir, no hay un orden definido para la transacción a través de los canales. 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* [17] 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 exclusivamente a entornos de desarrollo, y consiste en un único nodo ordenador. En la actualidad es considerado como obsoleto, ya es posible usar un único nodo ordenador con el protocolo Raft para entornos de desarrollo.

2.1.9. Nodo Ordenador

Los nodo ordenador participan 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.

2.1.10. Tamaño de bloque

Las transacciones se procesan por lotes en el nodo ordenador y se entregan como un bloque a los nodos pares directamente o 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.

[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.

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

2.1.11. 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.

2.1.12. 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.
- Control de Multiversión de concurrencia [25] *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 fantasmas la consulta se vuelve a ejecutar y se comparan los *hash* 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.

2.1.13. 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 [13], que es *NP-Completo*. Adicionalmente, incluye verificar la identidad. La complejidad de la política de aprobación afectará los recursos y el tiempo necesario para recopilar y evaluar transacciones.

2.1.14. Servicio de membresía

El servicio de membresía o proveedor de servicios de membresía *MSP* [19] 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.

Autoridad certificadora

Para poder participar en la red *blockchain* un nodo debe tener, una identidad digital que es 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*) [4]. Las *CA* tienen un certificado, que ponen a disposición de todos para que los consumidores de identidades emitidas por una determinada *CA* puedan verificar que el certificado sólo pudo ser generado por el titular de la clave privada correspondiente. Véase en la figura 2.2.

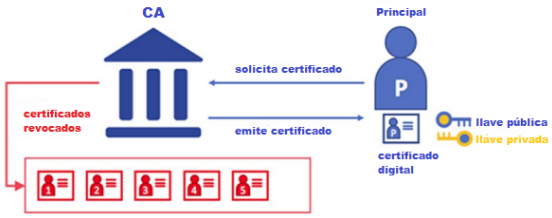


Figura 2.2: Autoridad certificadora.

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.

2.2. Flujo de transacciones en Hyperledger Fabric

A diferencia de otras redes *blockchain* que emplean un modelo de *ordenación-ejecución* [30] de transacciones, Fabric emplea un modelo de simulación, validación y confirmación de transacciones.

La figura 2.3 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 Fase de ordenación: Ordena las transacciones a través de un protocolo de consenso.

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

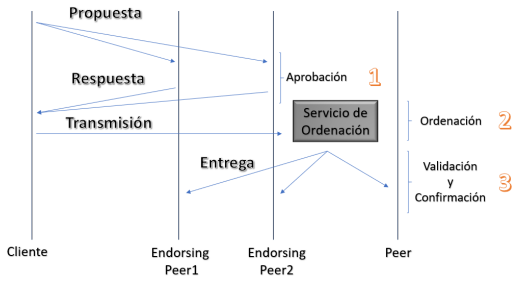


Figura 2.3: Flujo de transacciones.

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. Luego 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* [11], [24], [16], 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 la 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.

En primer lugar, 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*. 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 de *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 sean iguales. 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 y confirmación

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*. Por último, el libro mayor se actualiza con los conjuntos de escritura de transacciones válidas.

2.3. 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 [31]. 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 [14]:

- **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 creó 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 un *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 los *chaincodes* 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 e instalado con todas sus dependencias en su interior. Ahora es posible ejecutar un *chaincode* como un servicio externo, al cual el nodo par se conecta y utiliza para su ejecución.

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.
- ***Snapshot* 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.

A partir 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, que es cambiada por 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 por ejemplo, el *ledger*. 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.

2.4. Caliper

Caliper [3] es un *framework* para determinar el rendimiento de plataformas *blockchain*. Permite a los usuarios probar soluciones de *blockchain* con casos de uso personalizados y obtener un conjunto de resultados de pruebas de rendimiento. Actualmente soporta soluciones de Hyperledger Besu, Ethereum, Hyperledger Fabric y FISCO BCOS. Expone una serie de métricas, tales como: rendimiento, latencia (mínima, máxima y promedio) en el procesamiento de transacciones y el consumo de recursos del *hardware* (CPU, RAM, red).

2.5. Minifabric

Minifabric [20] es una herramienta que permite configurar una red de Hyperledger Fabric. Posibilita expandir la red con varias organizaciones externas y unirlas en un canal. También se pueden instalar y actualizar *chaincodes*, invocar transacciones, inspeccionar el libro mayor y cambiar la configuración del canal. Está integrada con Hyperledger Explorer y Caliper.

Para iniciar el trabajo en *minifabric* se debe tener, en un directorio, el script *minifab*[3] y un archivo denominado *spec.yaml*, donde se nombran los componentes que conforman la red (CA, nodos pares y nodos ordenadores). Luego, desde la terminal de *bash* se ejecutan los comandos para inicializar una red de Fabric y, de acuerdo a las necesidades, el programa descarga las imágenes de docker necesarias para su funcionamiento.

Entre sus principales comandos se encuentran:

- minifab up: Inicia, por defecto, la última versión de Hyperledger Fabric en la computadora. Para emplear una versión específica se utiliza el parámetro *-i*, ejemplo, *-i* 2.4.
- minifab cleanup: Elimina el contenedor de docker y las imágenes creadas. También elimina el contenido de la red en el directorio de trabajo.
- minifab down: Detiene la red, pero conserva el contenido en el directorio de trabajo.
- minifab restart: Inicializa la red si el contenedor de docker y las imágenes están en el sistema.

[3]Para obtener el script se debe ejecutar en una terminal de *bash*: curl -o minifab -sL https://tinyurl.com/yxa2q6yr

- minifab create: Crea un canal especificando su nombre seguido del parámetro *-c*.
- minifab join: Une las organizaciones al canal.
- minifab install: Instala un *chaincode*, especificando con los parámetros *-v*, *-n* y *-l*, la versión, el nombre y el lenguaje de programación respectivamente.
- minifab approve: Aprueba el *chaincode* en el canal.
- minifab commit: Confirma el *chaincode* en el canal.
- minifab invoke: Invoca un *chaincode*, especificando con *-n* y *-p*, su nombre y parámetros respectivamente.
- minifab caliperrun: Ejecuta Caliper para una configuración predeterminada por el sistema si en su directorio de trabajo no cuenta con una configuración personalizada.

Ofrece un entorno agradable y fácil de aprender para interactuar con una red de Fabric a menor escala. Es recomendado para todo aquel que quiera incursionar en la plataforma de Hyperledger Fabric y cuente con conocimientos básicos. Además con herramientas como Caliper o Explorer es posible determinar el rendimiento y escalabilidad de varias configuraciones, resultando provechoso también en ámbitos investigativos.

Capítulo 3

Propuesta

La variación en las configuraciones de Hyperledger Fabric tiene una influencia directa en el tiempo de confirmación de las transacciones y el rendimiento de la red *Blockchain*. Se realizó un análisis del comportamiento de varias configuraciones de redes (véase en la tabla 4.2), donde varían el número de nodos pares y el tamaño de los bloques en la configuración del canal de comunicación. Se tomaron como variables de estudio: latencia mínima, latencia máxima, latencia promedio y el rendimiento de la red, en función del número de transacciones procesadas en un segundo. Para evaluar el desempeño de los nodos pares, se fijó el número de nodos ordenadores en 1 para evitar sesgos en las mediciones. Cada nodo par fue configurado para cumplir con la política de aprobación. Las configuraciones se evaluaron, durante 60 segundos, para dos escenarios con un valor fijo de acuerdo al volumen de transacciones: bajo y alto, con 10 y 100 transacciones por segundo respectivamente. Como factor determinante se estudió la influencia del tamaño de los bloques en la latencia promedio y el rendimiento. Para esto se evaluaron tres posibles candidatos a tamaño de bloques: 10, 50 y 100. Para los escenarios de bajo y alto volumen de transacciones se propuso un conjunto de parámetros que, de acuerdo a las necesidades, maximizan el rendimiento de las componentes en la red o minimiza la latencia promedio de las transacciones.

Capítulo 4

Detalles de Implementación y Experimentos

El estudio se realizó en un sistema operativo GNU/Linux 22.04 LTS con una CPU Intel Core i3-5020U 2.20Ghz y 12GB de RAM a 1600MHz de velocidad. Se utilizó Docker 19.03.8 para las imágenes de Fabric v2.4.

Para configurar la red se utilizó Minifabric en su versión más reciente. Las pruebas de rendimiento se realizaron con Caliper v0.5.0 empleando el *chaincode samplecc* suministrado por Minifabric.

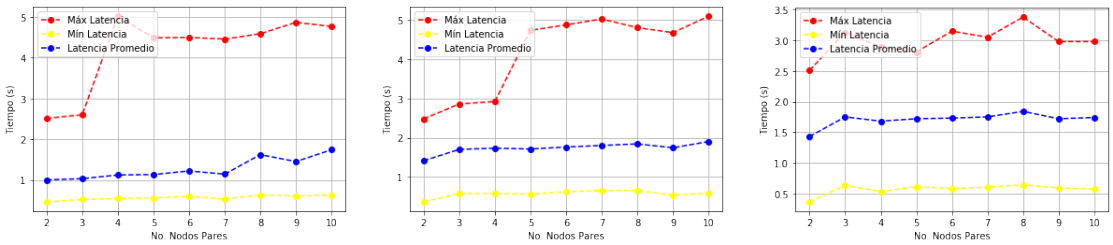
El nodo ordenador empleó el mecanismo de consenso *raft* en cada una de las configuraciones y los nodos pares, *GoLevelDB* como base de datos de estado. Se destaca el empleo del voto por mayoría en la política de aprobación, donde todos los nodos pares fueron igualmente configurados para participar en ella. Se tuvo en cuenta el estudio en una sola organización, pero de acuerdo a la política de aprobación implantada, su comportamiento simula para n nodos pares, una red con n organizaciones de un nodo par cada una, donde se evita el sesgo que propicia la conectividad en las organizaciones y a su vez se tiene un acceso homogéneo de los recursos del *hardware*.

4.1. Escenario de bajo volumen de transacciones

En las tablas 4.3, 4.4 y 4.5 se registran los datos obtenidos para el escenario de 10 TPS.

4.1.1. Análisis de latencia

De acuerdo a los datos suministrados se puede visualizar el comportamiento de las medida de latencia para las configuraciones con diferente tamaño en los bloques del canal.



(a) Tamaño de bloque igual 10. (b) Tamaño de bloque igual 50. (c) Tamaño de bloque igual 100.

Figura 4.1: Latencia en distintas configuraciones de nodos pares en una red con un solo nodo ordenador para escenarios de 10 TPS.

De acuerdo a la latencia mínima, para los tres casos mantienen un comportamiento similar, marcado por una leve tendencia al crecimiento con el aumento del número de nodos pares en la red. En el caso de la red configurada con un tamaño de bloque igual a 10, los valores de latencia promedio son más próximos a los valores de latencia mínima con respecto a las demás configuraciones. Esto sucede porque al llegar las transacciones al servicio de ordenación, como el tamaño del bloque es menor que el resto, y a su vez tiene una capacidad no superior al número de transacciones suministradas por los clientes, el tiempo de espera, en promedio, de las transacciones válidas para cerrar el bloque es menor. Esta afirmación contrasta con la latencia máxima que, a menor tamaño de bloques, mayores valores alcanza producto a que ocurre un proceso de llenado más rápido originando una validación más frecuente en los nodos pares, que a su vez participan en el proceso de simulación de las transacciones elevando la probabilidad de saturación en su proceso de ejecución.

En la figura 4.2 se percibe que para un tamaño de bloque igual a 10, el valor de latencia promedio es mínimo para todos las configuraciones de nodos pares. Luego,

el tamaño de bloque 10 es un parámetro que optimiza la latencia promedio en la red, en comparación a los demás, por tanto, es un posible candidato a óptimo.

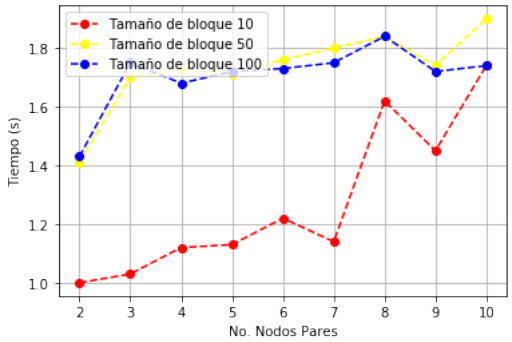
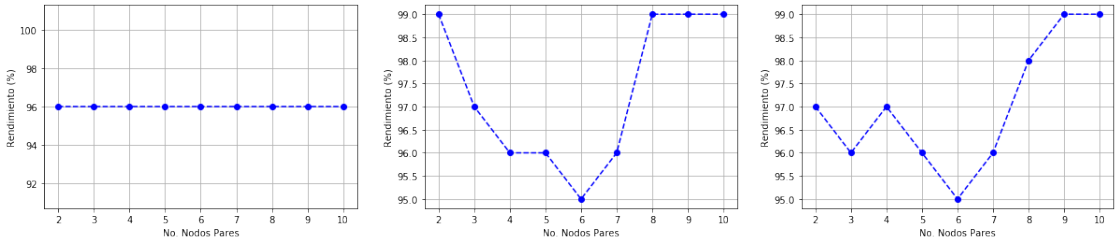


Figura 4.2: Latencia promedio en redes con distinto tamaño de bloque para escenarios de 10 TPS.

4.1.2. Análisis de rendimiento



(a) Tamaño de bloque igual 10. (b) Tamaño de bloque igual 50. (c) Tamaño de bloque igual 100.

Figura 4.3: Rendimiento en distintas configuraciones de nodos pares en una red con un solo nodo ordenador para escenarios de 10 TPS.

Para los tamaño de bloque planteados, los valores de rendimiento de la red oscilan entre el 95 % y 99 %. La red configurada con tamaño de bloque 10 mantiene un rendimiento constante de 96 % para las configuraciones de nodos pares estudiadas.

En el caso de la red con bloques de tamaño 50 se alcanza un mejor rendimiento en la mayoría de las configuraciones de nodos pares, como se puede ver en la figura 4.4.

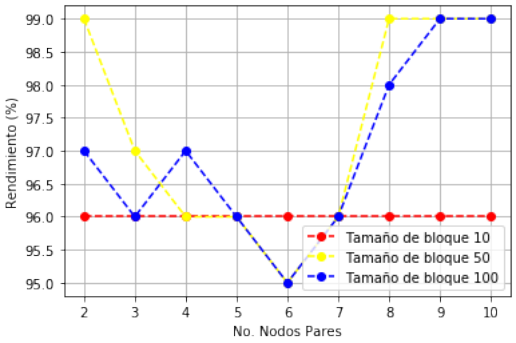


Figura 4.4: Rendimiento de redes con distinto tamaño de bloque para escenarios de 10 TPS.

Para determinar una configuración óptima, en relación a los valores de parámetros estudiados, se tuvo en cuenta el rendimiento en la red y la latencia promedio. Para establecer una comparación se llevaron las métricas a una escala global, normalizando sus valores. Estos valores son estimaciones promedio para configuraciones con una cantidad de nodos pares entre dos y diez por organización, donde el tamaño en los bloques representa el factor a optimizar. En la figura 4.5 podemos ver que el mejor rendimiento se alcanza para canales con tamaño de bloque igual a 50, pero a su vez la latencia promedio es mayor que la estimada para las configuraciones de canales de tamaño 10. Por tanto, los valores 10 y 50 representan óptimos locales, que en dependencia del escenario de uso, minimizan la latencia promedio y maximizan el rendimiento de la red respectivamente.

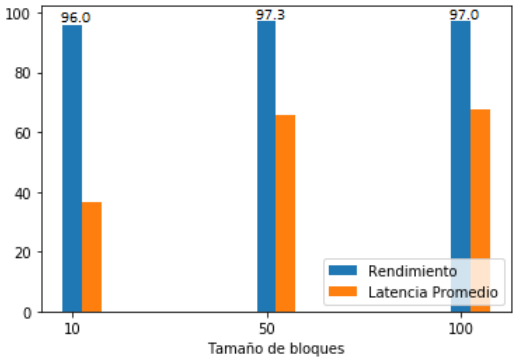


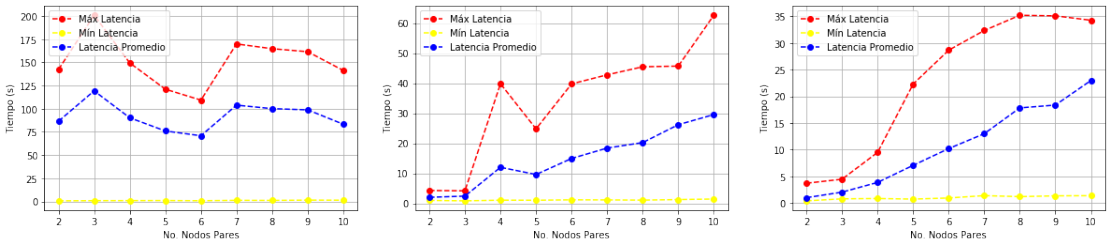
Figura 4.5: Evaluación de resultados para escenarios de 10 TPS.

4.2. Escenario de alto volumen de transacciones

En las tablas 4.6, 4.7 y 4.8 se registran los datos obtenidos para el escenario de 100 TPS.

4.2.1. Análisis de latencia

Las siguientes gráficas muestran el comportamiento de la latencia promedio en la red configurada con cada tamaño de bloque estudiado.



(a) Tamaño de bloque igual 10. (b) Tamaño de bloque igual 50. (c) Tamaño de bloque igual 100.

Figura 4.6: Latencia en distintas configuraciones de nodos pares en una red con un solo nodo ordenador para escenarios de 100 TPS.

En la primera gráfica podemos observar que los valores de latencia promedio oscilan entre los 70 y 120 segundos aproximadamente. Estos valores son, en extremo, elevados en comparación con las restantes configuraciones. Para el caso de los bloques de tamaño 100 el mayor valor alcanzado de latencia promedio, no supera los 25 segundos, quedando todos sus valores por debajo de los alcanzados por la configuración de bloques de tamaño 10. Lo mismo sucede en las redes con bloques de tamaño 50 que no superan los 30 segundos.

Para determinar el tamaño de bloque que ofrece mejor desempeño en redes con un número de nodos pares que varía de dos a diez por organización, con respecto a la latencia de las transacciones, se calculó la media de las latencias promedios de las configuraciones de nodos pares, por tamaños de bloques, y nos quedamos con la configuración de bloque que corresponda a la menor de ellas.

Los bloques de tamaño 10, 50 y 100 promedian una latencia media para las configuraciones de nodos pares de 92.1, 15.0 y 10.7 segundos respectivamente. Por tanto, las redes configuradas con bloques de tamaño 100 son óptimas respecto al conjunto

estudiado, propiciando la menor latencia promedio. Para establecer una comparativa visual nos podemos remitir a la figura 4.7.

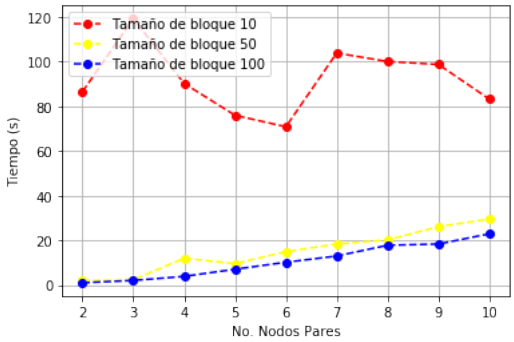
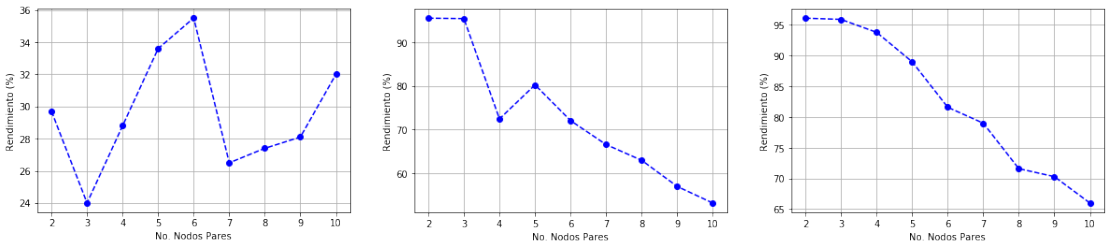


Figura 4.7: Latencia promedio en redes con distinto tamaño de bloque para escenarios de 100 TPS.

4.2.2. Análisis de rendimiento



(a) Tamaño de bloque igual 10. (b) Tamaño de bloque igual 50. (c) Tamaño de bloque igual 100.

Figura 4.8: Rendimiento en distintas configuraciones de nodos pares en una red con un solo nodo ordenador para escenarios de 100 TPS.

De acuerdo a la gráfica que representa el rendimiento para las redes con bloques de tamaño 10, se destaca el bajo rendimiento que logran en escenarios de altos volúmenes de transacciones. Entre las causas fundamentales que lo ocasionan está el rápido llenado de los bloques por el servicio de ordenación, que luego son enviados, con mayor frecuencia, a los nodos pares para el proceso de validación y a su vez se conjuga con el alto volumen de transacciones que deben simular en cada intervalo de tiempo. Los bloques de tamaño 50 y 100 manifiestan un mejor rendimiento debido a que compensan la frecuencia de validación por los nodos pares, con un aumento del

tiempo de conformación de bloques.

Si ilustramos las curvas de rendimiento en una sola gráfica, como en la figura 4.9, apreciamos que las redes con bloques de tamaño 100 superan para todas las configuraciones de nodos pares, a las redes con bloques de tamaño inferior. Se aprecia además, que para escenarios de altos volúmenes de transacciones por segundo, el aumento del número de nodos pares en las organizaciones reducen el rendimiento de forma considerable.

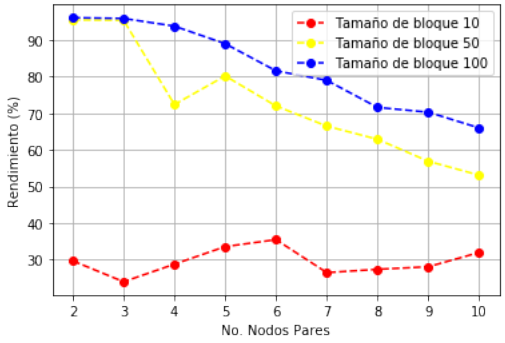


Figura 4.9: Rendimiento de redes con distinto tamaño de bloque para escenarios de 100 TPS.

En este escenario, la configuración para bloques de tamaño 100, propicia la menor latencia promedio en la red y el mejor desempeño en el rendimiento, del conjunto evaluado. En la figura 4.10 se puede ver el marcado contraste dado por la diferencia en el tamaño de los bloques. Se tiene una diferencia máxima de latencia superior a los 80 segundos, igual comportamiento tenemos en el rendimiento, donde los bloque de tamaño 10 ofrecen las métricas más desfavorables. A su vez, esta coincide con la configuración, por defecto, de la red para los canales de comunicación. Por tanto, confirma la necesidad de evaluar los parámetros del sistema antes de su puesta en producción.

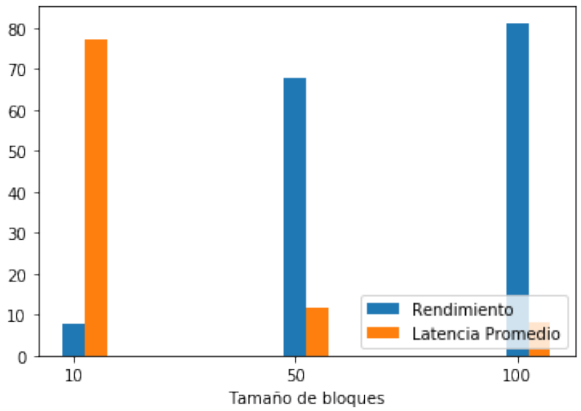


Figura 4.10: Evaluación de resultados para escenarios de 100 TPS.

Conclusiones

Los resultados obtenidos en el desarrollo de la investigación posibilitan un desempeño, en la plataforma de Hyperledger Fabric, más eficiente para los escenarios planteados. Si se desea operar una red con un bajo volumen de transacciones por segundo, se ofrecen configuraciones que garantizan para 10 TPS un elevado valor de rendimiento o una baja latencia media de la red, de acuerdo a los requerimientos particulares del operador de red. Para optimizar el rendimiento las tres configuraciones de tamaño de bloque para canales de comunicación: 10, 50 y 100; ofrecen un elevado por ciento de eficiencia, siendo estos mayores o iguales al 96 %. En el caso de la latencia para canales de tamaño 50 y 100 sus valores son suficientemente homogéneos como para decantarnos por cualquiera de ellos, pero son superiores en aproximadamente un 25 %, al valor de latencia para redes con canales de tamaño 10. Siendo la configuración, por defecto, de Hyperledger Fabric con valor 10, una excelente apuesta, para este tipo de escenarios de bajos volúmenes de transacciones por segundo.

Para escenarios donde el volumen de transacciones por segundo sea elevado y a su vez próximos a las 100 TPS, se ofrece a 100 como candidato a óptimo local de acuerdo al estudio de los tamaños de bloque de: 10, 50 y 100. Este valor garantiza un promedio de rendimiento superior al 80 % y un valor en la latencia por debajo de los 11 segundos. La configuración, por defecto, de Fabric para el tamaño de los bloques en los canales, en estos escenarios de altos valores de TPS ocasiona muy bajo rendimiento, el cual no supera el 10 % de TPS y un valor de latencia superior a los 90 segundos. Por tanto, confirma la necesidad del estudio de parámetros óptimos para determinados casos de uso, no siendo siempre la configuración establecida por el proveedor de la plataforma una opción a tener en cuenta.

Anexos

No. Organizaciones	No. Nodos Pares	No. Nodos Ordenadores/Nodos Kafka
1	2, 4, 8, 16	1
2	2, 4, 8, 16	1
4	4, 8, 16	1
8	8	1
2	2	2, 4, 8
4	4	2, 4, 8

Tabla 4.1: Configuraciones de la red. *Fuente: Performance analysis of hyperledger fabric 2.0 blockchain platform.*

No. Nodos Pares	No. Nodos Ordenadores	Tamaño de los bloques
2,3,4,5,6,7,8,9,10	1	10,50,100

Tabla 4.2: Configuraciones de la red.

No. Nodos Pares	No. Nodos Ordenadores	TPS	Máx. Latencia	Mín. Latencia	Latencia Promedio	Rendimiento(%)
2	1	10	2.51	0.46	1.00	96.0
3	1	10	2.60	0.52	1.03	96.0
4	1	10	5.02	0.54	1.12	96.0
5	1	10	4.49	0.56	1.13	96.0
6	1	10	4.50	0.59	1.22	96.0
7	1	10	4.46	0.53	1.14	96.0
8	1	10	4.59	0.63	1.62	96.0
9	1	10	4.87	0.60	1.45	96.0
10	1	10	4.77	0.63	1.74	96.0

Tabla 4.3: Evaluación escalando el número de nodos pares en una red con un tamaño de bloque igual a 10 y un valor estimado de envío de 10 TPS.

No. Nodos Pares	No. Nodos Ordenadores	TPS	Máx. Latencia	Mín. Latencia	Latencia Promedio	Rendimiento(%)
2	1	10	2.48	0.36	1.41	99.0
3	1	10	2.86	0.57	1.70	97.0
4	1	10	2.92	0.57	1.73	96.0
5	1	10	4.74	0.56	1.71	96.0
6	1	10	4.88	0.62	1.76	95.0
7	1	10	5.03	0.65	1.80	96.0
8	1	10	4.81	0.65	1.84	99.0
9	1	10	4.68	0.53	1.74	99.0
10	1	10	5.10	0.58	1.90	99.0

Tabla 4.4: Evaluación escalando el número de nodos pares en una red con un tamaño de bloque igual a 50 y un valor estimado de envío de 10 TPS.

No. Nodos Pares	No. Nodos Ordenadores	TPS	Máx. Latencia	Mín. Latencia	Latencia Promedio	Rendimiento(%)
2	1	10	2.51	0.36	1.43	97.0
3	1	10	3.13	0.63	1.75	96.0
4	1	10	2.89	0.53	1.68	97.0
5	1	10	2.81	0.61	1.72	96.0
6	1	10	3.15	0.58	1.73	95.0
7	1	10	3.05	0.60	1.75	96.0
8	1	10	3.38	0.64	1.84	98.0
9	1	10	2.98	0.59	1.72	99.0
10	1	10	2.98	0.57	1.74	99.0

Tabla 4.5: Evaluación escalando el número de nodos pares en una red con un tamaño de bloque igual a 100 y un valor estimado de envío de 10 TPS.

No. Nodos Pares	No. Nodos Ordenadores	TPS	Máx. Latencia	Mín. Latencia	Latencia Promedio	Rendimiento(%)
2	1	100	142.46	0.65	86.55	29.7
3	1	100	200.83	0.88	119.27	24.0
4	1	100	149.46	0.92	90.26	28.8
5	1	100	121.09	1.01	76.02	33.6
6	1	100	109.29	0.79	70.88	35.5
7	1	100	169.89	1.21	103.89	26.5
8	1	100	164.75	1.12	100.10	27.4
9	1	100	161.51	1.43	98.79	28.1
10	1	100	141.25	1.36	83.28	32.0

Tabla 4.6: Evaluación escalando el número de nodos pares en una red con un tamaño de bloque igual a 10 y un valor estimado de envío de 100 TPS.

No. Nodos Pares	No. Nodos Ordenadores	TPS	Máx. Latencia	Mín. Latencia	Latencia Promedio	Rendimiento(%)
2	1	100	4.28	0.98	2.01	95.5
3	1	100	4.16	0.83	2.41	95.4
4	1	100	39.80	1.06	11.99	72.4
5	1	100	24.78	1.03	9.60	80.2
6	1	100	39.75	1.17	14.93	72.0
7	1	100	42.77	1.14	18.40	66.5
8	1	100	45.44	1.07	20.22	62.9
9	1	100	45.67	1.26	26.17	56.9
10	1	100	62.59	1.44	29.54	53.1

Tabla 4.7: Evaluación escalando el número de nodos pares en una red con un tamaño de bloque igual a 50 y un valor estimado de envío de 100 TPS.

No. Nodos Pares	No. Nodos Ordenadores	TPS	Máx. Latencia	Mín. Latencia	Latencia Promedio	Rendimiento(%)
2	1	100	3.73	0.43	1.03	96.1
3	1	100	4.48	0.81	2.06	95.9
4	1	100	9.50	0.87	3.88	93.8
5	1	100	22.29	0.76	7.08	89.0
6	1	100	28.63	0.97	10.21	81.6
7	1	100	32.32	1.41	13.00	79.0
8	1	100	35.17	1.22	17.83	71.6
9	1	100	35.06	1.37	18.36	70.3
10	1	100	34.23	1.40	22.96	66.0

Tabla 4.8: Evaluación escalando el número de nodos pares en una red con un tamaño de bloque igual a 100 y un valor estimado de envío de 100 TPS.

Recomendaciones

El estudio se realizó sobre una muestra de parámetros de configuración (véase en la tabla 4.2), y se determinó con respecto a ellos, los que optimizan, tanto escenarios de 10, como de 100 transacciones por segundo. Se propone extender la investigación para una muestra de tamaño mayor en el número de nodos pares y tamaño de los bloques, que permita realizar un análisis en pruebas de hipótesis para tratar de obtener una generalización en el comportamiento de la variación de los parámetros.

Sería de utilidad extender las pruebas con *CouchDB* para establecer una comparativa con *GolevelDB*, y determinar su influencia en la latencia general de la red.

Determinar un tamaño adecuado de los bloques en un canal de comunicación, resulta una tarea compleja producto a la influencia directa que tiene sobre varias de las componentes de Fabric, como los nodos pares y los nodos ordenadores. Por tanto, trabajar en una herramienta que itere de manera eficiente para converger hacia un posible candidato a tamaño óptimo ayudaría en gran medida a encontrar los valores que mejoren el rendimiento en un escenario determinado.

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. 9).

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

[3] *Caliper*. URL: `https://hyperledger.github.io/caliper/v0.5.0/getting-started/` (visitado 19-11-2022) (vid. pág. 22).

[4] *Certificate Authorities*. URL: `https://hyperledger-fabric.readthedocs.io/en/latest/identity/identity.html` (visitado 26-10-2022) (vid. pág. 15).

[5] *Chaincode*. URL: `https://hyperledger-fabric.readthedocs.io/en/latest/fabric_model.html` (visitado 02-11-2022) (vid. pág. 10).

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

[7] *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. 9).

[8] Julian Dreyer, Marten Fischer y Ralf Tönjes. «Performance analysis of hyperledger fabric 2.0 blockchain platform». En: *Proceedings of the Workshop on Cloud Continuum Services for Smart IoT Systems*. 2020, págs. 32-38 (vid. pág. 4).

[9] *Everledger—Tech for Good Blockchain Solutions*. URL: `https://www.everledger.io/` (visitado 12-10-2022) (vid. pág. 9).

[10] *Fabric 2.0 Configurator*. 2020. URL: `https://github.com/JulianD267/Hyperledger-Fabric2-0-configurator` (visitado 05-11-2022) (vid. pág. 4).

[11] *Go SDK for Fabric Client/Application*. URL: `https://github.com/hyperledger/fabric-sdk-go` (visitado 12-10-2022) (vid. pág. 16).

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

[13] Jens Gottlieb, Elena Marchiori y Claudio Rossi. «Evolutionary algorithms for the satisfiability problem». En: *Evolutionary computation* 10.1 (2002), págs. 35-50 (vid. pág. 14).

[14] Fabric Hyperledger. *Hyperledger fabricdocs documentation*. 2018 (vid. pág. 18).

[15] *Hyperledger Fabric*. URL: `https://github.com/hyperledger/fabric` (visitado 12-10-2022) (vid. pág. 9).

[16] *Java SDK for Fabric Client/Application*. URL: `https://github.com/hyperledger/fabric-sdk-java` (visitado 12-10-2022) (vid. pág. 16).

[17] Flavio Junqueira y Benjamin Reed. *ZooKeeper: distributed process coordination*. .ºReilly Media, Inc.", 2013 (vid. pág. 12).

[18] 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. 2).

[19] *Membership Service Providers (MSP)*. 2022. URL: `https://hyperledger-fabric.readthedocs.io/en/latest/membership/membership.html` (visitado 26-10-2022) (vid. pág. 14).

[20] *Minifabric*. URL: `https://github.com/hyperledger-labs/minifabric` (visitado 20-11-2022) (vid. pág. 22).

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

[22] Arvind Narayanan y col. *Bitcoin and cryptocurrency technologies: a comprehensive introduction*. Princeton University Press, 2016 (vid. pág. 2).

[23] Qassim Nasir y col. «Performance analysis of hyperledger fabric platforms». En: *Security and Communication Networks* 2018 (2018) (vid. pág. 6).

[24] *Node SDK for Fabric Client/Application*. URL: `https://github.com/hyperledger/fabric-sdk-node` (visitado 12-10-2022) (vid. pág. 16).

[25] 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. 13).

[26] *SecureKey: Building Trusted Identity Networks*. URL: `https://securekey.com/` (visitado 12-10-2022) (vid. pág. 9).

[27] *Smart Contracts and Chaincode*. URL: `https://hyperledger-fabric.readthedocs.io/en/latest/smartcontract/smartcontract.html` (visitado 02-11-2022) (vid. pág. 10).

[28] 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).

[29] 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).

[30] Marko Vukolić. «Rethinking permissioned blockchains». En: *Proceedings of the ACM workshop on blockchain, cryptocurrencies and contracts*. 2017, págs. 3-7 (vid. pág. 16).

[31] *What's new in Hyperledger Fabric v2.x*. URL: `https://hyperledger-fabric.readthedocs.io/en/latest/whatsnew.html` (visitado 26-10-2022) (vid. pág. 18).

[32] 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. págs. 1, 2).