Universidad de La Habana
Facultad de Matemática y Computación

Parámetros óptimos para redes blockchain de Hyperledger Fabric basada en el análisis de la configuración de transacciones (configtx).

Autor:
Bryan Machín García

Tutores:

Camilo Denis González
Carlos Miguel Legón

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

13 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

Introducción

Hyperledger Fabric es una plataforma blockchain respaldada por una arquitectura modular que ofrece un alta grado de confidencialidad, resiliencia, flexibilidad y escalabilidad. Está diseñada para admitir implementaciones de diferentes componentes y adaptarse a las complejidades que existen en el ecosistema empresarial.

Propone un nuevo enfoque en su arquitectura para transacciones, conocido como $ejecutar - ordenar - validar$. Ejecutar una transacción implica, primero, comprobar su corrección, luego en el proceso de ordenación se efectúa un protocolo de consenso, y finalmente se chequea la validez de la transacción contra una política de endoso específica de la aplicación antes de confirmarla en el ledger. Su rápida expansión en el mercado ha incentivado a la comunidad de desarrolladores a contribuir a su constante evolución. Actualmente ocupa diversos casos de uso de relevancia, como $GlobalTradeDigitization$ [33], $SecureKey$ [15], $Everledger$ [5].

Debido a los numerosos componentes y fases, Fabric proporciona varios parámetros configurables, como el tama´ no del bloque, la política de respaldo, los canales, base de datos de estado. De ahí que uno de los principales retos a la hora de establecer una red blockchain eficiente es encontrar el conjunto correcto de valores para estos parámetros.

Motivación:

Con los [**watson53**] nuevos cambios introducidos en Hyperledger Fabric hace más demandada la plataforma en el sector empresarial, producto a su adaptabilidad en escenarios de consensos y políticas de gobierno para varias entidades. Resulta esencial el rendimiento de la plataforma en la transacción de información, lo cual nos motiva a realizar un estudio que posibilite obtener un conjunto de configuraciones óptimas.

Capítulo 1

Estado del Arte: Arquitectura de Hyperledger Fabric y parámetros configurables

Hyperledger Fabric [10] es una de las plataformas blockchain más populares administrada por Linux Foundation. Se basa en una plataforma privada de diferentes tipos de entidades, nodos pares, nodos ordenadores y clientes, pertenecientes a diferentes organizaciones. Cada una tiene una identidad en la red que es proporcionada por un Proveedor de Servicios de Membresía MSP [12], generalmente asociado con una organización. 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 [4], SecureKey [15] y Everledger [5].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.

1.1. Principales conceptos de Hyperledger Fabric

1.1.1. Chaincode

El chaincode [2] es un software que define uno o más activos junto a las instrucciones de la transacción para su modificación. Es el encargado de ofrecer la lógica del 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.

1.1.2. 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.

1.1.3. 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.

1.1.4. 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 [14] $MVCC$: 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. Esto es similar 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 compara 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.

1.1.5. 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.

1.1.6. Servicio de membresía

Un servicio de membresía o MSP abstrae todos los mecanismos y protocolos criptográficos detrás de la emisión de certificados, la validación de certificados y la autenticación de usuarios. Un MSP puede definir su propia noción de identidad y las reglas por las cuales se rigen esas identidades (validación de identidad) y se autentican (generación y verificación de firmas).

Una red blockchain de Hyperledger Fabric puede estar gobernada por uno o más MSP. Esto proporciona modularidad de las operaciones de membresía e interoperabilidad entre diferentes estándares y arquitecturas.

1.1.7. 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.

1.1.8. 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*[1]. 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 tenga un número máximo de nuevas transacciones desde el último bloque conformado, o se llegue al tiempo de espera configurado desde la última transacción producida.

1.1.9. 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.

1.2. Flujo de transacciones en Hyperledger Fabric

A diferencia de otras redes Blockchain que emplean un modelo de $ordenación - ejecución$ [16] de transacciones, Fabric emplea un modelo de simulación, validación y confirmación de transacciones.

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

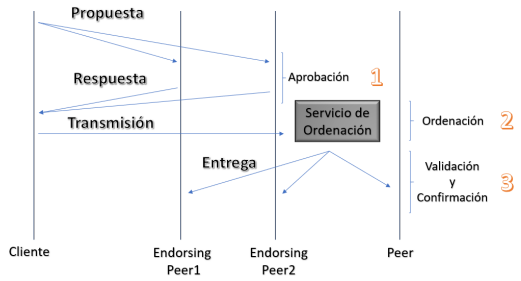

Figura 1.1: Flujo de transacciones.

La figura 1.1 muestra el flujo de la transacción que consta de 3 fases:

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

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

3 Fase de validación: validación seguida de confirmación en el libro mayor.

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.

1.2.1. Fase de aprobación

Una aplicación cliente que utiliza Fabric SDK [6], [13], [11], 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 actual estado 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.

1.2.2. 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*.

1.2.3. 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 $MVCC$.

1.2.4. 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.

1.3. 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 de nivel de componente.

1.3.1. Tamaño de bloque

Las transacciones se procesan por lotes en el nodo ordenador y entrega 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.

1.3.2. 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 [8], 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.

1.3.3. 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.

1.3.4. Recursos

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

1.3.5. Base de datos del libro mayor

Fabric admite dos alternativas para almacenar $llave - valor$, $CouchDB$ [3] y $GoLevelDB$ [7] 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 $JSON$.

1.4. 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. 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 [9]:

- **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 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 *Dockerdaemon* 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*.

Capítulo 2

Propuesta

Capítulo 3

Detalles de Implementación y Experimentos

Conclusiones

Conclusiones

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. 2).

[2] *Chaincodes*. 2017. URL: http://hyperledger-fabric.readthedocs.io/en/release-1.1/msp.html (visitado 12-10-2022) (vid. pág. 3).

[3] *CouchDB*. URL: http://couchdb.apache.org/ (visitado 12-10-2022) (vid. pág. 9).

[4] *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. 2).

[5] *Everledger—Tech for Good Blockchain Solutions*. URL: https://www.everledger.io/ (visitado 12-10-2022) (vid. pág. 2).

[6] *Go SDK for Fabric Client/Application*. URL: https://github.com/hyperledger/fabric-sdk-go (visitado 12-10-2022) (vid. pág. 6).

[7] *GoLevelDB*. URL: https://github.com/syndtr/goleveldb (visitado 12-10-2022) (vid. pág. 9).

[8] Jens Gottlieb, Elena Marchiori y Claudio Rossi. «Evolutionary algorithms for the satisfiability problem». En: *Evolutionary computation* 10.1 (2002), págs. 35-50 (vid. pág. 8).

[9] Fabric Hyperledger. *Hyperledger fabricdocs documentation*. 2018 (vid. pág. 9).

[10] *Hyperledger Fabric*. URL: https://github.com/hyperledger/fabric (visitado 12-10-2022) (vid. pág. 2).

[11] *Java SDK for Fabric Client/Application*. URL: https://github.com/hyperledger/fabric-sdk-java (visitado 12-10-2022) (vid. pág. 6).

[12] *Membership Service Providers (MSP)*. 2017. URL: .http://hyperledger-fabric.readthedocs.io/en/release-1.1/msp.html (visitado 12-10-2022) (vid. pág. 2).

[13] *Node SDK for Fabric Client/Application*. URL: https://github.com/hyperledger/fabric-sdk-node (visitado 12-10-2022) (vid. pág. 6).

[14] 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. 4).

[15] *SecureKey: Building Trusted Identity Networks*. URL: https://securekey.com/ (visitado 12-10-2022) (vid. pág. 2).

[16] Marko Vukolić. «Rethinking permissioned blockchains». En: *Proceedings of the ACM workshop on blockchain, cryptocurrencies and contracts*. 2017, págs. 3-7 (vid. pág. 5).