ISBN: 978-628-96613-3-0. ISSN: 2414-6390. Digital Object Identifier: https://dx.doi.org/10.18687/LACCEI2026.1.1.845

Aplicación de Patrones de Diseño Sobre Contratos Inteligentes, Soportados en Hyprledger Fabric

Roger Calderón Moreno[1]; Santiago Martínez Díaz [2]
[1,2] *Universidad de los LLanos, Colombia, rcalderonmoreno@unillanos.edu.co*, santiago.martinez@unillanos.edu.co

Resumen – Un patrón de diseño permite estructurar y optimizar soluciones en entornos tecnológicos complejos, como lo es caso de la tecnología de blockchain, facilitando la reutilización de conocimientos y experiencias previas. Este artículo presenta un análisis orientado a la incorporación de patrones de diseño en aplicaciones soportadas sobre en Hyperledger Fabric, una blockchain de tipo permisionada. Mediante una revisión de la literatura y la clasificación de estudios previos, se identificaron patrones de diseño aplicables a contratos inteligentes, dichos patrones fueron implementados en una red de prueba para evaluar su comportamiento y efectividad, considerando métricas de rendimiento, mejoras de seguridad y de escalabilidad. Los resultados obtenidos demuestran que la aplicación de estos patrones no solo mejora la eficiencia en la ejecución de transacciones, sino que también facilitan el proceso de desarrollo y mantenimiento del software.

Palabras clave-- Blockchain, Contrato Inteligente, Chaincode, Hyperledger Fabric, Ingeniería de Software, Patrones de diseño..

I. Introduccion

Dentro del proceso de construcción de software de calidad, se sugiere la aplicación de principios de diseño de software, buenas prácticas y la aplicación de patrones de diseño de software. Para el ámbito de blockchain de tipo permisionadas se propuso un trabajo de investigación que aplicó patrones de diseño en la tecnología blockchain, utilizando como plataforma base a Hyperledger Fabric, en adelante se denominará Fabric. El estudio se desarrolló en dos fases: una revisión del estado del arte mediante la recopilación y análisis de 53 artículos científicos relacionados al área de estudio, y la implementación experimental de diversos patrones de diseño en contratos inteligentes. Entre los patrones analizados se incluyen Checks Effects Interactions, Contract Registry, Limit Storage y Proxy, cuyo impacto se evaluó mediante pruebas de rendimiento en escenarios de creación de activos digitales. Los resultados evidenciaron mejoras significativas en términos de velocidad, eficiencia y mantenimiento del código, demostrando que la incorporación de estos patrones optimiza la ejecución de transacciones y mejora la mantenibilidad de los contratos. En particular, patrones como Limit Storage y Batch Processing lograron reducir los tiempos de ejecución en más del 50%, alcanzando incluso un 98% de optimización en casos específicos. Sin embargo, no en todos los patrones evidenciaron beneficios, es el caso de soluciones como Escrow o Proxy, los cuales presentaron limitaciones inherentes al contexto de Fabric, destacando la importancia de adaptar las estrategias a las particularidades técnicas de la plataforma. Además de las ganancias en eficiencia, se observaron mejoras transversales en seguridad (mediante la encapsulación de lógica crítica) y escalabilidad (gracias a la reducción de operaciones redundantes), lo que refuerza el valor de estos enfoques en el desarrollo de aplicaciones descentralizadas robustas.

II. Estado del arte

A. Blockchain

La tecnología blockchain se ha consolidado como una solución innovadora para garantizar la integridad y transparencia de los datos; ha sido definida de diversas maneras en la literatura. Según Belotti et al. [1], una blockchain se define como "una estructura de datos inmutable de solo lectura, en la que nuevos bloques se añaden al final del registro mediante el enlace con el identificador hash del bloque anterior." Esta característica de inmutable, garantiza la integridad y veracidad de los datos almacenados. Además, como señalan Six et al. [2], la blockchain posee propiedades fundamentales que la hacen especialmente robusta para entornos en los que la confianza es un factor crítico. Entre estas propiedades se destacan su descentralización por naturaleza, la transparencia y resistencia a manipulaciones e inmutabilidad. Estas propiedades intrínsecas consolidan a la blockchain como una solución ideal para aplicaciones en las que la seguridad, la confianza y la integridad de los datos son esenciales.

Desde la disciplina de la Ingeniería de Software se han propuesto varios cambios a partir de los retos propuestos por la implementación de la tecnología de blockchain para crear aplicaciones de software de calidad. Existen diversos estudios que evidencian y sugieren cambios, los autores de [3] resaltan el abrumador número de proyectos alrededor del uso de la tecnología blockchain y sus posibles campos de acción, pero se evidencia que muchos proyectos fracasan rápidamente, y aquellos que resultan en ejecución dejan mucho que pensar respecto de su calidad y la correcta aplicación de los principios básicos de la Ingeniería de Software. Existen otros trabajos similares como [4]- [6], que identificaron la necesidad de incluir dentro del proceso de construcción de software aquellos aspectos propios de la tecnología blockchain.

B. *Hyperledger Fabric*

En esta investigación se utilizó una tecnología blockchain conocida como Fabric, el cual es un framework de blockchain permisionado desarrollado por la Fundación Linux. A diferencia de las blockchain públicas, Fabric se caracteriza por su arquitectura modular y su capacidad para configurarse según las necesidades específicas de cada organización, según IBM, Fabric "es una plataforma de tecnología de libro mayor distribuido (DLT) de código abierto y de grado empresarial, diseñada para su uso en contextos empresariales." [7]. Mientras que muchas de las primeras plataformas blockchain están siendo adaptadas para uso empresarial, Fabric ha sido diseñada para uso empresarial desde el principio.

C. *Contratos Inteligentes(Chaincodes)*

Los contratos inteligentes (Smart Contracts) llamados Chaincodes en Fabric, son definidos por Kannengießer et al. [8]: "son programas de software que expresan la lógica formalizada en código para la aplicación confiable de acuerdos comerciales entre entidades definidas (por ejemplo, individuos, organizaciones o máquinas)". Esta definición resalta el papel fundamental de los contratos inteligentes en la automatización y el cumplimiento confiable de los acuerdos, eliminando la necesidad de intermediarios. Al codificar las condiciones de un acuerdo en un lenguaje de programación, los contratos inteligentes permiten la ejecución automática de transacciones y la actualización del estado en sistemas basados en blockchain. Asimismo, un contrato inteligente debe cumplir con ciertas propiedades esenciales para asegurar su correcto funcionamiento, tal como en [9] donde indican que los contratos inteligentes deben ser deterministas, terminales, aislados e inmutables.

D. *Patrones de diseño*

Number Un patrón de diseño es una solución a un problema de diseño, el cual debe haber comprobado su efectividad resolviendo problemas similares en el pasado, también tiene que ser reutilizable, por lo que se deben poder usar para resolver problemas parecidos en contextos diferentes [10]. Estos patrones constituyen una herramienta poderosa para estandarizar las soluciones a un mismo problema, lo que se traduce en mejoras en la eficiencia, la seguridad, la escalabilidad y la mantenibilidad del código. Para el caso de blockchain, los autores Wohrer y Zdun [11], indican que los patrones ayudan a resolver requisitos de aplicación que surgen de manera recurrente y abordan problemas comunes y vulnerabilidades relacionados con el diseño de contratos inteligentes, y según su ámbito de operación, se pueden clasificar en cinco categorías: 1) acción y control, 2) autorización, 3) ciclo de vida, 4) mantenimiento y 5) seguridad. Aunque algunos patrones pueden parecer muy básicos, su verdadero valor práctico se manifiesta cuando se utilizan como prerrequisito o en combinación con otros patrones.

III. Proceso de revisión

Para identificar y analizar los patrones de diseño aplicables al desarrollo de chaincodes en el entorno de Fabric, se siguió un proceso sistemático dividido en tres etapas principales: recolección de la literatura, filtrado y evaluación, extracción y selección de patrones.

A. *Recolección de la literatura*

Para iniciar el proceso de recolección de literatura se recurre a una estrategia de búsqueda para recopilar información relevante sobre patrones de diseño, blockchain y Hyperledger, para esto se definió una ecuación de búsqueda específica que incluyó términos clave como "blockchain", "smart contract", "patrón de diseño" y "design pattern", la cual es la siguiente : ("blockchain" OR "ethereum" OR "smart contract" OR "chaincode" OR "hyperledger") AND ("design patterns" OR "patterns" OR "blockchain patterns") AND ("Hyperledger Fabric" OR "Fabric") AND (publication year: [2018 TO]). La ecuación fue utilizada en distintas bases de datos académicas reconocidas por su calidad y cobertura en ciencias computacionales e ingenierías, las cuales fueron principalmente, IEEE Xplore, Google Scholar, Springer Link y ACM Digital Library, con lo cual se logró recuperar 53 artículos científicos que abordaban soluciones reutilizables en el ámbito del desarrollo de aplicaciones descentralizadas..

B. *Filtrado y evaluación de estudios*

Posteriormente, se aplicaron criterios de selección y exclusión para asegurar la calidad y relevancia de los documentos. Se seleccionaron aquellos estudios que ofrecían un mayor aporte a la investigación en materia de patrones de diseño o en calidad de información que pudieran aportar al desarrollo del conocimiento en los ámbitos de blockchain, patrones de diseño, contratos inteligentes, o en la propia Fabric, también se descartaron aquellos artículos que no cumplieran con un aporte directo a la investigación, dejando así una lista de 20 artículos que fueron usados como referencia para el desarrollo de esta investigación.

C. *Extracción y selección de patrones*

A partir de los artículos seleccionados, se extrajeron 8 patrones los cuales fueron sometidos a un análisis en el entorno de Fabric. La Tabla 1 resume el listado de patrones de diseño seleccionados, junto con una descripción.

Tabla I
PATRONES DE DISEÑO SELECCIONADOS PARA ANALIZAR.

Patrón de Diseño	Descripción
Limit Storage	Optimiza el uso del almacenamiento en blockchain al registrar únicamente los datos críticos y gestionar el resto de la información off-chain [9].
Contract Registry	Centraliza el registro y la gestión de contratos inteligentes, facilitando actualizaciones, auditorías y la gestión de versiones [12].
Emergency Stop	Incorpora un mecanismo para detener la ejecución del

	contrato en situaciones críticas, protegiendo el sistema de comportamientos maliciosos o erróneos [12].
Checks Effects Interactions	Define un orden de ejecución: primero se verifican condiciones, luego se aplican cambios de estado y, finalmente, se realizan interacciones, evitando vulnerabilidades como la reentrancia [12].
Escrow	Gestiona de manera automatizada la retención y liberación de fondos a través de un contrato neutral, asegurando que se cumplan las condiciones acordadas [13].
Proxy	Funciona como intermediario entre el cliente y el contrato principal, permitiendo actualizar la lógica sin modificar el contrato original, gracias a la delegación de funciones [9].
Minimize On-Chain Data	Reduce la cantidad de información almacenada directamente en la blockchain, disminuyendo costos y mejorando la eficiencia [9].
Batch Processing	El procesamiento por lotes implica agrupar múltiples transacciones en una sola unidad para su ejecución simultánea en la blockchain [14].

Posterior a definir los patrones que se analizaron, se optó por un enfoque de patrones que propusieron una mejora en el rendimiento, como lo son: Limit Storage, Minimize On-Chain Data, Proxy y Batch Processing, con el fin de someterlos a pruebas de ejecución y analizar su comportamiento en Fabric.

D. Consideraciones Adicionales

Es importante destacar que, si bien los patrones presentados en la Tabla I representan una selección cuidadosa y fundamentada basada en la literatura revisada, no constituyen la totalidad de las soluciones aplicables a Fabric. Existen numerosos patrones adicionales, identificados en otros estudios por ejemplo [15]-[16], que pueden ser igualmente relevantes según el contexto y las necesidades específicas de cada implementación. La exclusión de estos patrones en la presente revisión se debió a criterios de relevancia y utilidad en el desarrollo de chaincodes en entornos blockchain, sin desmerecer su potencial aplicación en otros escenarios o en futuras investigaciones.

IV. Caso de estudio

Se implementó un entorno de prueba en Hyperledger Fabric utilizando una máquina virtual con el sistema operativo Ubuntu, cuatro procesadores y 8 GB de RAM. La red de prueba estuvo compuesta por tres organizaciones con un nodo peer cada una, y los chaincodes se desarrollaron con el lenguaje de programación Go. A partir de esta base se realizaron pruebas para aquellos patrones que proponen mejoras en el rendimiento, en las cuales se tuvo en cuenta el tiempo de ejecución al crear activos (assets), realizando estas ejecuciones tanto para un nodo peer como para todos los nodos peers de la red. Para realizar estas pruebas se propusieron dos escenarios: para el primer escenario, se ejecutaron los chaincodes sin la aplicación de patrones, y el segundo, en el cual se ejecutan los chaincodes con la aplicación de los patrones de diseño seleccionados. Después de realizar cada prueba en un escenario se eliminaba la red, con el fin de que no quedaran rastros que puedan perturbaran los resultados entre pruebas.

V. Resultados y análisis

Los patrones relacionados en la Tabla I, fueron revisados y analizados, según su posible implementación Fabric. Cabe destacar que no todos los patrones fueron sometidos a pruebas de rendimiento, ya que algunos están orientados a mejorar aspectos como la seguridad, escalabilidad y mantenibilidad, como lo fueron: Checks Effects Interactions, Contract Registry, Emergency Stop y Escrow.

A. Checks Effects Interactions (CEI)

El patrón CEI estructura la ejecución del código en tres fases: primero se realizan las verificaciones (checks), luego se actualiza el estado (effects) y finalmente se ejecutan las interacciones externas (interactions) [17]. En el contexto de Fabric, la aplicación de este patrón facilita el mantenimiento del código y mejora la seguridad del chaincode al minimizar la posibilidad de ataques de reentrancia la cual es una vulnerabilidad que implica ejecutar o re ejecutar un código en medio de la ejecución original de una subrutina o del programa. Al implementar este patrón no se espera obtener mejoras en términos de tiempo de ejecución, ya que su principal aporte reside en dar claridad y robustez al diseño, aspectos críticos para entornos empresariales.

B. Contract Registry

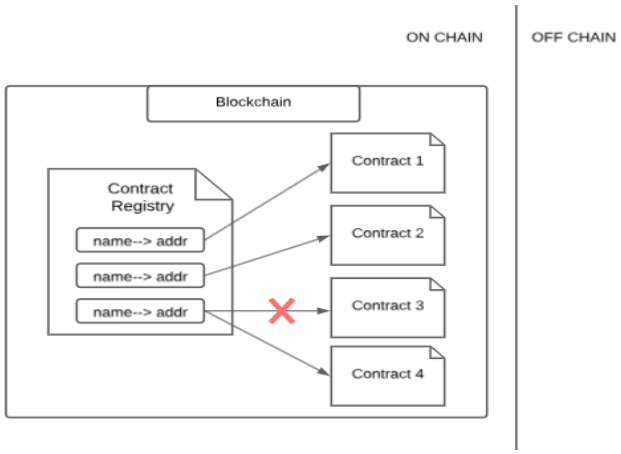

Fig. 1. Representación del funcionamiento de contract Registry.

Este patrón se utiliza para centralizar el registro y la gestión de chaincodes, permitiendo auditar, actualizar y administrar versiones de forma centralizada, facilitando así el mantenimiento de un proyecto desplegado en una blockchain, esto se logra creando un chaincode el cual se encarga de

mantener un registro con el nombre y la dirección de la última versión de cada chaincode [12]. En Fabric, donde la trazabilidad y el control de acceso son fundamentales, el Contract Registry facilita el mantenimiento y escalabilidad de sistemas que involucren múltiples chaincodes, una forma fácil de entenderlo es con la Fig 1, que representa el funcionamiento de este patrón.

C. *Emergency Stop*

El patrón Emergency Stop introduce un mecanismo que permite detener la ejecución del chaincode en situaciones críticas. Aunque su utilidad es ampliamente reconocida en blockchains públicas (donde la falta de control sobre los participantes puede derivar en ejecuciones maliciosas) [17], en el caso de Fabric, la necesidad de este patrón se reduce considerablemente debido a la naturaleza de los actores confiables de la red, y aunque este patrón se puede aplicar fácilmente su funcionalidad tiene menos relevancia.

D. *Escrow*

El patrón Escrow está diseñado para abordar la falta de confianza entre partes involucradas en una transacción. Consiste en utilizar un chaincode como un tercero neutral para retener fondos o activos hasta que se cumplan las condiciones predefinidas de la transacción. Este patrón resuelve el problema de confianza al permitir que un chaincode gestione los fondos de forma automática, eliminando la necesidad de un intermediario tradicional [13].

Si bien el patrón Escrow es ampliamente útil en blockchains públicas como Ethereum, su aplicación en Fabric es menos relevante debido a las siguientes razones: Fabric es una red permisionada por lo tanto hay cierto nivel de confianza preexistente o acuerdos regulatorios, Fabric tampoco cuenta con tokens nativos por lo tanto esto implica que se tiene que implementar un mecanismo de tokens los cual implica un costo mayor en el desarrollo.

E. *Limit Storage*

El patrón Limit Storage se centra en optimizar el uso del almacenamiento en la blockchain, almacenando únicamente los datos críticos en el ledger y manteniendo el resto de la información de forma temporal en memoria [9].

Para evaluar la optimización proporcionada por el patrón Limit Storage, se realizaron una serie de pruebas experimentales. Se desarrolló un chaincode que contiene dos funciones de creación de activos. En la función, se manejan tres variables de tipo String (color, id y propietario) y dos de tipo entero (valor y tamaño), simulando así un producto físico en un almacén. Una de las funciones implementa el patrón Limit Storage, mientras que la otra lo omite. Adicionalmente, se incluyeron dos funciones auxiliares: una que invoca la función de creación de activos cien veces y otra que la ejecuta mil veces, con el objetivo de simular transacciones en lote. Durante las pruebas, se despliega la red de Fabric, se instala el chaincode y se ejecuta la función de creación de cien activos en cuatro iteraciones, registrando el tiempo de ejecución en cada caso; este procedimiento se repite para la función que realiza mil ejecuciones. Las pruebas se llevan a cabo en dos configuraciones: una en la que la transacción se ejecuta en todos los nodos peers de la red y otra en la que se ejecuta únicamente en un solo nodo peer. Los datos recolectados se pueden ver reflejados en la Fig 2.

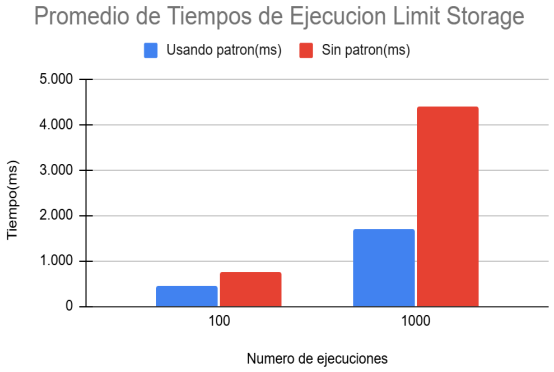

Fig. 2. Promedio de tiempos de ejecución usando y sin usar el patrón Limit Storage.

TABLA II
PORCENTAJE DE MEJORA DE LAS EJECUCIONES AL USAR EL PATRÓN LIMIT STORAGE.

Configuración	100 Activos	1000 Activos
Todas las organizaciones	13.1%	58.7%
Una organización	51.9%	63.1%

Como muestra la Tabla II, el patrón Limit Storage en las pruebas evidencia una mejora de más del 50% de los tiempo de ejecución, aunque esta mejora puede variar dependiendo del caso en el cual sea aplicado, sin embargo los resultados confirman que la aplicación del patrón Limit Storage es una estrategia eficaz para optimizar el rendimiento de los chaincodes, y se recomienda considerarlo en proyectos donde la eficiencia en el procesamiento de transacciones y la reducción de la sobrecarga sean aspectos críticos.

F. *Proxy*

El patrón Proxy actúa como un intermediario entre el cliente y el chaincode real, delegando las operaciones y añadiendo funcionalidades, como caché, validación previa, control de acceso y registro de auditoría. Para evaluar el impacto del patrón Proxy en Fabric, se diseñó una prueba comparativa entre dos implementaciones de chaincodes: una que utiliza el patrón Proxy y otra que lo omite. En ambas implementaciones se incluyeron las funciones CreateAsset y ReadAsset, por otro lado, se creó un archivo ejecutable el cual se encarga de hacer una serie de pruebas, las cuales consisten en ejecutar cien veces la función CreateAsset y ejecutar mil veces esta misma función sumando los tiempos de cada ejecución, por otro lado, se ejecuta la función ReadAsset tres veces con datos que existen y tres inexistentes, estas pruebas

fueron ejecutadas tal y como se explica en el caso de estudio, el resultado de la ejecución de una prueba se puede evidenciar en la Fig 3.

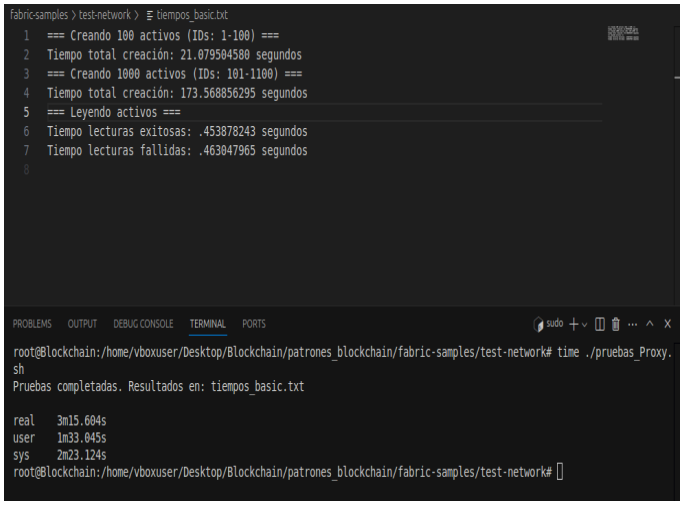

Fig. 3. Resultado de la ejecución de las pruebas de proxy.

TABLA III
PROMEDIO DE TIEMPOS DE EJECUCIÓN PROXY.

Configuración	Creación 100 activos (s)	Creación 1000 activos (s)	Lecturas exitosas (s)	Lecturas fallidas (s)
Usando Proxy – Todas las organizaciones	19,942	183,113	0,465	0,434
Usando Proxy – Solo una organización	14,814	142,796	0,417	0,391
Sin Usar Proxy – Todas las organizaciones	17,901	158,198	0,426	0,395
Sin Usar Proxy – Solo una organización	14,299	134,312	0,411	0,406

En los resultados que se pueden evidenciar en la Tabla III, no se puede observar ninguna mejora en el tiempo de ejecución usando el patrón Proxy, dicha situación se presenta por el propio funcionamiento de Fabric, ya que, en la ejecución de un chaincode, el cliente prepara una propuesta de transacción, que incluye la operación que quiere ejecutar (por ejemplo, actualizar un dato) y la envía a uno o varios "endorsers" (nodos autorizados). Cada endorser toma la propuesta y, de forma independiente, la simula ejecutando el chaincode en un contenedor Docker aislado. Durante la simulación, el chaincode no mantiene ningún estado en memoria de manera persistente entre invocaciones, ya que el proceso se ejecuta en un entorno efímero (contenedor Docker) y cualquier caché temporal se descarta al finalizar la ejecución. Esto garantiza que, aunque se puedan aplicar técnicas de cacheo durante la simulación, sus beneficios no se trasladan a las invocaciones posteriores [18]. Situación evidenciada y explicada por Androulaki [19], la Fig 4 representa el procesamiento dado.

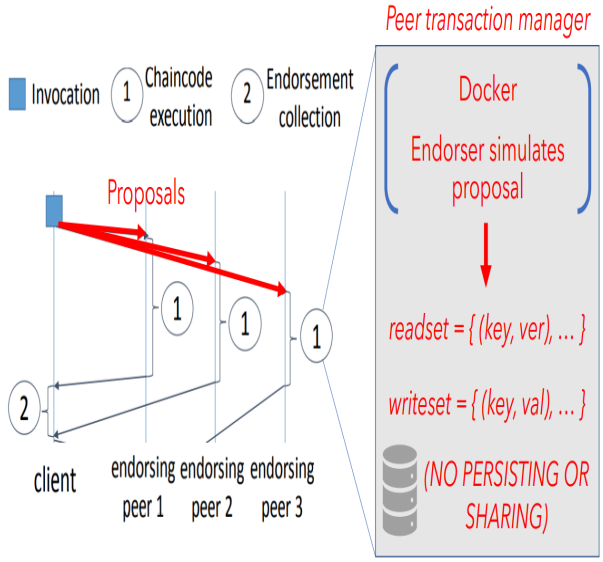

Fig 4. Fase de ejecución de una transacción [19]

Según lo expresado por Androulaki, cada invocación de chaincode se ejecuta en un contenedor efímero que reinicia su estado en cada transacción. Esto implica que, aunque se implemente una caché en memoria mediante el patrón Proxy, dichos datos se pierden al finalizar la transacción y no pueden ser reutilizados en invocaciones posteriores. Como resultado, la caché no logra reducir la latencia de lectura, ya que su información no se mantiene de manera persistente entre transacciones, limitando de este modo su efectividad en la optimización del rendimiento, sin embargo, esto no significa que el patrón proxy no tenga ninguna utilidad ya que cuenta con otro tipo de funcionalidades como lo es la validación previa, el control de acceso y el registro de auditoría.

G. Minimize On-Chain Data

El patrón Minimize On-Chain Data busca optimizar el uso del almacenamiento en blockchain al reducir la cantidad de datos guardados directamente en la cadena [9]. Esto es especialmente importante porque el almacenamiento en la cadena de bloques es significativamente más costoso que la memoria, lo que debería traducirse en una mejora del rendimiento.

Para evaluar el comportamiento en Fabric de este patrón, se crearon dos chaincodes con las mismas características, con la diferencia en que uno de ellos implementa el patrón y en el otro no. Partiendo de esto se efectuaron pruebas para observar las diferencias en dichos chaincodes los cuales fueron probados utilizando un archivo jpg de 181 kb. Para el caso del chaincode que utiliza el patrón, se usó la ruta de la imagen y con esta se crea un activo tal y como lo propone Minimize On Chain Data, para el caso que no aplica el patrón se realizó de la siguiente forma, se convirtió la imagen jpg a base64 con el fin de representar la imagen en texto, de esta forma se puede usar el texto para crear un activo, el resultado ejecutar un

comando con este texto fue un error “Argument list too long” el cual podemos ver en la Fig 5.

Fig. 5. Error: Argument list too long

El error "argument list too long" es un mensaje que se genera cuando se supera el límite máximo permitido para los argumentos de un comando (ARG_MAX), el cual define la longitud máxima que pueden tener los argumentos pasados a la función exec [20]. Esto indica que se han suministrado demasiados argumentos en una sola operación como lo es el texto generado al pasar la imagen a base 64, pero en Fabric el límite preestablecido es de 10 MB para el tamaño máximo de una transacción y es definido por una variable llamada máximos absolutos de bytes la cual puede ser modificada [21], por lo tanto, sabemos que la red puede soportar transacciones de un mayor tamaño, pero es limitada por el sistema operativo.

Con el objetivo de probar la eficacia de este patrón, se recurrió a usar una REST API desarrollada para el proyecto en el que se cimento la presente investigación. Utilizando la herramienta Postman se realizaron las siguientes pruebas: se crearon 10 activos que contienen la cadena generada al convertir la misma imagen de 181 KB a Base64 y se crearon 10 activos que contienen un hash simulando el uso del patrón Minimize On Chain Data. Posteriormente, se realizaron 10 pruebas de lectura a los bloques creados con la cadena completa y 10 a los activos con hash. Los resultados obtenidos se pueden ver en la Tabla IV.

TABLA IV
PORCENTAJE DE MEJORA DE LAS EJECUCIONES AL USAR EL PATRÓN MINIMIZE ON CHAIN DATA

Operación	Promedio sin el patrón	Promedio con el patron	% Mejora
Escritura	3.282 s	2.418 s	26.3%
Lectura	0.1148 s	0.0324 s	71.7%

Como se muestra en la Tabla IV, el patrón Minimize On-Chain Data proporciona una mejora del 26% en términos de escritura y del 71% en lectura. Estos resultados evidencian que, usando este patrón se evita no solo el error de sobrecarga, sino que se optimiza significativamente el rendimiento en la creación y consulta de activos. Este enfoque se presenta como una estrategia eficaz para optimizar programas basados en Fabric, permitiendo un uso más eficiente de los recursos y mejorando la escalabilidad de la red en escenarios de alta demanda.

H. *Batch Processing*

El procesamiento por lotes implica colocar múltiples transacciones en una sola unidad para su ejecución simultánea en la cadena de bloques. Esta metodología difiere de los métodos anteriores, en los que cada transacción se procesa de forma independiente. Al agrupar las transacciones, se puede aumentar el rendimiento, reducir la latencia y optimizar el consumo de recursos, mejorando así la eficiencia de la red [14].

Con el fin de comprobar el impacto que tiene este patrón, se desarrolló un script, el cual se encargó de crear cien activos cinco veces y posteriormente de crear mil activos tres veces para cada uno de los casos detallados en el caso de estudio, con la finalidad de medir la velocidad de creación de activos usando y sin usar el patrón, los promedios de resultados obtenidos en estas pruebas se pueden visualizar en la Tabla V.

TABLA V
PROMEDIO DE TIEMPOS DE EJECUCIÓN BATCH PROCESSING.

Configuración	100 Activos (s)	1000 Activos (s)
Usando Batch Processing en Todas las organizaciones.	0,304	0,993
Usando Batch Processing en una organización	0,299	1,1796
Sin Batch Processing en Todas las organizaciones.	20,051	200,728
Sin Batch Processing en una organización	17,215	158,366

TABLA VI
PORCENTAJE DE MEJORA DE LAS EJECUCIONES AL USAR EL PATRÓN BATCH PROCESSING

Condición	100 Activos	1000 Activos
Todas las organizaciones	98,49%	99,50%
Una organización	98,26%	99,25%

En el experimento se evaluó el impacto de Batch Processing en la creación de activos en Fabric, comparándolo con el procesamiento individual. Los datos presentados en la Tabla VI evidencian una notable mejora en los tiempos de ejecución tanto al crear 100 como 1000 activos, bajo condiciones de despliegue en una única organización o en todas las organizaciones. Esta mejora se debe a que, al utilizar el procesamiento por lotes, se evita ejecutar individualmente todos los procesos asociados a cada transacción; en cambio, se realiza una única transacción que abarca la totalidad de los activos, lo que reduce drásticamente el tiempo total de procesamiento.

V. Conclusiones

La presente investigación permitió la adaptación e implementación de patrones de diseño en el desarrollo de contratos inteligentes sobre una blockchain de tipo permisionada (Hyperledger Fabric), lo que resulta en una estrategia eficaz para abordar los desafíos inherentes a la tecnología blockchain,.

Los resultados experimentales evidencian que patrones como Limit Storage y Batch Processing permiten optimizar significativamente el rendimiento de las transacciones, reduciendo los tiempos de ejecución en más del 50% y hasta un 98% en algunos escenarios, sin embargo, la optimización no es el único beneficio que tienen los patrones de diseño ya que también trae consigo mejoras en la complejidad, la seguridad, la escalabilidad y la mantenibilidad.

Esto no significa que todos los patrones de diseño sean efectivos ya que depende en gran medida del contexto de la blockchain en que se aplique tal y como se pudo notar con el patrón Escrow el cual va en contra de la propia naturaleza de Fabric al ser una plataforma permisionada que no emplea tokens nativos, o con el patrón proxy el cual uno de sus atractivos principales es la capacidad de cachear información para mejorar el rendimiento, pero este no muestra los mismos beneficios en el entorno de Hyperledger debido a la naturaleza efímera de sus contenedores.

En conjunto, este trabajo ofrece un marco metodológico robusto que adapta las mejores prácticas de la Ingeniería de Software a las particularidades de la tecnología blockchain. Se demuestra que la reutilización y estructuración de soluciones probadas no solo mejora la ejecución de transacciones, sino que también facilita el mantenimiento y la evolución de las aplicaciones descentralizadas. Como línea futura, se sugiere ampliar el análisis a otros patrones de diseño y evaluar su aplicabilidad en escenarios de producción, explorando la integración de múltiples estrategias que potencien aún más las capacidades de las plataformas blockchain.

Agradecimientos

Este trabajo fue financiado por la Universidad de los Llanos bajo el proyecto C03-F02-012-2022. Los autores expresan sus agradecimientos a la Universidad de los Llanos, a la Dirección General de Investigaciones y a los miembros del grupo de investigación GITECX por su colaboración, orientación y apoyo logístico, lo cual hizo posible esta investigación.

Referencias

[1] M. Belotti, N. Božić, G. Pujolle, and S. Secci, "A Vademecum on Blockchain Technologies: When, Which, and How," *IEEE Communications Surveys and Tutorials*, vol. 21, no. 4, pp. 3796–3838, Oct. 2019, doi: 10.1109/COMST.2019.2928178.

[2] N. Six, N. Herbaut, and C. Salinesi, "Blockchain software patterns for the design of decentralized applications: A systematic literature review," *Blockchain: Research and Applications*, vol. 3, no. 2. Elsevier Ltd, Jun. 01, 2022. doi: 10.1016/j.bcra.2022.100061.

[3] S. Porru, A. Pinna, M. Marchesi, and R. Tonelli, "Blockchain-oriented software engineering: Challenges and new directions," in Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering Companion, ICSE-C 2017, Institute of Electrical and Electronics Engineers Inc., Jun. 2017, pp. 169–171. doi: 10.1109/ICSE-C.2017.142.

[4] Carleton Anita *et al.*, *Architecting the Future of Software Engineering: A National Agenda for Software Engineering Research & Development*, vol. 1. Carnegie Mellon University, 2021.

[5] G. Destefanis, M. Marchesi, M. Ortu, R. Tonelli, A. Bracciali, and R. Hierons, "Contratos inteligentes vulnerabilities: a call for blockchain software engineering?," in *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, 2018, pp. 19–25. doi: 10.1109/IWBOSE.2018.8327567.

[6] Demi, S., Colomo-Palacios, R., & Sánchez-Gordón, M. (2021). Software Engineering Applications Enabled by Blockchain Technology: A Systematic Mapping Study. Applied Sciences, 11(7), 2960. https://doi.org/10.3390/app11072960

[7] IBM, "Qué es el Hyperledger Fabric," IBM, [Online]. Available: https://www.ibm.com/es-es/topics/hyperledger, Accessed: Feb. 21, 2025.

[8] N. Kannengieser, S. Lins, C. Sander, K. Winter, H. Frey, and A. Sunyaev, "Challenges and Common Solutions in Smart Contract Development," *IEEE Transactions on Software Engineering*, vol. 48, no. 11, pp. 4291–4318, Nov. 2022, doi: 10.1109/TSE.2021.3116808.

[9] L. Marchesi, M. Marchesi, G. Destefanis, G. Barabino, and D. Tigano, "Design Patterns for Gas Optimization in Ethereum," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, 2021.

[10] Oscar Javier Blancarte Iturralde, "Introducción a los Patrones de Diseño un Enfoque Práctico," 2016, pp. 11-12.

[11] M. Wohrer and U. Zdun, "From Domain-Specific Language to Code: Contratos inteligentes and the Application of Design Patterns," *IEEE Software*, vol. 37, no. 5. IEEE Computer Society, pp. 37–42, Sep. 01, 2020. doi: 10.1109/MS.2020.2993470.

[12] V. Rajasekar, S. Sondhi, S. Saad, and S. Mohammed, "Emerging design patterns for blockchain applications," in *ICSOFT 2020 - Proceedings of the 15th International Conference on Software Technologies*, SciTePress, 2020, pp. 242–249. doi: 10.5220/0009892702420249.

[13] Q. Lu, X. Xu, H. M. N. D. Bandara, S. Chen, and L. Zhu, "Patterns for Blockchain-Based Payment Applications," Feb. 2021, [Online]. Available: http://arxiv.org/abs/2102.09810

[14] Dutt Rajput Punar and Rinki, *Advanced Solana: Mastering Development and Optimizaton*. C# Corner, 2024, pp. 47-53.

[15] X. Yang, "Blockchain-Based Supply Chain Finance Design Pattern," in *Proceedings - 2021 13th International Conference on Intelligent Human-Machine Systems and Cybernetics, IHMSC 2021*, Institute of Electrical and Electronics Engineers Inc., Aug. 2021, pp. 200–203. doi: 10.1109/IHMSC52134.2021.00053.

[16] H. D. Bandara, X. Xu, and I. Weber, "Patterns for Blockchain Data Migration," in *ACM International Conference Proceeding Series*, Association for Computing Machinery, Jul. 2020. doi: 10.1145/3424771.3424796.

[17] M. Wohrer and U. Zdun, "Contratos inteligentes: security patterns in the ethereum ecosystem and solidity," in *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, 2018, pp. 2–8. doi: 10.1109/IWBOSE.2018.8327565.

[18] E. Androulaki *et al.*, "Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains," in *Proceedings of the 13th EuroSys Conference, EuroSys 2018*, Association for Computing Machinery, Inc, Apr. 2018. doi: 10.1145/3190508.3190538.

[19] "Contract Registry," [Online]. Disponible: https://research.csiro.au/blockchainpatterns/general-patterns/contract-structural-patterns/contract-registry/ [Accedido: 23 Feb. 2025].

[20] "bash: /usr/bin/rm: Argument list too long – Solution" [Online]. https://linuxblog-io.translate.goog/bash-usr-bin-rm-argument-list-too-long-solution/?_x_tr_sl=en&_x_tr_tl=es&_x_tr_hl=es&_x_tr_pto=wa [Accedido: 16 Mar. 2025].

[21] "Checklist for a production ordering node" [Online]. Disponible:https://hyperledger-fabric.readthedocs.io/en/release-2.5/deployorderer/ordererchecklist.html#. [Accedido: 23 Feb. 2025].