Journal of Computing and Electronic Information Management
ISSN: 2413-1660 | Vol. 18, No. 1, 2025

Fine-grained control strategy for high-concurrency transactions in the consortium chain Hyperledger Fabric

Ruicheng Guo, Junjie Wu, Penghui Cheng

School of Taiyuan Normal University, Jinzhong 030619, China

Abstract: Hyperledger Fabric is a consortium chain framework that is widely used at home and abroad. When conducting business with Fabric technology, high-concurrency transaction conflicts are often encountered. To optimize the high concurrent transaction processing performance of Hyperledger Fabric, this paper proposes a targeted fine-grained control strategy, relying on the Dynamic Priority Reordering - Chain Code Cache mechanism (DPRC-Cache) framework and attribute-based Access Control (ABAC). Experiments show that fine-grained resource access control can effectively reduce the transaction conflict rate. Meanwhile, the optimized transaction sorting mechanism plays a positive role in improving the overall throughput of the system.

Keywords: High concurrency transaction processing; Fine-grained control; Reordering - chain code caching; Attribute-based access control.

1. Introduction

Since blockchain technology was proposed by Satoshi Nakamoto in 2008, it has undergone significant development and evolution. As a distributed ledger, blockchain integrates technologies such as P2P networks, cryptography, distributed consensus protocols, smart contracts, and databases, endowing it with features such as decentralization, tamper-proofing, traceability, shareability, trusted verification, and privacy protection [1].

Hyperledger Fabric, developed by IBM, is a modular blockchain platform specifically designed for enterprise-level applications [2] It offers a flexible and modular architecture for handling transactions. The transaction processing flow typically includes: Proposal stage: The client sends a transaction proposal to a peer node in the network. Endorsement stage: The proposal is sent to the designated endorsement node for verification and signature. Submission phase: The transactions that have been endorsed are sent to the order service node for sorting and are ultimately packaged into blocks and distributed to all peer nodes for verification and submission. That is, by adopting the "Execution-OrderValidate" (EOV) three-stage architecture [3], smart contracts and consensus trust are separated, which enhances the flexibility of contract trust and the scalability of the system. However, it also exacerbates the problem of high concurrent transaction conflicts [4]. Therefore, how to optimize the performance of high-concurrency transaction processing in Fabric has become a key research issue at home and abroad at present.

For this issue, many studies have proposed some solutions to concurrent conflicts in Fabric. Xu [5] et al. designed the LMLS algorithm based on lock mechanism and ledger storage, which to some extent improved the problem of high concurrency conflicts. However, it had an impact on the historical information of the state database and did not conform to the application scenarios of actual business. Sharma [6] et al. proposed Fabric++, which addresses intra-block conflict issues in high-concurrency scenarios through cyclic detection of transaction conflicts and early termination of transactions. However, it is unable to handle concurrent conflict transactions initiated by multiple clients. FabricSharp [7] proposed a transaction reordering scheme to detect loop conflicts in the form of a dependency graph. However, the time complexity of its detection algorithm is too high to cope with application scenarios when the node scale is large.

This paper proposes a fine and efficient fine-grained transaction control strategy in combination with aspects such as resource access control and transaction sorting optimization, aiming to improve system performance and make the consortium chain Hyperledger Fabric better meet the actual production needs.

2. Dynamic Priority Reordering - Chain Code Cache Framework (DPRC-Cache)

For the high-concurrency transaction scenarios of Fabric, the Dynamic Priority Reordering - Chain Code Caching Framework (DPRC-Cache) framework collaboratively optimizes through a dual mechanism of dynamic priority adjustment and chain code instance caching [8], addressing the performance bottleneck caused by static priority allocation and frequent chain code instantiation in traditional caching frameworks under high concurrency. Its core goals include:

(1) Reduce transaction execution latency: Decrease the waiting time of high-priority transactions through priority reordering;

(2) Enhance system throughput: Reduce instantiation overhead through chain-code caching to support processing more transactions per second;

(3) Enhance fine-grained control: By integrating ABAC access control, achieve dynamic management of transaction-level permissions.

This solution takes the Dynamic Priority Reordering - Chain Code Cache framework (DPRC-Cache) as its core, integrates attribute-based access control (ABAC), and utilizes Prometheus and Grafana for monitoring and feedback to form a closed-loop feedback mechanism, achieving fine-grained

control over high-concurrency transactions in Fabric.

3. Design of Fine-grained Control Strategy Scheme for High-Concurrency Transactions in Consortium Chain Fabric

The fine-grained control strategy based on the Dynamic Priority Reordering - Chain Code Cache framework (DPRC-Cache) generally consists of four links: input layer, task classification and scheduling, cache security, and execution and feedback. The flowchart is shown in Figure 1. Each link has a different function in this strategy, as follows.

(1) Input Layer: The input layer is the starting point of the entire fine-grained control strategy and is mainly responsible for receiving various task requests and related attribute information. While receiving a task, the input layer extracts various attribute information related to the task. These attributes include the urgency of the task, the type of business it belongs to, the sensitivity of the data, etc.

(2) Task Classification and Scheduling: Task classification and scheduling is based on the task attribute information provided by the input layer. After classifying the tasks, a dynamic priority algorithm is used. Combined with the classification results of the tasks and the real-time status of the system (such as resource occupation, cache usage, etc.), the corresponding priority is assigned to each task, and the execution sequence of the tasks is determined.

(3) Cache Security: The cache security aspect aims to ensure the security, integrity and availability of the chain code cache data. Based on ABAC verification, access to cached data is strictly authorized according to the attributes of the task and the user's permission information. Only tasks that meet specific attribute conditions and permission requirements can access the corresponding cache data.

(4) Execution and Feedback: The execution and feedback phase are responsible for the actual execution of tasks and feeding back the execution results to the system for subsequent optimization and adjustment. During the execution process determined by the task scheduling sequence, the system will monitor the execution status of the tasks and feed back the execution results to the system. If any abnormal situations occur during the task execution process, such as incorrect chain code execution or abnormal data access, the system will promptly capture the abnormal information and handle it accordingly.

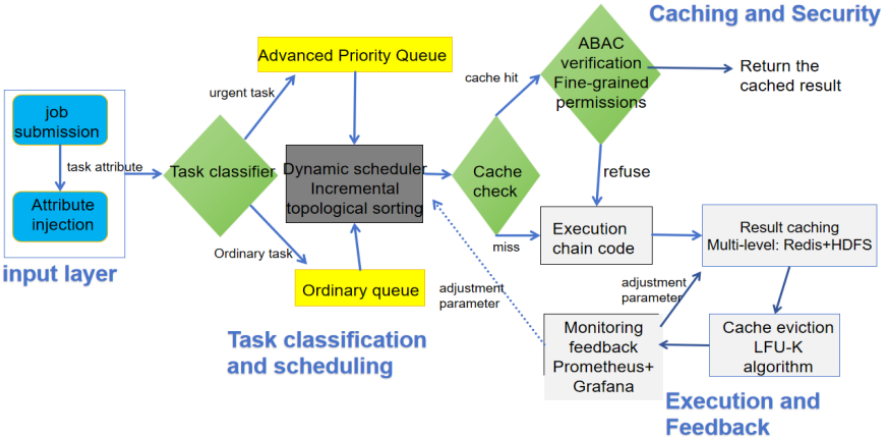

Figure 1. Flowchart of the DPRC-Cache framework

3.1. Dynamic Priority Reordering Engine (DPR-Engine)

Pr-engine assigns initial priority values to transactions based on their type (query transaction or update transaction) and resource sensitivity (such as transactions involving high-value assets).

In order to enable the priority to be dynamically adjusted according to the real-time status of the system, DPR-Engine adopts the improved SLAD algorithm (dual-Principle Scheduling Framework). The priority calculation formula is as follows:

$$P = v_{deadline} \cdot \frac{t_p}{p_d} + \alpha \cdot ConflictRisk(T_i) \quad (1)$$

Among them, is the deadline weight, which reflects the degree to which the transaction deadline affects the priority. It is the remaining time slice, indicating how much time the transaction has left until the deadline. It is the duration of the cycle, used as a benchmark for measuring time. It is the conflict risk coefficient, which is calculated based on the historical conflict frequency [9].

By comprehensively considering the deadline factor and the conflict risk factor, this algorithm can more accurately and dynamically adjust the priority of transactions.

3.2. Chain Code Cache Acceleration Module (Cache-Module)

In a blockchain network, the execution efficiency of the chain code is crucial to the performance of the entire system. To improve the execution speed of the chain code, Cache-Module adopts the hot spot chain code preloading strategy.

The system will continuously count the access frequency of each chain code. Let the set of chain codes be, and the number of accesses to the chain codes within the time interval be. The access frequency of the chain code can be obtained by calculating the number of accesses per unit time.

Set an access frequency threshold θ. When >θ, the chain code is considered a high-frequency chain code. The system will cache these high-frequency chain codes in the memory of the endorsement node (Peer). This can prevent the chain code from being loaded from the disk or other storage media each time it is executed, significantly reducing the startup time of the chain code execution.

Due to the limited memory space, the chain code cannot be cached without limit. Therefore, Cache-Module adopts the LFU-K (Least Frequently Used) elimination mechanism to regularly clean up low-frequency chain codes and free up Cache space.

The system maintains an access counter for each cache chain code. Let the chain code cache set be, and the corresponding access counter set be, where represents the number of accesses to the chain code within a certain time window.

When the cache space is insufficient and the chain code needs to be evicted, first sort the cache chain code in ascending order of the value of the access counter. Then select the K chain codes with the fewest visits for elimination.

3.3. ABAC Access Control Module (ABAC-Module)

Under the DPRC-Cache framework, the attribute management of the ABAC access control module provides basic information for the secure access of the entire system.

(1) Subject attribute: The subject is the entity that initiates the access request. The Role determines the responsibilities and permission scope of the subject in the system. For example, the "Admin" role may have write operation permissions for sensitive resources [10]. The Department attribute helps to achieve access control based on organizational structure. The device ID (DeviceID) can be used to restrict certain operations to be performed only on specific devices, enhancing security and preventing unauthorized device access.

Let the subject set be, and each subject has corresponding role, department, and device ID attributes.

(2) Resource attributes: Resources are the objects accessed in the system. Resource types (AssetType) distinguish different kinds of resources, such as documents, databases, etc. Sensitivity measures the importance and confidentiality of resources. Highly sensitive resources require stricter access control. The "Owner" attribute clearly defines the ownership of resources, and resource owners usually have relatively high access rights.

Let the resource set be, and each resource has a resource type, sensitivity, and owner attribute.

In the ABAC (Attribute-Based Access Control) access control model, policy definition is the core link, which precisely stipulates the access rights of subjects to resources under different combinations of attributes [11].

When an access request is received, the system needs to make a logical judgment based on the policies in the policy set to determine whether to allow the access. For each policy, the system will successively check whether the subject attributes in the request meet the SubjectAttrs condition in the policy, whether the resource attributes meet the ResourceAttrs condition, and whether the operation type matches the Action field and determine the final control result based on whether the environmental conditions in the "Condition" field are met.

3.4. Experimental Design and Results

Experimental objective: To evaluate the improvement effect of the overall DPRC-Cache framework on the network performance of Hyperledger Fabric. Specifically, quantitative analysis is conducted from aspects such as transaction processing time, throughput, transaction abort rate, and resource utilization rate to verify the effectiveness of the collaborative work of each module of the framework.

Table 1. Experimental Environment Configuration

Parameter	Value
CPU	Intel Xeon Platinum 8280
RAM	512G
System	Unbutu 20.04LTS
Network	10 Gbit/s
Working load	Caliper
Docker version	20.10.06
Status database	DouchDB

Control group: Using the traditional Hyperledger Fabric network without the integrated DPRC-Cache framework.

Experimental group: Hyperledger Fabric network integrated with DPRC-Cache framework.

Scene 1: Low-load scenario

Simulate 100 concurrent clients, with each client sending one transaction request per second, and run continuously for one hour. Transaction types include 60% query transactions and 40% update transactions.

Scene 2: Medium load scenario

Simulate 300 concurrent clients, each sending 2 transaction requests per second, and run continuously for 1 hour. The proportion of transaction types is the same as that in low-load scenarios.

Scene 3: High-load scenario

Simulate 500 concurrent clients, each sending 3 transaction requests per second, and run continuously for 1 hour. The proportion of transaction types remains unchanged.

The experimental results are as follows:

Table 2. Average Transaction Processing Time (ms) in Different Scenarios

Experimental scene	control group	experimental group	Increase the proportion
Low-load scenario	120	95	20.8%
Medium-load scenario	250	180	28%
High-load scenario	450	300	33.3%

Table 3. Throughput (TPS) in Different Scenarios

Experimental scene	control group	experimental group	Increase the proportion
Low-load scenario	800	950	18.7%
Medium-load scenario	1200		37.5%
High-load scenario	1100	2000	81.8%

Table 4. Transaction Abort Rates in Different Scenarios

Experimental scene	control group	experimental group	Increase the proportion
Low-load scenario	2	1	50%
Medium-load scenario	5	2	60%
High-load scenario	12	4	66.7%

4. Experimental Results and Summary

Through this experiment, it was verified that the overall framework of DPRC-Cache has significantly improved the

network performance of Hyperledger Fabric. This framework effectively reduces transaction processing time, increases throughput, lowers transaction abort rate, and optimizes resource utilization through the collaborative work of the dynamic priority reordering engine, chain code cache acceleration module, and ABAC access control module.

In practical applications, the DPRC-Cache framework can provide a more efficient and secure operating environment for blockchain systems, and it also fully demonstrates that the solution based on the DPRC-Cache framework is an effective way to enhance the fine-grained handling of high-concurrency transaction conflicts in the Fabric network.

References

[1] Yuan Yong, Wang Feiyue Blocks chain technology development present situation and prospect of [J]. Journal of automation, 2016, and (4): 481-494. The DOI: 10.16383 / j.a as. 2016. C160158.

[2] Androulaki E, Barger A, Bortnikov V, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains [C]//Proceedings of the thirteenth EuroSys conference. 2018: 1-15.

[3] Androulaki E, Barger A, Bortnikov V, et al. Hyperledger Fabric:A distributed operating system for permissioned blockchains [C/OL]//Proc of the 13th EuroSys Conf (EuroSys). New York: ACM, 2018[2023-09- 09].

[4] Jiang Lili, Chang Xiaolin, Liu Yuhang, et al. Performance analysis of Hyperledger Fabric platform: A hierarchical model approach [J]. Peer-toPeer Networking and Applications, 2020, 13(3): 1014-1025

[5] Xu Xiaoqiong, Sun Gang, Luo Long, et al. Latency performance modeling and analysis for hyperledger Fabric blockchain network [J]. Information Processing & Management, 2021, 58(1): 102436-102437

[6] Sharma A, Schuhknecht F M, Agrawal D, et al. Blurring the lines between blockchains and database systems: the case of hyperledger Fabric [C]//Proc of the 37th Int Conf on Management of Data (SIGMOD). New York: ACM, 2019: 105-122

[7] Ruan P, Loghin D, Ta Q T, et al. A transactional perspective on executeorder-validate blockchains [C]//Proc of the 38th Int Conf on Management of Data(SIGMOD). NewYork: ACM, 2020: 543-557

[8] Ren Jianfeng, Hao Wanru, Liu Qing. Multi-stage supply chain network node wisdom dynamic prioritization system [J]. Journal of electronic design engineering, 2024, 32 (14): 64-67 + 72. DOI: 10.14022 / j.i ssn1674-6236.2024.14.013.

[9] Chen Yao, Chen Liquan, Wu Hao. A Dynamic Secure Searchable Encryption Scheme Supporting Priority Sorting [J]. Cyberspace Security, 2020, 11(08): 51-55+80.

[10] Pang Yuxiang, Chen Zemao. Unmanned Aerial Vehicle Flight Control Security Scheme Based on Attribute Access Control Policy [J]. Computer Science, 2024, 51(04):366-372.

[11] Dong Xiguo, Xu Ziheng, Zhu Yan. ABAC Access Policy Index Optimization Scheme for Blockchain [J]. Radio Engineering, 2025, 55(06):1318-1326.