2021 IEEE 6th International Conference on Intelligent Computing and Signal Processing (ICSP 2021)

Blockchain storage middleware based on external database

1st Zhaoyi Zhang
Guilin University of Electronic Technology
Guilin, China
576915249@qq.com

2nd Yanru Zhong
Guilin University of Electronic Technology
Guilin, China
1467536235@qq.com

3rd Xiaofan Yu
Guilin University of Electronic Technology
Guilin, China
827349539@qq.com

2021 6th International Conference on Intelligent Computing and Signal Processing (ICSP) | 978-1-6654-0413-6/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICSP51882.2021.9408752

***Abstract*—Because of the anti-tampering, decentralization, anonymous transaction, and other characteristics of the blockchain, it is considered to be a technology that can bring huge changes to the Internet field, especially the financial Internet. As a huge distributed ledger, the blockchain has obvious deficiencies in query processing module design and client query performance. In addition, the current blockchain storage engine cannot provide version control and multi-person collaboration functions. This article proposes to solve the problem of query performance and query semantics not rich by linking other databases, the version control function is realized by separately designing the version control semantics, and the function of deleting the duplicate data by the Pos-tree can be effectively used to solve the problem of waste of storage space during version control. Experiments show that under normal network delay (<80ms), the middleware system proposed in this paper has a slight improvement in query performance, and can also save storage space as much as possible during version control.**

Keywords—Decentralized; Version control; Pos-tree

I. INTRODUCTION

Blockchain is a chained data structure of distributed, decentralized storage. It is a distributed ledger, each node has a complete ledger. The most notable features of the blockchain itself are: distributed, decentralized, and information cannot be tampered with. All transactions and status in the network are recorded on the blockchain.

The blockchain consists of a series of blocks, which are referenced by the previous block to form a chain structure. Through peer-to-peer communication technology and encryption technology to form the entire blockchain system. From the perspective of the protocol, the application can establish a consensus mechanism to promote the realization of various functions.

In terms of applications, Bitcoin and Ethereum use peer-to-peer networks[3] to run consensus algorithms[5-6] to infer what transactions can be chained and the data on the chain will not be tampered with and deleted. As a distributed storage system, the blockchain does not store account data on a specific secure node or server, but all nodes in the network have a backup of information. As a public database, the blockchain can record and update all transaction information between the Internet. The use of cryptography technology can avoid data tampering. In terms of databases, there are many storage engines with different storage models and operations, so that their storage models can serve a variety of applications. For example, there are levelDB and AmazonDB based on key-value storage. This data model is highly scalable. There are also relational databases such as MySQL, which supports more complex relational models and more powerful operational semantics. Its transaction needs to follow the ACID principle, so it is relatively poor in scalability. There are also data models between relational and key-value systems, such as BigTable and Yahoo data platforms. We have a lot of choices, but in modern applications, there is a certain gap between the functions provided by the storage system and the requirements required by the application.

The bottom layer of most blockchain systems uses level DB to store data, which is a storage system specially designed for writing data. He used the read performance of data in exchange for write performance. In fact, in most blockchain systems, the requirements for write performance are not high. In the Bitcoin system, the current transaction write speed is one stroke per second. In Ethereum, it is about 7-10 transactions per second. This greatly wastes the high-speed write performance of levelDB. All nodes in the blockchain need to synchronize the latest data on the chain at all times, so the performance requirements for reading are higher. With the increase of blockchain data and the expansion of applications, it is often necessary to process frequent queries. Especially in the DApp application of Ethereum, it is necessary to frequently use external frameworks to query smart contracts and data on the chain. The lack of reading performance is the main bottleneck of query performance.

II. RELATED WORK

At present, various mainstream blockchain systems can be divided into three major modules: data storage module, consensus protocol module, and query processing module[7]. The main task of the data storage module is to achieve efficient data persistence and provide external access interface functions. The underlying data storage system determines the data storage structure and reads and writes performance. At present, the research on stand-alone high-performance data storage systems is very mature, but there is very little research on the application of storage systems[8-10] in the blockchain. Most of the research is focused on the new blockchain architecture[1]. Therefore, first, introduce the blockchain Data storage system used in. Maintaining the Integrity of the Specifications.

A. *LSM-Tree structure storage*

levelDB is a key-value storage engine open-sourced by Google. And a large number of blockchains are currently used as storage engines for development. It has the following characteristics:

- Arbitrary length key value: The key and value are regarded as a simple byte array, so their length can be arbitrarily long.
- Compressed storage: Use the Snappy compression library to compress data, which can reduce disk storage pressure.

The bottom layer of levelDB uses the log sorting merge tree (LSM-Tree)[4] as the data storage structure. In the big data analysis scenario, the data operation to be processed by the system is the continuous reading and writing of block data, and the query optimization strategy of the traditional data management system does not bring obvious performance improvement. The LSM-Tree used at the bottom of levelDB has extremely high write performance. Unlike the traditional B/B+ tree, this structure can reduce a large amount of random read and write overhead caused by updating the index structure during the data writing stage. Complete sorting of fixed-size shard data, and then write it to disk together. The background gradually merges small ordered files into large ordered files to improve search efficiency. However, the B+ tree only generates very little memory sorting spending during the writing phase, and the file sorting is performed in the background, and this operation also only brings the spending of sequentially reading and writing to the disk. This strategy can greatly improve the efficiency of data writing.

B. *Traditional storage engine and ForkBase*

Traditional storage engines do not have good storage support for data version control, fork semantics, and query performance improvement. If you want to implement this support, either rebuild on top of these systems or deploy your implementation from scratch. Both methods increase development costs and delays, and the former may also cause unnecessary performance spending. Building data structures on current blockchain platforms do not scale well with large data sets, nor does it support rich query processing.

ForkBase[2] implements a version control that can be implemented in modern applications and can maintain small storage and computational overhead when maintaining large data versions. Another function is to provide small, flexible, and powerful query statements for various applications. First, to be able to quickly retrieve and verify the integrity of the data, the version number is used to uniquely identify the data content and its history. Second, the large objects are divided into data blocks, and a new index structure Pos-Tree is used to organize. This index structure combines the concepts of Merkle trees and B+ trees. This structure can effectively identify and delete duplicate blocks between objects, thereby greatly reducing the storage overhead of version forks. Third, the Pos-tree supports copy-on-write during the fork to eliminate unnecessary copies. Finally, ForkBase provides a simple API and many structured data types, which helps reduce development costs and find the best balance between query performance and storage efficiency. In summary, ForkBase made the following contributions: supports common requirements in modern blockchain applications: version control and high-speed queries. These attributes are not well supported in traditional storage systems. ForkBase supports common fork semantics and open-source simple and elegant API. It also provides an index structure Pos-tree to manage large objects to reduce the storage overhead of multiple versions of objects.

C. *Externally linked database*

ForkBase can implement version control of data, and use it as an external database to add an external cache to improve query performance, and this method can quickly synchronize the copy in the blockchain database to the local node. The method of this kind of thinking mainly comes from the EtherQL system, which mainly includes three main modules: data monitoring, parsing module, and external database query API. In the operation stage of the blockchain system, the data monitoring and analysis module monitor the block data through the API provided by the blockchain and import it into an external database. The query is mainly realized by the query interface provided by the external database. However, no externally linked database can achieve version control and has high performance.

III. METHODOLOGY

This article proposes a storage middleware that supports multiple version control and has more efficient query processing than the Ethereum client. The middleware is developed using Java language and based on the spring framework, it can be easily integrated into modern applications. He provides query services for applications by providing Restful interfaces.

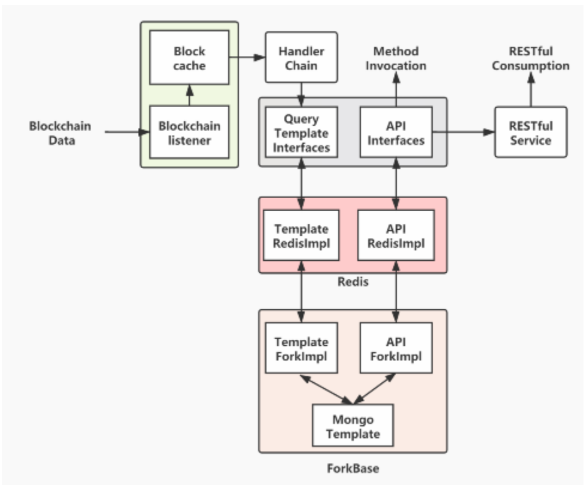

Figure 1. Pos-Tree database middleware

The whole system framework is shown in Figure 1. To be able to synchronize the latest block data in time, the system sets a blockChain listener at the beginning of the model layer to continuously monitor whether the block has changed. If the blockchain data is found to change, the event listener will put it in the Block cache. Handler Chain continuously extracts data from the cache and extracts the data into three data structures: blocks, transactions, and accounts, each of which corresponds to a different processing function. The processed data will be persisted to the database with Pos-Tree as the underlying structure. Developers can interact with the underlying data through the RESTful interface provided by the middleware. This module provides more flexibility from maintenance to deployment. In addition to the data that can be obtained by the underlying database, every time a user fetches data, they will first access the cached Redis. If Redis does not have the required data, then the data will be obtained from the Pos-Tree structured storage system.

IV. Design Detail

A. *Pattern-Oriented-Split Tree*

The existing index structure is not suitable for a data environment with multiple versions and versions can be merged. In the data structure supporting B+ tree and its variants, the final structure will be different because of the different insertion order of the data. For example, the structure of inserting the value A first and then inserting the value B is different from inserting B and then A. There are two consequences of maintaining two versions at the same time. First, it is difficult to share the same elements contained in the index node and the leaf node. Second, because of structural differences, it takes time to find the differences between the two versions and merge them. In order to solve the above problems, ForkBase proposed a novel Pattern-Oriented-Split tree structure. He has the following attributes:

- Quickly find and update elements.
- Quickly find differences and merge two trees.
- Effective in deduplication.

The working mode of Pos-Tree is to scan from the beginning. If a predefined pattern is encountered, a new node is created to save the scanned content. Because internal nodes have different degrees of randomness, they need to define different modes for them.

B. *Version control management*

The blockchain system is a practical example of a data version control system, where each block represents a version of the global state. If you want to support the data version, you need to reduce storage consumption. Because a large number of repeated branches will cause the storage system to crash. The most common method is to check only the incremental data and check its duplicate data, where the increment is the version difference. The underlying storage system of middleware proposed in this paper also applies incremental deletion technology. In addition, it also uses the branch management system provided by ForkBase to support a more diverse version management process. In order to achieve a simpler version control method, this article system has improved the semantics of ForkBase. There are two types of version control semantics commonly used in applications: on-demand bifurcation and conflict-forking bifurcation. In the git system, users are allowed to fork out their own branches from other branches. In the blockchain system, the occurrence of branch conflicts is a mechanism that uses the longest chain to connect. Therefore, in this system, it is necessary to naturally support forking operations based on conflicts.

TABLE I. Fork Api

API	Method
Get	Get(key,branch)
Put	Put(key,branch,value)
Fork	Fork(key,ref_brh,new_brh)

Table 1 illustrates the basic operational semantics supported. The latest version of the branch becomes the branch header. A branch can only be modified in the head. However, to change the historical version, you can create a new branch (branched version) to make it modifiable. And each branch has no limit on the number of forks.

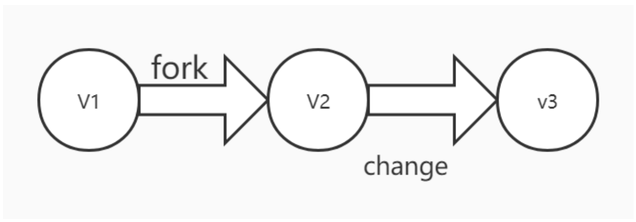

Figure 2. fork a new branch.

Before making any changes, you need to create a branch in advance. For example, in Figure 2, first use the Fork statement to create a new branch(v2 branch), and then update the branch(v3 branch).

C. *Outer Design*

The bottom layer uses the storage engine organized by Pos-Tree, and the outer layer uses EtherQL and cache for encapsulation. Provide convenience to the entire system and the query statement of the blockchain and more efficient query performance. Therefore, we will introduce the architecture of the outer layer system from query statements and query performance.

The query statements of the underlying storage system do not meet the needs of daily application development. Therefore, a set of APIs with richer query statements is needed for application developers to use. The supported APIs can be applied to three types of data types: accounts, transactions, and blocks. According to some common query methods in blockchain applications, the main API is shown in Table 2.

TABLE II. API Interface

Functionality	Account	Transaction	Block
Range query	balance	transaction hash	Block hash
Aggregate	balance	transaction value	Block number
Top K by	balance	transaction value	-

It can be seen that if the database is connected to the outer layer, top-K and range queries can be realized. The query module API interfaces provide an abstraction to decouple from the underlying database. So this system is very scalable. In the future, more underlying storage systems can be supported.

V. Performance Evaluation

The main test is the QPS value between the middleware proposed in this article, the Ethereum client, and EtherQL with mongoDB as the cache. The data set is based on the transaction data of the Ethereum main network from 2017 to 2020. The total transaction volume is 700 million, the total number of blocks is 2.3 million, and the average size of each block is 30kb. In middleware, the block data obtained each time can be regarded as a tuple of <address, balance, nonce, code, storage>. The role of nonce is to record the number of transactions sent by the current user, and can also be used as the unique ID of the transaction, which can be used to cancel a transaction. The transaction can be seen as a tuple composed of <hash, sender, receiver, value, gas, gasprice, data>. Gas is used as an incentive in Ethereum. When the miner processes the transaction successfully and puts it on the blockchain, it will be used as a fee for the miner.

All experiments are run on the PC. The configuration of the PC is: Intel Core I7 CPU of 8 cores, 3.6GHz. The memory is 8GB. We divide the test into three parts: get account balance, get transaction, get block. Because of the delay, the results are averaged over 30 test results. label, present them within parentheses. Do not label axes only with units. In the example, write "Magnetization (A/m)" or "Magnetization {A[m(1)]}", not just "A/m". Do not label axes with a ratio of quantities and units. For example, write "Temperature (K)", not "Temperature/K".

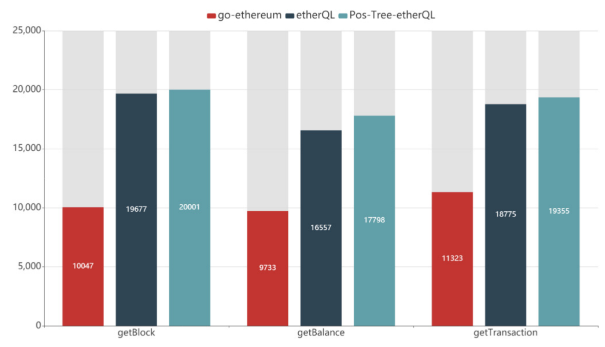

Figure 3. Query performance.

A. QPS

As shown in Figure 3, the QPS in EtherQL is almost twice that of the Ethereum client (go-ethereum), and if the Pos-Tree structure is used as the underlying storage structure, the performance will be slightly improved compared to EtherQL.

B. Multi-version storage capacity

Because the underlying database supports version control and fork semantics, compared to EtherQL, it will provide more available interfaces for applications. Because it is impossible to compare the storage capacity of the fork version before and after the EtherQL comparison. Therefore, only the storage capacity of the Pos-Tree storage system can be tested. Only 20 tuples can be accessed in the cache. Therefore, the test is based on the limit of 20 tuples. Table 3 shows the comparison of storage capacity before and after fork.

TABLE III. Storage Capacity Performance

Category	**Typical semantics**	**Storage(deta>=2MB)**	
		Before operation	**After operation**
fork	Fork(key,ref,new)	6.3MB	8.5MB
merge	Merge(key,taget,ref)	(1)6.3MB (2)4.2MB	6.01MB

VI. Conclusion

Due to the limited query support of the underlying data storage of the blockchain application, the original blockchain application development efficiency is low and the performance is poor. With the advent of EtherQL, the external database enriches the query semantics of Ethereum applications. In addition, because of the introduction of blockchain listeners and cache, EtherQL middleware far exceeds the Ethereum client in query performance. However, EtherQL does not support version control. This article refers to the ForkBase storage system on the basis of EtherQL, and replaces the underlying MongoDB organized by B+ tree with the storage system organized by Pos-Tree, and adds version control semantics on this basis. Experiments show that the storage middleware not only improves query performance slightly, but also supports blockchain applications for multi-person collaborative development.

References

[1] Bhattacharya, Amiya,Roy, Abhishek,Das, Sajal.2001. Towards a Novel Architecture to Support Universal Location Awareness,page 182.

[2] Wang S, Dinh T T, Lin Q, et al. Forkbase: an efficient storage engine for blockchain and forkable applications[C]. very large data bases, 2018, 11(10): 1137-1150.

[3] Pass R, Seeman L, Shelat A, et al. Analysis of the Blockchain Protocol in Asynchronous Networks[C]. theory and application of cryptographic techniques, 2017: 643-673.

[4] Oneil P, Cheng E Y, Gawlick D, et al. The log-structured merge-tree (LSM-tree)[J]. Acta Informatica, 1996, 33(4): 351-385.

[5] Kosba A E, Miller A, Shi E, et al. Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts[C]. ieee symposium on security and privacy, 2016: 839-858.

[6] Halpin H, Piekarska M. Introduction to Security and Privacy on the Blockchain[C]. ieee european symposium on security and privacy, 2017: 1-3.

[7] Li Y, Zheng K, Yan Y, et al. EtherQL: A Query Layer for Blockchain System[C]. database systems for advanced applications, 2017: 556-567.

[8] Wu X, Xu Y, Shao Z, et al. LSM-trie:an LSM-tree-based ultra-large key-value store for small data[C]. usenix annual technical conference, 2015: 71-82.

[9] Agarwal R, Khandelwal A, Stoica I, et al. Succinct: enabling queries on compressed data[C]. networked systems design and implementation, 2015: 337-350.

[10] Lim H, Fan B, Andersen D G, et al. SILT: a memory-efficient, high-performance key-value store[C]. symposium on operating systems principles,2011:1-1.