awesome-parallel-blockchain

Faster, robust, and high-performanced blockchain systems are the cornerstone of tomorrow.

1. `Contributions from the community are welcome; make sure the material you add is worth reading, at least you have read it yourself.`
2. About the Format:
    - for the papers: [Type] [Conference(If any)] Name, Author, Year, Accessible links
        - Type: [Academic] Paper that demonstrates more about theoretical algorithms or theoretical analysis.
        - Type: [Engineering] Paper that introduces a new system architecture or practical algorithms
    - for the repos: [Type] Repo Name, Org, Year, Accessible links
        - Type: [PoC] Basic runnable codebase with some basic functions to evaluate the ideas.
        - Type: [Developing] Systems under development that have some basic features but are not yet ready for use in a stable production environment.
        - Type: [Product] Systems that are still under maintenance are already ready to be used in production environments.
        - Type: [Archive] Non-maintained code base.

Parallelism and concurrency in brief

Background

Parallelism and concurrency are two sharp swords that could improve the transaction execution performance of Blockchain and Blockchain-related VMs(i.e., EVM and SVM).

This knowledge base collects information about state-of-the-art research and engineering works about parallel and concurrent blockchain, blockchain VM, and other related systems, with some relevant examples for better understanding.

Parallelism and **concurrency** are two easily confused terms; although they are both used to improve the efficiency of the execution of the system, they differ in meaning and in the scenarios in which they are used.

Parallelism, typically refers to multiple processes or threads working on different tasks at the same time. For example, ten threads are executing ten different transactions simultaneously. Or think of ten dogs eating from ten food bowls.

Concurrency, typically refers to a thread simultaneously working multiple tasks to increase efficiency. For instance, certain operations, like reading data from disk(i.e., sLoad in EVM), might need to wait for a while in order to be completed. In such case, the main thread could move ahead and do other duties, like computation. Then, It returns to continuing execution after the disk's data has been read in complete. Again, think about using a bowl to feed ten dogs, and being in the middle of a meal the dog is chewing on a bone, when you could start by letting him out of the bowl for a while and letting the other dogs eat first.

For further reading: Difference between Concurrency and Parallelism

Core challenges

In general, the core challenge of adopting a parallel or concurrent method is the data race problem, read-write conflict, or data hazard problem. All these terms describe the same issue: different threads or operations are trying to read and modify the same data at the same time.

Consider a transfer scenario on Ethereum where Alice and Bob both want to transfer 10 ETH to Carl at the same time. Suppose Carl's initial balance is 100 ETH. After these two transactions are completed, Carl's balance should be 120 ETH. Let's consider a fully concurrent scenario where Alice's and Bob's transactions

start executing at the same time. The initial balance of Carl that their transactions read would be 100 ETH (see the error?). Eventually, they will update Carl's balance as 110 ETH.

To resolve a data race issue, there are usually three ways:

- Protecting competing data by adding **locks**,
- **Scheduling module** to avoid conflicts,
- **Optimistic approach** that doesn't worry about conflicts but **rolls back** transactions after encountering the conflict issue.

So how do we implement these solutions in the blockchain world? Let's dig in!

Case Study: Parallel/Concurrent EVM

In the current EVM architecture, the most fine-grained read and write operators are **sload** and **sstore**, which read and write data from the state trie, respectively. So the easiest entry point is how to make sure that different threads don't conflict on these two operators. In fact, there is a special kind of transaction in Ethereum that includes a special structure named Access list, which allows transactions carrying storage addresses that it will read and modify. Therefore, this is a good entry point for implementing scheduling-based concurrent methods.

In terms of system implementations, there are three general forms of Parallel/Concurrent EVM.

1. Multi-threads with one EVM instance.
2. Multi-threads with multi-EVM instances on one node.
3. Multi-threads with multi-EVM instances on multi-nodes (basically, this is system-level sharding).

So, what makes Parallism/concurrency on blockchain different than in the database system?

1. **Unreliable Timestmap** which makes timestamp-based concurrency methods hard to deploy in the blockchain world.
2. **Absolute determinism** on a blockchain system to ensure that transactions are re-executed between different validators is the same.
3. The ultimate goal of validator is higher earnings, not faster executing transactions.

What do we need?

1. System-level consensus is required, and faster execution will bring higher returns.
2. **Multivariate scheduling algorithms** that, given a block limit, capture more revenue while being able to complete the execution faster.
3. More fine-grained data operations, including **opcode-level data locks**, in-memory cache layers, etc.

Parallelism and Concurrency in Blockchain

Research Papers

- [Academic][TPDS] PaVM: A Parallel Virtual Machine for Smart Contract Execution and Validation, 2023, [Paper]
- [Academic] Parallel and Asynchronous Smart Contract Execution, 2021, [Paper]
- [Academic][SIGPLAN] Practical smart contract sharding with ownership and commutativity analysis, 2021, [Paper]

- [Academic][VLDB] SlimChain: scaling blockchain transactions through off-chain storage and parallel processing, 2021, [Paper]
- [Academic][DASFAA] PEEP: A Parallel Execution Engine for Permissioned Blockchain Systems, 2021, [Paper]
- [Academic][SIGMOD] A Transactional Perspective on Execute-order-validate Blockchains, 2020, [Paper]
- [Academic] An Empirical Study of Speculative Concurrency in Ethereum Smart Contracts, 2019, [Paper]
- [Academic][PDP] An efficient framework for optimistic concurrent execution of smart contracts, 2019, [Paper]
- [Academic] FastFabric: Scaling hyperledger fabric to 20000 transactions per second, 2019, [Paper]
- [Academic][SIGMOD] Blurring the Lines between Blockchains and Database Systems: the Case of Hyperledger Fabric, 2019, [Paper]
- [Academic][ICDCS).] Parblockchain: Leveraging transaction parallelism in permissioned blockchain systems, 2019, [Paper]
- [Academic][PODC] Adding Concurrency to Smart Contracts, 2017, [Paper]
- [Engineering] Block-STM: Scaling Blockchain Execution by Turning Ordering Curse to a Performance Blessing, **Aptos**, 2022, [Paper], [Video]

Engineering Repos

- [PoC] Parallel-go-ethereum, **ABCDELabs**, 2022, [Codebase]
- [Developing] Evmone-compiler, **MegaETH**, 2023, [Codebase]

Other Materials

- [Blog] Speeding up the EVM (part 1), Flashbots, 2022, [Blog]

Parallelism and Concurrency in DBMS

Research Papers

Parallel Query Optimization

- On Optimistic Methods for Concurrency Controll, 1981, [Paper]
- Scheduling problems in parallel query optimization, 1995, [Paper]
- Efficient and accurate cost models for parallel query optimization, 1996, [Paper]
- Parallelizing query optimization, 2008, [Paper]
- Flow algorithms for parallel query optimization, 2008, [Paper]
- Query optimization for massively parallel data processing, 2011, [Paper]
- Communication steps for parallel query processing, 2017, [Paper]

Engineering Repos

Other Materials

- [Tech Doc] Concurrency Control in PostgreSQL, PostgreSQL, [Doc link]

Parallelism and Concurrency in OS

Research Papers

Engineering Repos

Other Materials

Appendix