Midterm Report

bmjain2, yuquanw2, vinithk2, ss77
Code repository: https://gitlab.engr.illinois.edu/vinithk2/prism_voting_chains

We have implemented Prism voting chains to reduce transaction confirmation latency.

Overview

The Prism protocol aims to scale the blockchain protocol (increase throughput and reduce transaction confirmation latency) by decoupling the roles carried by a single block. The authors factorise the blocks into three types: proposer, transaction and voter blocks. The proposer blocktree anchors the Prism blockchain and contains references to transaction blocks. There are `m >> 1` voter chains, and voter blocks vote for proposer blocks. However, only votes in the longest chain are counted and *each voter chain* can only vote for one proposer block at a particular level. The leader proposer block at each level is the one with the highest number of votes among all the proposer blocks at that level. The elected leader blocks provide a unique ordering of the transaction blocks to form the ledger.

Block

Our project is aimed at reducing latency, hence we implement two types of blocks -- proposer and voter blocks and include transactions as a part of content of proposer blocks. The proposer content contains parent hash, references to other proposer blocks and a list of transactions. The voter content contains parent hash, votes for proposer blocks and the chain number they belong to.

Blockchain

Blockchain is the central structure that keeps track of the blocks received over the network and mined locally. It stores additional data structures to aid the mining and ledger manager modules:

- `proposer_chain` is a hashmap that maps the block hash to the metablock (block + level).
- `proposer_tip` stores the tip of the longest chain in the proposer tree
- `proposer_depth` records the depth (distance from the genesis block) of the proposer tip.
- We maintain a vector of similar structures for voter chains.
- `unref_proposers` stores a list of proposer block hashes that have not been referenced yet. When a new proposer block arrives, we add its hash to the list and remove the parent and referenced proposer hashes from the list.

- Voter blocks vote for the first proposer block received at a particular level. `level2proposer` maps level to the hash of the first proposer block inserted at that level.
- `level2allproposers` is a hashmap that maps a level to the vector of all proposer block hashes at that level.
- `proposer2votecount` is a hashmap that maps a proposer hash to the total number of votes it has. When a new voter block arrives, there can be two scenarios:
    - *New voter block does not change the longest voter chain:* Simply update `proposer2votecount` by adding the votes from the new block.
    - *New voter block changes the longest voter chain*: In this case, we should subtract counted votes from the old chain and add votes corresponding to the new chain. Find the least common ancestor (LCA block) of the two chains, subtract votes for blocks from old tip to the LCA block and add votes from new tip to the LCA block.
- `orphan_buffer`: An incoming block is inserted into the blockchain only when it has no missing references. Otherwise it is stored in a hashmap which maps missing reference hash to the block. Indexing by missing reference allows us to process orphaned blocks when their missing references arrive.

Miner

Miner creates a superblock containing contents of proposer block + `m` voter blocks belonging to different chains. It polls blockchain for the proposer tip and list of unreferenced proposers and includes transactions from the mempool. For the ith voter blocks, it polls the ith voter tip and includes votes for proposer from (the max level voted by the parent + 1) to the `proposer_depth`. It votes for the first proposer received at a particular level (available in blockchain `level2proposer`). For the header, it includes a random nonce, the merkle root of the contents list and some other uninteresting fields. If the block hash is less than required difficulty, it is sortitioned into the proposer tree or one of the `m` voter trees. A processed block containing the header, relevant content and sortition proof is created. It is inserted into the local blockchain and the new block hash is broadcasted to peers.

Mempool:

Mempool contains the following members:

- A BtreeMap for speedy insertion and removal of objects stored in FIFO order (helps extract transactions in order of arrival).
- Hashmap that maps transaction hashes to transaction that allows for convenient hash based indexing.

- Hashmap that maps transaction input to transaction. Used for detecting double spends and removal of dependent transactions.
- Counter to store the size of BtreeMap.

It provides access to the following methods:

- `insert` : Inserts transaction into all the structures in Mempool
- `get`: Retrieves a transaction given its hash.
- `contains`: Checks whether a transaction is present in Mempool
- `is_double_spend`: Checks whether transaction is using inputs that had already been consumed by a transaction in Mempool.
- `delete_dependent_transaction`: Removes transactions that depend on the outputs of a deleted transaction.
- `get_transactions`: Returns `n` transactions in FIFO order form Mempool.
- `mempool_len`: Returns size of Mempool.

Network

The network transmits blocks and transactions between peers. We do not perform transaction validation at this stage. Here we directly insert transactions into the mempool. Transactions are validated by the ledger manager once the order of transactions is finalised. Blocks are validated against the following checks:

- satisfies proof-of-work (block hash < difficulty)
- is correctly sortitioned into proposer trees or one of the `m` voter trees.
- sortition_proof provided in the block verifies.

We need to implement validation against content semantics like each voter chain votes for a level only once, voter starts voting levels after parent block, etc.

Ledger Manager

It is responsible for making sure that the transactions are confirmed and UTXO is updated. It runs asynchronously in a separate thread. It does following three activities:

- First, it goes over the proposer blocks and selects a leader at each level.
- Second, after the leader selection process, it iterates over the leader sequence and forms a list of ordered transactions. For a leader, transactions from the referenced proposer blocks are included before its own transactions.
- Third, we iterate over the transaction sequence and remove any transactions which are already processed and are not valid. Once the list is sanitized, we update the UTXO state. Validity of transactions is checked in the similar way we implemented for the normal Bitcoin client.

We maintain a LedgerManagerState which consists of structures such as

- `last_level_processed`, an integer which indicates the level progresses till now
- `proposer_blocks_processed`, a HashSet which keeps track of processed proposer blocks
- `tx_confirmed`, a HashSet which keeps track of processed transactions
- `leader_sequence`, a list of proposer blocks chosen as leaders

We poll the Blockchain to obtain the proposer candidates at a particular level (`level2allproposers`) and the number of votes scored by the proposer blocks (`proposer2votecount`) to help elect the leader proposer block at each level.

Finally, we have an instance of `UtxoState` which we use to update the Utxo Database (In memory HashMap).

Future Work:
We have a simple implementation ready. Next, we plan to remove bottlenecks and improve performance metrics. Here are some of those optimizations:

- Fine-grained locking over Blockchain components: When the miner or ledger manager needs to poll information from the Blockchain, we obtain a coarse lock over the entire structure. We can switch to fine-grained locking over components to reduce contention.
- Blockchain signals Miner of new blocks: In our current implementation, miner creates the superblock from scratch in each iteration. This is wasteful. If there are no new blocks, the miner can reuse the superblock, change nonce and continue mining. For example, if a voter block in chain `i` arrives, it doesn't affected the proposer or other voter block contents in the superblock.
- Confirming transaction in parallel threads: Currently, the transactions are confirmed sequentially in ledger manager. Prism has a high throughput and thus this leads to a bottleneck. Hence, we will assign a pool of workers to confirm transactions which will reduce confirmation latency.

We also plan to simulate attacks such as balance attack and see how our confirmation latency fares against increasing adversarial hash power.