Adding Concurrency to Smart Contracts

Presentation by : Shravika Mittal(2016093)
Mehak Gupta(2016163)

Introduction to Ethereum

- Ethereum is a modern blockchain system, which has ether as its currency.
- It differs from Bitcoin as it has another layer between the users and the blockchain system called the smart contracts.
- This system has three types of people in it.
  a. Users - people who make transactions in cryptocurrency.
  b. Miners - people who propose new blocks to be added to the chain.
  c. Validators - people who re-execute the current state of the chain.
- Smart Contracts - It is a piece of code which performs the logic needed to provide a complex service, like authentication and changing values of parameters.

Smart Contracts

A smart contract code keeps record of the sender, receiver and the amount to be transferred.

In reference to the Ballot contract example given in the paper the contract has information about:

- The array of voters
- Weight of votes cast by each voter
- Voting status of each voter.
- Array of candidates.

Introducing Concurrency to the Blockchain System

Why concurrency??

- Modern day computer systems are very advanced with high processing speeds. If we limit the miners to serialised execution of smart contracts then we lag in the following:
  a. We do not use the resources available to us efficiently.
  b. We limit the throughput and so the miners who could have earned a lot more have to settle for a less pay.

What would happen if we add concurrency without thinking much?

- Existing systems are non deterministic, that means they do not store the schedule followed by the miners while executing the smart contracts. Hence when a validator again executes the transaction concurrently then there is a high possibility that he chooses a different order of execution resulting in a different final state.
- This would lead to no block being added to the blockchain, which was not what we intended to do :(
- Also there may be situations similar to a deadlock when two transactions access the same shared data.
- Hence before adding concurrency we need to be careful about its repercussions.

What this paper achieves?

The paper makes the following contributions:

1. To avoid low throughputs we can use parallel execution of smart contracts, on the miner's front.
2. Storing the schedule followed by the miner to hint the order to be followed by the validator.
3. To calculate the overall speed-ups for miners and validator when tested on a benchmark data.

Storage Operations on EVM

- Two storage operations commute if they provide the same final output, irrespective of their order of execution. Referring to the ballet example, Alice giving her vote to candidate 17 and Bob to candidate 14 commutes. While a deletion of alice's vote does not.
- To maintain concurrency there are two mechanism:
  a. Abstract locks - associated with each storage operation.
  b. Inverse logs - used when we have to undo the operation(abort).
- It may be possible that before a transaction is completed an error occurs, so the system should be able to go back to its prior state, so we use logging mechanism, which are stored as inverse logs.

Welcome back to Advanced Programming

- When a smart contract makes a call to another, it creates a child which acquires the same abstract lock from its parents.
- The child starts executing its code and makes changes to the inverse log in case it aborts.
- But if the child commits its changes do not get updated until the parent commits.
- When the child exits the lock is passed back to the parent.
- And when the parent commits or aborts the lock is freed for another thread to acquire.

Miner side execution

- Since the miner has to deterministically make the schedule for the validator to use, every abstract lock has a counter and a log. Both of them are reset when a new block is created.
- Whenever each action is committed the counters of the locks which were acquired in the process get incremented and the lock profile gets stored.
- Then the minor uses topological sort based on this data to produce a happened before graph which is then used by the validator.

Validator side execution

- Based on the graph provided by the miner, the validator constructs a fork join program which upon execution would result in the same sequence as executed by the miner.
- The validator uses Work Stealing to increase its parallelism.
- Work Stealing is the process in which an idle thread makes itself useful by stealing work.
- It is similar to implementing DFS on the happened before graph.
- The validator then compares the lock profiles after completing execution. If the miner’s and the validator’s profiles differ then the block gets rejected.

Evaluation

There are 2 conditions to be considered while evaluating the concurrent implementation :

1. For a given amount of data conflict, what is the effect on the speedup as we increase the number of transaction.
2. How does the speedup change as data conflict increases?

For the first case speedup increases as the number of transactions increases, while in the second case as the data conflicts increase there is a drop in the speedup. But both of these are limited by the underlying hardware, i.e. the core availability.

It’s all about the Numbers

The two standards used for calculating the speedup are:

1. 10 to 400 transactions at 15% data conflict.
2. 200 transactions at 0% to 100% data conflict.

Observation:

The impact of data conflict on speedup depends on the contract implementation.

	SimpleAuction		Ballot		EtherDoc		Mixed	
	Conflict	BlockSize	Conflict	BlockSize	Conflict	BlockSize	Conflict	BlockSize
Miner	1.23x	1.58x	1.57x	1.35x	0.78x	1.09x	1.57x	1.45x
Validator	1.35x	1.60x	1.73x	1.58x	2.04x	1.75x	1.86x	1.64x

Conclusion

- We have thus seen, how multi core systems can be used for increasing the throughput of the work performed by miners and validators.
- Miners execute the smart contracts in the blocks in parallel and they include this parallel schedule for the validators sake.
- The validators then re-execute these transactions using the schedule provided by the miners parallely using work stealing and fork-join program.
- Overall speedups of 1.33x for miners and 1.69x for validators was seen when three threads were used.

What’s its future?

- Adding multithreading to Ethereum Virtual Machine.
- Scheduling the metadata in blocks so that this technique is compatible with other platforms as well which do not have the concept of smart contracts.
- Giving proper incentives to miners so that they are encouraged to compute concurrently.
- Implementing soft forks in this system so that we can achieve backward compatible changes.