Parallel Execution

Monad executes transactions in parallel. While at first it might seem like this implies different execution semantics than exist in Ethereum, it actually does not. Monad blocks are the same as Ethereum blocks - a linearly ordered set of transactions. The result of executing the transactions in a block is identical between Monad and Ethereum.

Optimistic Execution

At a base level, Monad uses optimistic execution. This means that Monad will start executing transactions before earlier transactions in the block have completed. Sometimes (but not always) this results in incorrect execution.

Consider two transactions (in this order in the block):

1. Transaction 1 reads and updates the balance of account A (for example, it receives a transfer from account B).
2. Transaction 2 also reads and updates the balance of account A (for example, it makes a transfer to account C).
3.

If these transactions are run in parallel and transaction 2 starts running before transaction 1 has completed, then the balance it reads for account A may be different than if they were run sequentially. This could result in incorrect execution.

The way optimistic execution solves this is by tracking the inputs used while executing transaction 2 and comparing them to the outputs of transaction 1. If they differ, we have detected that transaction 2 used incorrect data while executing and it needs to be executed again with the correct data.

While Monad executes transactions in parallel, the updated state for each transaction is "merged" sequentially in order to check the condition mentioned above.

Related computer science topics areoptimistic concurrency control (OCC) andsoftware transactional memory (STM).

Optimistic Execution Implications

In a naïve implementation of optimistic execution, one doesn't detect that a transaction needs to be executed again until earlier transactions in the block have completed. At that time, the state updates for all the earlier transactions have been merged so it's not possible for the transaction to fail due to optimistic execution a second time.

There are steps in executing a transaction that do not depend on state. An example is signature recovery, which is an expensive computation. This work does not need to be repeated when executing the transaction again.

Furthermore, when executing a transaction again due to failure to merge, often the account(s) and storage accessed will not change. This state is still be cached in memory, so again this is expensive work that does not need to be repeated.

Scheduling

A naïve implementation of optimistic execution will try to start executing the next transaction when the processor has available resources. There may be long "chains" of transactions which depend on each other in the block. Executing these transactions in parallel would result in a significant number of failures.

Determining dependencies between transactions ahead of time allows Monad to avoid this wasted effort by only scheduling transactions for execution when prerequisite transactions have completed. Monad has a static code analyzer that tries to make such predictions. In a good case Monad can predict many dependencies ahead of time; in the worst case Monad falls back to the naïve implementation.

Further Work

There are other opportunities to avoid re-executing transactions which are still being explored.

Previous Execution Next MonadDb Last updated5 months ago

On this page