This paper discusses various concurrency issues with Ethereum contracts.

I think many of these issues come from a single fact that EVM allows a contract to directly call another contract.

This leads to tightly coupled contracts and two major problems

The first problem is security issues due to a possibility of re-entrant behavior (as was manifested by the DAO vulnerability)

The second problem is that it is hard to parallelize contract execution because if is not known in advance which contract is going to be called inside of a given contract so there is potentially lots of effectively shared variables. Currently ETH clients execute contracts sequentially, so you can essentially use only one CPU core.

In the Big Data world it is known that tightly connected architectures are hardly parallelizable and have lots of concurrency issues, so people are switching to so called Actor model, where each actor interacts with other actors only by sending asynchronous messages, and not by issuing blocking calls. Once you have a collection of actors you can execute them

in any way you want and it is guaranteed that there will be no concurrency issues. The actor model is used by many modern frameworks such as Apache Spark and Apache Akka.

In order to make smart contracts actors one needs to prohibit contracts from calling each other directly. Instead, each call from contract A to contract B needs to be done asynchronously by sending a message.

Once contracts are decoupled, contract execution can be made highly parallel in the following way

1. For a given block, first go through all messages and group them according to the contract called by the message. Only calls to the same contract need to be processed sequentially, calls to different contracts can be executed in different threads.
2. As contracts are executed, they will send messages to call other contracts. These virtual messages will form virtual block 1, that is “virtually appended” to the current block. Once the original block is processed, the virtual block 1 messages will need to be sorted and then executed. The messages can be deterministically sorted by first using the hash of the receiving contract address, so that messages calling the same contract are be grouped together, and then according to the hash of the message. In this way virtual block 1 will be uniquely deterministically ordered.
3. Then virtual block 1 will be processed to yield virtual block 2 and so on until there are no more messages.
4. When the miner sends the block to other nodes, only the original block needs to be sent, and the virtual blocks

are reconstructed by nodes during the validation.

Benefits of this model:

a) more secure - all reentrant bugs dissapear

b) much more parallel - a typical Ethereum block will be split into groups of transactions according to the contract address, and each group can be executed in parallel.

c) Faster transaction rate for the Ethereum network - sequential contract execution is currently a performance bottleneck

d) Ethereum client can be re-engineered by using an actor frameworklike this.

e) For sharding, with “Actor” contracts there is an interesting possibility to assign a shard to each contract. This can make both storage and compute highly parallel, and you will not need any cross-shard token transfers, since a particular token can live completely on its shard, and all operations with this token would not cross the shard. Essentially the only token that will need to be transferred across shards will be ETH.