MOAC Yellow Paper

Implementation details

1. Basic setup

MOAC is using a layered consensus stack, to scale transaction processing through layered structure and sharded process, but to keep everything synced in a single Blockchain.

For the underlying layer, we utilize POW as the main consensus protocol, because POW is a widely tested protocol and most robust solution to a large-scale network setup. Currently MOAC uses POW similar to Ethereum. However, we will make the POW consensus pluggable. If needed, we can easy swap the POW with another efficient protocol.

The drawback of POW is compensated in the top layer. Only critical transactions and control flow transactions are processed in the POW layer. The top layer adopts configurable consensus protocol with sharding technique to provide faster and higher throughput solution.

The lower layer node is called v-node. Each v-node has one Smart Contract Server node. The Smart Contract Server (SCS) identity is fully verifiable by the corresponding v-node. Each SCS node will present bond to be able to process the top layer contracts.

Note that SCS processes Smart Contract calls. All transaction in the top layer is in form of Smart Contract calls. Not all SCS will process single transaction at the same time. Rather, part of selected SCS will process specific transaction.

The selection of SCS is through initialization of Smart Contract call or reshuffle call. The init/ reshuffle call is actually passed to the underlying layer and achieves consensus. The init/ reshuffle call will include the selection criteria including percentage of processing nodes. Then each v-node will invoke that call on its SCS with a predefined algorithm. SCS will determine if itself is selected to process this Smart Contract. Note this is a deterministic process and SCS participation can be verified by anyone.

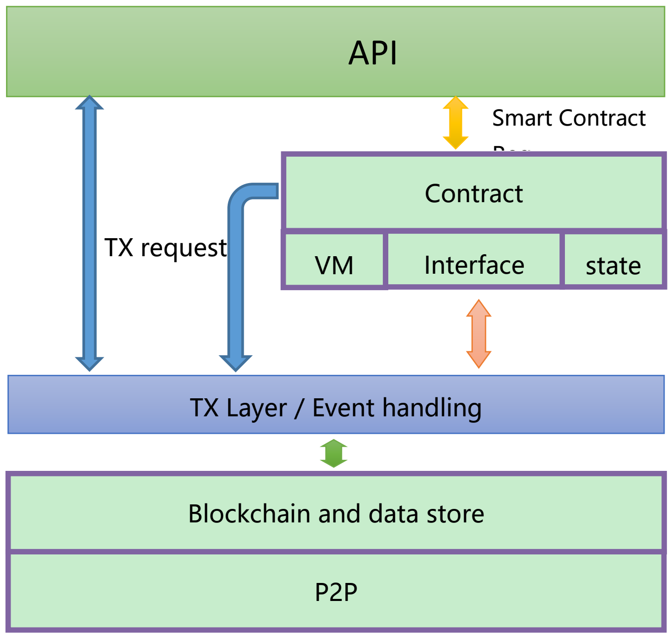
Once group of SCS is selected for certain Smart Contract, they will communicate with each other to form a small consensus group. This group will process the following Smart Contract calls on that Smart Contract. Also, the behavior on how they reach the consensus among them could be specified by the init call. Effectively these SCS nodes form a sub Blockchain and perform the consensus based on the predefined protocol or user defined protocol. Please note the consensus protocol is different from the actual Smart Contract code.

Smart Contract state is saved in each SCS. However, that is not on the actual global Blockchain. To achieve the benefit of global Blockchain, the state needs to flush into the underlying Blockchain periodically or on demand.

When flushing, in the consensus mode, SCS node will initiate a data store request on the underlying v-nodes. This is done by initiating a TX from consensus shard members with proper authentication. The TX will be broadcasted to all v-node. Its validity can be verified by any node. In the same TX, it also specified the rewarding fee to each shard participant. Additional process could be implemented to forfeits SCS's bond if the TX is proved to be invalid.

In MOAC, most contracts will be processed in the top layer, while only small portion of control flow transactions and asset transactions are processed in the v-node layer. This is feasible because top layer provides fast, flexible and low-cost service, while v-node layer provides slow, reliant and expensive service.

Layered Architecture



1) P2P network layer. This layer defines p2p protocol.

2) Blockchain layer. This layer handles all operation related to Blockchain operation, like consensus, data access, etc.

3) TX layer. This layer handles TX request and reply. It also processes the Control flow TX request and if necessary, invokes Smart Contract related operations.

4) Smart Contract layer. This layer performs smart contract execution inside virtual machine and also keeps a temporary contract state.

5) API handles end-user input and gets the output from lower layers.

2. P2P network

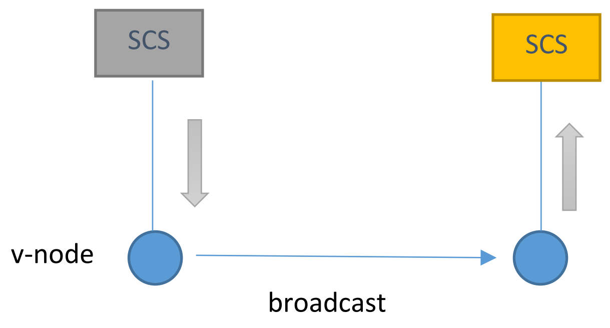
Peer-to-peer network is the underlying layer of communication between nodes running the same protocol. Nodes communicate by sending messages using RLPx, an encrypted and authenticated transport protocol.

Peers are free to advertise and accept connections on any TCP ports they wish, however, a default port on which the connection may be listened is specified explicitly.

Each node connects to its peers and maintain a list of peers. There are three different categories of nodes that we connect to: Bootstrap nodes, Static nodes and Trusted nodes.

The maximum number of peers can be set by each node. A typical number is 25. When broadcasting, node will send pkt to all the peers. And these peers will further send the pkt to their peers. In multiple hops, the pkt will reach all nodes in the network.

The protocol provides the basic networking plumbing. The lower layer consensus is built upon that. Also, the SCS layer will form another consensus layer. However, there is not direct communication between SCSs. So SCS will pkg its protocol msg into the lower pkt. V-node is responsible to broadcast the SCS's pkt to its peers. And each peer will rebroadcast, as well as pass the pkt to its SCS. Thus, SCSs will form a logic consensus with the underlying p2p network.



3. Sharding in general

   Current blockchain has a scalability problem. A blockchain processing capability is limited to the processing power of a single node. That limits the processing speed of Blockchain. It also wastes a lot of processing power of the whole network, as more nodes join the network, they will never increase the system processing power. Instead, it will lower the system performance as network traffic increase.

   Sharding is the solution to this problem. It subdivides the whole network into multiple shards, as long as there are sufficiently nodes in each shard. The system is still highly secure, with many transactions processed in parallel.

   Each shard randomly picks the member to processing a single Smart Contract. All the contract calls on the same contract will be processed by the same shard members. The shard members can be reshuffled either frequently or explicit invoked by proper contract calls.

   Please note that even though only a few nodes are verifying and creating blocks on each shard at any given time, the level of security is in fact not much lower than what it would be if every single node was verifying and creating blocks. Detailed calculation is shown in later section.

4. Distributed random generator

   The sampling requires randomized pick of shard members from all registered Smart contract servers. We will use deterministic threshold signatures for this purpose and have validators collaboratively generate the random value. This scheme is robust against all manipulation unless a majority of validators collude (in some cases though, depending on the implementation, between 33-50% of validators can interfere in the operation, leading to the protocol having a 67% liveness assumption).

   Threshold signature is also used in the commit stage of shard member trying to flush the contract states into the main blockchain. We will cover that later.

5. Selection of shard members

   The selection of shard members is done through following steps:

   1) All SCS is registered in the System Contract with bond.

   2) System contract will pick k% percentage of SCS nodes randomly who have enough bond. The selection algorithm is public. Each v-node can verify if its SCS's ID is selected. The SCS node will be notified by v-node.

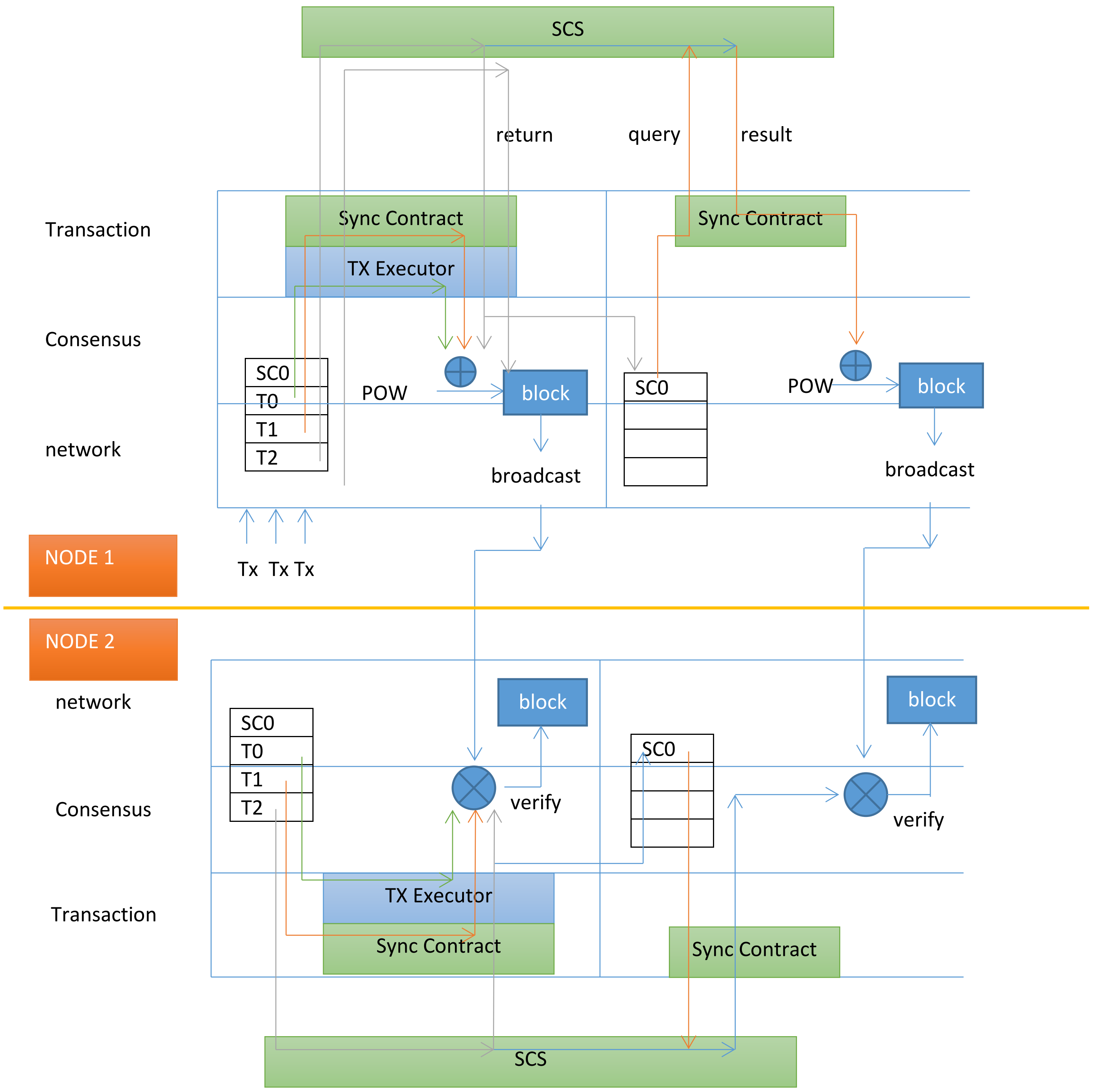
   3) If SCS is active and willing to participate, it will reply with signed Ack to v-node. V-node will broadcast its enrollment.

   4) Within certain period of time, if acknowledged enrollment meets the minimal percentage, then the enrollment is successful.

5) Each node will be notified by a share of secret s_i.

6) If enrollment times out, another round of selection is initiated.

6. Asynchronous Workflow



7. Condition of Asynchronous CS call

   Under following conditions, contract cannot be executed in asynchronous mode:

   a. The output of contract will change external state

   b. Embedded call to other contract in the contract

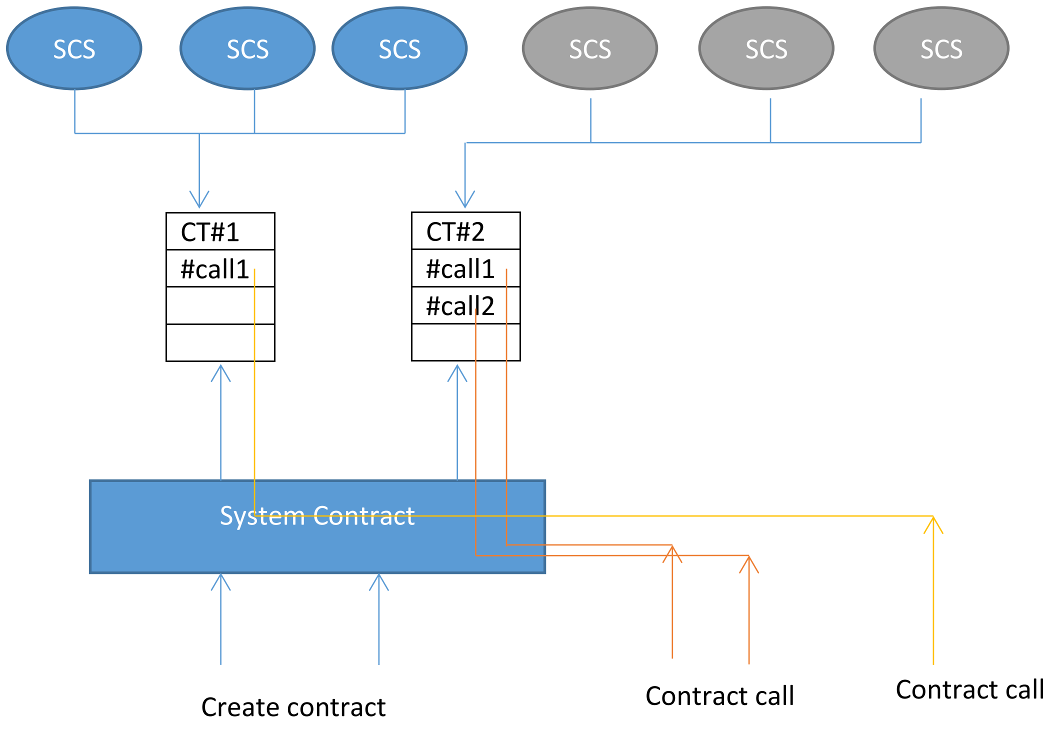
   c. Contract is dependent on input external state. This condition could be relaxed if the dependency is time insensitive, give that each call is executed in a way referencing the source of the dependent input state.

8. Shard with contract calls

   a. Using random generator to pick % nodes that perform one single contract. System contract (SC) will record the participants of that selected contract.

   b. The same contract call following will be directed to that specific shard.

   c. If synchronous call has to be made according to item #b, then commit must be made before that call.



   d. Execution result of contract is not immediately returned by the contract; also, no additional system contract will query the result of Shard contract call result. The execution result is returned by explicit commit.

9. Shard configuration

In the Sharding case, execution result for each contract is stored at each participating nodes in the form of Merkel tree. Each shard can keep growing its Blockchain until explicit commit request is made or predefined commit schedule is reached.

The nodes in the same shard will form some type of consensus. The protocol is very flexible and can be specified by the contract creation settings.

Each SCS is connected to a lower-level consensus node (v-node). The SCS fully trusts the information passed from the v-node.

Each SCS will firstly register itself to the system contract to indicate that it will participate in the smart contract processing. It will need to submit certain amount of bond. This bond will be deducted if any fraud is detected.

For each contract processing, the required bond is *b*. SCS could submit n x *b* if it has enough processing power and bandwidth to support more contract processing. Please note SCS is required to perform synchronous contract call without requirement of bond.

When user submit a contract creation, he will need to specify the following parameters:

Mode: prefer asynchronous or synchronous

Shard percentage: 10%, 20% with minimal shard size

Auto-commit frequency: no. of blocks

One typical contract call workflow is like following:

a. User submits a contract all initialization TX. This TX is processed in v-node. And system contract is notified of this initialization.

b. Node is selected randomly in a deterministic way. The selection algorithm will run at each v-node, thus decide if current SCS node is the part of the selection.

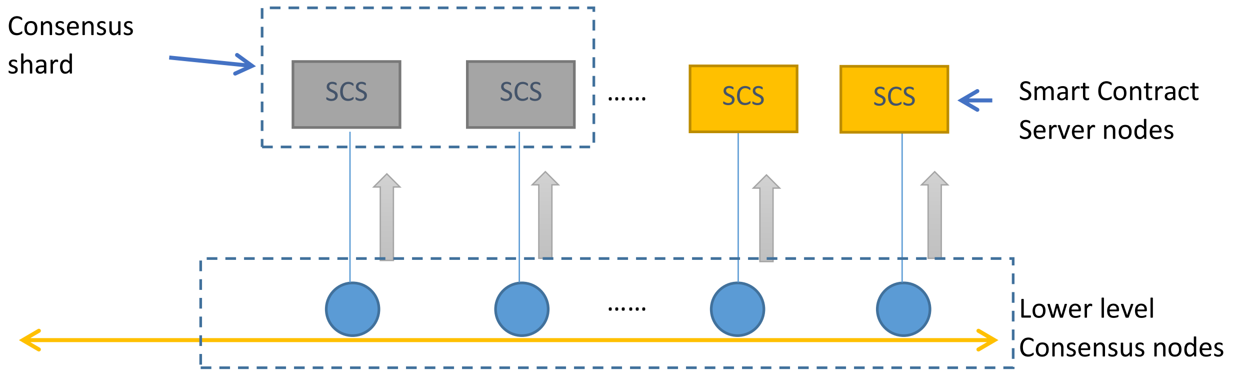
c. Each scs node will then send out enrollment bond to the system contract to confirm its agreement to join the shard. The bond is defined by system. Contract creator could select higher bond requirement if needed.

d. Each selected node will send out broadcast pkt through connected v-node to identify themselves. The broadcast is through lower lever p2p network. This process should be only needed at contract creation time or contract reshuffle time. Please note, one v-node may not know the other's ip address due to multiple-hop connection. So it will prevent direct DoS for smaller shard size.

e. Each node then starts to execute the initialization of the contract and generate local state.

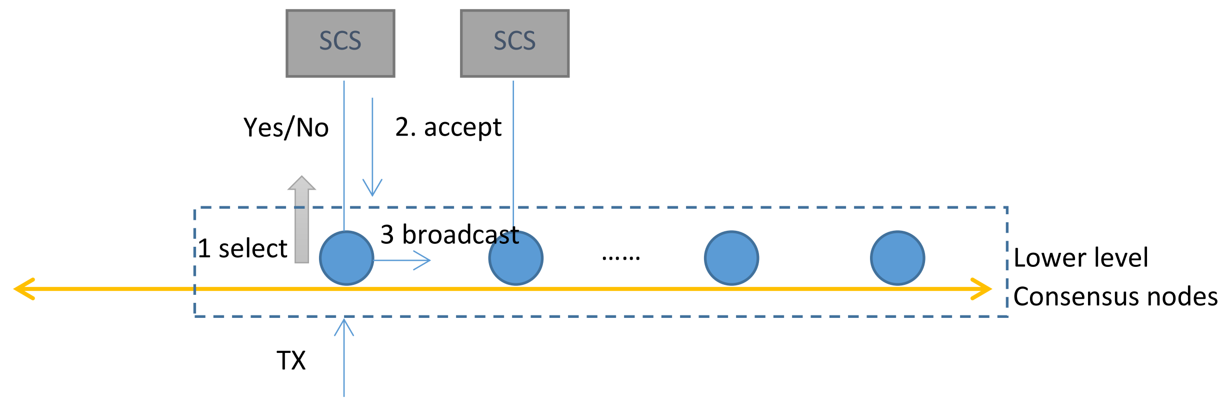
f. Any asynchronous call on the contract will be directed to each SCS node and queued. The order of unexecuted calls will need to reach consensus among nodes. This can be done using POS protocol. And the frequency is totally determined by the incoming contract calls, with some minimal threshold.

Consensus shard



SCS will form a consensus sub chain representing a shard.

Create new shard

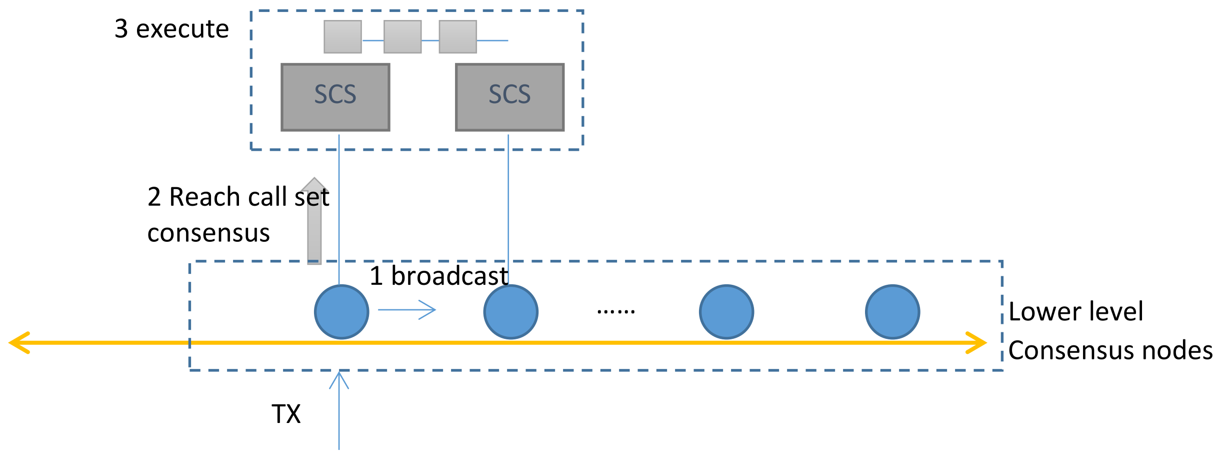


In the first stage of creation, user will submit a TX to initiate a contract creation. V-node will first perform necessary job like address generation, configuration, etc. It will also decide deterministically if connected SCS is selected to process this contract or not. If yes, SCS is notified and verify its participation by authorizing bond to system contract. SCS will also utilize underlying p2p network to broadcast its participation of that contract. The communication pkg includes information like (SCS_ID, ContractAddr, Config). Please note this communication pkg could be combined with other contract creation to reduce transmission pkts. Other selected SCS will perform the same operation. The total pkt needed for this communication is:

$$C = \sum^{k} S$$

k: total # of contracts to perform broadcast. S: shading size.

Contract Function call



User submits contract calls to the v-node. V-node will broadcast to all other shard members through underlying p2p network. V-node will check if current SCS need to execute this call. If yes, it will pass the call to SCS. And SCS will execute it in an asynchronous way.

If multiple contract call reaches v-node in the same block, these calls may arrive in different order for different nodes. In order to make sure all nodes have the same order, consensus is required to make sure each member of shard will have the same tx set.

Query contract status

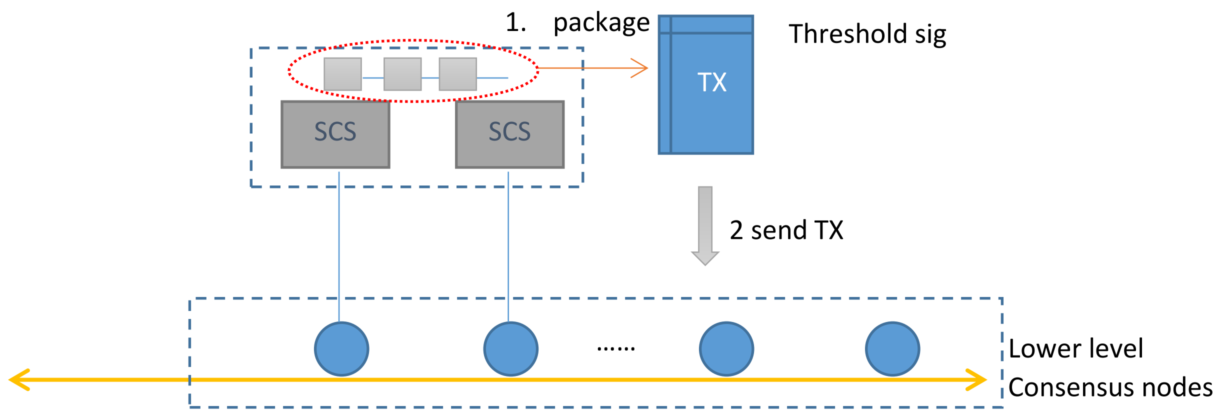
It's slightly different for contract status check for synchronous contract and sharded (asynchronous) contract. In synchronous contract call, all information will be available in the main/root chain. However, the sharded contract all will not have the latest contract state in root chain, rather, it is stored in consensus shard.

The status check for consensus shard is done in two steps:

a. Any node who wants to check the contract will need to register with system contract as the shard watcher. It does not need to execute contract or perform consensus. It just receives the information from shard activity, such as consensus result, block update

b. Query request will be handled by checking the watcher's information regarding the consensus shard.

Pre-Commit

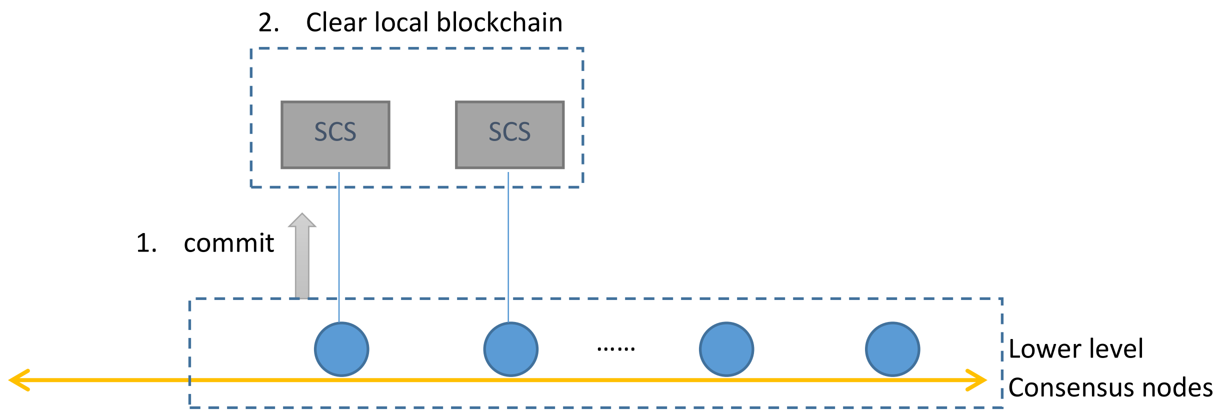


Consensus shard will maintain a Blockchain about the execution of the contract. Periodically, these contract call result in the Blockchain will be committed to the main/root chain.

The pre-commit can be invoked by a contract call to the contract shard. Or, if pre-defined, it can be invoked by system contract call. All honest SCS will send its share of signed TX and broadcast to other shard members. In these cases, consensus shard will reach a threshold signature agreement. And each SCS will package all uncommitted TX packaged inside one Commit transaction. The commit transaction will be sent to its v-node and further broadcasted to the network. Because that commit TX can be verified by any one, it will be eventually included into the root Blockchain.

Once this TX is included in the root Blockchain, v-node will notify its SCS to commit.

Commit

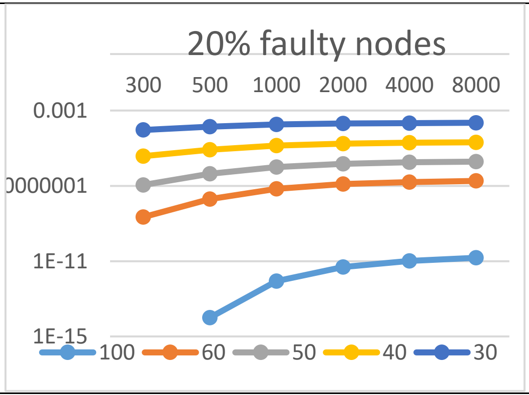


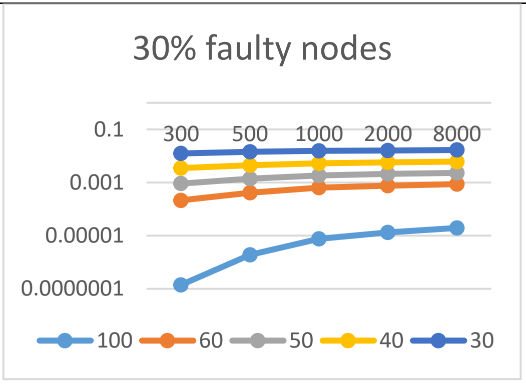
V-node will submit commit command to SCS with proper root Blockchain merkle tree hash. Since SCS will trust its v-node, SCS will clear local Blockchain with the information that already committed into the root chain.

Threshold Signature Usage

We are using threshold signature to sign the consensus TX in the commit stage.

For a typical setup of 1000 nodes, with 30% of Byzantine failure. Each shard is 100 with threshold size 51. The probability of not getting 51 valid signatures are 10e-5.



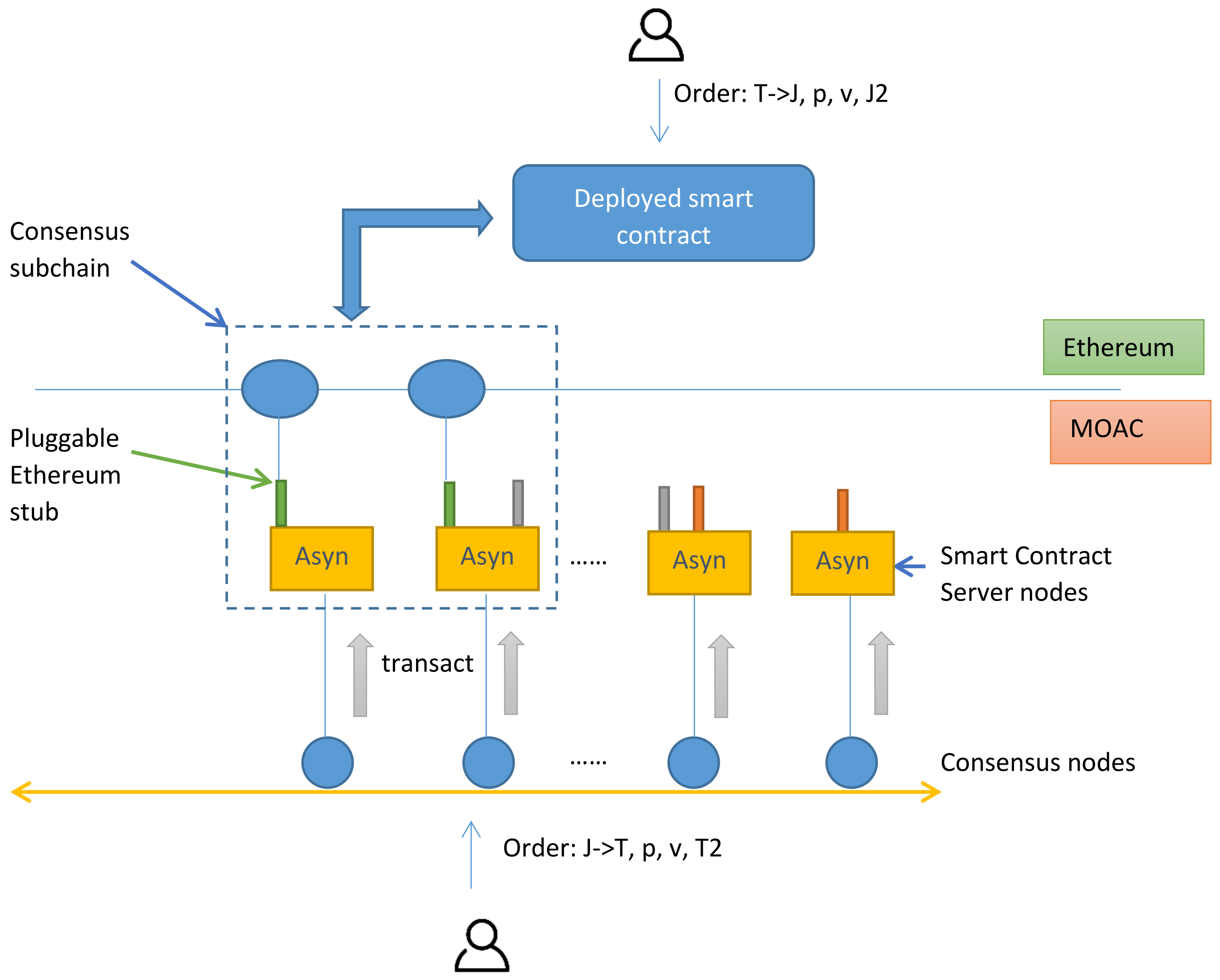


So, the shard size cannot be too small. A minimal size of 100 nodes is required. This requirement should be easily satisfied as MOAC is designed for broader internet applications. Also, the entry barrier of SCS is not very high, and the incentive of SCS processing will encourage more SCS to join the processing pool.

10. Cross chain support

This section will describe the details of how Moac Blockchain and Ethereum Blockchain work together to achieve the atomic exchange between tokens from these Blockchains.

a) Assume in MOAC network, some user places an offer of MOAC token J for exchange of Ethereum token T, with price, volume, and recipient address specified in a form: Order: J->T, p, v, T2.
b) This request is submitted to Smart contract SC1 with proper authorization and thus forwarded to the order book.
c) Some other user in Ethereum wants to sell token T to get J in MOAC with parameters: Order: T->J, p, v, J2.
d) This order is submitted to a deployed contract in Ethereum smart contract SC2 with proper authorization.
e) The asynchronous contract server in MOAC could have pluggable stub connected to Ethereum public chain. These servers will form a consensus sub chain using efficient consensus algorithm. These contract servers will monitor the activity of the Ethereum and add the order to the order book.
f) Validators in MOAC system will find a matching transaction (T<->J). It will generate a proposal { (J1->Js, int(Js->J2), (T1->Ts, int(Ts->T2)), Js is the address of the SC2, and Ts is the address of SC1. Int() is the intention transaction not yet executed.
g) J1->Js will be transacted in MOAC network and T1->Ts will be transacted in Ethereum network. The sub chain will make sure the two transactions happened and verifable. This can be done with the asynchronous call nature. It normally need to wait for some period of confirmation blocks, which user can define.
h) Each contract will wait to see both transaction finished. Then each will finish Ts->T2 or Js->J2 transaction. This is done in deterministic way.
i) If for any reason, step #g could not finish, then contract is able to refund back to original owners.



The role of asynchronous server in MOAC network serves several purpose:

a. Form a BFT sub chain to communicate between MOAC network and other Blockchain system.
b. Initiate smart contract call to the deployed smart contract call in other Blockchain.
c. Monitor and return specific activity to the ordering procedures.
d. Define the logic of sending MOAC based tokens.

Similarly, the contract deployed in Ethereum system will simply define the logic of sending Ethereum based tokens

11. System contract

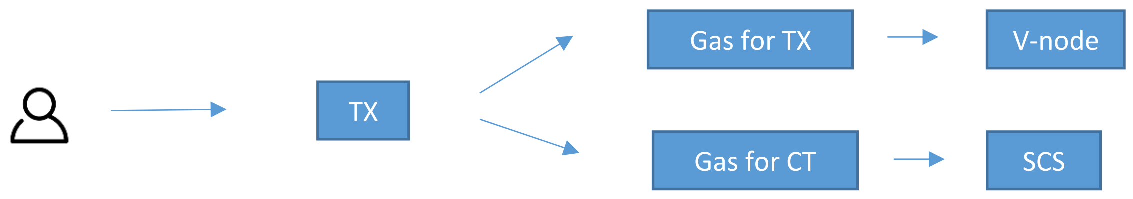
System contract plays a critical role in the workflow. It has following properties:

a) Build-in contract that will be executed first by each node
b) Hold SCS registration information w/ bond.
c) Hold internal TX call stub for asynchronous contract calls if needed.
d) Initiate contract query, pre-commit, commit calls
e) Take gas from TX and distribute gas to SCS participants
f) Handle versioning process based on distributed voting scheme

System contract is executed in each v-node and should reach the same result for each node.

12. Mining and reward

The reward of mining is divided into two parts. One is lower layer that does POW (for now). The other one is SCS that executing contract.



The POW part is very straightforward. Anyone who finds the block will get the block reward as well as the TX gas fee. However, since the asynchronous contract execution is done in the SCS nodes, the corresponding reward will be distributed to the SCSs. So when user submits a transaction, the payment gas includes two parts: one is fix value part for TX, the other is variable value calculated based on contract code. If the Contract is executed in synchronous way, it will be treated as TX and belong to miner who finds the block. If contract is executed in an asynchronous and sharded way, that part of gas will be distributed among the sharding member. In particular, the sharding member who makes consensus of each subchain block will be awarded even share of the gas. The share is recognized and recorded by system contract. It will only be distributed when blocks are commited to the root chain.

In the packaged TX sent by SCS shard will include the vested gas distribution to each shard member.