Ultrain's Multi-chain architecture

By Ultrain Team, March 2019

Scalability is an unavoidable topic for any permissionless blockchain project that is serious enough to seek mass adaption, research on scalability and its implementability has always been one of the top debated topics, the current solutions include on-chain/off-chain scaling, horizontal/vertical scaling, and multi-chain/sharding scaling. We will not compare and contrast these options in this article; rather we will focus on justifying the multi-chain solution we chose for Ultrain's permissionless blockchain. A complex blockchain architecture design requires various trade-offs, for example, between the theoretical complexity and implementability, long-term vision and short-term executability, efficiency and fairness. In this article, not only we documented how we designed Ultrain's blockchain architecture, but also we explained the rationale behinds our decisions.

Overview of multi-chain/sharding architecture

The architectural solution of Ultrain can be considered as one of the on-chain horizontal scaling solutions. The fundamental of the solution is to run all the Dapps on shard-chains, while the main-chain manages and coordinates all the shard-chains. Each shard-chain has a maximum capacity on the hardware resources – when Dapps on the shard-chain reaches the maximum capacity, a new shard-chain will be created to accommodate the new demands, this is also known as horizontal scaling. Each shard-chain comprises a group of Dapps, which is segregated from other shard-chains, as a result of such design, we naturally segregated the computation and storag resources between shard-chains. There will be no user level Dapps running on the main-chain, which its tasks involve users' accounts creation/maintenance, miners' registration/token staking, and scheduling of mining nodes among different shard-chains.

Discussion 1: DO WE REALLY NEED A LAYERED MULTI-CHAIN ARCHITECTURE?

Taking "Quadratic Sharding" architecture of the Ethereum or the infinite scalability of Polkadot's "Hierarchical Scaling" as examples, we understand the fascinating aspect of fractal and recursive design of blockchain architecture, however, we believe through a single-layer multi-chain architecture, we can support tenth of thousands of nodes (single shard-chain: 80-320 miner nodes * 50 shard-chains), such capacity is sufficient for Ultrain to support commercial application on a large scale, for the reasons above, in the first launched version of Ultrain maninet, we opted an easier single-layer multi-chain architecture.

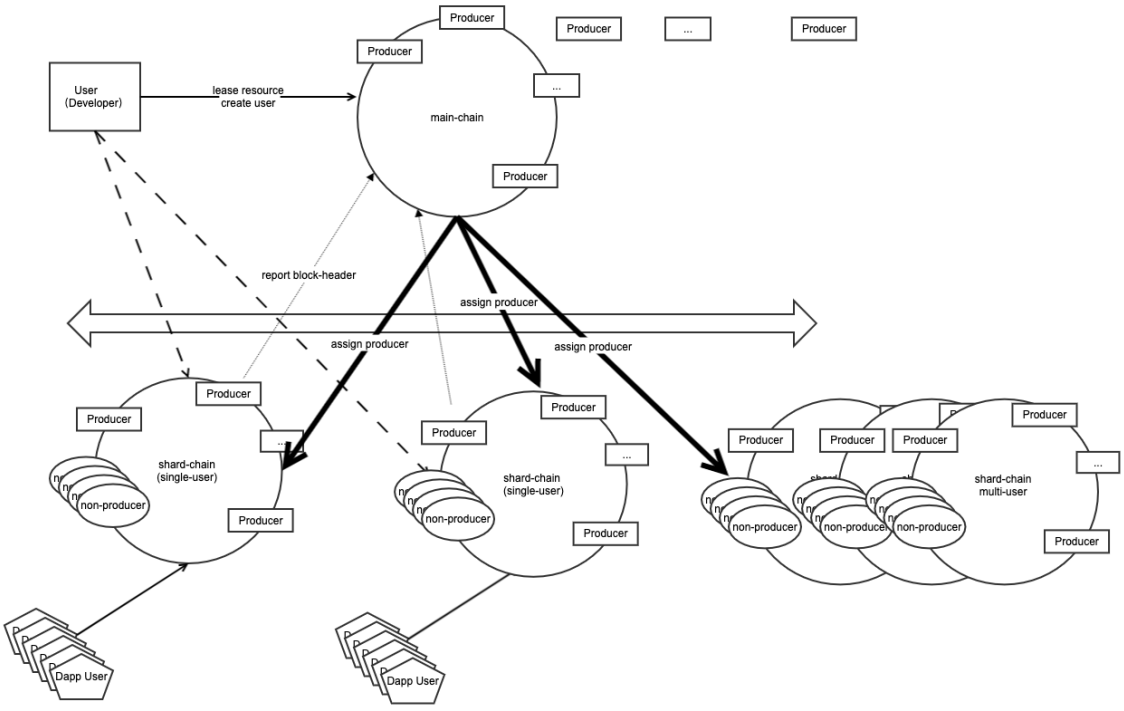
Discussion 2: DO WE REALLY NEED A HETEROGENEOUS MULTI-CHAIN STRUCTURE?

Because we positioned Ultrain as a permissionless, high-performance public chain project instead of an interoperable multi-chain project, we opted the homogeneous structure, this means both main-chain and shard-chains are based on our R-POS (Random POS) consensus mechanism, different shard-chains still follow the same finality assumption (although different shard-chains can have different

confirmation times, their transactions' finality are all deterministic). This structural design simplifies our Inter-Blockchain Communication (IBC) mechanism.

Following is a brief description of the main-chain/shard-chain workflow:

1. User registers on main-chain to be the miner node, during the registration procedure, the user is required to stake native token of Ultrain (UGAS); once registration is successful, miner will be queued for dispatch, the system smart contract on main-chain will randomly assign the miner to a shard-chain.
2. The miner will perform block proposition and validation (mining) and the results are reported to the system smart contract on main-chain, and once the main-chain verifies the validility of the reported block, the block reward will be distributed to the miner.
3. To enhance the network safety, the system smart contract on the main-chain will reshuffle miners to different shard-chains, therefore there will be no miner stay permanently on a particular shard-chain. The random reshuffling of miners is executed by the smart contract, the randomness is verifiable and transparent on the blockchain, the long-term economic return of the miner will not be affected by this.



4. For Dapp users, the registration of account must be performed on the main-chain; the main-chain has the snapshot of all the accounts of the network; when user needs to interact with a Dapp on a particular shard-chain, the user needs to apply for authorisation to access (empower to) the shard-chain, the IBC mechanism between the main/shard-chain will clone the user's account to the designated shard-chain.

5. For Dapp developers, when they want to deploy a Dapp on a particular shard-chain, they need to purchase appropriate resource packages (CPU/Storage/Network) on the main-chain, the purchased packages will be propagated to the designated shard-chain, where the developers can deploy their Dapps.

Discussion 3: WHY RESOURCE PACKAGE?

When we discuss horizontal scaling, we often overlooked the role of the resource pricing model. We believe many permissionless blockchain projects failed to consider this when they designed their resource pricing model, for example, ETH and EOS are both adapting the "overselling model" – with more developers purchase their resource while the overall resource available remains unchanged, each developer/user will have fewer resources for the money they paid. We use "preselling model" – we divide the resource of a shard-chain to a fixed number of equivalent portions, the developer can forecast how much CPU/storage/network power its Dapp needed and purchase accordingly. We would like to use the train ticket analogy to distinguish the "overselling" and "preselling" models, in an "overselling" scenario, when more tickets are sold than the number of seats a train has, all passengers' seat space is reduced. In a "preselling" model, we only sell tickets no more than the total seats available, this ensures everyone's seat space remains unchanged. In essence, the "overselling" model disregards the outcome of inflation while reaping the full benefit from selling the resource packages, the "preselling" model, on the other hand, balances our package purchasers' cost and user experience.

The native token on the Ultrain blockchain is UGAS, it is designed as a utility token to serve the purpose of settlement between the resource user (Dapp developer) and resource provider (miners). For this reason, the use case of UGAS is limited to the main-chain: to be staked by miners, to serve as block reward for mining, to purchase resource package and to pay for smart contract execution. As a settlement tool, we foresee UGAS will be needed when Dapp on shard-chain requires some forms of settlement, we hence designed the cross-chain settlement function between the main-chain and shard-chain, we will cover this in the following chapter.

Miners reshuffling among shard-chains

Multi-chain, sharding and similar horizontal scaling solutions are all facing one security risk – how to maintain the safety of one shard-chain while the network's hash power/staking been diluted. This risk is also known as 1% attack: when we shift from single-chain to multi-chain/sharding architecture, if there are 100 shard-chain, each shard-chain will have 1% the hash power/stake of the original network; then attacking one shard-chain will only need 1% of the original single-chain hash power/stake. Currently there is no known proven solution to this security risk, we believe the practical solution to this risk is to randomly assign (reshuffling) miners across different shard-chains: that means, miners at a given time can only propose and validate block on a shard-chain assigned by the main-chain, and only on this shard-chain the miner can earn block reward; the main-chain reshuffles miners randomly across different shard-chains, therefore for M amount of shard-chains, although each shard-chain's total hash power/stake is only 1/M of the total hash power, it is "difficult" for the adversary to wholly control the 1/M hash power/stake. How do we quantify the "difficulty" level? If we define N = number of committees on a shard-chain, p = % of adversary's hash power/stake versus total blockchain network's hash power/stake. Then the probability of the adversary to effectively control a particular

shard-chain is $\sum_{k=\frac{2N}{3}}^{N} p^k (1-p)^{N-k} \binom{N}{k}$ (In R-POS consensus mechanism, it requires 2/3 committees to reach consensus):

	N=40	N=80	N=160
p = 0.4	0.0004	6.1e-7	6.1e-12
P = 0.33	8.5e-8	3.0e-10	2.2e-18
P = 0.25	1.9e-8	1.6e-15	9.7e-29

We can see as one shard-chain committee reaches 80 members, even the adversary can take 40% of the whole network's hash power/stake, the probability of the adversary to reach 2/3 of the total numbers of committees on a shard-chain is extremely low. Hence under our multi-chain architecture, we set the default starting numbers of the committee to 40, and the recommended committee members to at least 80. Although we can see one shard-chain improves on the security when the number of committees increases, it also increases the number of the miner nodes and subsequently the cost of the network communication, resulting in the reduced performance of the shard-chain and the capacity of the whole blockchain (as the number of available shard-chains reduces). Base on the information above, we recommend within one shard-chain, the desired number of the miner node is between 80 and 160 and should not exceed 320.

We have mentioned previously that data among each shard-chain are segregated, the benefit is that miner nodes only need to maintain the data of one shard-chain at a time, this reduces the on-chain data storage pressure; however, data sync is required when miners are reassigned to a new shard-chain by the main-chain's smart contract. The most common way of data sync on the blockchain is the block replay model –it syncs all the history blocks and replay all the transactions on the shard-chain to build the latest world state for the miner node before it can participate in the consensus process. As the time needed for data sync is proportional to the block height, when the height grows, it can take days for the data sync to complete, this is unfriendly to miner nodes, to tackle this issue, we designed the world state snapshot mechanism:

1. For a fixed period of time, a world state snapshot is taken and stored locally from all mining nodes on a shard-chain, the hash value of this world sate snapshot is recorded on the main-chain.
2. Main-chain's smart contact is responsible to record the shard-chains' hash value of the world state snapshot. Only when the world state is validated by miner nodes no less than 2/3 of the committee members on the shard-chain, the main-chain deems the hash value of the world state snapshot valid.
3. When miner nodes are reshuffled to a new shard-chain, they can retrieve the hash values of recent world state snapshots from the main-chain; when new miner nodes sync with existing miner nodes on the shard-chain for the world state snapshot, the new miner nodes can then know if the world state is, in fact, safe and sound.
4. Sync the world state is far more efficient than sync and replay all the blocks, this subsequently reduces the economic loss from the miner nodes reshuffling.

Discussion 4: WHAT SHOULD BE THE FREQUENCY OF THE MINER NODES RESHUFFLING?

We naturally feel the more frequent the reshuffling the better, for example, frequent reshuffling give adversary little time to prepare the attack, however, we need to make practical considerations: 1) according to table above, even for 40 committee members, adversary nodes controls 40% of the network hash power/stake, then adversary has 0.004% probability to take 2/3 committee members on a shard-chain. This probability is not too trivial to neglect, this means the faster the reshuffling of the miner nodes, the higher the probability adversary fulfills the attacking criteria. 2) there is cost involves when reshuffling the mining nodes, reshuffling frequency versus the time required for the mining nodes to sync with the new shard-chain needs to be optimised to make economic sense. For these two reasons, the mining nodes reshuffling frequency cannot be purely based on theory, but to take considerations of other practical variables. We have observed Ethereum is experimenting "Stateless Client", which enables nodes to "instantaneously" switch among shard-chains without the need of data sync, resulting in hyper-speed reshuffling, we agree this is an innovative idea but far from realisation, we will keep monitoring the progress of this idea.

IBC between main/shard-chain and light client

With multi-chain architecture, we must consider the notion of inter-blockchain communication mechanism. From a designing perspective, we need to consider what is needed to be communicated? To achieve inter-blockchain communication, what native infrastructures are needed by layer 1? And what can be supplied by the library or Dapp on layer 2? Take all considerations into account, we came up with the following design:

1. Both main-chain and shard-chains provide mutually receptive light-client support, meaning main-chain and shard-chains store each others' block header information, they can verify each other and confirm a particular transaction on other's chain indeed happened.
2. Provide an IBC mechanism based on committee voting, for example, when shard-chain A needs to sync particular data from the main-chain, committees on shard-chain A will initiate multi-sig validation toward this data on the main-chain. When no less than 2/3 of the committee members validate, we can consider the data on the main-chain is verified by the shard-chain A. This is an excellent use case of using Oracle (off-chain data to on-chain) in inter-blockchain communication.
3. For the specific type of IBC, the system achieves natively: 1) main-chain's block headers are synced to all shard-chains; 2) each shard-chain's block header is synced to main-chain; 3) reshuffling results of the miner nodes among shard-chains is synced from main-chain to shard-chains; 4) account information on main-chain is synced to shard-chain; 5) the purchased resource package of Dapp developer on main-chain is synced to shard-chains; 6) UGAS can be transferred across the user's main-chain and shard-chains accounts.

Discussion 5 WHY WE DIDN'T EXPLORE INTER-BLOCKCHAIN CONTRACT CALLING SCHEME?

Multi-chain/sharding mechanism with data segregation design works in opposite way to cross-chain synchronous contract calling scheme, because there is neither data, nor code (execution environment) to call; if we must call contract on another shard-chain to provide some form of computation service, we can only do it through the async model, and send the query to the targeted contract/shard-chain, the result should be returned through some form of async waiting. If the consensus protocol doesn't have deterministic finality, then the outcome will be disastrous. Therefore we believe native cross-chain contract calling scheme at present is too complex and there is no intuitively coding paradigm.

We believe similar functionality can be achieved through library provider, relayer service provider, and Dapp developer on Layer 2.

We will then discuss the challenge of designing light client under POS model. Firstly, we need to clarify our scope, which is between chain A and B, chain A needs to confirm transactions on chain B is truthful and vice versa. This means in the transaction-driven blockchain world, chain A needs to confirm a transaction on chain B indeed happened and recorded in chain B's irreversible blocks, then the most intuitive way is to make chain A a full node of chain B; after all, if chain A is a turing complete virtual machine, creating a full node of chain B through smart contract on chain A is theoretically plausible, but in fact, chain A doesn't need to be part of the consensus on chain B, but to validate transactions of chain B, the traditional way is to have chain A maintains the block header information of chain B, rather to sync full block history or world state of chain B, we call chain A the light client of chain B. The question is, what exactly does chain A need to maintain of chain B? Under POW scenario (BTC, ETH), the light client only needs to maintain the targeted chain's block header information to 1) validate the blockchain 2) validate a transaction has been packed to a block (through transaction's Merkle path and Merkle root in blockhead). Under the POS scenario, how does light client maintain the block header data to validate targeted blockchain?

(We assume readers are already familiar with our R-POS consensus design) In our design, the block head header not only contains the block proposer, transaction Merkle, previous hash but also contains two optional data structure periodically:

1. Confirm point: it contains the current committee members' multi-sig on the previous block; as we know, block header in a POW scenario is considered untampered if the nonce is correct; but a POS blockchain's consensus is based on committees' validation (vote), so naturally we should put all the committees' votes for each consensus round in to the block header, but this will consume the storage space on blockchain rapidly (we also face the issue that the committees' votes of the current block can only be recorded to the next block); we periodically choose a block and store the multi-sig vote in the block header; therefore for light client that only maintain the block header, it can and only can validate previous blocks when it receives block header with confirm point. The disadvantage of such method is in a cross-chain scenario, the latency for block header/transaction validation is at the mercy of the interval between two confirm points, this is a trade-off we have to make between the efficiency and storage/computation resources.
2. Epoch point contains the current list of all committee members; epoch point only appears in block header when there is a change to committee members; we have mentioned that confirm point is the aggregation of the multi-sig of all the committee members, but light client doesn't maintain the world state of the targeted shard-chain, how does it know who's the current committee members? It needs to retrieve the full committee member lists from the latest Epoch point.

From above, we can see confirm point and epoch point's design concept is to store the key data of POS to block header: who has become committee members and the changelog of the committee members; which committee members validate the block; Only when the light client has sufficient confidence in the data from the block header, the inter-blockchain communication is truly possible.