Contentos 2.0 Sharding

Version 1.0

Contentos R&D Team, August 2020

1. Background

Contentos is a public blockchain system focusing on digital contents. In addition to regular token transfers, it also supports recording a wide range of content creation and social behaviors. The built-in economic system automatically settles rewards to block producers, dAPP creators, content creators and curators in accordance with the reward plan formulated by the Contentos Foundation. Turing-complete smart contract system and rich contract APIs provide a very broad customization space for marketing. By introducing the novel saBFT consensus algorithm on the basis of DPoS, Contentos has an industry-leading service response speed. The average block finalization delay is 1 to 2 seconds, which provides solid support for smooth user experience of dAPPs.

Contentos Mainnet 1.0 has been running for 10 months since its launch on September 25, 2019. With the contributions of COS.TV, PhotoGrid and other dAPPs, it has served hundreds of thousands of users, recorded more than two million pieces of digital contents, and over 200,000 users have benefited from the system rewards. Contentos Mainnet 1.0 has a stable processing capacity of about 3,000 TPS. The current system load rate is less than 1% but in fast growth. The Contentos team decided to introduce the sharding mechanism in Contentos 2.0 to further expand the system scalability and prepare for greater dAPP traffics in the future.

The TPS problem, or the scalability problem, has always been the most infamous technical bottleneck in the public blockchain world. For example, the average TPS of the most popular blockchain projects Bitcoin and Ethereum is less than 20. Such a low throughput severely limits the applicable scenarios of these blockchain systems.

There are two main reasons for the scalability problem.

- PoW is excellent in terms of security and decentralization, but it relies on heavy calculations.
- Each node in the blockchain network must process all transactions, because the whole transaction history is required by the verification of a new transaction.

Therefore, the throughput of the entire system comforts to the law of minimum, i.e. it is determined by the performance of a single node, and the addition of new nodes cannot improve the overall system performance. For the first reason, consensus algorithms that do not rely on intensive calculations, such as PoS, DPoS, have been proposed by the industry and have gradually become mainstream because of their significant advantages in efficiency. For the second reason, there are generally two kinds of solutions. One is the super node solution, which forces people to run their nodes on high-performance and high-bandwidth machines. Expensive nodes have higher processing capabilities, but also raise the threshold for participation and reduce the level of decentralization. Solana is an outstanding example of this kind. By thorough algorithm

optimization and GPU usage, it managed to increase the TPS to hundreds of thousands. Meanwhile, other projects favored more moderate trade-offs. Super node solutions are just mitigations because the system nature never changes. Another solution is sharding, which distributes the system load to different nodes and each node is only responsible for processing a part of the transactions. This is the typical pattern of a distributed system, in which scalability is achieved. However, sharding is not easy when it comes to blockchains. Many projects, such as Ethereum 2.0, Zilliqa, Ontology, PolkaDot, Cosmos, Harmony and NEAR, have proposed their own sharding solutions, which are great experimentations in this direction.

A node can process a transaction if and only if it stores state data on which the transaction depends. In practical applications, the storage of state data may become a bottleneck earlier than the computing power, because the state data will always increase even if the system load is constantly low. Zilliqa only distributes transaction processing tasks while each node still stores complete state data. Other solutions distribute both processing and state, so that each node stores only part of the state data. State sharding can significantly reduce the storage requirement of a single node, but introduces several cutting-edge challenges in terms of state verifiability, data availability, and data synchronization efficiency.

Contentos Mainnet 1.0 is a DPoS-based super node solution. Although the node criteria is not too restricted, it does exclude most PCs. The goal of Contentos 2.0 is to implement sharding while ensuring both the validity of Mainnet 1.0's data and the continuation of the Contentos reward plan.

2. Overview

Contentos 2.0 uses a dAPP-based sharding strategy.

More than 99% of Contentos Mainnet 1.0 transactions come from dAPPs, such as COS.TV website, COS.TV app and PhotoGrid app. In practice Contentos Mainnet is not a direct-to-user system, but an infrastructure that provides blockchain service for dAPPs. This is the main business feature of Contentos today and in the future. dAPP is an effective grouping of users and transactions. In most cases, a transaction from a dAPP only affects users of the same dAPP. In a dAPP-based sharding system, each dAPP will be assigned to one and only one shard. Each shard is responsible for processing transactions of one or several dAPPs, and only stores state data related to these dAPPs.

The benefits of dAPP-based sharding are mainly reflected in,

- Well clustered state data.
- Better performance due to the small proportion of cross-shard transactions.

The disadvantage is that a dAPP can only be supported by one shard. The system may still find it difficult to afford super dAPPs which require thousands of TPS.

We believe that it would be really helpful for the prosperity and diversity of Contentos ecosystem if we managed to support hundreds of midsized dAPPs using a horizontal sharding solution. For the limitation of support for super dAPPs, we hope to improve that by vertical sharding in the future.

3. System Model

Contentos 2.0 sharding uses a Beacon+Shard structure, which is the same as Ethereum 2.0.

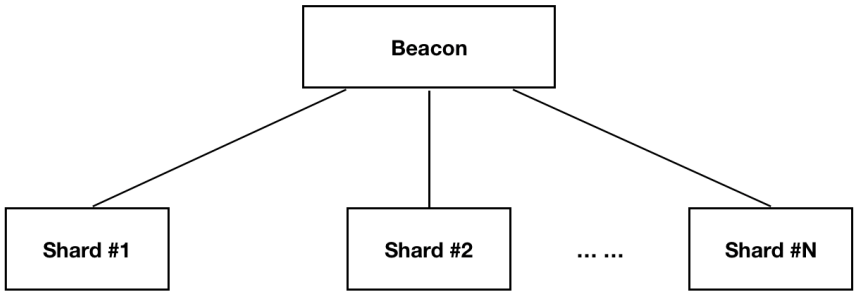


Figure 1, Beacon and Shards

As shown in Figure 1, the Beacon chain and shard chains form a simple two-layer tree structure with the Beacon as root and shards as leaves. All shards are siblings. There's one and only one Beacon chain, while shards are optional and the number of shards is only limited by the capacity of the Beacon.

In Contentos 2.0, all nodes must join the Beacon chain. A Beacon node has the choice to join one or more shards. In other words, if N is a shard node, it is also a Beacon node. N must maintain two sets of state data at the same time, that is, the state data of the Beacon chain and the state data of the shard it is in. Therefore, Beacon's state data is shared by all shards, as shown in Figure 2.

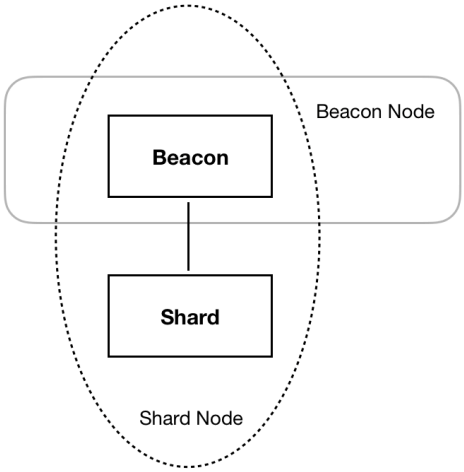


Figure 2, Beacon Node and Shard Node

4. Cross-shard Messaging

Cross-shard messaging is the message passing among Beacon and shards.

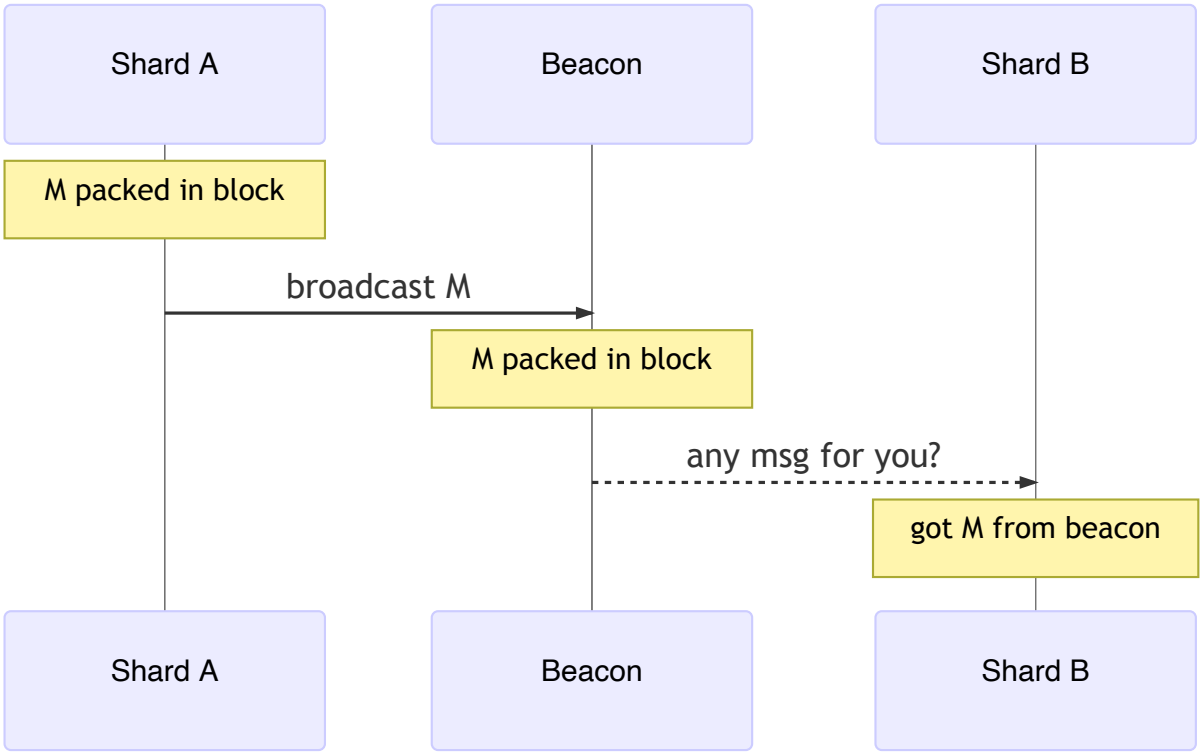
A typical application of cross-shard messaging is token transfer among shards. Suppose that U_a , U_b are users from $Shard_a$, $Shard_b$ respectively and U_a wants to send some tokens to U_b. Since $Shard_a$ does not maintain the state of U_b, it can only complete the deduction of U_a's balance and then send a message $M_{a->b}$ to $Shard_b$. When $Shard_b$ receives $M_{a->b}$, it increases U_b's balance. In this example, $M_{a->b}$ is the key part that guarantees the eventual consistency.

We have mentioned in Section 3 that all Contentos 2.0 nodes must join the Beacon chain. No matter which shard a node belongs to, they can communicate freely because they are all participants of Beacon's P2P network. But P2P messaging may not always be successful due to network jitters. In extreme cases like a short-term network partition, shards could be completely disconnected from each other so that any cross-shard messaging would fail. Howerver, cross-shard messages must be successfully received and processed, which is the fundamental requirement to keep things working. This requires the system to continuously retry until the message is successfully delivered. In a gossip network, a retry mechanism is hard to build because of the lack of network error detection and feedback channels. Most projects in the industry do not implement a retry mechanism, but turn to a client-driven solution, which asks users to initiate retry requests when they find critical inconsistency issues in cross-shard transfers. The client-driven solution provides an opportunity to clean up the mess, but it doesn't help fix the consistency flaw. We prefer a non-user-interactive and more secure system.

Our solution records all cross-shard messages into blocks of the Beacon chain. Take the cross-shard transfer between U_a and U_b as an example,

- Step 1, U_a initiates and broadcasts the transaction T_a in $Shard_a$.
- Step 2, A block producer of $Shard_a$ produces block B_a, which contains T_a. Since T_a is a cross-shard transfer, the block producer will construct a message $M_{a->b}$, telling $Shard_b$ that the balance of U_b should be increased. $M_{a->b}$ is also included in B_a.
- Step 3, When B_a reaches finality, all cross-shard messages it contains, including $M_{a->b}$, are broadcast to the Beacon chain.
- Step 4, All candidate block producers of the Beacon chain who receives $M_{a->b}$ will store it in a buffer queue, which is the data source for future blocks. Eventually, $M_{a->b}$ is packed into B_{beacon} by one of the producers.
- Step 5, When B_{beacon} reaches finality, each shard node searches the cross-shard messages contained in B_{beacon} for its own interests. Candidate block producers of $Shard_b$ find $M_{a->b}$ and store it to their buffer queues.
- Step 6, A block producer of $Shard_b$ packs $M_{a->b}$ into B_b, and broadcasts B_b in $Shard_b$. Each node of $Shard_b$ who receives B_b updates U_b's balance in response to $M_{a->b}$.
- Step 7, B_b reaches finality and the cross-shard transfer is done.

Here is a simple diagram,



In the above procedure, the cross-shard message $M_{a->b}$ will be stored in the buffer queue of each block producer. A producer always makes its best to pack buffered messages into blocks so that $M_{a->b}$ gets maximum chance of delivery. In fact $M_{a->b}$ will always survive as long as the communicating shards are not completely disconnected at the moment $M_{a->b}$ is sent. The whole procedure can be broken when a network partition happens to prevent $M_{a->b}$ from being received by any Beacon producers in Step 4. In such situation, nodes of $Shard_a$ will discover the problem easily because $M_{a->b}$ is supposed to be packed into a Beacon block but it's not. Any node of $Shard_a$ can try fixing the problem by automatically broadcasting $M_{a->b}$ to the Beacon chain again after a predefined timeout. The re-broadcast is repeated until the network recovers and $M_{a->b}$ is finally processed.

The cross-shard messaging of Contentos 2.0 guarantees the eventual consistency, which we believe should be respected by any sharding system.

4.1 saBFT Compatibility

Sharding is essentially a multi-chain collaboration system. Each chain participating in the system must be able to reach the finality state, otherwise the consistency will be at risk. Taking cross-shard transfer as an example, the entire transfer process has two steps, U_a pays and U_b earns. The two steps are performed asynchronously on two chains $Shard_a$ and $Shard_b$. At some point after the transfer is done, $Shard_a$ applies a fork switch and rolls back the transaction so that U_a's tokens come back. In order to ensure data consistency, $Shard_b$ must also roll back U_b's balance. If U_b has spent these tokens or transferred them to other shards, these subsequent transactions must be rolled back too. The chain effect makes the problem extremely complicated and unsolvable with high probability. That's why finality is required.

saBFT is a novel consensus protocol used in Contentos Mainnet 1.0. It provides block finality as other BFT family members do, but in a different way. Forks are allowed and actually quite common in saBFT. It doesn't matter at all for a mono-chain system like Contentos Mainnet 1.0, but it is somewhat dangerous for sharding. Fortunately we only need the irreversiblity of the exchange data across shards, which can be guaranteed if all cross-shard messages are created and delivered at moments of block commitments. Therefore, saBFT is safe to be deployed in both Contentos 2.0 Beacon chain and shards. We can still benefit from its low finalization lantency without introducing any risk to cross-shard transactions.

4.2 Limitation

A cross-shard transaction always contains several operations that will be done in different shards. An asynchronous relay framework can't support cross-shard transactions in which a successor operation could fail and thus require cancelling its predecessor. It's simply because the predecessor operation has been successfully executed and finalized. Cross-shard token transfers are easy because the addition of receiver's balance is unconditionally achievable.

Contentos 2.0 will only support cross-shard token transfers and stakings. Cross-shard contract calls are not supported.

4.3 Cross-Shard Message

Cross-shard messages are messages passed among the Beacon and shards. The general form is defined as,

$M := (src, dst, type, reason, data)$

where src and dst are the sender and receiver shards respectively. The Beacon can also be regarded as a special shard. $type$ is the message type and indicates what kind of operation dst should take. $reason$ is why src sends the message. $data$ is customized user data that should be parsed according to $type$. Some cross-shard messages defined by Contentos 2.0 are shown in the following table.

type	reason	data
TRANSFER	ID of cross-shard transaction	receiver account and amount of tokens
STAKE	ID of cross-shard transaction	receiver account and amount of tokens
REWARD	(triggered by Beacon economic system)	amount of the reward
VALIDATOR	information of the block at which validator set changes	change of validator set

5. Account System

A user must create an account on the Beacon chain before he/she can join a shard. The Beacon account system works in the same way as Contentos Mainnet 1.0. Each account has its own name, cryptographic key, COS balance, staking VEST, etc. Once the user has a Beacon account, he/she automatically has accounts on all shards. The name and cryptographic key of the shard accounts are the same as the Beacon account, but the shard asset balances are initialized as zero. Even if a new shard is created later, the user will automatically have an account on the new shard.

Contentos 2.0 uses a lazy mapping way to deal with the shard accounts. By default, a shard account is just a mapping of the Beacon account, which means that the shard account is generated in memory everytime it's needed. Not until the user changes his/her shard assets is a real shard account record created in the shard state database.

Users can transfer assets from the Beacon chain to the shard and vice versa, which is very similar to the cross-shard transfer process described in Section 4.

6. Beacon Chain

The Beacon chain is the main chain of Contentos 2.0. It provides shards management and cross-shard message relaying. Establishing the Beacon chain is one of the main goals of Contentos 2.0.

The infrastructure layer of the Beacon chain is almost the same as the current Contentos Mainnet 1.0.

- DPoS + saBFT consensus algorithm.
- Dual identity account model based on user name and cryptographic key.
- Token transfer and staking.
- The resource limitation model of CPU and bandwidth.
- Block producer election procedure.
- Smart contract virtual machine.
- The release-by-block mechanism for rewards.

The differences are mainly reflected in,

- No social features such as content publishing, comments, likes, and following.
- Cross-shard messages involvement in blocks.
- Shard management features to create, suspend, join or exit a shard.
- Different reward distribution among the Beacon and shards.

6.1 Shard Management

A shard can be defined as,
$shard := (name, state, validators, init\_stakes, stats, criteria, rewards)$,

where $name$ is the display name and the unique identifier of the shard. $state$ is the status of the shard, which can be one of $Inactive$, $Active$ and $Suspended$. $Inactive$ indicates that the shard has not started, $Active$ indicates that the shard is running, and $Suspended$ indicates that the shard has been suspended. $validators$ is the validator set of the shard with each $validator := (name, pub\_key)$, where $name$ is the validator's account name and pub_key is the

public key. $init\_stakes := \{name : stakes\}$ records the amount of tokens staked by $validators$ before the shard starts. $stats$ contains a bunch of statistical numbers of the shard. $criteria$ is the minimum requirement of a running shard, including the minimum number of validators, the minimum staking of a single validator, the minimum total staking, etc. $rewards$ is the proportion of system rewards the shard deserves.

Contentos foundation holds a privileged account for shard management , which has the authority to create, update, start or suspend shards. After a shard is successfully created, all accounts on the Beacon chain can join or exit the shard's validator set.

Shard Creation

The privileged account initiates the shard creation transaction $T_{create} := (name, criteria, rewards)$, specifying the name, running conditions, and reward ratio of the new shard. The system will create a new shard record in response to the transaction,

$$shard := (name, state = Inactive, validators = \emptyset, init\_stakes = \emptyset, stats = \{0\}, criteria, rewards)$$

Shard Updates

The privileged account initiates the shard updating transaction $T_{update} := (name, criteria, rewards)$, specifying the updated running conditions and reward ratio of the shard. If the shard is already in the $Active$ state but does not meet the updated running conditions, it will automatically switch to the $Suspended$ state.

Shard Startup

The privileged account initiates the shard startup transaction $T_{start} := (name)$, specifying the name of the shard to start. If the shard meets the running conditions and is not in the $Active$ state, it will switch to the $Active$ state and start running. In the genesis of the shard, the system will register the validator set for the shard, and send validators their VESTs in accordance with $init\_stakes$.

Shard Suspension

The privileged account initiates the shard suspension transaction $T_{suspend} := (name)$, specifying the name of the shard to suspend. If the shard is in the $Active$ state, it will switch to the $Suspended$ state and stop the service.

Note that shards could also be suspended automatically in case of insufficient number of validators or insufficient amount of staking, so that the running criteria is no longer met.

Shard suspensions are considered very critical events. Aftercare works will be done by Contentos core team.

Validator Registration

Any account can initiate a validator registration transaction $T_{register} := (name, stakes)$ to join the shard, specifying the shard name and the amount of COS to stake. If the shard is in the $Inactive$ state and $stakes$ meets the shard's $criteria$, the initiator account becomes a validator. $validators$ and $init\_stakes$ of the shard will be updated to record the new validator.

For shards that are already in $Active$ state, a user should register himself as a validator in the shard chain insead of the Beacon.

Validator Unregistration

A shard validator can initiate an unregistration transaction $T_{unregister} := (name)$, specifying the name of the shard to exit. If the shard is in the $Inactive$ state, the system will remove the validator account from $validators$ and return his previously staked COS according to $init\_stakes$.

For shards that are already in $Active$ state, a validator should unregister himself in the shard chain insead of the Beacon.

The VALIDATOR Message

All nodes of the Beacon chain must know the validator set of each shard, so that they can verify the digital signatures along with messages sent from the shards. Therefore, each shard must report its validator set changes to the Beacon via VALIDATOR messages. After receiving the message, the Beacon updates $validators$ of the sender shard. Although shards also need the Beacon's validator set, there's no need for the Beacon to send messages to them, because shard nodes can fetch that directly from the state database.

6.2 Reward Distribution

Like the Contentos Mainnet 1.0, the Beacon chain also has a built-in economic system, which conforms to the Contentos' reward plan of releasing 3.5 billion COS tokens over a 12-year term.

After each new block is produced, the economic system will mint a certain numbr of rewards and deposit them into the prize pool. In addition to awarding itself, the Beacon chain distributes rewards to all shards in the following way,

- Step 1, Each time a Beacon node completes the block commitment, it calculates the amount of rewards that each shard deserves since the last commitment and packs the result into a cross-shard message R of REWARD type.
- Step 2, Beacon nodes store R in their buffer queue.
- Step 3, R is packed into a Beacon block B.
- Step 4, When B is applied on the Beacon chain, the total reward of R is deducted from the Beacon's prize pool.
- Step 5, After B reaches finality, nodes of shard S store R in their buffer queue.
- Step 6, R is packed into a shard block B'.
- Step 7, When B' is applied on shard S, prize pool of S receives the reward recorded in R .

7. Shards

Shards are dAPP carriers that actually deal with digital contents and social behaviors. In the framework of Contentos 2.0, current Mainnet 1.0 will be reconstructed into one or more shards according to dAPPs running on it.

Shard chains inherit the DPoS + saBFT consensus algorithm and all features of the Mainnet 1.0. In order to cooperate with the Beacon, shards must also,

- Implement a lazy mapping account system.
- Handle all kinds of messages from the Beacon.
- Send various messages to the Beacon at the right time.
- Refactor its reward system so that all kinds of rewards must come from a single prize pool. Real time settlements, such as block producer rewards of Mainnet 1.0, need to be re-designed.

In fact, a shard will work as long as it implements the collaboration interface with the Beacon. Shards can be very different from each other in terms of consensus algorithm, business features and reward distribution scheme.

8. Conclusion

In this paper, we outlined the sharding framework of Contentos 2.0, which is a Beacon + Shard tree structured system model with a dAPP-based sharding strategy. Without covering all the details, it focused on a few most important components,

- A cross-shard messaging scheme that ensures eventual consistency.
- Feasibility analysis of saBFT protocol in a sharding system.
- A user friendly design of the account system.
- Shard management on the Beacon chain.
- A reward distribution scheme in compliance with Contentos Reward Plan.

We'll keep studying and make changes on this paper when necessary.