KHIPU

Towards an Enterprise-Level Blockchain Platform

Khipu Foundation

V 1.0

CONTENTS

FOREWORD

For years, we have been eagerly looking forward to a free, open and trustworthy distributed system that can be used to realize or refactor enterprise-level applications. For the first time, Bitcoin has given us a rare glimpse of what a truly free, open and trustworthy distributed system may look like. Since then, it has grown into a new distributed computing ecosystem called blockchain through the expansion of Ethereum and EOS, promising revolutionary changes to the industrial landscape.

From Satoshi Nakamoto's Bitcoin to Vitalik Buterin's Ethereum, exactly ten years have passed. Over this period, blockchain technology has evolved from a small-scale encrypted currency experiment to a large-scale distributed application involving tens of millions of participating accounts every day.

At the same time, enterprise application architectures, from several mainframes carrying enterprise businesses in the past to the advent of parallel computing and large-scale distributed clusters, also provide services for billions of customers each day.

As far as computing technology is concerned, blockchain represents a typical distributed application. Its core design elements essentially incorporate the **"event sourcing + state snapshot"** model, which is increasingly used by enterprise applications to expand into distributed systems.

From the perspective of distributed computing, we define blockchain as：

An event sourcing-based, peer-to-peer, trustworthy and distributed sharded cluster with unreliable but fully redundant nodes in a WAN environment.

From this definition, we can see that blockchain is actually an approximation of the ultimate form of distributed computing. For the first time, Bitcoin allows us to see such an "ultimate" distributed system in operation.

For those engaged in enterprise application development and familiar with Domain-Driven Design (DDD), a deep understanding of the blockchain will find its similarity with the DDD model in almost every aspect. This naturally leads to the following conjecture: in theory, most enterprise applications can be refactored to the blockchain, while DDD is the best approach for the task.

Of course, there is something much more than that. For enterprise applications, refactoring to the blockchain clearly brings in an all-new perspective. It involves the adoption of technologies related to the above definition, including but not limited to cryptography, point-to-point networks, consensus algorithms to implement or refactor enterprise applications into free, open and trustworthy distributed systems.

From this, we can also draw a more meaningful conclusion: from a social and economic perspective, it will be possible for anyone to freely participate in an enterprise-level service in the future, with no permission required or restriction imposed. This no doubt represents a tremendous change.

Chapter 1 Enterprise-Level Blockchain - Design Model

When analyzing the design of blockchains, we firstly notice its good fit with the DDD model for modern enterprise applications. The blockchain design precisely embraces the core elements of DDD, such as Event-Sourcing, CQRS (Command and Query Responsibility Segregation (CQRS) and Aggregate. These core elements are the right way for distributed computing.

1.1 DDD for Enterprise Applications

1.1.1 Event Sourcing[1]

Martin Fowler summarizes Event Sourcing as collecting all changes in an application state into a series of events. This implies that we do not have to intentionally store the state (such as account balance) of a storage entity object at a certain point in time. Rather, we store a series of events that affect its state (such as a series of transfers that lead to changes in the account balance). To query the state of a certain time point, we can reconstruct the state snapshots of the time point by sequentially replaying a series of events related to that state up to the time point (such as calculating the balance real time through a series of transfer events, instead of reading some data directly from somewhere).

An object will go through many events from creation to extinction. Under the development model of the past, every time an object participates in a business action, it will persist the latest state of the object into the database. The data in the database always reflects the latest state of the object. Event sourcing is just the opposite. It does not save the latest state of the object but every event that the object has experienced. All events that affect the change of state will be stored in the database in chronological order.

A system adopting event sourcing keeps all the experienced events that describe the whole lifecycle of an object, which are the most complete and objective information. It can restore the state and scenarios of any point at any time by event replay at any time, thus better guaranteeing system certainty and consistency (different people can reproduce the same result at every replay).

---

[1] "Event Sourcing - Martin Fowler." 12 Dec. 2005, https://martinfowler.com/eaaDev/EventSourcing.html. Accessed 7 Nov. 2018.

So how does an event actually affect the state of a domain object? When a business triggers an action of a domain object, the domain object will firstly generate an event, then the object will respond to the event and update its own state, while persisting in the event.

To re-establish, copy or transfer an object, we only need to create an empty object first, and then sequentially replay all the events related to the object to get exactly the same object (and state). This process is called event sourcing.

On the other hand, because the state of an object is represented by events and events only increase in numbers but make no modification, this makes the data representing the object in the database very stable, and there can be no operations such as DELETE or UPDATE. An event represents a fact, and no fact can be erased or modified. This feature makes the domain model very stable and causes no problem of concurrently updating the same data at the database level.

Of course, event sourcing does have its downside. When there are many events, replaying them can be a time-consuming process, and querying the state of a certain point in time can become very troublesome, because events have to be replayed to rebuild the state of the time . On these occasions, snapshots can be used. It is not necessary for the system to create snapshots for every change. Instead, it can create snapshots periodically. When making state query, you can get the desired state by replaying a few events that have occurred after the snapshot time.

A system which is designed according to event sourcing can be summarized as follows:

- The whole system is event-driven, and all businesses are completed through event driving;
- Events are first-class citizens, system state changes are driven by events, and events need to be stored in some kind of storage in an orderly manner; and
- The state is a kind of event-driven view that does not necessarily need to be saved to the database

1.1.2 CQRS[2]

CQRS (Command Query Resposibility Segregation) segregates the read operation (query) from the write operation (add, delete, change command), which not only makes the logic clearer but also allows their respective optimization. There are multiple ways to realize CQRS, including:

- Segregate the write from the read operation of a database
- Distinguish nodes responsible for handling commands from those for making queries in a network (cluster)
- Store event sequences and phasal state snapshots on the command side; and create the latest state of the saved object for query on the query side, especially in memory as much as possible.

1.1.3 Aggregate[3]

In DDD, an aggregate is a set of domain objects. It is the smallest unit of multiple state/data changes and is encapsulated as true invariance. Changes/transactions in an aggregate must maintain atomicity. In other words, strong consistency within the aggregate is required, while final consistency between aggregates will be achieved. As an aggregate, it can have a unique ID to facilitate association and reference between aggregates.

Simply put, the relationship between aggregate, entity and value can be expressed as follows:

- An entity has an ID, lifecycle, and state (described by value). Entities use IDs to distinguish their instances.
- An aggregate root is an entity. The ID of an aggregate root is globally unique, but the IDs of entities under the aggregate root only need to be unique in the aggregate root.
- Value has no life cycle. It is essentially a value. Its equivalence is judged by whether the value is the same, and value is used to describe the state of an entity.

---

[2] "CQRS - Martin Fowler." 14 Jul. 2011, https://martinfowler.com/bliki/CQRS.html. Accessed 7 Nov. 2018.
[3] "DDD_Aggregate - Martin Fowler." https://martinfowler.com/bliki/DDD_Aggregate.html. Accessed 7 Nov. 2018.

For example, an order can have more than one detail, while order details cannot exist without an order, and orders cannot be without details. In such a situation, the order and order details are an aggregate, while the order is the aggregation root of the aggregate. The order and order details are within the boundaries of the aggregate.

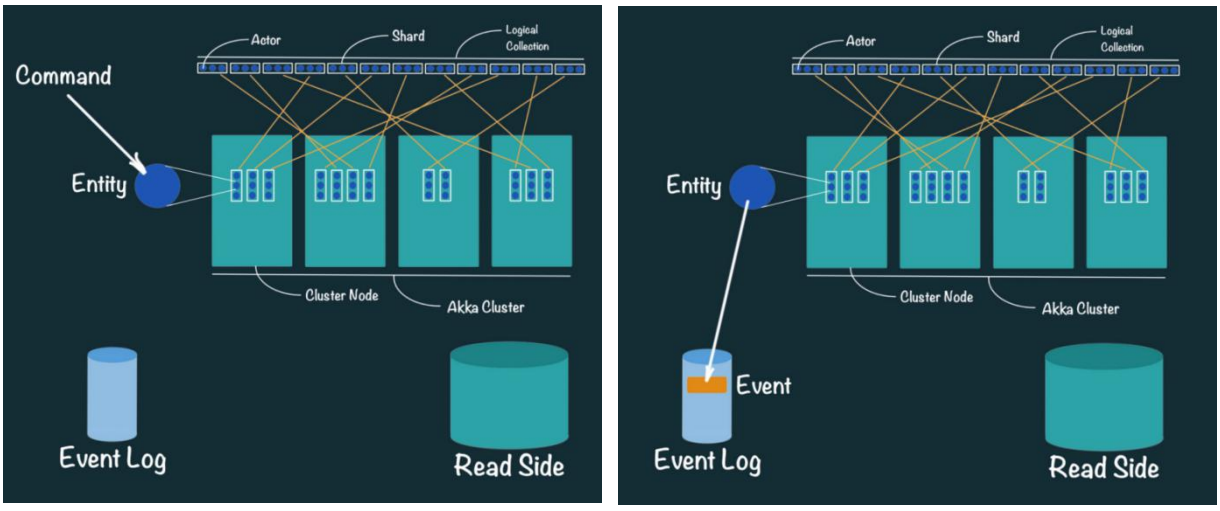
Data modifications inside an aggregate are of ACID strong consistency, while cross-aggregate data modifications are of ultimate consistency. By following this principle, we can minimize concurrent conflicts and maximize the throughput of the whole system.

Meanwhile,aggregates use invariance to enforce their own data consistency/integrity. Invariance is a simple rule. No matter how the demand changes, there are always several parts connected at the same time, either changing or remaining unchanged.

The concept of "aggregate" points to the key aspects when we refactor enterprise applications to the blockchain and design domain entity models.s.

As a distributed infrastructure for centralized services, Akka is a typical service designed according to the principles of Event Sourcing, CQRS and Aggregate (Figure 1).

Figure 1 Event Sourcing and CQRS in an Akka Cluster



Source: How Akka Works: "Akka A To Z", An Illustrated White Paper

If the states and events are truly known, it is theoretically possible to replay them on any computer from the genesis to any point in time and ensure that the results are the same. Then, how to confirm and disseminate real states and events? For centralized systems, this is easy to do, but with the drawback of no longer requiring that anyone can make the replay. For scattered peer-to-peer network systems, the task is difficult . But, the blockchain has taken an important step forward in this direction.

"Event flow + state snapshot + aggregate" constitute the essential model for us to understand and simulate the world. Enterprise applications designed and implemented according to this model are also the closest to the nature of the system, which makes it easy to implement concurrent and distributed applications through business expansion and implementation. Blockchain precisely follows this model.

1.2 DDD Blockchain

Blockchain is blockchain because it possesses the following key features:

1. Because all events from genesis are sequentially kept, it is possible to relay and validate the state of any point in time on any machine; it is also possible to replay all subsequent events from the (state) snapshot of a certain time point.
2. It ensures that every event is unforgeable and tamper-free (encryption, signature) and that the sequence and logic of the events are consistent.
3. It ensures that no event or playback logic is lost. For example, events are synchronized and stored on numerous peer nodes, and those nodes whose events should follow are specified (consensus).
4. Token becomes a necessary condition for the blockchain to prevent its abuse without a cost (which is in fact an attack).

These features can be more specifically mapped to the various elements of the DDD model.

1.2.1 Command - Transactions

The command is Tx, the core of which lies in data transfer and increase/decrease. The code contained in Tx sends out a series of commands.

1.2.2 Event Sourcing - Chained Blocks

Blockchains sequentially retain all events from genesis, while allowing the replaying and validation of the state of any point in time on any machine. It is also possible to replay all subsequent events that happen after the snapshot of a point in time (state).

1.2.3 Time - Block Number

The block number is time, which can be seen as the timestamp under the key scale of the blockchain. The sequence number of tx in a block can be regarded as the timestamp of a finer scale under the block number. These timestamps are identified and sealed by the consensus mechanism, and become consistent timestamps in a distributed environment. For the state (account balance code, etc.) of all specific moments, all their "moments" correspond to this timestamp (block number - Tx sequence number).

1.2.4 Entity and State

All important entities are associated with an Address. The most important entities under the Address include Account, as well as Contract (Code) and Storage. Account, Contract, and Storage all have historical states, which, as mentioned earlier, are snapshots of the Timestamp (block number - TX sequence number) moments.

Ethereum executes and packages a batch of codes at certain intervals, which can be seen as the state snapshots extracted after running the world for a short interval, and the block number is precisely the timestamp.

1.2.5 Everything is Hash

Instances of all entity objects are identified by their (value) hash and can be referenced by hash only. Even the Address, the only stable identifier, is itself a hash. In the philosophy of blockchain, everything in the world is changing all the time. For entities at different times, such as Account, Contract, Storage, etc., the value of their instances is changing. As a result, their identification and Key (value hash) are also changing. The only thing that does not change is the Address, which itself is hash. The Address does not change, but its associated Account needs to be found in the latest Account trie, as it is also the case of Storage.

Account Trie and Storage Trie is like a state map. The states of the world are connected to a verifiable, tamper-free and easy-to-find state map through the nodes of these Tries.

There are several interconnected state maps in the timestamp of Ethereum at any time, including Account Trie, StateStorage Trie, Receipt Trie, Transaction Trie and so on. The change of any node will lead to a series of changes in the corresponding nodes, and finally be reflected in the root node.

Comparing the blockchain design model with the DDD model for enterprise applications, we can see that the two fit with each other. For this reason, the path to refactor enterprise applications to the blockchain is to adopt DDD to analyze and design enterprise application systems, and a "peer-to-peer trustworthy distributed cluster in the WAN environment" is constructed by generally adopting cryptography technology, peer-to-peer networking, event sequencing and sealing, and state snapshot storage and query in blockchains.

While theoretically feasible, do the related blockchain technologies mentioned above have the ability to refactor large-scale enterprise applications?

Chapter 2 Enterprise-Level Blockchain - Implementation Path

Taking Ethereum as a typical case, we will engage in discussions from the aspects of single node computing performance, distributed shard architecture and domain entity model. The reason why Ethereum is chosen as an example is that it is a blockchain implementation that does not make any compromise on the way towards the ultimate form of distributed computing.

2.1 Single Node Computing Performance

Firstly, single node computing performance. The computing performance of a single node determines the performance of a single chain (single world).

The single chain performance of Ethereum is mainly determined by the following three factors:

- Network broadcasting delay
- Time to reach consensus
- Contract execution and validation time by a single node

What are roughly the proportions of these three factors in the average block generation time of approximately 15 seconds?

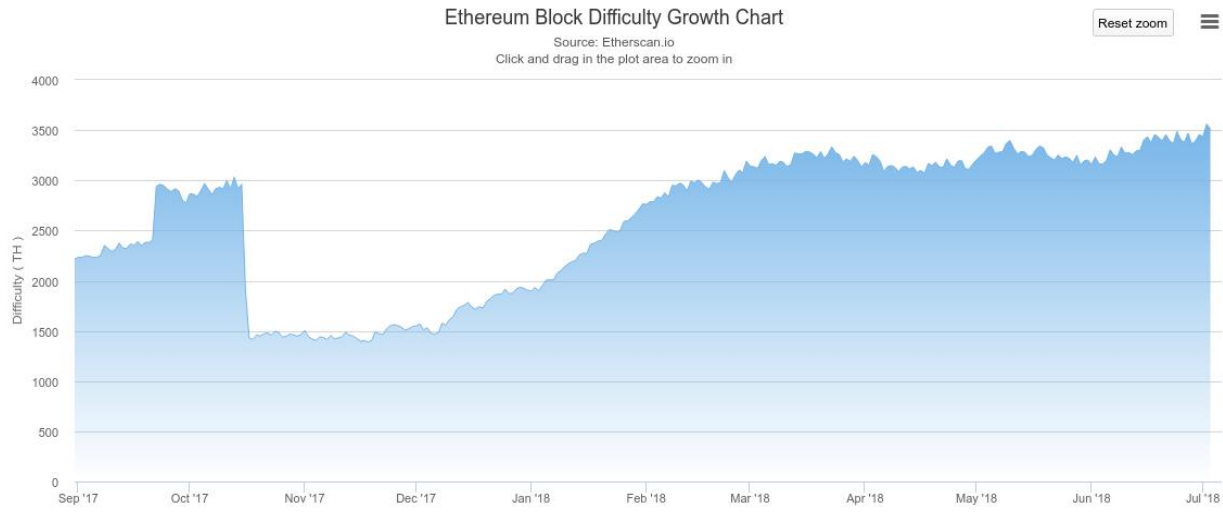
According to multiple studies, in the existing Internet environment, the network broadcasting delay of a 0.1M block can be taken as 1 to 3 seconds. When block size doubles, the delay will increase by 25%-50%. The current block size of Ethereum is less than 33k. Considering the advancement of network technology, it is reasonable to set the delay to 1-3 seconds or even shorter. We can take 2 seconds to make various theoretical estimates.

The time to reach consensus can be adjusted by dynamically increasing or reducing the proof-of-work difficulty coefficient. However, under the PoW consensus mechanism, the proportion of this time cannot be too low. Otherwise, the reliability and fairness of PoW will be undermined by network transmission time and block execution time.

The figure below is the difficulty coefficient curve of the Ethereum Mainnet from September 2017 to July 2018. It is obvious that in the heat of cryptokitties, the time

used by miners to execute blocks increased substantially, and the difficulty coefficient had to be adjusted dynamically to ensure block generation time.

Figure 2 Ethereum Difficulty Coefficient Curve - September 2017 to July 2018



Source: https://etherscan.io/chart/difficulty

Therefore, how many transactions can be processed in 15 seconds will mainly depend on the contract execution and validation time of a single node. In other words, it will depend on how many promptly processable transactions each block can accommodate.

Each transaction in Ethereum is not a simple transfer of account assets (increase or decrease). Rather, it involves the handling of complex logic through intelligent contracts. Such complex logic will call different instructions, or read the stored states, or run certain complex computations (such as SHA). These different instructions consume different computing resources.

The Ethernet VM well brings in gas metrics to measure the resources consumed by each instruction. In a node with typical configurations, the ideal gas metric design should achieve the following: gas consumption per transaction approximates a linear relationship with its execution time so as to ensure that validation is completed within a predictable time. At the same time, by adjusting the gas ceiling of each block, it is possible to adjust the total amount of computing resources or the number of transactions that can be consumed in each block.

Blockchains usually use the simplest transfer transaction to theoretically estimate the number of transactions per second (TPS). The block gas ceiling of Ethereum is currently 8 million gas, while a simple transfer consumes 21,000 gas. Therefore, the maximum number of transactions in each block is 800/2.1 = 381. With a block generation time of 15 seconds, the maximum TPS is 381/15 = 25.

Therefore, in the current Ethernet architecture, increasing TPS means raising the gas ceiling of blocks. This involves the proportion of block validation time in the 15s block generation time for most nodes in the network, especially the mining nodes. If this time is too long, the delay of forwarding (broadcasting) new blocks by other nodes in the network after validation will get longer, the probability of uncle blocks appearing on the chain will be higher and the price for reaching the irreversible determinacy of blocks will also be higher. We may therefore assume that the block execution time/ block generation time ratio should be limited to less than one fifth, or 3 seconds.

Based on the above assumption, we get the upper limits of TPS which a single chain in Ethereum can reach (Table 1).

Table 1. TPS Upper Limits of Ethereum Under Different Computing Power (Mgas/S) and Execution Time Proportions

$$\boldsymbol{TPS = (100 \times mgas \div 2.1 \times duration) \div 15}$$

Block execution time & its proportion of block generation time	1s 7%	2s 13%	3s 20%	4s 27%	5s 33%	10s 67%	12s 80%	15s 100%
1 mgas/s	3	6	10	13	16	32	38	48
2 mgas/s	6	13	19	25	32	63	76	95
3 mgas/s	10	19	29	38	48	95	114	143
5 mgas/s	16	32	48	63	79	159	190	238
8 mgas/s	25	51	76	102	127	254	304	381
10 mgas/s	32	63	95	127	159	317	381	476
15 mgas/s	48	95	143	190	238	476	571	714
20 mgas/s	63	127	190	254	317	635	762	952
25 mgas/s	79	159	238	317	397	794	952	1,190
30 mgas/s	95	190	286	381	476	952	1,143	1,429
60 mgas/s	190	381	571	762	952	1,905	2,286	2,857
120 mgas/s	381	762	1,143	1,524	1,905	3,810	4,571	5,714

Table 1 also lists two extreme cases:

- One is that after a PoS-like consensus mechanism is adopted, the time to reach consensus is close to zero, in which case only the scenario for network broadcasting time needs to be reserved. Suppose that the block execution time can be relaxed to 80% of the block generation time (12 seconds out of the 15 seconds – leaving 3 seconds for network broadcasting). This scenario is precisely like the current implementation of EOS (blue).
- The other is that the block execution time equals the block generation time. This scenario is actually a completely centralized scenario in current enterprise applications (green).

It is clear that for Ethereum to achieve higher TPS, such as from 25 at present to the order of 100, the capacity of gas execution per second on a single node should reach more than 10 mgas. To reach the 1,000 level, a 120 mgas capacity will be required.
So, what is the block execution limit on a single node in Ethereum? Where is the bottleneck?

As the first step to enable blockchains to provide enterprise-level application capabilities, Khipu's first goal is to answer these questions.

2.1.1 Khipu's Exploration of Ethereum's Single Node Capacity Limits

As mentioned earlier, TPS is now Ethereum's main bottleneck for enterprise applications. One way to increase TPS is sharding. For example, at the current processing capacity (the theoretical value of TPS is about 25), 100 shared chains may reach a TPS of 2,500 at the cost of 500 endorsing nodes for each chain, only 1/10 of the 50,000 nodes endorsing a chain in the past. When TPS goes up, security and credibility both fall.

So, raising single-node TPS (i.e. the TPS of a single node) is always most critical. For example, if the TPS of one chain can be increased to 1,000, then 10 shared chains will have a TPS of 10,000, while each chain can still guarantee the endorsement of 5,000 nodes.

It can be foreseen that disks' IO performance will be improved through SSD and other technologies in the next three to five years, and network bandwidth will also

continue to increase. The improvement of CPU clock frequency has now reached a bottleneck, and computing power needs to be boosted by increasing the number of CPU cores in a computer. However, if transactions can only be serially executed by single treads, it will not be possible to overcome the CPU bottleneck. As a result, the ability to execute transactions in parallel is particularly critical.

2.1.1.1 Parallel

The difficulty of parallel contract execution within a block is that we do not know in advance the relationship of dependency between contracts. There are three possible race conditions in Ethereum contracts:

- Access Account of the same address
- Access Storage of the same address
- Access Evm Code of the same address

If users are asked to identify and specify the address ranges over which condition conflicts occur when they compile contracts, it is clearly unrealistic to expect no errors or omissions.

Whether, where and under what condition bifurcations will race conditions occur can only be judged after the current state is deterministically obtained. With existing contract programming languages, it is almost impossible to obtain through static code analysis completely correct results without missing items.

However, this does not mean that the parallel execution of contracts in a block is absolutely impossible. Interestingly, this question was raised for Ethereum several years ago, but nobody has really tried to solve it. In fact, the current problem is more of an engineering rather than a theoretical or design one. In the process of project implementation, it is possible to find problems in the design process and then come up with better designs.

In this respect, Khipu has made comprehensive efforts and achieved project implementation.

Khipu’s implementation solution is to respectively execute all transactions in parallel starting from the world state of the block of the previous period. In the process of execution, the above three race conditions encountered on all ideal paths are

recorded. After the parallel stage ends, the merger stage is entered where the parallel world states are merged one by one. When merging a transaction, firstly judge from the recorded race conditions whether they are in conflict with the preceding merged race conditions. If not, merge the transaction directly. If yes, execute the process again with the previously merged world state as the starting point. Finally, use the hash of the block as the final check on the merged state of the world. This is the last line of defense. If an error is found in the check, abandon the preceding parallel scheme, and serially execute the transactions in the block again.

In the process, Khipu introduces a degree-of-parallelism index, namely the proportion of transactions within a block that can merge results directly without requiring re-execution. Khipu's actual test results show that this proportion (degree of parallelism) can reach 80% on average.

On the whole, if computing tasks can be fully parallelized, single chains will have unlimited scalability: one can always add more CPU cores to each node. If this is not the case, then the maximum theoretical rate will be limited by Amdahl's law: the limit of speed which one can increase for a system will depend on the reciprocal of the parts that cannot be parallelized. If one can parallelize 99% of them, the speed will be increased up to 100 times. But, if only 95% parallelization can be achieved, the speed will only be raised 20 times.

In the example of Ethereum, if there is 80% parallelization, then 20% of the computing tasks cannot be parallelized. The reciprocal of 20% is 5, so the limit of speed-up is 5 times.

In particular, for block producers, because transactions can be deployed on high-performance computers, if they can actively identify race conditions between transactions and arrange these transactions according to the best strategy, the degree of parallelism can be substantially increased, thus achieving higher speed-up. This is exactly the solution adopted after EOSIO Dawn 3.0:

Automatic Scope Detection [4]

Under EOSIO Dawn 2.0, every transaction was required to declare which data ranges it would access. This was error prone and verbose for developers. Under Dawn 3.0, the block producers are responsible for determining which data ranges are accessed and to decon ict them. This makes all transactions smaller and moves the scheduling overhead to the block producer rather than pushing it back on the user, developer, or full nodes.

2.1.1.2 Storage

Ethereum still has a serious bottleneck in single-node performance, namely access efficiency when there are hundreds of millions of trie nodes on one computer. This bottleneck was also the target of the DDos attack on Ethereum in 2016. The attacker repeatedly created empty accounts by utilizing the low gas of some storage access-related instructions (EXTCODESIZE, EXTCODECOPY, BALANCE, SLOAD, CALL, DELEGATECALL, CALLCODE, SELFDESTRUCT, etc.) of Ethereum at that time, which led to frequent IO disk operations and dramatic performance degradation of the nodes in the whole Ethereum. Later, although the attack was finally stopped by increasing the gas of the relevant instructions, it no doubt exposed Ethereum's access bottleneck when the number of trie nodes increased to tens of millions and even hundreds of millions.

In the subsequent main Ethernet implementations, Geth, Parity and others have all made considerable kv storage engine optimization. However, as the number of trie nodes continues to increase, almost all traditional mechanical hard disks have lost the ability to run the entire nodes.

An Ethereum block currently needs to execute proximately 4,000 random reads. We can estimate how long it takes to complete the data read and write of one block under different hard disk IOPS and different CacheHitRates in the most ideal case (only one disk IO is needed for one cold data read).

Firstly, set the IOPS indicators for several groups of hard disks:

- Set the IOPS of HDD random read to 100
- Set the IOPS of SATA SSD random read to 1,500
- Set the IOPS of NVMe SSD random read to 15,000

The time needed to complete the data read of one block in the ideal case (only one disk IO is needed for one cold data read) is given in the table below.

Table 2 Time Needed for Reading the Data of One Block in the Ideal Case

$$4000 \times (1 - CacheHitRate) \div IOPS$$

Disk (IOPS) /Cache Hit rate	HDD (100)	SATA SSD (1,500)	NVMe SSD (15,000)
50%	20s	1.3s	0.13s
60%	16s	1.1s	0.11s
70%	12s	0.8s	0.08s
80%	8s	0.5s	0.05s
90%	4s	0.3s	0.03s

It can be seen that as far as current Ethereum blocks are concerned, HDD nodes need to meet the following conditions to keep up with the mainnet (15s block generation time). Otherwise, it is theoretically impossible:

- Excellent cache algorithm, with a cache hit rate of more than 70%.
- Excellent storage engine, with one disk IO for cold data at most.

Geth, Parity and almost all other Ethernet implementations use LevelDB or its subsequent optimized version RocksDB to store kV data including those of trie nodes. LevelDB and RocksDB are both efficient kV databases implemented by LSM algorithm, which can convert random write into sequential write to substantially improve write performance. However, this algorithm cannot solve the problem of random read. Especially on mechanical hard drives, there is virtually no solution for random read, and caching has to be relied on. Or, SSD hard drives are cheaper to use than increasing caching through bigger memory.

But for Ethereum, a unique problem is that as long as the value of trie node data changes, the key also changes, because the key is nothing but the hash of the value. For example, even for the same account, its key also changes with the value. kv cached for the key is meaningful only when the value remains unchanged for a considerable period of time. Otherwise, the hot data is basically invalid. Such data features make it difficult to construct an appropriate cache algorithm. In other words, the simplest FIFO cache strategy may also be the most effective.

In Ethereum and many blockchain kv data, the key varies with the value. In fact, such kV data have high requirements for making random read directly from the hard disk. Even though a considerable amount of data is cached by memory augmentation, if the ability to read cold data randomly from the hard disk is not as good as O (1), then as:

- As data volume increases unidirectionally
- Early and late data become equally important

You will not be able to guarantee the completion time of any transaction, or as long as some out of hundreds of queries for a transaction are very slow, the processing of the transactions will be also substantially slowed down.

There are at least three types of data on the Ethereum chain:

1. Account nodes and storage nodes are world states (WS) organized in Trie form. Each block has a snapshot of WS at that block time. Each block is only related to the WS of the previous block. Snapshots of different blocks have redundant nodes (the same without change). Compaction can be done for these data, namely only the snapshots of a previous moment and the subsequent increments can be retained, without affecting the check of subsequent blocks.

2. Evm-code, block-header, and block-body. These three items plus account/storage nodes are all data which are needed for block validation. But, these three items correspond to block time one by one. They have no redundancy and need no compaction. And, these three items may always be used for block execution and validation.

3. Locations of Receipts and Transactions. The number is huge (each block can contain up to 300), and they are only used for queries.

In particular, for the first type of data, the states (trie nodes) of the whole world, i.e. the current states of all entities since the genesis block has evolved to the present, must be used in the calculation and validation of the current block. The location and value of all these entities in trie will be needed. Although calculating the current block only involves fragments of it, you cannot predict which fragments will be involved. In other words, you must store all the trie nodes that have not been deleted to complete the calculation and validation of the current block. This brings about a difficult implementation problem (contradiction): on the one hand, Ethereum

hopes that every whole node can participate in validation for the purposes of security, non-counterfeiting and non-tampering; on the other hand, these nodes must have the current state of the whole world. At present (in more than 6 million blocks), the total number of these trie nodes exceeds 250 million, occupying more than 40G of disk space. Computing a block will add approximately thousands of new nodes, while thousands of nodes also become useless.

So, is there a design that can keep LevelDB/RocksDB's write power as strong as possible, while achieving random read close to O(1) at the same time?

After carefully analyzing the characteristics of blockchain data, Khipu has especially developed a storage engine called Kesque which is suitable for blockchain kv data.

First, in the kv pairs of the blockchain, the key is usually the Hash of 256 bits. Loading such a big key into the memory will undoubtedly take up an enormous space. But since the key is the Hash, if we only get the last four bytes of the key (equivalent to a kind of effective compression) to represent the key, then the probability of collision will be very low. Actual test results also show that for hundreds of millions of Ethereum kv records, the probability of collision is only a few thousandths.
In other words, if we load the last four bytes of the key and the offset (represented by four bytes) of the record on the hard disk file into memory as an index, then in 99.x% of the circumstances, it will be possible to load the given key into the record by one disk IO at most.

The memory space occupied by this index per record is approximately: 4 bytes + 4 bytes = 8 bytes, and approximately 134 million records can be stored in one G of memory.

Following this idea, Khipu has developed and implemented a Kesque storage engine, whose key points of design are as follows:

- Use the log file of kafka as the carrier of kV entities of the hash table.
- Design an Int-Int map to save conflict-free key-> offset
- Design an Int-Ints map to save in conflict key-> offsets

- When searching, search in the int - > int map first

(1) If found, it will directly position to the offset recorded in the log file, and the record can be retrieved by one disk IO, with a complexity of O(1). This is the situation for 99.x% of the times.

(2) If not, then find n offsets from the int - > ints map. For each offset, read the record from the hard disk and compare it with the complete key. The same record is what needs to be found. Here, the n value is 2 in most cases, and the higher the probability, the lower the value. Although it is no longer one disk IO time (they may be two times or more points), the complexity is still approximately O(1).

When processing hundreds of millions of typical Ethereum data, this newly designed storage engine can have a random read capacity which is an order of magnitude higher than that of LevelDB.

2.1.2 Khipu-0.2.0-Beta

As the goal of the first stage, Khipu has mainly explored the extreme capacity limits of single nodes in Ethereum. Khipu 0.2.0 Beta comes out as the achievement of the first stage. It completes routine Ethereuem optimization and parallel transaction execution on single nodes, and has an especially designed storage engine Kesque.

With these optimizations, Khipu has reached the following performance when batch processing 6,184 blocks with more than 6.4 million numbers on a computer with a 32G memory, SATA SSD hard disk, and Intel® Xeon® E3-1231 v3 @ 3.40GHz quad-core CPU: 4.11 blocks are processed per second, actual TPS reaches 384, 23.5 MGAs are executed per second, the degree of parallelism is 80%, and processing speed is twice that of Parity-2.1.5 developed using Rust language, which is the fastest Ethereum implementation.

When compared with the theoretical estimates of Table 1, the processing capacity of 23.5 MGAs per second is equivalent to a 224 TPS under PoW and an 895 TPS under PoS, approaching the order of magnitude of 1,000. This ability of a single chain can well meet the needs of a considerable number of enterprise services.

If Ethereum 2.0 is implemented on this basis, then shard chains in the order of 10x can have a TPS of the 10,000 order.

2.2 Distributed Shard Architecture

2.2.1 Sharding Design of Ethereum 2.0

Ethereum 2.0 puts forward the concept of sharding design. Sharding is key to the distributed scalability of enterprise applications. The sharding design of Ethereum is composed of multiple chains, which in turn consists of multiple parallel worlds. Each chain corresponds to a single world or shard. A shard carries non-repetitive main data, which is known as the Data Layer. Data exchanges between shards are coordinated by the Coordination Layer.

To solve the scalability problem, the concept of on-chain state partition is introduced into the sharding of Ethereum 2.0 in order to achieve higher throughput. Ethereum 2.0 divides the Ethernet into two layers. The upper layer is the existing Ethereum main chain, which is basically kept unchanged, while the lower layer is shard chains, which are mainly used for sharded computing.

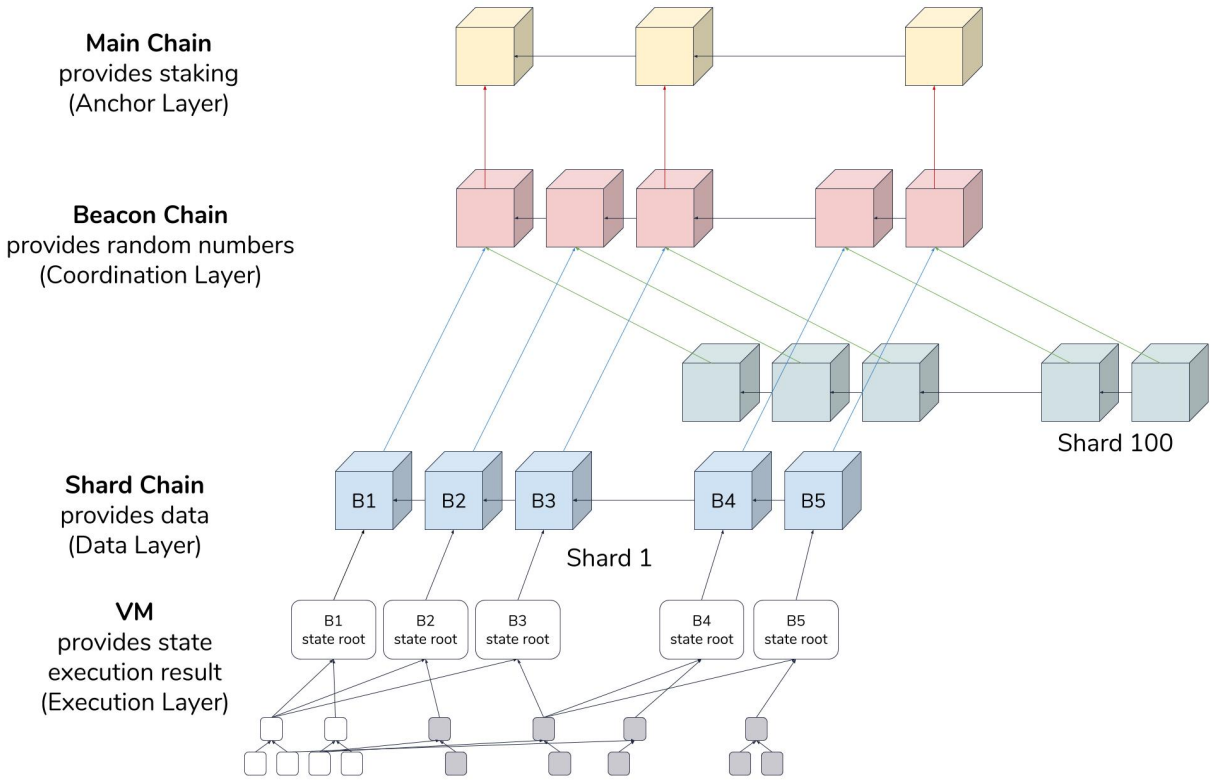
It can be simply understood as all transactions in a shard being loaded into collation. Similar to side chains, only a small portion of the collator will be recorded in the main chain.

1. Transactions on the shard chain are located in their own independent space, and the shard validator only needs to validate the shards of concern to it.
2. Shard chains are attached to the main chain through the POS mechanism to achieve a higher level of consensus.

The specific process is that a shard contains multiple nodes, namely shard validators. They can complete transaction validation in the shard through the POS mechanism. After validation, a check block will be generated, and one piece of head information of this check block is added to the main chain. But, the specific transaction is not saved on the main chain.

On the one hand, by introducing independent shards, Ethereum 2.0 offers parallel processing capability to the whole network and improves the throughput of the whole system; on the other hand, each shard only keeps part of the historical state data, and the main chain does not need to save specific transaction information, which can greatly reduce the storage pressure of nodes.

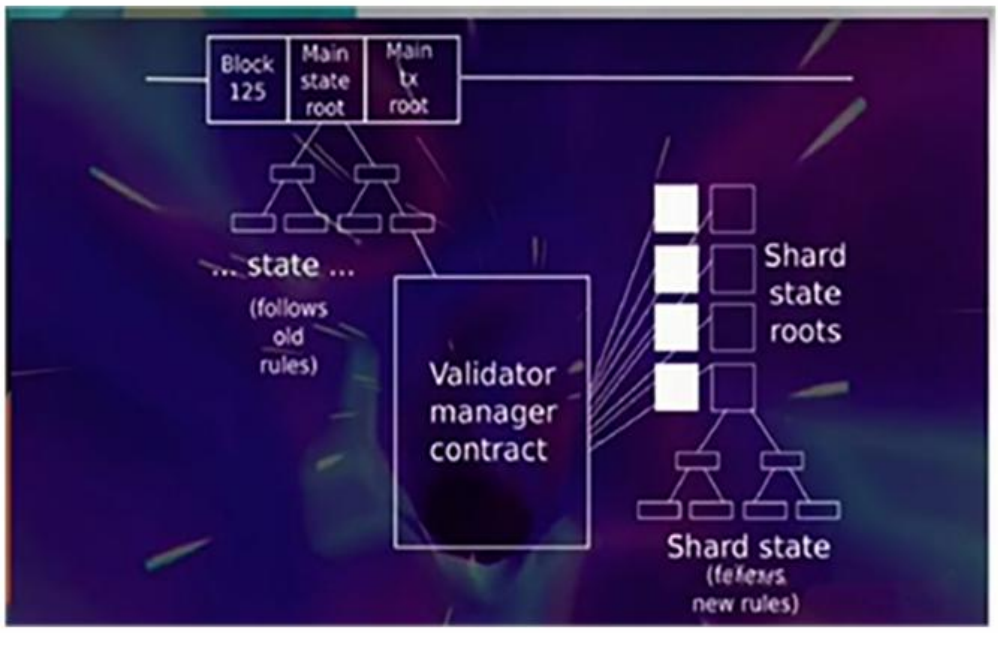
Figure 3 Sharding Design Concept of Ethereum 2.0



Source: What you can do for Ethereum 2.0 a.k.a. Sharding

2.2.1.1 Consistency of States

MainChain	ShardChain
Block	Collation
BlockHeader	CollationHeader
Block Proposer (or Miner in PoW Chain)	Collator



Source: zhihu@alabs block chain research

The PoW consensus is still used in the first layer, except that the state_root in the block converges many shard state roots. With the execution and validation of transactions in different parts of a block, the evolution of world accounts is executed concurrently, and TPS is increased.

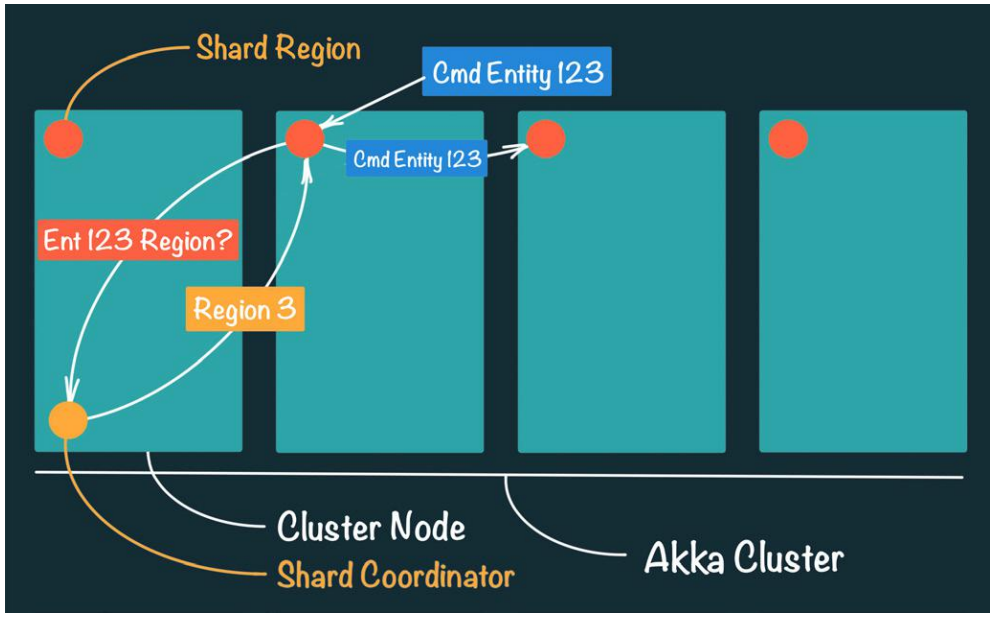
2.2.1.2 Consensus of Each Shard

Generally speaking, the high cost of computing in the world of PoW makes miners consciously converge to the longest chain. Then in the POS world, how to make block producers follow a unified strategy of longest chain selection? We need to set rules for this. Ethereum 2.0 adopts PoS + BFT-type consensus to reach ultimate consensus on the relevant rules (casper) and solve these fatal problems, But, BFT-type consensus carries a communication cost of N^2, and node scalability is also constrained. This is also why sharding (the number of nodes in each partition is limited) is chosen as the capacity expansion solution for Ethereum 2.0.

2.2.2 Sharding Design of Typical Distributed Systems

The sharding design of Ethereum is a typical way of sharding a distributed system. Compared with the popular distributed system Akka, we can see that each of its shared chains (parallel worlds) corresponds to the shard node in Akka's shard cluster, while the coordination layer is a single case of the whole universe. For a single chain in Ethereuem, it is a single cluster example of Akka's shard clusters.

Figure 4 Typical Distributed Shard Cluster – Design of Akka Distributed Shard Cluster



Source: How Akka Works: "Akka A To Z", An Illustrated White Paper

Akka already has many successful cases of enterprise applications. Similar to it, the sharding design of Ethereum allows us to smoothly distribute enterprise applications across shard chains. However, we need to extend Beacon Chain's stored state Markle trie (such as receipt) in the Ethereum 2.0 design to general enterprise domain entities.

2.3 Domain Entity Model

After its performance and architecture problems are solved, Ethereum is close to providing total enterprise application capability. However, there is still a major topic to be introduced – the domain entity model.

So far, Ethereum has provided an aggregate root under the concept of "aggregate", which is the aggregate of Account. This aggregate is the smallest unit which has encapsulated changes in a transaction, and realizes the association and reference between aggregates through ID (an Address with is the hash value). This aggregate has the following three values or entities:

- Balance - Direct deposit value
- Storage (limited to the kv pairs of 32 bytes keys and 32 bytes value) - referenced by ID (stateRoot)
- Contract (code) - Referenced by ID (codeHash)

For enterprise applications, these are not enough. Khipu will therefore expand its support for various business objects through the MPT mode of Ethereum storage state in accordance with the DDD "aggregate" concept.

2.3.1 Aggregate - Value Node of MPT

As stated above, an aggregate is a set of domain objects in DDD. It is the smallest unit of multiple state/data changes and is encapsulated as true invariance. Therefore, each aggregate instance can exactly correspond to the value node of the MPT tree, and the hash of all its values is its key.

Chapter 3 Khipu – Towards a Blockchain Enterprise Application Platform

After understanding the relationship between blockchain and DDD and the nature of sharding design, we can extend the blockchain to enterprise applications.

3.1 Blockchain Enterprise Application Scenarios

If blockchains truly evolve to complete businesses which are increasingly controlled and monopolized by "centers", then the first organizations to refactor their businesses to the blockchain will only need to be responsible for maintaining blockchain infrastructure, while the whole businesses can be completely out of the control of the organizations. If a business does this, it may defeat all its competitors. If it does not, someone else will do it. Perhaps, almost every IT service giant will come across a blockchain that provides the same service in the future, because this is the choice of users. And, it is almost the inevitable choice of users.

After business refactoring by the blockchain, a service is structured by individuals or companies. It can then be provided by anyone on any other node wholly and consistently to anyone anywhere.

There is another scenario of enterprise application. Several equal partners, who use the same set of data and services, have equal rights to publish and use data/services but can run their own nodes, while still maintaining data and logical consistency.

When blockchain infrastructure is immature, some data and services need to be put on the chain, while other data and services will still exist in the form of "centralized" services. With the improvement and upgrading of blockchain infrastructure, more and more data and services can be gradually transferred to the blockchain, until the data and services of the whole system are on the chain.

3.2 Khipu’s Goal

The goal of Khipu is to explore and implement a whole set of design patterns and infrastructure to refactor enterprise applications onto the blockchain, as well as to

establish, improve and develop corresponding ecosystems with communities. Its planned products will include :

- Easy-to-implement-and-deploy enterprise blockchain platforms and SDK
- Intelligent contract integrated development environment
- Enterprise blockchain real-time monitoring platforms

3.3 Khipu Interstellar

In the concept of Ethereum, every shard chain under a shard is a parallel world, and multiple parallel worlds constitute a complete service. In the field of enterprise applications, each complete enterprise service will therefore consist of multiple parallel worlds. Then, how will information be exchanged between different enterprise services?

The goal of Khipu Interstellar is to build a medium to communicate information and exchange value between different enterprise blockchain services. It will be a blockchain between blockchains, which transfers trust, information and value between different blockchains.

Chapter 4 Enterprise Blockchain Application Cases - Distributed Search Engine Blockchain

Khipu will explore a series of ecosystem tools for refactoring traditional enterprise businesses into the blockchain. As a starting point, it will select several traditional enterprise businesses as typical cases for implementation. These cases will be implemented from the following aspects:

- The case implementation process will be a process of continuously discovering and solving engineering and technical problems. A whole set of platforms and their development and maintenance tools will gradually take shape in this process.
- The case implementation process is also a process of validating the methodology and approaches described in this white paper.
- Case implementation will adopt a step-by-step approach. In other words, the various technologies designed for businesses, such as business data extraction, validation and storage, will be distributed or temporarily centralized where it is fit. For technologies such as large-scale data storage that can only be centralized for the time being, there will be a gradual transition towards the purely distributed model in the implementation process as technology advances.
- A case itself is a business model. In other words, the implementation of a case will bring about a completely distributed new business model to the business. With the gradual establishment of the business blockchain, the business will naturally become a completely new distributed business and compete with the existing centralized businesses in its field.
- The industry blockchains implemented on the basis of the Khipu platform will use general KIP tokens.

The first industry blockchain case implemented by Khipu is a distributed search engine blockchain.

4.1 Return Search Engines to the People

“Centralized” search engines that are now concentrated in large companies have the following problems:

- Data is controlled by a handful giant companies, forming monopoly over information sources.
- They can control what people see and what they cannot see.
- The privacy of personal access is also controlled in the hands of a few big companies.
- Data volume is so huge that only big companies have the computing resources to process it.
- How to ensure the fairness of human information access.

The Khipu-based distributed search engine blockchain will become a shared platform economy of "non-centralized" point-to-point direct communication. It adopts distributed ledger technology to encourage community participants and promote non-centralized platform management and operation. It also utilizes the automatic compliance of intelligent contracts to respect government and community regulations on urban supervision, thus giving concurrent consideration to universal benefits, regulatory compliance and efficiency. It ensures that search engines do not become engines controlled by any company or individual, and that the search engines are returned to the hands of the people.

4.2 Design

The distributed search engine blockchain can be summarized as follows:

A blockchain which anyone can participate in and which reaches an index record chain with chain-wise consensus after distributed web content extraction and parsing.

- Anyone can trustfully add his or her computer as a node to the business blockchain.
- Users who write web page extraction rules (dApp) can get token incentives upon validation.

- Extraction tasks will be distributed to the nodes on the chain through blockchains. Each node can complete capture service independently and get token incentives upon validation.
- Extracted web content will be locally parsed, segmented and summarized by the node to generate index records.
- Index records are distributed to the chain and validated by PoS and other consensus mechanisms to form a consistency index (consensus) of the whole chain.
- Extracted content is cached at the node for 30 days and then deleted to reduce disk resource usage and avoid copyright risks.
- Nodes participating in content extraction and parsing can get token incentives upon validation.
- Users can write dApp to make second-time processing of index records on demand and then use it. dApp needs to pay gas usage fees.
- Ordinary search is performed through a built-in dApp, with very low gas consumption.

4.3 Technical Difficulties

- Virtual Machine (VM) needs to support network IO, text processing (parsing, word segmentation), etc.
- What are stored on the chain are no longer just account balances and short data, but also index records that can be queried.
- Validation of content and processing results extracted from any node
- Rational distribution of web page extraction tasks

4.4 Legal Issues

- Ensure that there is no risk of copyright infringement when extracted web content is cached at a node for 30 days and then deleted;and
- Ensue that it is within the scope of legal norms on content in various countries when only index information is recorded on the chain.

Chapter 5 Khipu's Economic Model

Token KIP on the Khipu platform is a Utility Token, which anchors the value of resources on the chain, including but not limited to the computing power and storage consumed when running EVM, as well as valuable information (such as web index, web content) and so on.

Because Khipu is compatible with and extends Ethereum, its resource consumption and usage are measured in the form of Gas, and are deducted in the process when EVM runs instructions.

KIP Token is a general Token for any application instance based on the Khipu platform, although these different instances may run on different, independent public chains.

The first group of application instances on the Khipu platform are the search engine chain and the advertisement delivery chain.

Khipu's total KIP Token issuance volume is 10 billion, 20% or 2 billion of which are rewards for block generation.

5.1 Token Circulation Mechanisms of Search Engine and Advertisement Delivery Chain

There are four types of actions with the search engine and advertisement delivery:

- Crawling web pages, parsing and generating web index according to specific rules
- Packaging to form chain-wide Token circulation consensus and global web indexing consensus
- Searching web index
- Delivering advertisement

5.1.1 Crawling web pages, parsing and generating web index according to specific rules

"Crawling requests" come from users or website operators who want to have their website contents cited by search engines. KIP payment is a payment to meet such needs.

Khipu's web crawling tasks are implemented by writing dApp. After dApp is submitted, *n* nodes will be randomly selected from the chain. The correctness of the results executed by these nodes will be verified by comparing *m* results (m < n).

dApp consumes Gas. Therefore, corresponding KIPs must be paid for submitting dApp so as to ensure that the submitted “crawling requests” are executed by the nodes on the chain.

Among the *n* nodes that actually execute crawling tasks, the *m* nodes with the correct crawling results will share these KIPs with the packaged nodes according to a certain proportion.

5.1.2 Packing to form chain-wide token circulation consensus and global web index consensus

Mining or packaging nodes are responsible for verifying whether there are *m* copies of the web page and index results crawled back by the *n* nodes are identical. When m is > t (t is the minimum score threshold required for verifying the correct result), the index result will be merged into the global index and broadcast to the whole network.

During PoA, those packaging nodes with Authority will have the right to package and get the corresponding rewards for block generation.

During PoW, those nodes that complete the computational workload will get rewards for block generation.

5.1.3 Searching web index

Web index searching is what ordinary users most frequently do. To avoid the abuse of network resources, searching actions also consume Gas and have to pay corresponding KIPs. However, searching is a fixed action, and very low Gas consumption will be set for it so that ordinary users can afford it. Users do not need to pay immediately, but use mortgage KIPs to make monthly settlement. At the same time, when used along with the advertisement delivery chain, KIPs paid by searching users will be returned, and users may even get income depending on the advertisement delivery situation.

If ordinary users run all the nodes at the same time, they will not only get quicker searching responses, but also have the opportunity to obtain income by completing crawling tasks.

5.1.4 Delivering advertisement

Advertisers also use dApp to deliver advertisements. Packaging nodes will choose which advertisements to package first according to the gas price which advertisers are willing to pay, and deliver the advertisements to the Khipu client.

5.2 Participants in the KIP Token circulation process

5.2.1 KIP spenders

- Users who have specific content requirements submit crawling requests by writing dApp and paying the corresponding gas fee KIP
- Website operators who want to have their own content indexed by search engines submit crawling requests by writing dApp and paying the corresponding gas fee KIP
- Advertisers who deliver advertisements by writing dApp and paying the corresponding gas fee KIP

5.2.2 KIP income recipient

- Packaging income obtained from packaging nodes (Mining Mode 1)
- Income obtained by ordinary user nodes from the correct completion of crawling tasks (Mining Mode 2)
- Income obtained by ordinary users from advertisement display (Mining Mode 3)

5.2.3 KIP Mortgagor

- Packaging nodes

5.2.4 KIP Issuer

- Initial issuance (private placement, community, etc.)
- Gradual issuance by packaging nodes (a total of 2 billion)

5.3 Early PoA mechanism

5.3.1 Background

The early test network of Ethereum was called Modern. Later, due to compatibility problems between two Ethereum clients and some technical bugs, Modern was quickly rebuilt into Ropsten.

At the end of February 2017, Ropsten suffered a huge attack. Originally, packaging one block would cost only 4.7 million Gas. But, the attack raised that number to 900 million Gas, mainly due to the low cost of testing money in the Ethereum test network and the very low computing power of the entire network (there was no cost of rewards to support computing power). Subsequently, the Parity client developed a test network called Kovan. Soon, Geth also came up with a version called Rinkleby, which is compatible with Geth. As a result, Ethereum has three test networks.

Kovan is characterized by the use of the PoA (Proof of Authority) mechanism. Transition through this PoA mechanism is essential for any network with low

computing power. PoA can well protect networks with low computing power from malicious attacks when the whole network's computing power is not big enough.

5.3.2 Khipu’s early PoA mechanism

Both the search engine chain and the advertisement delivery chain used PoA when early testing networks transited to the main network. PoA requires a certain threshold but also has huge rewards. Khipu will use 10% of its total tokens to reward PoA nodes. After the transfusion from PoA to PoW and then to the PoW+PoS mixed mechanism, these nodes will automatically turn into ordinary mining nodes and be entitled to normal mining benefits.

To apply for a PoA node, more than 10 million KIP ERC20 Tokens must be pledged in exchange for the node rights and test tokens of the test network. The more the pledge, the more the test tokens obtained. The node reward mechanism is dynamic and gives overall scores based on the number of corresponding nodes packaged, online duration, failure rate, the total number of packaged contractual token transactions and other dimensions.

The testing node which comes first in the end can get a minimum reward 3 times the number of pledged tokens, up to 200 million tokens in total; the 2^{nd} to 5^{th} nodes will get a minimum reward 1.1-2 times the number of pledged token, up to 100 million tokens in total, the 6^{th} to 10^{th} nodes will get a minimum reward 0.5 times the number of pledged tokens, up to 60 million tokens in total, and nodes below the 10^{th} place will be rewarded with 0.1 times the number of pledged tokens, with a single maximum reward of no more than 10 million tokens and a total reward of no more than 100 million tokens.

Chapter 6 Khipu[4] Roadmap

6.1 Khipu Roadmap

[4] "GitHub - khipu-io/khipu: A Scala/Akka implementation of the Ethereum ...." https://github.com/khipu-io/khipu. Accessed 14 Apr. 2019.

Chapter 7 Team

7.1 About Us

The Khipu team is one of the earliest teams in the industry to engage in the development of enterprise-level parallel distributed applications. It has more than ten years of experience of successful development and deployment in this field. Projects which have been developed and deployed by its leading members include:

- Scala and Erlang language integrated development tools
- Large-scale parallel distributed real-time securities analysis and trading platforms
- Parallel distributed network crawlers
- Parallel distributed content service systems
- Parallel distributed messaging engines
- Parallel distributed real-time monitoring systems

The team boasts strong analytical and engineering capabilities, as typically evidenced in the analysis and optimization of Ethereum's capacity limits

7.2 Team leader

Caoyuan Deng / KHIPU Founder

- ❖ Bachelor of Engineering, Tsinghua University
- ❖ Master of Philosophy, Tsinghua University
- ❖ CTO, TOM (China) Investment Co., Ltd.
- ❖ Chief Expert, Financial Engineering Department
- ❖ Founder Securities Co., Ltd.
- ❖ Architect for the Wandoujia platform
- ❖ Architect for the Qingmang platform
- ❖ Member of the NetBeans Dream Team

◆ Caoyuan Deng's Social Meida Accounts: Twitter : https://twitter.com/dcaoyuan
Github: https://github.com/dcaoyuan Weibo: https://www.weibo.com/dcaoyuan

KHIPU

Towards an Enterprise-Level Blockchain Platform

Khipu Foundation

V 1.0