AIP[1]-008: Sharding

liayoo, platfowner 2020-07-21

Goals

- Define minimal requirements for a scalable blockchain and provide a design proposal
- Set directions for further upgrades of the minimal version
- Provide a roadmap

Requirements

Minimal Requirements for Scalable Blockchain

- Provide a scheme to decompose the global states into partitions and run parallel blockchains (parent blockchain and child blockchains) for the partitioned states to allow parallel transaction throughput
- Provide state proof for the child blockchains by linking the states of the child blockchains to the states of the parent blockchain with proof hashes
- Support multi-layer sharding

Advanced Requirements for Scalable Blockchain

- Support cross-shard communication
- Support more decentralized proof hash reporting
- Support deposit and exit protocol

Proposed Design

[1] AI Network Improvement Proposal. Visit https://docs.ainetwork.ai for the full list.

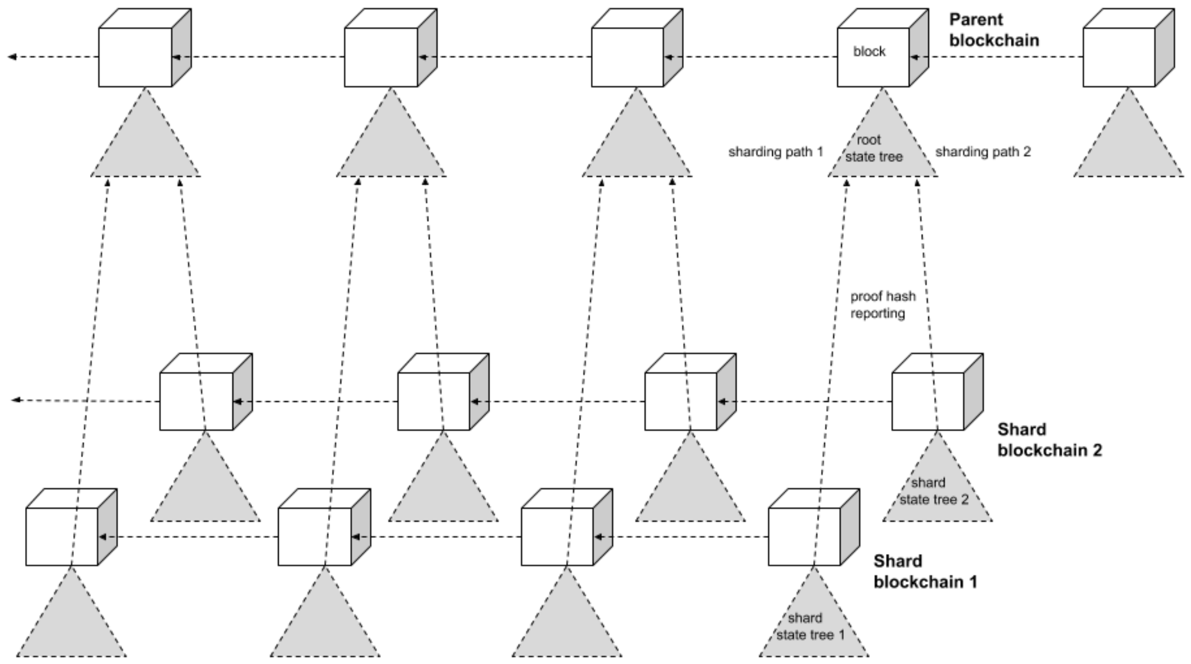

Figure 1. Sharding framework.

Key Ideas

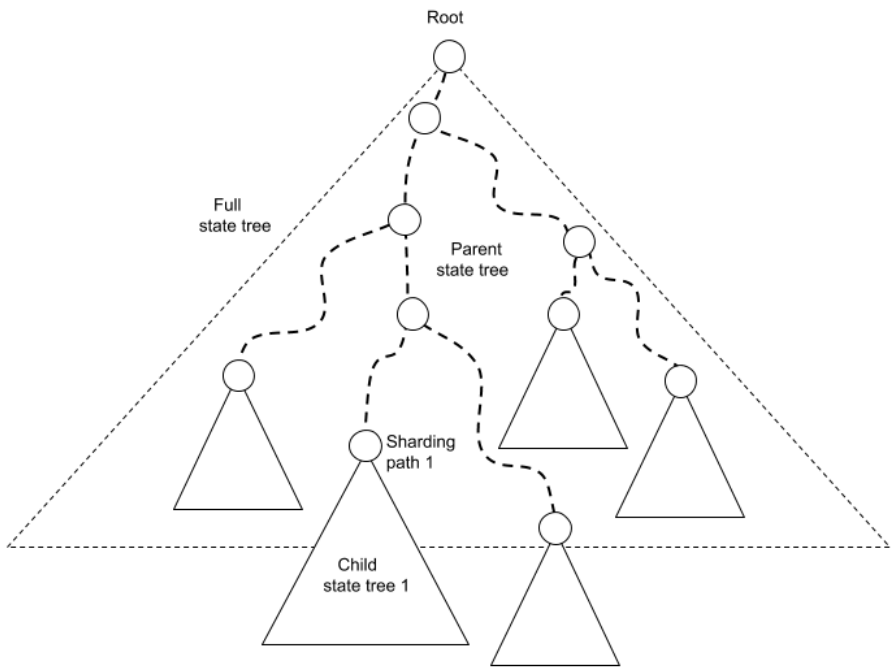

Figure 2. State decomposition for sharding.

State Tree Decomposition

For sharding, the blockchain states can be decomposed as follows:

- In AI Network Blockchain, states are defined and managed in a tree structure, so it can be naturally decomposed into a root tree and subtrees. The states of the root tree and subtrees are called ***root states*** and ***sub-states***, respectively.
- Each root of the subtrees is linked to a leaf node of the root tree.
- We call the path to the leaf node ***sharding path*** of the corresponding subtree and its blockchain.
- A subtree can be further decomposed into a root tree and subtrees for multi-layer sharding.
- When a state tree is decomposed into a root tree and subtrees, they are referred to as parent tree and child tree, their states as ***parent states*** and ***child states***, and their blockchains as ***parent chain*** and ***child chain (or shard chain)***.

POA-based Configurable Proof Hash Reporting

We can support PoA (Proof of Authority) based configurable proof hash reporting for child blockchains as follows:

- The sharding part of a child blockchain is configured by its owner by providing the following information to the genesis block of the blockchain:
    - Sharding path
    - Proof hash reporting period
    - The point-of-contact (POC) node of the parent chain
- Given the configuration, which is called ***sharding configuration***, the first node of the child blockchain sets up the sharding path of the parent blockchain by adding:
    - Owner configuration of the sharding path
    - Rule configuration of the sharding path
    - Initial value configuration of the sharding path

Proof Hash Linking and Provable Child States

In order to make the child blockchain provable via its parent blockchain, the root hash of the child states (i.e., proof hash) should be linked to a state of the parent blockchain. It's done by writing the proof hash to the sharding path of the parent states:

- A proof hash value written to the sharding path is keyed by the block number of the child blockchain.
- In order to prevent the data size bloats in the state tree, old hash values are automatically removed by the first peer node of the child blockchain.

Design Details

State Tree Decomposition

- When a shard is created, the subtree that the shard network manages is "carved out" of the root state, and its proof hashes take the place of the original subtree.
- AIN Blockchain supports 4 types of states, namely values, rules, owners, and functions. One state path may have all or some of the types. For example, if the value at /apps/afan is "Barbara" and the write rule is "auth === ‘0xBARBARA’", the state tree’s internal representation will look like the following:

Table 1. Global state tree before a shard at /apps/afan is created.

```
{
  "values": {
    "apps": {
      "afan": "Barbara"
    }
  },
  "rules": {
    "apps": {
      "afan": {
        ".write": "auth === ‘0xBARBARA’"
      }
    }
  }
}
```

When a shard is created at path /apps/afan, its current values will be removed from the root state tree, and the rules, functions, and owners will be replaced with new sharding configurations. Note that it is the shard owner’s responsibility to copy the existing values, rules, owners, and/or functions to the sub-state.

Table 2. Root state after a shard at /apps/afan is created.

```
{
  "values": {
    "sharding": {
      "shard": {
        "apps”: {
          “afan”: {
            “sharding_protocol”: “POA”,
            “sharding_path”: “/apps/afan”,
            “shard_owner”: “0xBARBARA”,
            “shard_reporter”: “0xALICE”,
            “parent_chain_poc”: “node.ainetwork.ai”,
            “reporting_period”: 10,
            “shard_reporter_endpoint”: “34.83.238.42”,
          }
        }
      }
    },
    "apps": {
      "afan": null
      }
    }
  },
  "rules": {
    "apps": {
      "afan": {
        “.write”: {
          “auth === ‘0xALICE’”
        }
      }
    }
  },
  “owners”: {
    “apps”: {
      “afan”: {
        “.owner”: {
          “owners”: {
            “0xBARBARA”: {
              “branch_owner”: true,
              “write_function”: true,
              “write_owner”: true,
              “write_rule”: true
            }
          }
        }
      }
    }
  }
}
```

Table 3. Sub-state of a shard at /apps/afan.

```
{
  "values": {
    “sharding”: {
      “config”: {
        “sharding_protocol”: “POA”,
        “sharding_path”: “/apps/afan”,
        “shard_owner”: “0xBARBARA”,
        “shard_reporter”: “0xALICE”,
        “parent_chain_poc”: “node.ainetwork.ai”,
        “reporting_period”: 10,
        “shard_reporter_endpoint”: “34.83.238.42”,
      }
    }
  },
  “rules”: {
    “.write”: “auth === ‘0xBARBARA’”,
    “sharding”: {
      “shard”: {
        “.write”: “auth === ‘0xALICE’”
      }
    }
  },
  “owners”: {
    “.owner”: {
      “owners”: {
        “0xBARBARA”: {
          “branch_owner”: true,
          “write_function”: true,
          “write_owner”: true,
          “write_rule”: true
        }
      }
    }
  }
}
```

Sharding Configuration

Roles and Permissions

In sharding, there are two different roles:

- ***Shard owner***:
  The role to manage the shard configuration. In POA protocol, it's set to be the owner address of the child blockchain.
- ***Shard reporter***:
  The role to report the proof hashes of the child blockchain to the parent blockchain. In POA protocol, it's the first child blockchain node.

The permissions for the roles are controlled using the owner and rule settings of the parent and child blockchains using the accounts given in the sharding configuration.

Configuration Specification and Sharding Protocol

When creating a shard, a node first needs to configure the shard by creating a "genesis_sharding.json" file that contains the following information:

Table 4. Sharding configuration properties.

Property name	Description	Example values
`sharding_protocol`	The type of the sharding protocol. Initial version supports "POA" (Proof of Authority). Default value is "NONE", which means it's a root chain.	“NONE”, “POA”
`sharding_path`	The path of the parent chain at which the shard will be created. The created shard will manage the subtree at the path.	"/apps/afan"
`shard_owner`	The owner of the sharding configuration.	“0xAAA...”
`shard_reporter`	The blockchain wallet address of the shard reporter that will communicate with the parent_chain_poc.	“0x0001...”
`parent_chain_poc`	The Point of Contact (PoC) at the parent chain's network that the shard will communicate with. Default setting is *node.ainetwork.ai*, an address mapped to some main chain's nodes.	“node.ainetwork.ai”
`reporting_period`	How often the shard will report its proof hashes. The unit is block, so if reporting_period is set to 10, the shard should report the proof hashes once every 10 blocks.	10

`shard_reporter_endpoint`	The network address (ip address or url) of the shard reporter.	“34.83.238.42”

In POA sharding protocol, the shard reporter node is implicitly determined to be the first child blockchain node that initializes the shard. The selection mechanism can be improved in the future, for example, by incorporating random sampling or having a committee of reporters. The configuration will also be recorded in the shard chain’s state, at /sharding/config. See the Initialization section for more details.
Subsequent non-first blockchain nodes who wish to participate in the shard network need to obtain the configuration file from child_chain_poc.

In the following sections, we describe based on POA sharding protocol.

Initialization

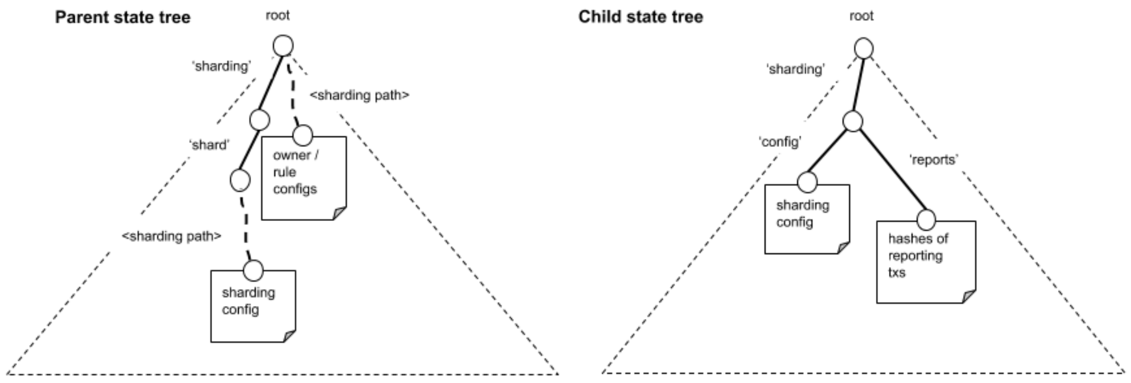

Figure 3. State tree initialization for sharding.

As the first child blockchain node begins its process, it needs to initialize the shard in two steps: one in the parent chain, and the other in the shard chain.

Parent Blockchain Initialization

Create and send an initialization transaction to the parent chain, which triggers a native function that sets owners, rules, and values at the sharding path so that only the reporter node would have the permission to report (write) at the path.

```
{
  type: "SET_VALUE",
  ref: "/sharding/shard/<Sharding Path>"
  value: {
    sharding_path: "/apps/afan",
    parent_chain_poc: "node.ainetwork.ai",
    reporting_period: 10,
    shard_owner: "0xAAA...",
    shard_reporter: "0x0001...",
    shard_reporter_endpoint: "34.83.238.42",
    sharding_protocol: "POA"
  }
}
```

Once the native function is triggered with the above data, it will

1. Set owners at `<Sharding Path>` to
   ```
   ".owner": {
     "owners": {
       <Shard Owner>: {
         "branch_owner": true,
         "write_function": true,
         "write_owner": true,
         "write_rule": true
       }
     }
   }
   ```
2. Set rules at /`<Sharding Path>` to `".write": "auth === <Shard Reporter>"`
3. Set value at /`<Sharding Path>` to `null`
4. Set function at /`<Sharding Path>` to `null`
5. Add a native function to the path `/<Sharding Path>/<Block Number>/proof_hash` that will update the latest reported block number.

Child Blockchain Initialization

Create a genesis block with the genesis configuration mentioned in the Configuration section. In addition to the basic genesis values, rules, and owners set-up, the configs need to be set as values at /sharding/config.

```
{
  type: "SET_VALUE",
  ref: "/sharding/config"
  value: {
    sharding_path: "/apps/afan",
    parent_chain_poc: "node.ainetwork.ai",
    reporting_period: 10,
    shard_owner: "0xAAA...",
```

```
    shard_reporter: "0x0001..."
    shard_reporter_endpoint: "34.83.238.42"
    sharding_protocol: "POA"
  }
}
```

Proof Hash Reporting

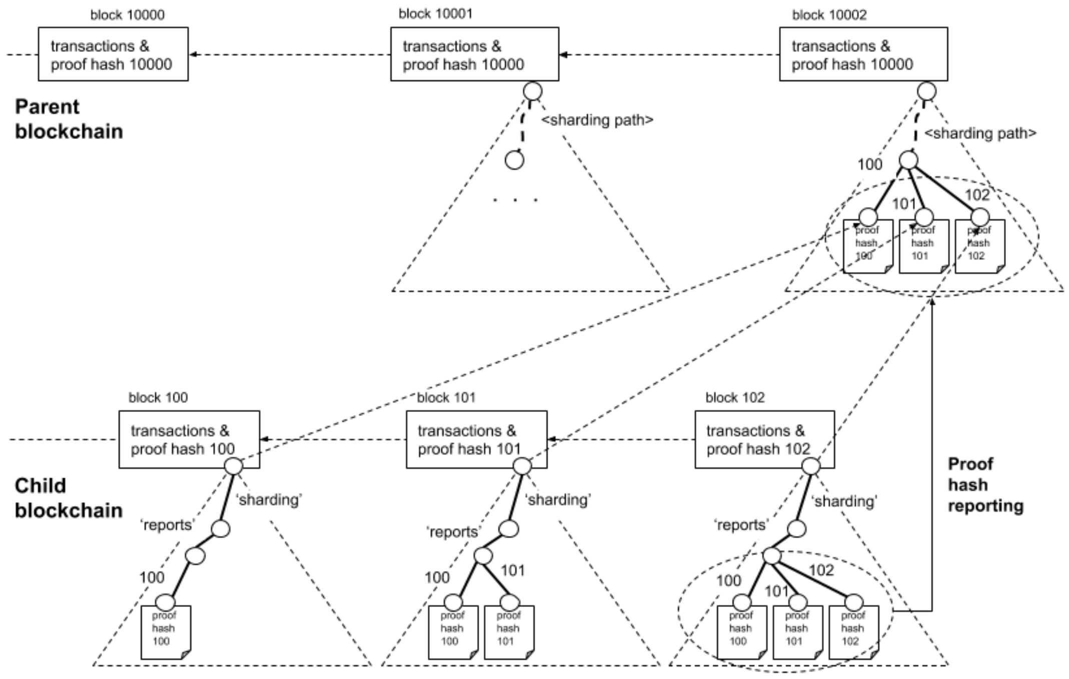

Figure 4. Proof hash reporting.

A designated reporter node needs to report a proof hash of the shard's updated state tree periodically to its parent chain so that other shard and parent networks can get the value of the sharding path and see the latest proof hash. Once the reporting transaction is included in one of the parent chain's blocks, the shard is considered "finalized" up to the recorded state. The reports can also be used in verifying the involved shards' states when processing a cross-shard transaction.

1. Get information on the last report by making a /get_value API call to the parent_chain_poc.

```
POST $parent_chain_poc/json-rpc/get_value
body: {
  …
  "ref": "<Sharding Path>/latest"
}
```

2. Compare it with the current shard chain's recently finalized block.

3. Report the current block, as well as any previous blocks that haven't been reported. Note that in order to prevent the reports from infinitely growing, the proof hashes at `/<Sharding Path>` will be removed in a round-robin fashion, once they reach a predefined threshold.

```
POST $parent_chain_poc/json-rpc/ain_sendSignedTransaction
body: {
  "signature": "0x...",
  "transaction": {
    "operation": {
      "op_list": [
        {
          "ref": "<Sharding Path>/100/proof_hash",
          "value": "0x...",
          "type": "SET_VALUE"
        },
        {
          "ref": "<Sharding Path>/110/proof_hash",
          "value": "0x...",
          "type": "SET_VALUE"
        },
        ...
      ]
      "type": "SET"
    }
    ...
  }
}
```

Upon setting a value at `/<Sharding Path>/<Block Number>/proof_hash`, the native function set during Parent Blockchain Initialization will update the `/<Sharding Path>/latest` accordingly.

4. Save the reported proof hashes and their transaction hash to `/sharding/report/<Sharding Path>/<Block Number>`. As the parent state does, the oldest of these proof hash and tx hash records at child state will be removed if they reach a threshold.

```
{
  type: "SET",
  ref: "/sharding/report/<Sharding Path>"
  "op_list": [
    {
      "ref": "100",
      "value": {
        "proof_hash": "0x...",
        "tx_hash": "0x...",
      },
      "type": "SET_VALUE"
    },
    {
      "ref": "101",
      "value": {
        "proof_hash": "0x...",
        "tx_hash": "0x...",
      },
      "type": "SET_VALUE"
    },
    ...
  ]
}
```

State Proofs

As defined and explained in AIP-007 Provable Blockchain States, AIN Blockchain provides state proofs for a given state, which is a combination of state hashes needed to calculate the root hash that proves whether the given state is valid or not when compared to the proof hash in a finalized block.
With sharding, a shard's state proof will only prove that the state is valid within the shard. If one needs to prove that the shard's state is also valid in the parent network's context, it would need to make sure that the particular state proof is included in the parent chain. This can be done by looking up the transaction hash of the state proof reporting transaction at `/<Sharding Path>/<Block Number>/tx_hash`, and asking the parent network if the transaction has been included in one of the finalized blocks, using the "ain_getTransactionByHash" JSON RPC API.

Advanced Topics

Cross-Shard Communication

To support cross-shard communication, AIN Blockchain sharding protocol can be extended to process a cross-shard transaction asynchronously in multiple shards, and each shard can check the proof hashes in the root state to make sure the transaction has been processed and finalized in other shards. A cross-shard transaction will be finalized, if and only if the transaction is finalized in all of the shards that the transaction affects the states of, and the shards' proof hashes can be found in the root state.

Decentralized Proof Hash Reporting

The proof hash reporting mechanism used in the POA sharding protocol is clearly not the most secure or decentralized solution. The reporter selection protocol can be extended in such a way that it can be more random and resistant to various attacks. One of the ways to do this is by creating a committee of reporters and performing a weighted random selection algorithm among them in each reporting period, a scheme much like the Proof of Stake (PoS) protocol that's often used for blockchain consensus.
In the future, as we support other improved types of proof hash reporting mechanisms, the shard owners can change the configuration file to adopt the new protocol. Specifically, one may change the `sharding_protocol` type and `shard_reporter` fields, as well as the write rule at `/<Sharding Path>` to something like `".write": "<Reporter Committee Map>[auth] === true"` so that anyone in the reporter committee has the write permission.

Deposit and Exit

A shard may improve its security and extend its functionalities by providing a deposit-and-exit scheme. Participants would deposit AIN at the root chain and a corresponding amount will be minted in the child chain. The deposit will be locked until the participant (the owner of the deposited and minted tokens) tries to exit from the child chain, in which case she can withdraw the tokens from the child chain and claim them in the root chain. The process will be similar to the deposit and exit processes in Plasma. For instance, if a shard at /app/afan has its native token FANCO, users can deposit AIN from the root chain, and an exchange rate will be applied to swap the AIN into FANCO and mint it in the child chain. Later when the user decides to exit, she will request for a withdrawal, and after a predefined lock-up period, the FANCO left in the child chain will be swapped to AIN in the root chain. In the simplest form, the exchange rates could be fixed and defined in `/sharding/config`, but shards may choose to support more complex variable exchange rate schemes that decentralized exchanges such as Uniswap, Bancor, and FancoSwap are based on.

Roadmap

The proposed features will be implemented with the following phases:

Phase 0: Implement POA sharding protocol

- Sharding config setting in shard blockchain
- Sharding config setting in parent blockchain
- Proof hash reporting

Phase 1: Support deposit and exit

Phase 2: Extend to POS sharding protocol

Phase 3: Support cross-shard transactions

Conclusion

The following topics were addressed by this document:

- Defined minimal requirements and advanced requirements for scalable blockchain
- Proposed design for minimal requirements
- Provided directions to extend the proposed design to meet the advanced requirements
- Provided a roadmap

Links

- Proof of Authority (wiki)
- AIP-007 Provable Blockchain States (link)
- Plasma (link)
- AI Network Blockchain Scalability (gitbook)

Document History

Date	Who	Change	Notes
2020-07-21	platfowner, liayoo	Initial flow and skeleton	
2020-08-07	liayoo, platfowner	Detailed drafting	
2020-08-13	platfowner, liayoo	Shared and reviewed	Eng design review
2021-05-07	platfowner	Published	
2021-05-12	platfowner	Github IDs Link to full list	
2021-05-12	platfowner	Republished	