Titan scheduler

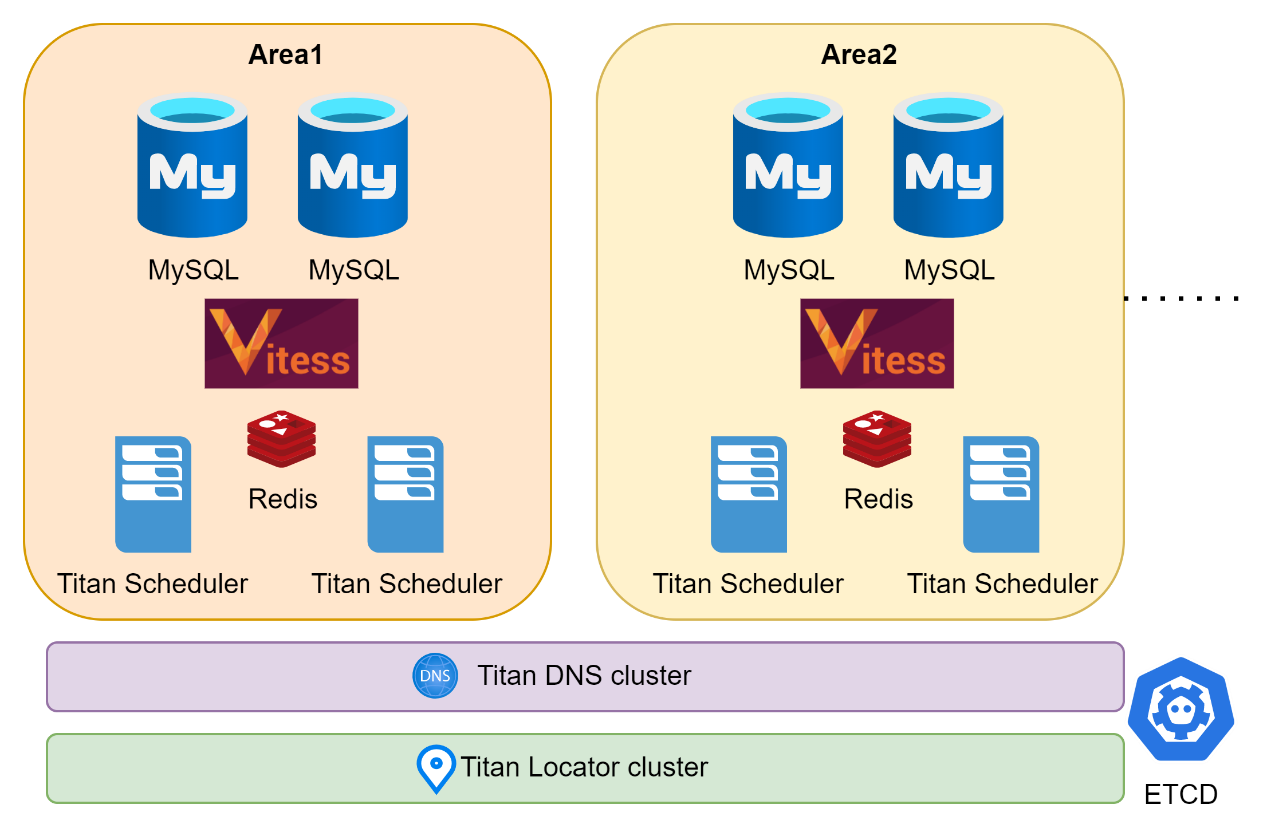
# Summary

As the hub of the titan network, the main responsibilities of the scheduler are as follows:

* manages the nodes in the network. All nodes in the titan network need to connect to some scheduler in order to join the titan network.
* manages assets in the network. The scheduler provides interfaces through which administrators can manipulate assets objects in the network, for example, by creating, deleting, querying or modifying assets objects.
* responds to the user's request to pull assets data and assigns the resource nodes that own the assets so that the user can pull assets data from those nodes.
* evaluates the contribution of nodes, generates an archive carfile of all nodes' contribution records, and publishes the CID of the archive carfile to the FVM contract. Scheduler performs storage and bandwidth spot-checks on the nodes in the network according to the set period, and records the results of the spot-checks. After a node provides download services to users, the scheduler records the node workload based on valid workload proofs submitted by the node and users; the sampling results as well as the node workload are the basis for the scheduler to evaluate the contribution of the node.
* The Scheduler sends rewards to nodes based on their contributions, periodically sends rewards to nodes based on their contributions according to the reward period and reward rules set by the titan network, generates an archive carfile of the reward results, and publishes the CID of the archive carfile to the FVM contract.

# Deployment

Deployment diagram of the Titan network:



The Titan network is deployed by area. For each area, there is a titan area network. These titan area networks form the entire titan network.

Generally, an area corresponds to a city, and all nodes in the Titan area network run in the metro Ethernet of that city, or in the metro network of that city. Therefore, it is very fast to transfer data between nodes, because they do not need to communicate across the national backbone.

As shown in the figure, for each area, deploy multiple titan scheduler servers, at least two are recommended to avoid the failure of a single titan scheduler causing the titan network in that area to be no longer available.

As shown in the figure, for each area, a Vitess distributed MySQL database system is deployed, and multiple titan scheduler instances share the same Vitess database system, so these scheduler server instances see the same data.

As shown in the figure, for each area, consider deploying one Redis cluster for accelerated data reads.

The titan locator in the figure, is responsible for discovering all titan scheduler servers and routing requests to the nearest scheduler server for that user when they are received. After each titan scheduler is started, it registers with the ETCD server of the titan network, so the titan locator is able to discover all the scheduler servers and the titan area network served by these schedulers through the ETCD service.

Titan DNS, shown in the figure, is responsible for mapping the node network name to the node's IP address. Each node in the titan network is given a network name, and clients need the titan DNS service to resolve the node's network name to the node's IP address in order to access the node.

# Database

## MySQL

Titan network has a lot of data, such as node information, Assets information, etc., which needs persistent storage, we choose MySQL as the persistence database.

## Vitess: Distributed MySQL

Considering the scale of data, we need a distributed MySQL database; considering the reliability of data, we need Master-Slave architecture to avoid data loss due to single point of failure.

We chose Vitess as our distributed solution for MySQL, a database solution for deploying, scaling and managing large clusters of MySQL instances. Vitess combines many of the important features of a MySQL database with the scalability of a NoSQL database.

Vitess' built-in split sharding feature allows us to scale a MySQL database cluster infinitely horizontally without adding sharding logic to the application.

In general, Vitess has the following advantages:

* performance

Vitess automatically rewrites queries that are detrimental to database performance. It also uses a caching mechanism to mediate queries and prevent duplicate queries from reaching the database at the same time.

* management

Vitess can support features such as automatic handling of primary failover and backups. It uses distributed metadata services to track and manage servers, eliminating the need for applications to care about database topology changes.

* connection pool

Vitess avoids the high memory overhead of MySQL connections. Vitess server easily handles thousands of connections at a time.

* shard management

MySQL itself does not provide split sharding, but we need to add clusters when the volume of business data increases to a certain level, and Vitess provides an online splitting feature that requires very little time to switch to a new cluster without adding any splitting logic to the application.

* workflow

Vitess keeps track of all metadata about the cluster configuration so that the cluster topology is always up-to-date and consistent across clients.

## Memory Database

The Titan network has some data that needs to be read frequently, for example, the CID of the carfile, and the list of edge nodes that own the carfile, etc. This data needs to be read every time a user downloads data from the titan network, and if this data is read from MySQL every time, the read from MySQL may become a bottleneck when there are a lot of requests.

Therefore, we can take this frequently read data and put it into an in-memory database, such as Redis, to speed up the read speed. Also, by using the LRU principle, the data that has not been read for a long time is evicted from Redis to avoid Redis' high memory usage.

# Core components

## Nodes management

All nodes need to join the titan network through the scheduler. The node starts and establishes a communication connection with the scheduler and completes the node login process.

The Scheduler manages the nodes, and the resources on the nodes: storage, bandwidth, compute, etc., by calling the JSON-RPC API interface of the nodes.

On the other hand, the node reports the node's status information, running events, etc. to the scheduler by calling the JSON-RPC API interface of the scheduler.

### Node Type

The Scheduler manages three types of nodes, namely, validator nodes, candidate nodes, and edge nodes. Among them, the validator node and the candidate node belong to L1 node, and the edge node belongs to L2 node.

The hierarchical relationship between the three nodes and the scheduler is shown in the following figure:



#### Candidate nodes

Candidate nodes are L1 nodes, which are fewer in number compared to edge nodes, but require much higher specific resources, such as large storage and bandwidth, as well as good computing power.

L1 nodes need to be responsible for performing most of the computational needs in the titan network, such as running trusted containers. This requires the L1 node to have a strong CPU and large memory.

Compared to L1 nodes, L2 nodes require less computational power, and L2 nodes deal more with access acceleration of assets data, thus tending to bandwidth and storage resources.

#### Validator nodes

Validator node is a candidate node that is elected as validator. validator node differs from candidate node in that the former needs to participate in the sampling process of titan network and is responsible for sampling the assigned nodes and reporting the results to the scheduler.

#### Edge nodes

The Edge node is the L2 node of the titan network and the outermost node of the titan network.

Edge nodes use their bandwidth and storage resources to pull assets data from the source to the node's own storage. When a user requests assets data from the titan network, the titan network assigns the edge nodes with the shortest distance from the user's network based on the user's location information, and the user pulls assets data from these edge nodes.

The shortest network distance means that the user and edge nodes are located in the same metropolitan area network (or, Metro Ethernet), and the data flow between them does not require cross-domain national backbone networks/submarine fiber optic cables, etc. Therefore, the network bandwidth is large, the latency is low, and the data transmission speed is fast.

### Node registration

Edge node, Candidate node to join titan network, need to complete the node registration first.

The node registration process consists of the following tasks:

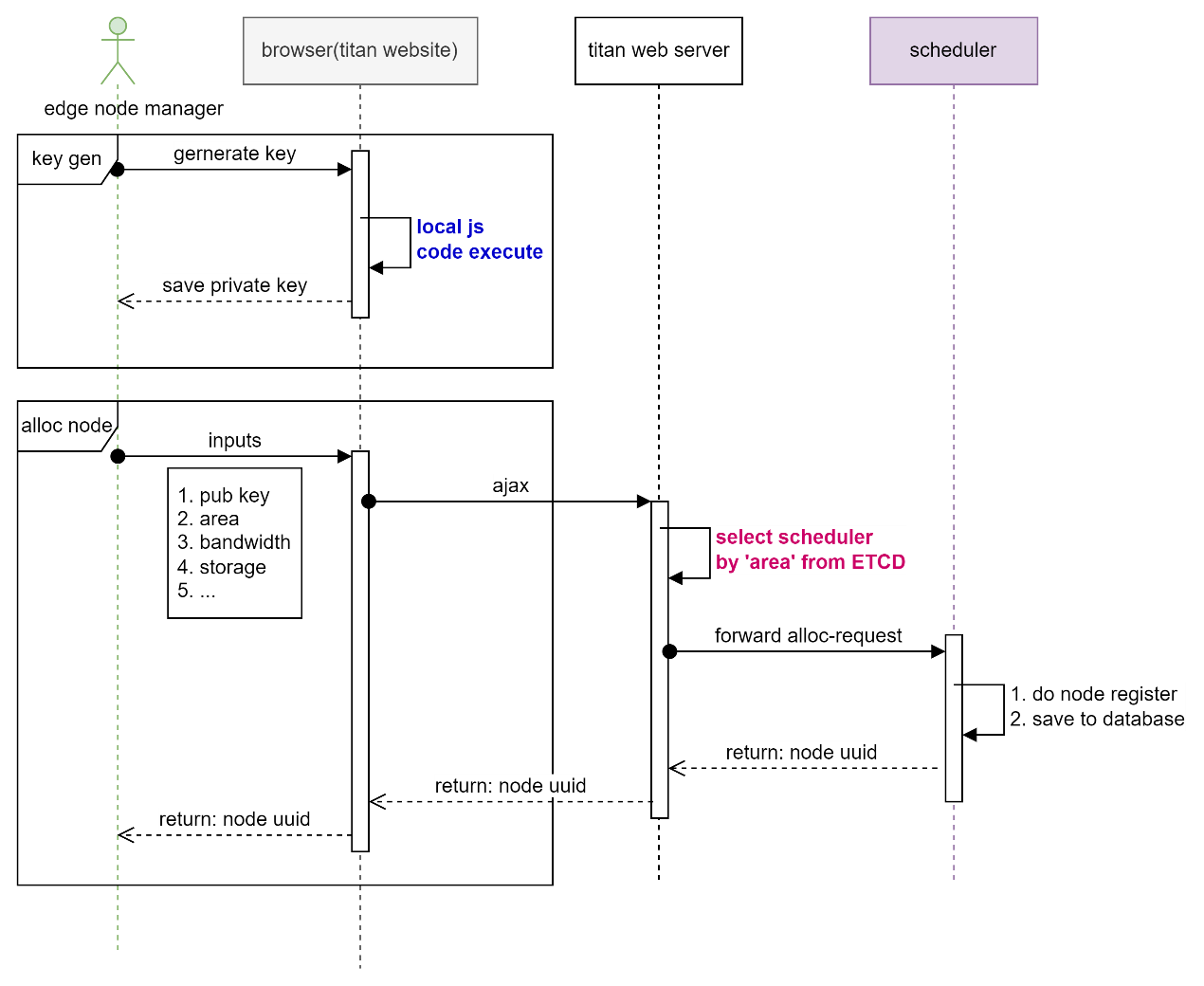
* Locally generated public/private key pair
* Select the area(city) of the node
* Select the quantity of resources to contribute

The node manager saves the private key to local storage.

Then, after completing the above information, click the node registration button, the web application will send the public key and other information to the titan scheduler of the corresponding area in the titan network according to the area(city), and the latter will complete the node registration.

After completing the registration, the node manager needs to import the private key, node's UUID and other information into the node's startup configuration, so that the node can interact with the scheduler through this information when startup.

The node registration process is shown in the following diagram:



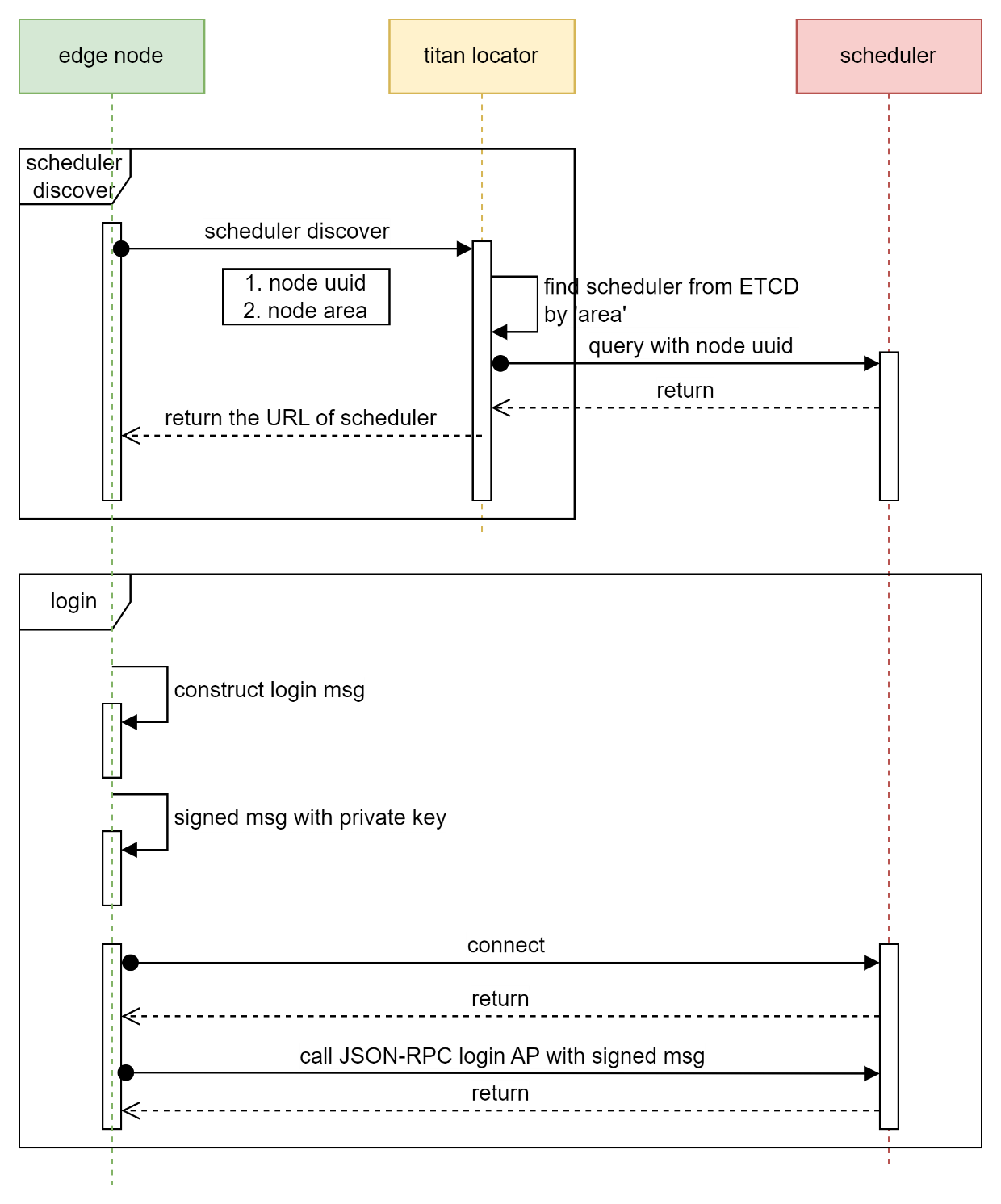
### Node login

Nodes need to be logged into the scheduler in order to join the titan network.

After completing the node registration, the node can get the node ID after registration, use the private key signature, generate the login request message, and login to the titan network.

When logging in, the node first connects to the titan locator to get the IP address information of the titan scheduler where the node is located, and then establishes a connection with the scheduler and sends a login message to complete the login.

The login process is shown in the following diagram:



### Node reliability score

The Scheduler assigns a property called reliability score to each node. The score is based on the online status of the node: the longer the node is online, the higher the score; the more frequently the node is offline, the lower the score.

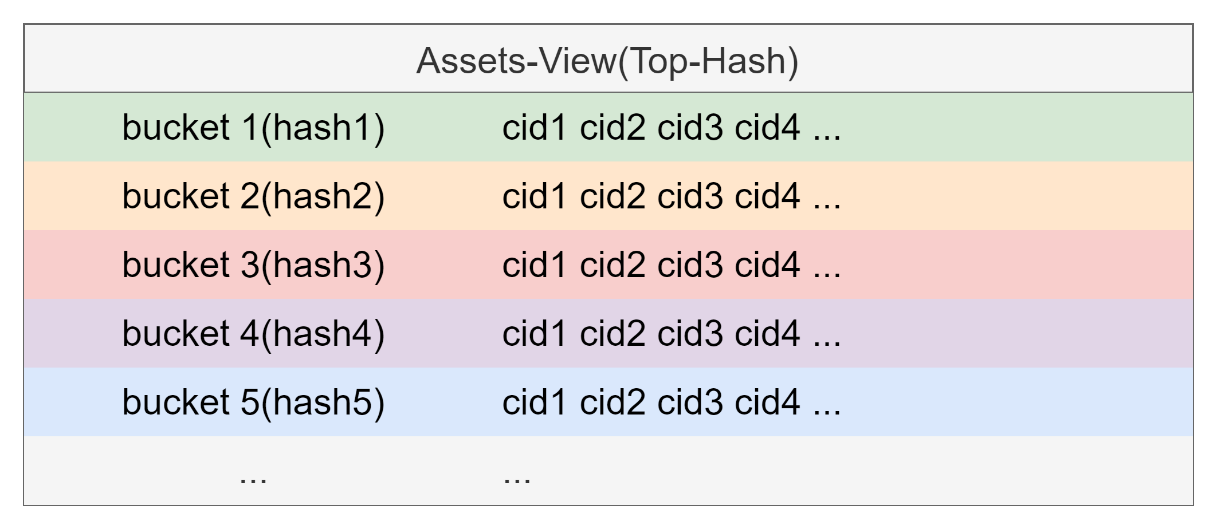
When Scheduler selects edge nodes for storing asset data, it gives preference to edge nodes with high reliability scores.

### Node data view

The Scheduler keeps track of Assets on each node in a data view. The node itself records all its Assets in a data view, and the Scheduler periodically aligns the data view with the node to ensure that the Assets information is consistent.

If the Assets information is inconsistent, for example, if the scheduler thinks that data for an Asset exists on a node, but the node does not have data for that Asset, then when the scheduler assigns that node to a user requesting that Asset resource, the user will fail to pull the Asset data from that node.

The diagram of the data view:



The data view takes all Assets of a node and divides them into multiple buckets. The hash of each bucket is obtained by concatenating the CIDs of all the Assets in the bucket and hashing them again.

The hash of each bucket is obtained by concatenating the CIDs of all the Assets in the bucket and then hashing them again to obtain the hash of the entire data view.

When Scheduler and nodes synchronize data views, they first compare Top-hash, and only when Top-hash is different, they compare the hash of each bucket to try to find the bucket with different hash, and when the inconsistent bucket is identified, they further compare the CIDs within the bucket until they find the inconsistent CID.

By dividing the Assets into multiple buckets, we can quickly find the inconsistency between two data views.

Scheduler and node will maintain their own data view, for example, when an Asset is added, or deleted, the bucket in the data view will change, and the hash of the bucket and the Top-hash will need to be recalculated.

### Data view synchronization

Scheduler records the hash information of Asset data on each node in the database and synchronizes with the node periodically to ensure that the node's data view is correct.

The synchronization is initiated by the scheduler, and the titan scheduler is currently set to synchronize once a day, and this synchronization period is configurable.

To speed up the synchronization process, the scheduler and the node divide the hash of the Asset data into a number of buckets, and the hash of each bucket is obtained by combining the hashes of all the Asset data in the bucket, while the top hash is obtained by combining the hashes of all the buckets.

When synchronizing, first compare the top hash, if it is consistent, it means the data view is consistent and the synchronization process is finished. If top hash is inconsistent, then compare bucket hash by bucket, find out the bucket with inconsistent hash, and then find the inconsistent Asset data from that bucket.

The node needs to fix inconsistent Asset data. to keep the data view and scheduler consistent.

## Assets management

The core goal of the Titan network is to utilize the bandwidth, storage, and computing resources of a large number of nodes within the network to cache the data demanded by users to the node with the shortest distance from the user's network, accelerating access to data and improving the user experience.

Each piece of data, in the titan network is called an asset.

Assets management is the core system of titan scheduler, responsible for the full lifecycle process management of assets.

### Asset

Asset is the basic data unit in the titan network, which manages many assets. titan scheduler's database records information about each asset, such as the status of the asset, which nodes have copies of the asset, etc.

Multiple copies of different assets exist in each node's storage.

Users pull data of an asset by requesting it from the titan network.

Titan network administrators can manage Assets through the APIs provided by the scheduler.

Three core APIs for Assets management are listed below.

#### Create Asset

Create an Asset via this API.

Among the parameters to be specified:

* The CID of the Asset
* Asset's reliability requirement (replicas count)
* The download bandwidth

The Scheduler checks if the asset specified by the CID already exists, and if it does, the scheduler returns basic information about the asset, such as its current status, creation time, number of replicas, etc.

Asset's source, if it is an HTTP source, the scheduler will use the HEAD command to check the status of the asset in the source, if an error occurs, the scheduler cannot create the asset object.

If source is an IPFS source, the scheduler does not check the status of the asset at source, this is because IPFS sources do not have an API interface similar to HTTP HEAD for quick checking of data status.

#### Query Asset

Through this API, query the basic information of the asset, such as the current status of the asset, the creation time, the number of replica, etc.

#### Delete Asset

Remove the Asset from the titan network through this API.

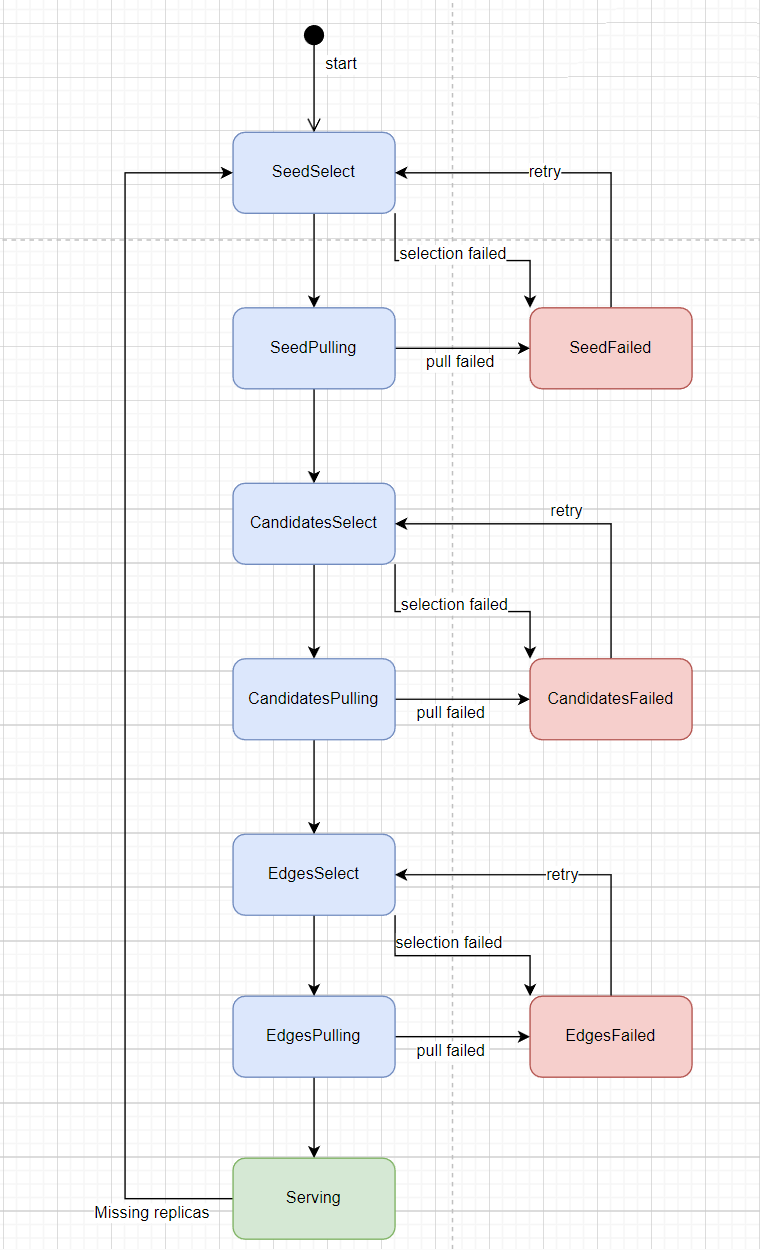
The deletion of Asset, including:

* Scheduler requests node to delete asset data and release storage resources
* Scheduler removes asset-related information from the database
* Scheduler logs asset deletion event
* Since the node deleted the asset data, the node needs to update the local data view
* Scheduler updates the node's data view simultaneously with the node's local data view

### Asset's state machine

After the Asset is created, the scheduler initiates the lifecycle management process for the asset.

Each asset has its own state machine, and the state transitions are shown in the following figure:



State of Asset:

* Seed-Select: is the initial state of the asset object, the scheduler selects the L1 node of N, N is configurable, the default is 1. This L1 node of N, as the seed node, pulls the asset data from source. The reason why it is called seed node is that at this time the titan network does not have the data of this asset, it needs to pull it from source, after the seed node is pulled, the titan network will have the data of this asset.
* Seeds-pulling：the selected seed L1 node starts pulling the asset data from source. This pulling process can take a long time, depending on the downlink bandwidth of the source server, and the size of the data.
* Candidates-select: the Scheduler needs to distribute this asset data to more L1 nodes in order to avoid pulling the asset data from the source server again when a single L1 node fails. Therefore, the scheduler needs to select more candidate nodes to distribute the asset data to these L1 nodes next.
* Candidates-pulling: the selected candidate node pulls the asset data from the seed L1 node. Note that the asset data is pulled from the seed L1 node, not from the source. That is, the asset data at this point originates from the titan network itself, not from an external source.
* Edges-select: after the L1 node has finished distributing the asset data, the scheduler then distributes the asset data to a number of edge nodes. the Scheduler needs to select the edge nodes, how many edge nodes to select, and which edge nodes to select are dynamically changing. the higher the reliability requirement of the asset, the more the scheduler selects the higher the reliability requirement of the asset, the higher the number of edge nodes selected by the scheduler, and the higher the reliability score of the edge nodes, the higher the possibility of being selected. The goal of this phase of the Scheduler is: to select enough edge nodes, and the total reliability of these selected nodes, needs to meet the reliability requirement of the asset. Another factor that the scheduler needs to consider is the download bandwidth requirement of the asset. The total bandwidth capacity of these selected edge nodes needs to meet the download bandwidth requirement of the asset.
* Edges-pulling: the selected edge node, pulls the asset data from the L1 node. scheduler makes the edge nodes pull from the L1 node in batches, i.e., if there are N L1 nodes, then each batch is N edge nodes. the purpose of this is to let some of the edge nodes finish pulling as soon as possible. If all edge nodes go to the L1 node to pull at the same time, each edge node gets a smaller bandwidth of the L1 node.
* Serving: when the distribution of asset data is finished, the status of the asset enters the service state, and then the user can pull the asset data from the edge node of the asset. In fact, since the Edges-pulling phase, some of the edge nodes finish pulling the asset from the L1 node first, so the user can already pull the asset data from the edge node before the Edges-pulling phase is finished.

### Asset availability

The Scheduler continuously monitors the status of the asset.

If some of the edge nodes of the asset fail and cause the reliability of the asset to decrease, or, cause the download bandwidth of the asset to be insufficient, then the scheduler will make the asset state switch from Serving state to Edges-select state and execute the process of Edges-select phase so that more edge nodes are selected to store the asset, thus restoring the reliability and download bandwidth of the asset.

## Validation (spot-checking)

The main responsibility of the validation(spot-checking) system in Scheduler is to periodically verify the storage and bandwidth resources of the nodes in the titan network, according to the set period, including:

* Periodically elects validators according to a set election cycle
* Randomly assign validator to the sampled nodes at the beginning of each check
* Sample results are verified and recorded, and these records are used as a basis for evaluating node contributions

The purpose of the spot-checking is mainly two:

* Storage check: check whether a node stores an IPFS block or a common file range correctly. scheduler gets a random number of seeds from the Filecoin main network in each round of sampling, so the block/range index is random and different for each round and for each node, which makes the node have to save all blocks/ranges.
* Bandwidth check: check the node's uplink bandwidth resources, this process occurs along with the storage check, because during the storage check, the node needs to send block/range to the validator, so the validator can calculate the node's sending speed and thus evaluate the node's uplink bandwidth.

### Validator

In the validation(spot-checking) process, data sending and data Hash (CID) calculation are required, which requires a lot of bandwidth and computation considering the large number of titan network nodes. As the focal point of the spot-checking process, the validator node needs to have a large bandwidth and computing power to complete the spot-checking process of the titan network efficiently.

If a validator fails, it may cause the edge node to have no validator for spot-checking; and the unstable bandwidth resources of the validator may cause the results of spot-checking to be wrong.

Therefore, not only does the validator have greater resource requirements, but its resource stability requirements are also higher.

On the other hand, as validator, the node is able to get more rewards.

For each round of spot-checking, we select a number (N) of valid validators from the validators (offline validators are considered invalid), and N is configurable. The default is to select all online validators.

Then, we assign edge nodes to each validator: according to the resources of the validator, we assign M edge nodes to him randomly, and these edge nodes are the edge nodes that the validator needs to spot-checking in this round.

When assigning edge nodes, the main consideration is that the sum of the uplink bandwidth of all edge nodes assigned to a certain validator is not greater than 70% of the validator's downlink bandwidth, so that the validator's downlink bandwidth leaves enough margin to avoid deviations in spot-checking results due to fluctuations in the validator's downlink bandwidth.

Since the random pairing of validator and edge nodes is re-paired in each round, a certain validator does not have the same edge nodes to be spot-checking in each round, and such an arrangement can avoid collusion between validator and edge nodes to cheat.

### Validator election

The Validator node is randomly elected from the Candidate nodes.

The Titan network sets an election cycle P. At the beginning of each cycle, a ratio of nodes from all candidate nodes is selected as validator nodes. This ratio R is variable and is adjusted appropriately according to the development of the titan network.

For example, in the early stage of titan network development, when there are fewer candidate nodes, R may be 100%; when the number of candidate nodes is larger, R will decrease.

Validator nodes need to be randomly elected in the hope that this will make it more difficult to spot check cheating.

### Pairing

At the beginning of each round, the scheduler randomly pairs the validator and edge nodes. The purpose of this is to avoid collusion between the validator and the edge nodes, and to avoid the same validator to the edge nodes and the biased results due to the validator.

We define the downlink bandwidth window v-window of the validator: each v-window has the same downlink bandwidth B, and B is configurable. The larger the downlink bandwidth of the validator, the more v-windows it has.

The process complexity of pairing is O(N+M), where N is the number of all v-window and M is the number of sampled nodes (edge nodes and candidate nodes).

### Spot-checking

At the beginning of the spot check, the scheduler calls the API of the edge node to inform the edge node of the parameters for this round of spot checks. The parameters include the random number Seed, the IP address information of the validator, etc.

Where the random number Seed is a distributed random number obtained by the scheduler from the Filecoin main network and is re-fetched by the scheduler for each round of spot checks.

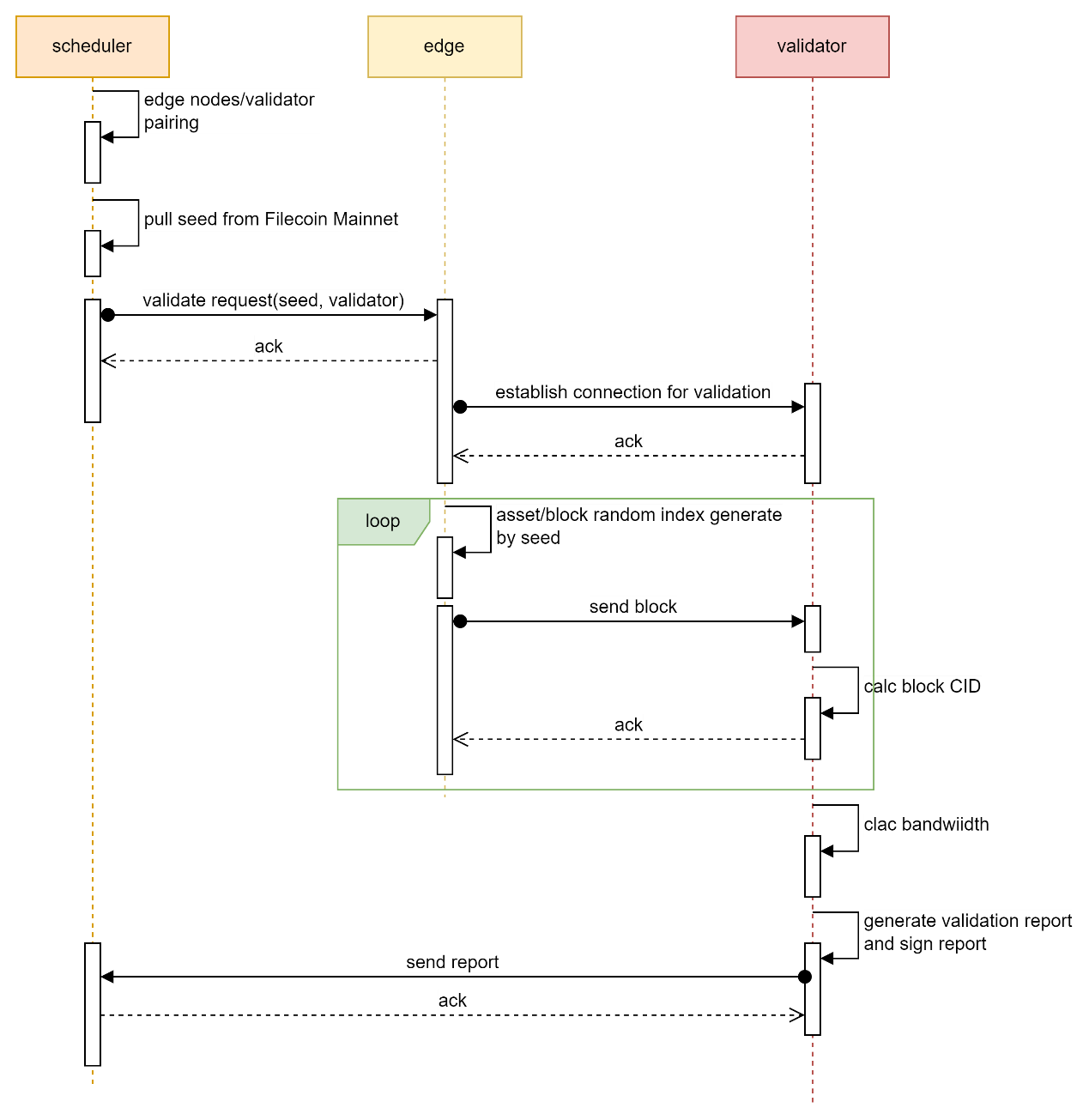
The Edge node establishes a communication connection with the validator for spot-checking based on the IP address of the validator.

After the communication connection is established, the edge node generates a random number with the random number Seed and gets a number of numbers from the random number as index, selects the bucket in the local data view and the asset object (carfile) in the bucket, and finally selects the block in the asset object (carfile) and sends the block to the validator.

After receiving the block data, Validator calculates the CID of the block. If the edge node sends multiple blocks, it receives and calculates the CID of each block one by one.

When the Edge node finishes sending, the validator gets the cid of all blocks sent by the edge node and calculates the average sending speed of the edge node, and then signs and sends this information to the scheduler as the spot-checking result of that edge node.

The process is illustrated below:



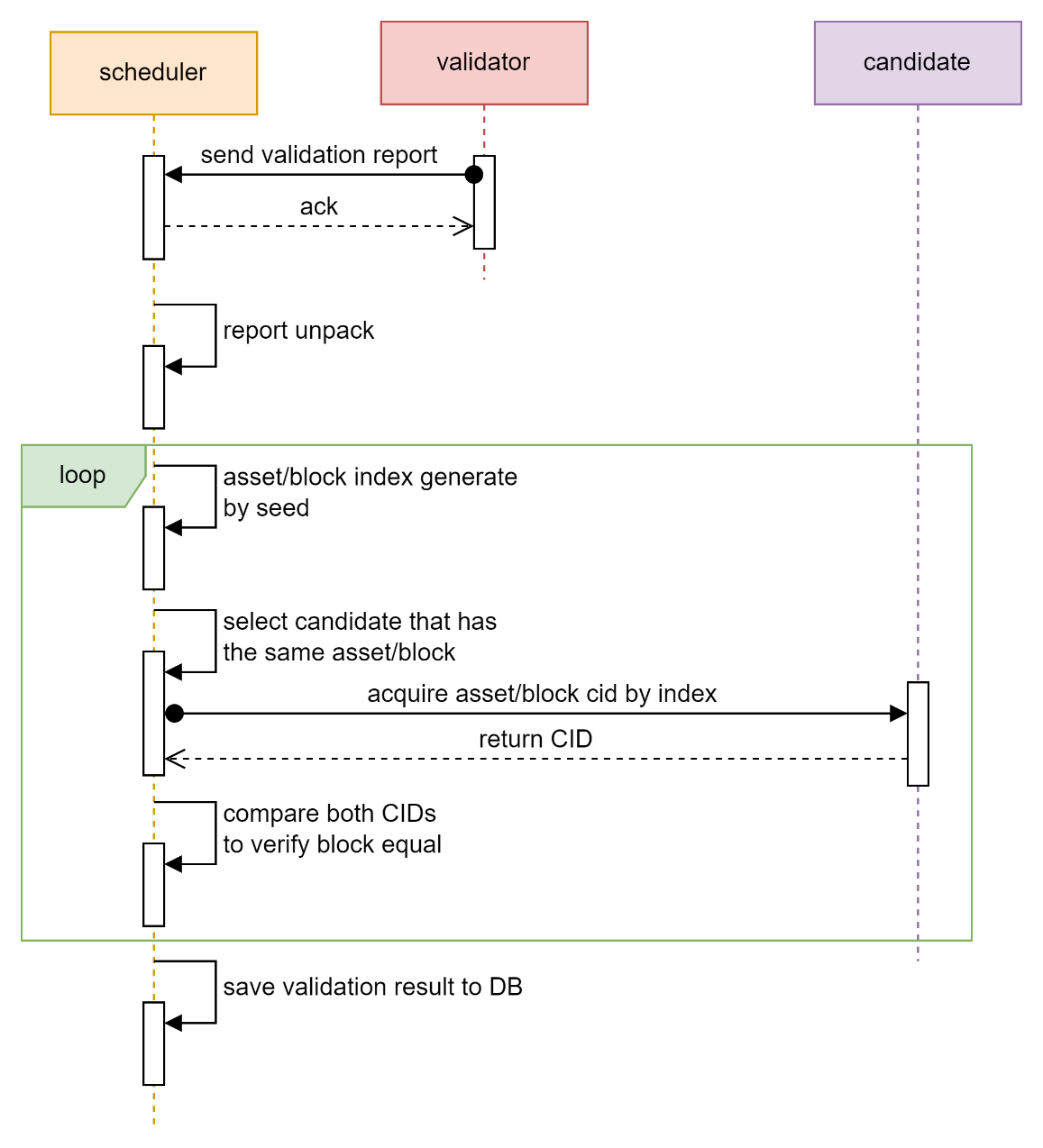
### Spot-check results processing

After the Scheduler receives the spot check results from the validator, it verifies the spot check results using the validator's pub key to ensure that the spot check results are from the validator.

The Scheduler server itself does not store the block data, but the scheduler knows which L1 nodes have the data, so the scheduler requests the L1 nodes with the same block data to send back the CID of that block and compare it with the CID in the spot-check result. If the two CIDs agree, the block stored by the edge node is correct, and vice versa, the block data of the edge node is wrong.

The spot check result contains, in addition to the CID of one or more blocks, the average speed at which the edge node sends data from these blocks. Scheduler records this average speed as the bandwidth contribution of the edge node for this time period.

The process is illustrated below:



### Bandwidth spot-check skip

Since the spot check verifies not only the correctness of the storage but also the bandwidth, by having the edge node continuously send a number of blocks, the validator is able to measure the uplink bandwidth of the edge node.

However, if at this time, the edge node is sending asset data to the user, and the bandwidth spot check is crowding the uplink bandwidth resources, it will inevitably affect the speed of the edge node sending data to the user; in addition, since the edge node is sending data to the user, the bandwidth obtained from the spot check measurement is not the real uplink bandwidth of the edge node.

Therefore, when the edge node is sending data to the user, the edge node receives a spot check command from the scheduler, and when the edge node and the validator establish a spot check communication connection, the validator is informed via parameters that the edge node needs a bandwidth spot check exemption(skip) for storage spot checks only.

At the end of the current round of spot check, scheduler checks whether the bandwidth spot check exemption for the edge node is valid: scheduler checks whether the edge node is assigned to a user during that time period, and if so, the exemption request is considered valid. Conversely, the exemption request is considered invalid.

When an exemption request is valid, although the edge node does not perform a bandwidth spot check, scheduler will take the bandwidth measurements of that edge node for the last several times and give the edge node a corresponding bandwidth bonus.

If the exemption is considered invalid, then the scheduler treats the edge node as absent from the bandwidth spot check and the edge node does not get the bandwidth bonus.

## User request system

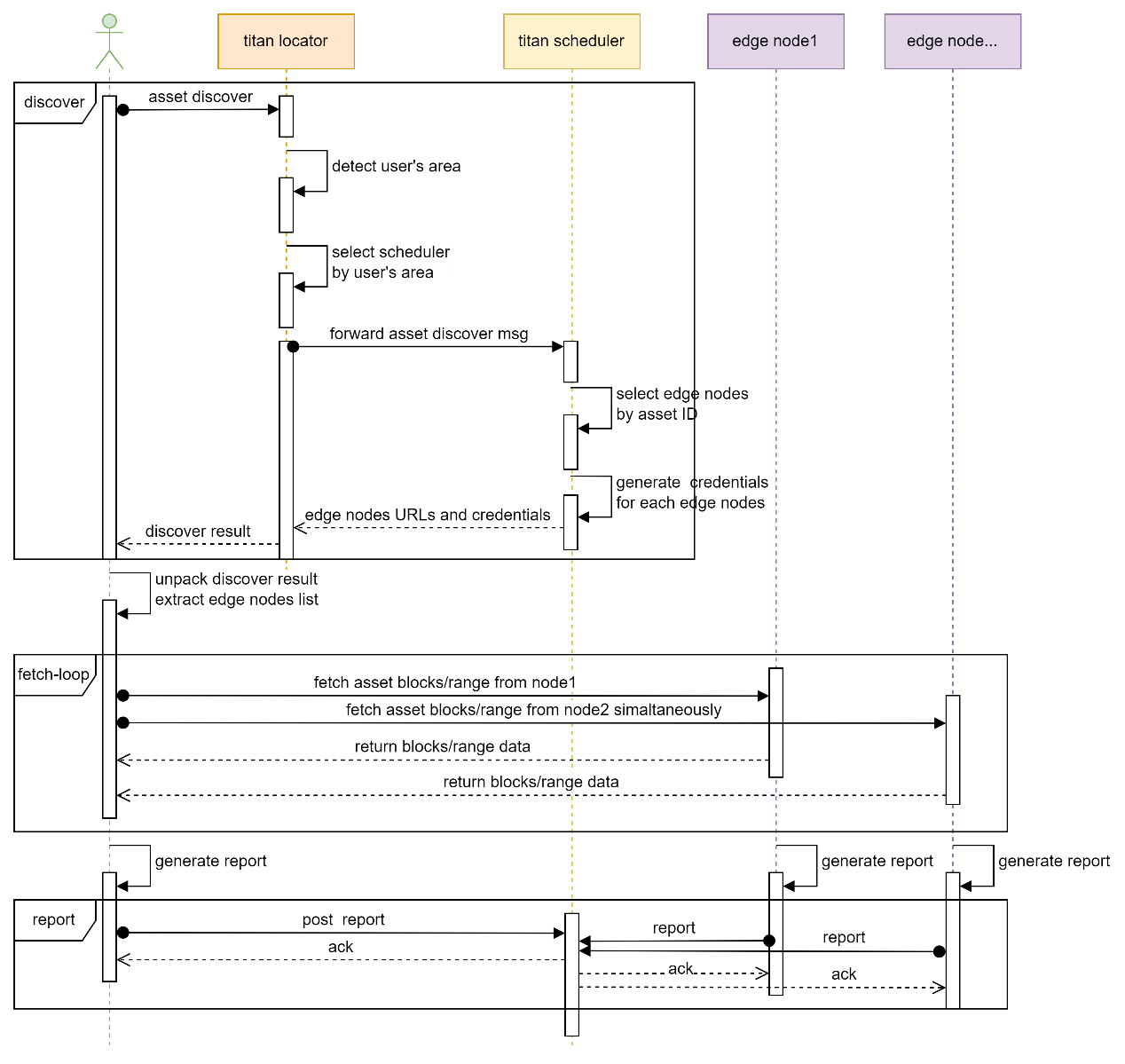
The goal of the Titan network is to accelerate data access speed and enhance user experience.

The Titan Network approach is to use the node resources of the titan subnet of multiple areas(cities) to send the asset data to the user via the shortest distance.

When a user requests for asset data, the titan network needs to assign the edge nodes of the same area(city) to the user based on the user's IP address and other location information, and then the user can pull the asset data from these edge nodes.

Since the edge nodes providing data and users are in the same AREA, i.e., the same MAN, data transmission does not need to go through busy national backbone networks or submarine fiber optic cables, so the transmission bandwidth is large and the latency is low, so the data acquisition speed is fast and the user's data acquisition experience is enhanced.

The process of users acquiring assets from titan network is as follows:



In summary, the user obtains asset data from the titan network in three steps:

* Asset discovery: The user requests the titan locator, which gets the user's area based on the user's IP address, etc., and then forwards the user request to the titan scheduler of that area, which returns to the user a list of URLs of several nodes with that asset data, as well as a list of the nodes' access credentials.
* Asset pulling: Once users get the list of edge nodes, they can pull resource data through the edge nodes, and users can concurrently pull the process: that is, pull different parts of the resource data from multiple edge nodes at the same time to speed up the data pull.
* Report: After the pull is complete, the user submits the ASSET data pull report to the scheduler.

### Asset access credentials

Asset access credentials have two roles, the first is the authorization to download resources, without the credentials, it is not possible to download resources from the edge node; the second is that the edge node needs to generate workload proofs based on the credentials, indicating that the edge node provides the download service of the resources specified by the credentials.

Asset access credentials are blocks of data that are encrypted by the titan scheduler server using the public key of the edge node and signed with the private key of the scheduler. The titan scheduler uses its own private key signature to ensure that the credentials are issued by the scheduler.

The data block of the credential, contains and is not limited to the following:

* Unique ID of the credential
* Carfile of the credential
* Download speed limit of the credential
* Client ID, the unique ID of the user side
* Expiration date of the credential, etc.

When the Scheduler returns the list of edge nodes to the user, the scheduler generates an access credential for each edge node.

The user needs to use the credentials in order to pull data from the edge node. The edge node checks the validity of the credentials, and if the credentials are invalid, the edge node rejects the user's pull request.

## Workload Management System

Workload management, the main purpose of which is to receive workload reports from nodes in the titan network and verify the validity of the reports. Based on the valid workload reports, the workloads of the nodes are recorded to the database. These valid workload records are used as a basis for evaluating the node's contribution to the titan network.

The user sends a report to the scheduler when the pulling of data from each edge node is completed.

The workload reports submitted by the edge nodes at regular intervals contain and are not limited to the following:

* Unique ID of the credential
* User ID
* Download speed
* Amount of data downloaded
* Download start time
* Download end time, etc.

A final valid workload report is recognized if and only if both the edge node and the user have submitted the same and valid workload proof. That is, a workload report submitted by an edge node, or a user, alone will not be considered a valid workload, because it will not be used as a basis for evaluating the contribution of the edge node either.

## FVM Smart Contracts

The Titan network uses the FVM contract as follows:

* Stores into the contract the CIDs of the carfile that are periodically generated about the contribution records (proof of workload). Thus, the contract provides backtracking for the titan network, and anyone is able to obtain the corresponding carfile by these CIDs, and analyze and verify the correctness of the contribution records within the carfile, either manually or by using tools.
* Utilizes the FVM DATA-DAO contract to enable titan network data backup to Filecoin storage provider.
* Accept CDN deals using FVM contracts.

### Node Contribution Record FVM Contract

Titan Network deploys a contract called "node contribution record" on FVM. The main role of this contract is to continuously keep the added CIDs, which are carfile CIDs, corresponding to one carfile after another, and the content of each carfile is about the titan network's The content of each carfile is a record of the contributions of nodes in the titan network.

The Scheduler periodically packs the contribution records of nodes, including spot-check records, downloads workload certificates, generates carfiles and saves them to the Filecoin DC deal (DC Sectors). Meanwhile, the scheduler will submit the cid of the generated carfile to the FVM contract of the titan network: the contribution contract. Anyone can get the cid of the contribution record carfile for a certain time period through the interface of the contract, and then get the carfile from Filecoin SP through the cid, and review the carfile in the node contribution records.

### Cold Backup FVM DATA-DAO Contract

The data in the Titan network is mainly used as hot cache data. In order to improve the reliability of data, titan network regularly backs up data to Filecoin Storage Provider. Titan network uses FVM's DATA-DAO contract to issue storage orders for backup data, and Filecoin's Storage Provider receives and completes storage orders through the contract. The Storage Provider receives and fulfills the storage order through the contract. The Storage Provider is rewarded for completing the storage order.

When the titan network needs to restore data, the titan network needs to retrieve data from the Storage Provider that completed the storage order, so the Storage Provider needs to provide enough bandwidth so that the titan network can retrieve as soon as possible. titan network through the FVM DATA-DAO contract The Filecoin Storage Provider receives and completes the retrieve order through the contract. the Storage Provider is rewarded for completing the retrieve order.