Ethereum Sharded Storage Protocol (SSTORAGE)

The `sstorage` protocol runs on top of [RLPx], facilitating the exchange of Web3q sharded storage between peers. The protocol is an optional extension for peers supporting sharded storage.

The current version is `sstorage/1`.

Overview

The `sstorage` protocol's goal is to synchronize sharded storage content from peers. The `sstorage` protocol does not take part in chain maintenance (block and transaction propagation); and it is **meant to be run side-by-side with the** `eth` **protocol**, not standalone (e.g. chain progression is announced via `eth`).

The `sstorage` protocol is supposed to be run after the node is synchronized to the latest block. The `sstorage` protocol itself is simplistic by design, it supports retrieving a contiguous segment of kvs from the peers' sharded storage. Those kvs can be verified by kv metadata stored in the sharded storage system contracts. After retrieving and verifying all kvs, they will be saved to sharded storage files locally. An additional complexity must be aware of, is that kv content is ephemeral and changes with the progress of the chain, so a syncer may receive kvs that mismatch with local kv metadata. In this case, the syncer will fetch the data from other peers until the download kvs are matched. Another special case is that a node may crash with inconsistent local kvs (i.e. Those kvs are not empty, but are inconsistent with metadata).Syncers need to support identifying inconsistent kv segments and heal kv inconsistencies by retrieving the kvs via `sstorage` protocol.

Relation to `eth`

The `sstorage` protocol is a *dependent satellite* of `eth` (i.e. to run `sstorage`, you need to run `eth` too), not a fully standalone protocol. This is a deliberate design decision:

- `sstorage` is meant to be a bootstrap aid for newly joining full nodes with sharded storage function enabled. `sstorage` protocol only keeps `eth` peers which also enables sharded storage with kvs the node needed.
- `eth` already contains well-established chain and fork negotiation mechanisms, as well as remote peer staleness detection during sync. By running both protocols side-by-side, `sstorage` can benefit from all these mechanisms without having to duplicate them.

In order to let the `sstorage` protocol know which storage shards are supported by the peer node, the storage shards information needs to be added to the protocol's `handshake` function.
In order to make sure every storage shard has enough peers to fetch kvs, the `minSstoragePeers` will be set for each shard, so a new peer will ignore `MaxPeers` setting for eth protocol and register, only if that peer contains a storage shard which the local node having the same shard, and do not reach the `minSstoragePeers` limit.

Synchronization algorithm

When starting a node with sharded storage enabled, it will check if the local storage content is correct. If any content is missing or not correct, a sync task will be added for that sharded storage file to sync data from peers. So the sstorage synchronization task will be added under the following conditions:

- Starting a node with new shards;
- Restarting a node that partially downloads the data of some shards
- Starting an existing node which failed to flush kv content from memory to sharded storage file when the node stops (e.g., the node crashes because of OOM).

The caveat of the sstorage synchronization is that the target data constantly changes (as new blocks arrive). This is not a problem because we will discard mismatched kvs and try to download the kvs of another peer. If the kvs are unavailable from all peers (likely happen when the node is a bit behind the network), the syncer will try to download it later or it will automatically synchronize the kvs by executing new blocks.

In the case of inconsistency of local kvs with metadata (likely to happen when the node crashes), we can self-heal.

Request ID

Every on-demand request message contains a `reqID` field, which is simply returned by the server in the corresponding reply message. This helps matching replies for requests on the client side so that each reply doesn't need to be matched against each pending request.

Protocol Messages

ShardsMsg(0x00)

`[ContractShardsList: [Contract: P, ShardIds: [ShardId: P, ...]]]`

Request is used by sstorage protocol handshake to share the shards the local node supported to the remote node.

- `ContractShardsList`: Contract and ShardIds pairs supported by the local node
    - `Contract`: Sharded storage system contract the local supported
    - `ShardIds`: List of shardIds supported
        - `ShardId`: Shard id supported

Notes:

- The response is not needed for this request.

GetKVRangeMsg(0x01)

`[reqID: P, contract: P, shardId: P, origin: P, limit: P, bytes: P]`

Requests an unknown number of KVs starting at the specified KV index and capped by the maximum allowed response size in bytes. The intended purpose of this message is to fetch a large number of KVs from a remote node.

- `reqID`: Request ID to match up responses with
- `contract`: Sharded storage system contract related to kv retrieve
- `shardId`: Shard ID related to kv retrieve
- `origin`: KV index of the first to retrieve
- `limit`: KV index after which to stop serving data
- `bytes`: Soft limit at which to stop returning data

Notes:

- Nodes **must** always respond to the query.
- If the node does **not** have the kv for the requested contract or shard id, it **must** return an empty reply. It is the responsibility of the caller to query kvs from the sharded storage file, not including content saved in the memory.
- The responding node is allowed to return **less** data than requested (own QoS limits), but the node must return at least one kv unless no kv exists between `origin` and `limit`.

Rationale:

- The response is capped by byte size and not by number of KV, because it makes the network traffic more deterministic..

Caveats:

- When requesting KVs from a starting kv index, malicious nodes may skip ahead and return a gapped reply. Such a reply would cause sync to finish early with a lot of missing data. If too many kvs are dropped, the peer will be dropped to prevent this attack.
- For the kvs being dropped, the kv index will be saved to the healing list to retrieve again.

KVRangeMsg(0x02)

`[reqID: P, contract: B, shardId: P, kvs: [[idx: P, data: B], ...]]`

Returns a number of kvs for the requested kv range (i.e. list of kv). GetKVRange requests will use this message as a response.

- `reqID`: ID of the request this is a response for
- `contract`: Sharded storage system contract related to kv retrieve
- `shardId`: Shard ID related to kv retrieve
- `kvs`: List of kvs
  - `idx`: index of the kv
  - `data`: Data content of the kv

GetKVs (0x03)

`[reqID: P, contract: B, shardId: P, kvList: [idx: P, ...]]`

Requests a list of kvs using a list of kv indexes. The intended purpose of this message is to fetch a large number of kvs from a remote node and refill a sharded storage file locally.

- `reqID`: Request ID to match up responses with
- `contract`: Sharded storage system contract related to kv retrieve

- `shardId`: Shard ID related to kv retrieve
- `kvList`: A list of kv index

Notes:

- Nodes **must** always respond to the query.
- If the node does **not** have the kv for the requested kv index, it **must** return an empty reply. It is the responsibility of the caller to query kvs from the sharded storage file, not including content saved in the memory.
- The responding node is allowed to return **less** data than requested, but the node must return at least one kv, unless none exists.

Rationale:

- The response is capped by the number of kvs set locally, because it makes the network traffic more deterministic.

Caveats:

- When requesting a range of kvs from a kv index list, malicious nodes may return incorrect kv content or missing some of them. Such a reply would cause the local node to spend a lot of time to verify the kv contents and drop them. So if too many kvs are dropped, the peer will be dropped to prevent this attack.
- For the kvs being dropped, the kv index will be saved to the healing list to retrieve again.

KVs (0x04)

`[reqID: P, contract: B, shardId: P, kvs: [[idx: P, data: B], ...]]`

Returns a number of kvs for the requested kv index (i.e. list of kv). GetKVs requests will use this message as a response.

- `reqID`: ID of the request this is a response for
- `contract`: Sharded storage system contract related to kv retrieve
- `shardId`: Shard ID related to kv retrieve
- `kvs`: List of kvs
    - `idx`: index of the kv
    - `data`: Data content of the kv

Change Log

sstorage/1 (July 2022)

Version 1 was the introduction of the sharded storage protocol.