11 may 2017
Ben
v1.0

Sawtooth+Burrow Integration

The primary problems to solve in order to integrate the Burrow EVM into Sawtooth are:

1. Define an efficient mapping between EVM World State addresses and Sawtooth Global State addresses.
2. Define an efficient method for maintaining accounts and account storage in Sawtooth Global State.

For more info on Ethereum addressing, see section 4 of the Ethereum yellow paper:
https://ethereum.github.io/yellowpaper/paper.pdf

For more info on Ethereum Accounts see:
http://ethdocs.org/en/latest/account-management.html

For more info on Sawtooth Global State see:
http://intelledger.github.io/architecture/global_state.html

Facts

Relevant facts about both systems are aggregated below as I learn them.

Addressing Facts

- Account addresses in the EVM are 20 bytes long.
- Storage addresses in Burrow are 20+32 bytes long (address+key)
- Global state addresses in Sawtooth are 3+32 bytes long (namespace+address)

∀ account (20 byte).
∃ key-value store (32,32) byte

State Facts

- Transaction Families can claim one or more 3 byte namespaces, each of which serves as the root of a Merkle-Radix trie with 32 byte addresses
- EVM accounts are stored on-chain at the account address
- EVM accounts have an associated on-chain key-value storage
- Storage used by EVM accounts is only limited by:
  - Keys must be 32 bytes
  - Values must be 32 bytes

in Burrow storage is stored in db
Storage Roots of Accounts are hashed into block commit result.

Naive Solutions

Below are several naive solutions for solving the addressing and state integration problem. None of these solutions is perfect but they should help drive further discussion.

1. One-Namespace, One-Address

- The Burrow EVM TP uses one prefix in Sawtooth Global State.
- All data associated with an account (including storage) is stored at the address formed by concatenating the EVM prefix with the account address and padding the address with 0s as necessary.
  - Eg., "abcdef" + account_address + "000..."
- Data is serialized using protobuf messages such as:

```
message EvmEntry {
      EvmAccount account = 1;
      repeated EvmStorage storage = 2;
}

message EvmAccount {
      string address = 1;
      int64 balance = 2;
      bytes code = 3;
      int64 nonce = 4;
      bytes other = 5;
}

message EvmStorage {
      string key = 1;
      bytes data = 2;
}
```

- When the transaction processor receives a transaction, the data stored at the account address associated with the transaction is retrieved from the Sawtooth Global State Context (SGSC) and cached at the Transaction Processor. The EVM interacts with this data directly.
- Additional addresses that are required are loaded from the SGSC as needed and cached at the Transaction Processor.
- When the EVM has finished processing the transaction, all addresses that have had their data modified are synchronized back to the SGSC.

Pros

- Conceptually simple to understand and implement

Cons

- Must transfer all data associated with an account for each transaction that is processed (i.e., slow)

2. Two-Namespaces, Many Addresses

- Two namespaces are used: one for accounts and one for account storage

- Accounts are stored in a serialized form at the address formed by concatenating the account namespace prefix with the account address and padding the address with 0s.
- Storage associated with an account is stored at the address formed by concatenating the account storage namespace prefix and a 32 byte hash of the address and key.
- The actual data stored in the SGSC at the address is a serialized list of key-value pairs to handle hash collisions, which is hidden from the EVM.

Pros

- Less data is transferred for each transaction

Cons

- Data stored in the Sawtooth Global State is less ordered, may be difficult to parallelize.

3. One-Namespace, Account Address + Storage Key Hash

- The Burrow EVM TP uses one prefix in Sawtooth Global State.
- Accounts are stored in a serialized form at the address formed by concatenating the EVM prefix with the account address and padding the address with 0s:
  - "abcdef" + account_address + "000..."
- Storage associated with an account is stored at the address formed by concatenating the prefix, the account_address, and a hash of the key. The actual data stored at the address is a serialized list of key-value pairs to handle hash collisions.
- When the transaction processor receives a transaction, just the account associated with the transaction is retrieved from the SGSC.
- Account storage is loaded and cached as needed. The map stored in SGSC is hidden from the EVM.
- When the EVM has finished processing the transaction, all addresses that have had their data modified are synchronized back to the SGSC.

Pros

- Less data is transferred for each transaction
- All data associated with an account is guaranteed to be stored under prefix+account_address, which may make parallel scheduling easier (using longer address prefixing)

Cons

- Possibly a lot of hash collisions when using a lot of storage
- Two different types of data stored under the same namespace (probably fine)

Notes

Open Questions

- What should the txid, origin, and params arguments be when instantiating the EVM?
- Should a new EVM be created for each transaction handled by the Burrow EVM TP?

- Where should caching be implemented and to what extent?

yes, in sawtooth-burrow
↳ per context
Q: can multiple Apply requests arrive in one context?

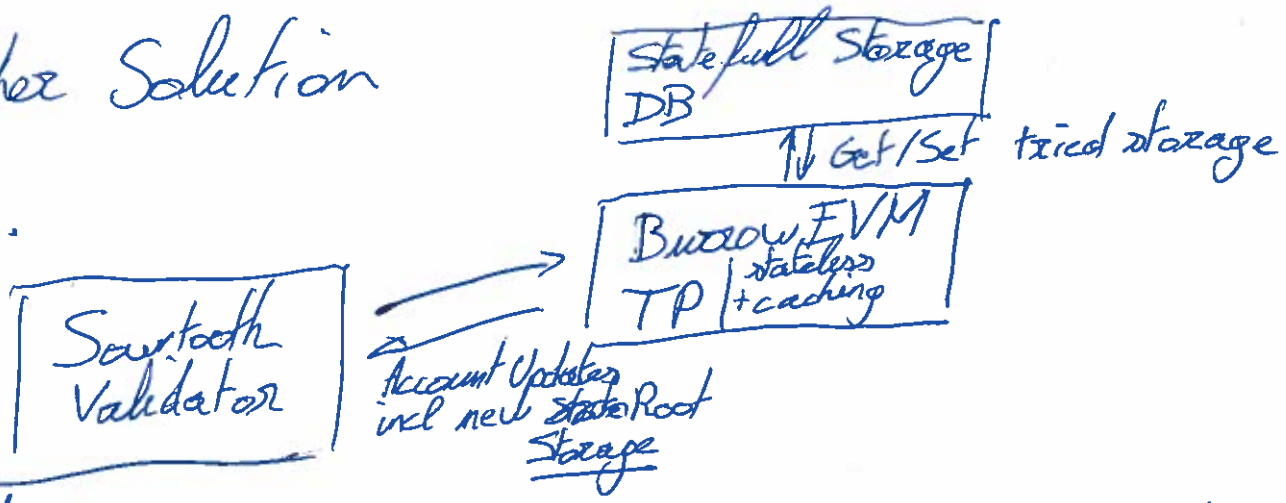
Q

Future Enhancement/Feature Ideas

- Denote "special" Sawtooth-EVM accounts that can be called at a specific address but run off-chain code and return the result to the on-chain code. This would demand that the off-chain code is deterministic.
- Need to define an event handling system. LOG opcodes will need this.
- Running non-deterministic code with a deterministic output in the EVM? (Silas can add more detail on this, I didn't completely follow what was being described.)
- Optimized and efficient caching

Other Solution

4.



addressing:
"abcdef" + "20byte account address" + "00..."
→ Store serialized account object

State Account {
(Address)
Sequence
Pubkey
Code
StorageRoot
Permissions
}

Store Account Storage (32byte, 32byte k-v store, not in STL global state but in separate DB