Virtual Machine Solution

Document Overview

Document contains high level description of the BitShares Blockchain and the Ethereum Virtual Machine Integration Solution.

Solution Overview

Provided solution allows to execute smart contracts in the BitShares blockchain using the Ethereum Virtual Machine (EVM).

Ethereum Virtual Machine

Ethereum Virtual Machine (EVM) is a quasi-Turing-complete machine; the quasi qualification comes from the fact that the computation is intrinsically bounded through a parameter, gas, which limits the total amount of computation done.

The EVM is a simple stack-based architecture. The word size of the machine (and thus size of stack items) is 256-bit. The memory model is a simple word-addressed byte array. The machine also has an independent storage model; this is similar in concept to the memory but rather than a byte array, it is a word-addressable word array. Unlike memory, which is volatile, storage is non volatile and is maintained as part of the system state.

The machine does not follow the standard von Neumann architecture. Rather than storing program code in generally-accessible memory or storage, it is stored separately in a virtual ROM interactable only through a specialised instruction.

The machine can have exceptional execution for several reasons, including stack underflows and invalid instructions. Like the out-of-gas exception, they do not leave state changes intact. Rather, the machine halts immediately and reports the issue to the execution agent (either the transaction processor or, recursively, the spawning execution environment) which will deal with it separately.

Fees (denominated in gas) are charged under three distinct circumstances, all three as prerequisite to the execution of an operation. The first and most common is the fee intrinsic to the computation of the operation. Secondly, gas may be deducted in order to form the payment for a subordinate message call or contract creation; this forms part of the payment for CREATE, CALL and CALLCODE. Finally, gas may be paid due to an increase in the usage of the memory.

EVM was chosen because it has a lot of existing tested smart contracts, that can be used in the BitShares network.

Accounts and Contracts

Contracts are objects that are located in the 1.16.X space and created by the BitShares accounts. Contracts don't have keys and can't create, sign and broadcast transactions. Assets can be transferred to the contract. Contract object contains address and a code, that will be executed during the contract methods calls.

Account Creation and Call

Account creates object using *create_contract* call, sending the byte code of the contract. Transaction is created and in case of its acceptance, new contract object in space 1.16.X is created. Methods of the contract object can be called using *call_contract* function with parameters: address, method name, arguments. During contract call, different addressing can be used. Contract call is an execution of operation, for which *do_evaluate, do_apply* are performed and fee is charged.

Addressing

Double addressing system is used:

- like in BitShares (space, type, instance);
- like in Ethereum (20-byte addressing) for compatibility with Solidity contracts and EVM.

I.e to each BitShares address uniquely corresponds a 20-byte value and vice versa.

Gas, Fee, Gas Price

Gas is a measurement unit of computational effort that is needed to be paid to commit the transaction to the blockchain network. The caller of the contract pay this cost. At the very high level, gas is the number of instructions used to execute a transaction in the EVM. It ensures that an appropriate fee is being paid by transactions submitted to the network. By requiring that a transaction pays for each operation it performs, it ensures that the network doesn't get misused.

Fee is a payment that must be changed for operation call.

Gas Price is an amount of fee, that must be payed for 1 Gas. Gas Price is set by the Committee.

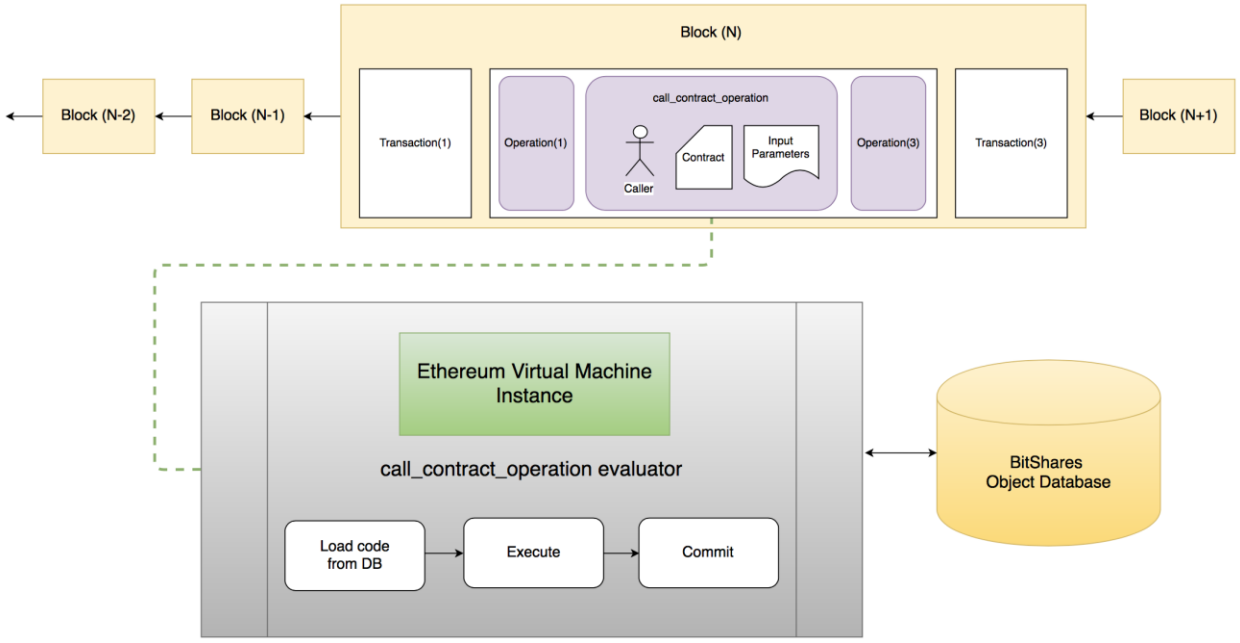
Payment, that is charged for contract call = Fee + Gas*Gas Price.

Fees for calling contracts methods are distributed like fees for any other operations in BitShares (*pending_vested_fees* and *pending_fees*).

System Contract

System contract allows to get access to blockchain functionality from the Solidity code. System contract has an address and can be called using this address.

Solution Architecture



Functionality

<table>
<tr><th>ID</th><th>Functionality</th></tr>
<tr><td>1</td><td>Solution allows to execute smart contracts on the BitShares blockchain (Any smart contracts, that can be executed using the Ethereum Virtual Machine. For example, contracts on Solidity). Smart contracts executes BitShares operations using BitShares system contracts.</td></tr>
<tr><td>2</td><td>Currently Solution supports execution of the following BitShares operations:
<table>
<tr><th>ID</th><th>Operation</th><th>Operation Description</th></tr>
<tr><td>1</td><td>getBalance(address) returns (balance)</td><td>returns balance of account or contract at address in core asset</td></tr>
<tr><td>2</td><td>getBalance(address, symbol) public returns (balance)</td><td>returns balance of account or contract at address in given asset (symbol)</td></tr>
<tr><td>3</td><td>transfer(receiver, amount, symbol)</td><td>transfer from this contract to an account or contract specified at receiver-address the amount in given asset (symbol)</td></tr>
<tr><td>4</td><td>transfer(receiver, amount)</td><td>transfer from this contract to an account or contract specified at receiver-address the amount in core asset</td></tr>
<tr><td>5</td><td>getMessageAsset() public returns (symbol, amount)</td><td>get amount and symbol of asset transferred to contract within call</td></tr>
<tr><td>6</td><td>createAsset(symbol)</td><td>create new asset 'symbol'</td></tr>
<tr><td>7</td><td>issueAsset(receiver, amount, symbol)</td><td>issue asset to receiver-address</td></tr>
<tr><td>8</td><td>transferFrom(sender, receiver, amount, symbol)</td><td>transfer from sender-address to an account or contract at receiver-address amount in given asset (symbol)</td></tr>
<tr><td>9</td><td>getAssetPrecision(symbol) public returns (precision)</td><td>returns precision of a given asset</td></tr>
<tr><td>10</td><td>getAssetSupply(symbol) returns (amount)</td><td>returns max supply of a given asset</td></tr>
<tr><td>11</td><td>setAssetSupply(symbol, amount)</td><td>set max supply of a given asset</td></tr>
</table>
In future, Solution will be allowed to perform any existing operations in the BitShares blockchain.</td></tr>
</table>

3	Solution includes implementation of the JavaScript Console that allows: • to perform operations (running scripts and commands), available in the BitShares blockchain; • to execute smart contracts in the BitShares blockchain. Console acts as an alternative for the cli_wallet, existing in the BitShares blockchain, but has the following advantages: • user-friendly interface; • allows to execute scripts; • allows easy-migration of the Ethereum console users to BitShares console users.

Solution Tests

Tests

The following tests were prepared and conducted for the Solution:

- Integration Tests (python): smart contracts loading and execution were checked.
- Unit Tests (C++).
- Smoke Tests (python).
- Performance Tests.

Performance tests results

Test Description	Test Result
Performance of the transfer operation was testing	5769 transfers per second (each transaction contains 1 transfer)
Performance of the contract call operation, that modifies status of the contract was testing	3504 calls per second (each transaction contains 1 call)

Testnet

Solution has a public testnet.

For getting the access to the testnet it is required:

- To pull the docker image from https://hub.docker.com/r/aetsoft/evm-demo/

docker pull aetsoft/evm-demo:1.0

- To connect to the node:

docker run --rm -it aetsoft/evm-demo:1.0

- Required operations can be performed.