Getting Started with

Hyperledger Bootcamp - São Paulo - June 24, 2019 - Juliana Passos

Hyperledger Burrow is a permissioned Ethereum smart-contract blockchain node. It executes Ethereum EVM smart contract code (usually written in Solidity) on a permissioned virtual machine. Burrow provides transaction finality and high transaction throughput on a proof-of-stake Tendermint consensus engine.

Agenda

- Different Consensus Mechanisms
- Hyperledger Burrow vs. other Blockchains
- Getting HL Burrow installed

About me…

Telegram

Consensus Mechanisms

Byzantine Fault Tolerant

Byzantine Fault Tolerance is the ability of a distributed computer network to remain fault tolerant with valid consensus despite imperfect information or failed components of the network. Prior to Bitcoin, the only way to maintain a BFT, P2P network was through employing a closed or semi-closed group of nodes. Additionally, traditional BFT algorithms such as Practical Byzantine Fault Tolerance (pBFT) use a different node selection method than what is currently used in Nakamoto Consensus.

Tolerance >> usually ⅓ byzantine nodes vs. ⅔ honest nodes.

Nakamoto Consensus

Created by Satoshi Nakamoto for Bitcoin, Nakamoto Consensus refers to the set of rules, in conjunction with the Proof of Work consensus model in the network, that govern the consensus mechanism and ensure its trustless nature. In doing so, Bitcoin became the first Byzantine Fault Tolerant (BFT) open and distributed Peer to Peer (P2P) network that utilizes a distributed network of anonymous nodes that are free to join and leave the network at will.

Nakamoto Consensus can be broken down into roughly 4 parts.

- Proof of Work (PoW)
- Block Selection
- Scarcity
- Incentive Structure

Ethash (formerly Dagger-Hashimoto)

- It’s a Mining Proof of Work Algorithm (not a Consensus)
- SHA3-256 variant Keccak hashing function
- Memory-hard computation
- Memory-easy validation
- Can’t use ASICs (Application Specific Integrated Circuits)
- Uses 4GB directed acyclic graph file (DAG) regenerated every 30000 blocks by miner

Tendermint

Tendermint is a BFT consensus mechanism, it makes the same assumptions as other BFT systems, specifically that no more than ⅓ of the nodes in the network can be byzantine at any given instance.

Tendermint uses an optimized version designed to scale to thousands of transactions per second and allow for easy plug-and-play functionality. We can break down the round in Tendermint BFT into 3 stages:

1) Block Proposal 2) Pre-vote 3) Pre-commit

Participants in the protocol are validators. Validators propose and vote on blocks in the network based on their overall stake in the round. Therefore, the ⅓ BFT assumption is predicated on the “weight” of each validator — determined by their correlated stake — rather than ⅓ of the total nodes participating.

Hyperledger Burrow vs. other Blockchains

Hyperledger Burrow in a Nutshell

Hyperledger Burrow is a permissioned blockchain node that executes smart contract code following the Ethereum specification. Burrow is built for a multi-chain universe with application specific optimization in mind. Burrow as a node is constructed out of three main components: the consensus engine, the permissioned Ethereum virtual machine and the rpc gateway. More specifically Burrow consists of the following:

Burrow consists of:

- Consensus Engine
- Application Blockchain Interface (ABCI)
- Smart Contract Application
- Permissioned EVM
- Application Binary Interface (ABI)
- API Gateway

Benefits of Hyperledger Burrow

High Performance
https://www.hyperledger.org/blog/2018/08/29/hyperledger-burrow-even-faster-and-easier-to-use

Governance clearly defined

Prevention of processing infinite loops

Security of a permissioned blockchain

Provides Transaction Finality

https://youtu.be/8abUvhSa9H4?t=28

Getting Started with Hyperledger Burrow

What you will need

- Golang installed
- Install Dependencies
- Create keys & quick boot
- Deploy and test contracts
- Set up burrow.js service
- Call your app over HTTP

Install Burrow

https://github.com/hyperledger/burrow/releases/latest

Node.js Optional (for your environment)

Curl (for your environment)

Git (for your environment)

```
 go get github.com/hyperledger/burrow

 cd $GOPATH/src/github.com/hyperledger/burrow

 # We need to force enable module support to build from within GOPATH (our protobuf
build depends on path, otherwise any checkout location should work)

 export GO111MODULE=on

  make build
```

Install Dependencies - Monax

```
burrow --version

git clone https://github.com/monax/burrow-workshop

cd burrow-workshop && ls -la

cd chains
```

Quick Boot a local network

First we will quick boot a local network just to show how burrow chain building tools operate

```
burrow spec --full-accounts 1 --toml > quick_spec.toml

cat quick_spec.toml

burrow spec --help

burrow configure --genesis-spec quick_spec.toml > quick_config.toml

cat quick_config.toml

burrow configure --help

burrow start --help

burrow start --config quick_config.toml --validator-index 0
```

Quick Boot & Cleanup

```
# don't run this now but in the future you can use

burrow spec -f1 | burrow configure -s- | burrow start -v0 -c-



# ctrl-c to stop this burrow chain

rm -rf .burrow
```

Create keys for collaborative network

Identities on Burrow are based on private key pairs. You need a key pair for two separate activities

1) Running a validator (for signing votes in the consensus mechanism)
2) Sending transactions

We will reuse the same key for both

```
# Start the keys service (from workshop dir)

burrow keys server &

# Generate a named key with no password

burrow keys gen --no-password --name myKey

# Make a note of the 20 byte string and save it as $myKey

pkill burrow
```

Start Node and join network

We've setup a baseload validator pool on our systems which you are going to join. For this you will need to:

- Move into the collab chain context (where burrow.toml and genesis.json reside)
- Start your node and join as a passive participant
- Use a special key to send a governance transaction to make your node a validator
- Profit

```
# Move to where burrow.toml (config) and genesis.json (chain def) are

mv genesis.json.bak genesis.json
cat burrow.toml
cat genesis.json

# Start your burrow node (where $myKey is the address from earlier)

burrow start --validator-address $myKey --validator-moniker YOUR_NAME
```

Deploy and Test contracts

- Create a deploy.yaml
- https://godoc.org/github.com/hyperledger/burrow/deploy/def
- Run burrow deploy with the yaml

```
cd ../04-deploy

cat simplestorage.sol
cat burrow.toml

burrow deploy --help
rootKey=$(cat ../chains/.keys/names/rootKey)

burrow deploy --address $rootKey
```

Become a validator

```
# Run a burrow deploy governance job that will add your account to the chain as
validator (take a look at the add-validator.yaml playbook to understand what is
happening).

cd ../03-join
burrow deploy -s localhost:10997 -e Target=$myKey -a $rootKey -f add-validator.yaml

# If the response from this looks good - you are now a validator. The total validator
pool power has just grown by 999999991

# Admire your validatiness by hitting our HTTP info service

curl localhost:26658/validators

# You should be able to find your validator listed by its $myKey address...

curl localhost:26658/status
```

Setup burrow.js service

Create a node application that will talk to the get-set contract over REST - http

First we need to connect it to YOUR contract on the collaborative chain and make sure the App has the ABIs

```
cd ../06-service

# copy in the needed files from the deploy directory
cp -r ../04-deploy/bin .
cp ../04-deploy/*.output.json .

npm install

# in an editor, change line 19 of the app.js

npm test
```

Call your App over HTTP

Now we will simulate talking to the app.js via an external user (front-end) or other business systems. We'll use curl.

```
url=http://127.0.0.1:3000

# Inspect current stored value
curl ${url}

# Set the value to 2000
curl -d '{"value": 2000}' -H "Content-Type: application/json" -X POST ${url}
```