A Blockchain Computer [DRAFT/NOTES]

Rasmus Erik Voel Jensen

2017

Abstract

TODO

Introduction

The vision is to have a decentralised infrastructure for webapplications, where each client brings its own backend-resources. This leads to a blockchain computer, with an economy for computational resources. The client can then save up resources to make its backend code continue run, even when it is offline. The blockchain is also needed to ensure trustable results on a trustless network. Using the blockchain computer as the application backend, has several interesting aspects:

- You can run the application 10 years from now, as it does not depend on a central server, that may have disappeared in the meantime.
- Security needs to move to the data layer, which leads to more privacy for the end user.
- The application will be able to run on networks that a disconnected from the Internet.
- No server management nor operations, - the resources scales with the number of clients.

The time is ripe for making this, as the needed building blocks (p2p networking and a performant sandbox) just arrived across major browsers late 2017.

Related Work

Difference from projects in the same sphere:

Ethereum (Wood 2014)

Golem (Golem 2016)

Computes.io (Matthieu 2017)

TrueBit (Teutsch and Reitwießner 2017)

iEx.ec (The iEx.ec Project 2017)

Architecture

The blockchain computer has a global memory on which computation runs in parallel. Both memory and computation are distributed across the nodes of the network.

A node is any application/web-browser that connects into the network, and contributes computation and storage.

Overlay Network

The address of a node is the hash of its public key. Nodes are connected in a kademlia-like topology.

Notice that honest nodes will be evenly distributed across the address space.

State Blockchain

The state of the computer consists of addressable entries with data. Every entry has a *credit balance*, that pays for keeping it in the state. Entries are CRDTs, timestamped, and cryptographically/computationally signed.

Every node stores and replicates entries in a neighbourhood around its own address. The size of the neighbourhood is determined by the median density of nodes across the address space. This gives a statistical guarantee of honest nodes for every entry. Signatures of the entries are checked during replication.

Due to the work of replication, entries are size limited, and large data are stored via merkle trees.

There are two kinds of entries: keyed entries and computational entries. *Keyed entries* are addressed by the hash of their public key, and are updated/signed by their private key. *Computational entries* are addressed by the hash of their origin, and are updated/signed through computations stored in the blockchain.

There is a keyed entry for each node, which contains the state of the node, and whoose credit balance is incremented for replicating the state (based on the local node address density to avoid cheating).

The blockchain keeps a history of the state. Each block contains a reference to the previous blocks, and a reference to a snapshot of the state. Both references are hashes of binary merkle trees, in order to minimise lengths of proofs.

The *snapshot* is a merkle tree of the state, that branches with the bits of the address. It is calculated with a divide-and-conquor consensus algorithm. Nodes are responsible for verifying and storing the path through the tree, that their address lay on. The nodes remembers the entries of their neighbourhood for several snapshots back in the past, and the merkle path to prove their value.

The theoretical time for computing the snapshot corresponds to (branching depth) $\times$ (network latency). For a huge global network, this would be in the order of magnitude of 10 seconds. The estimate assumes 1 billion nodes, binary branching(easily improved), and a high network latency(improvable by optimising topology).

Protection against evil nodes halting the consensus(by issueing delayed entry updates), can be implemented by requiring every entry update to be timestamped before a certain time by a random third party (in a similar way to the scheduling randomisation). This approximately doubles the time of the consensus.

Resource economy

The currency is bound to the value of computational work, and not based on artificial scarcity. Upper bound on value: solving a computational task gives the node currency corresponding to amount of computing power used. Lower bound on value: the currency can be used to schedule computational tasks,

Tasks

- task definition (and max amount of work)
- scheduling and computation
- proof of work done, without revealing value
- reveal result (and amount of work)
- update of ledger

Computational tasks

Sandboxed webassembly

The way to ensure the correct results of computations in an untrusted network, is to do the computation multiple times on different random nodes, and compare the result.

The list of nodes in the snapshot of the state is used to assign tasks to nodes (pseudorandomly, based on the hash of the state, such that it is deterministic, and cannot be determined beforehand). Thus a node cannot control which tasks it gets.

There is a tradeoff between the number of times the computation is done, and the probabilty that an adversary controlling a large part of network theoretically could return a wrong result.

A (computational) task has several steps:

1. The task is scheduled by storing it in the state snapshot in a block of the blockchain, - this can either be done by a previous task, or by a node.
2. Nodes solve the task. A proof-of-result is stored in the the state at the task. Only N proof-of-works are stored (with the lowest distance between the task, and the hash of the scheduling block combined with the addresse of the node (which must be in the list of nodes at scheduling time)).
3. When sufficient proof-of-result/time has arrived, the nodes release the actual result, and they are compared to check if they ar correct.
4. The nodes are credited by the system for their computational work.

Autonomous Computations

(Example use cases)

Conclusion

Future work

Finishing the actual implementation.

Generalise computational tasks to storage, bandwidth, etc.

Adding stake, in addition to proof of work for better security.

Bibliography

Golem. 2016. “The Golem Project - Crowdfunding Whitepaper.” http://www.golemproject.net/doc/DraftGolemProjectWhitepaper.pdf.

Matthieu, Chris. 2017. “Distributed Computing: Centralized Vs Decentralized; How Does Computes.io Work?” https://blog.computes.io/distributed-computed-centralized-vs-decentralized-c1d21202bde8.

Teutsch, Jason, and Christian Reitwießner. 2017. “A Scalable Verification Solution for Blockchains.” https://people.cs.uchicago.edu/~teutsch/papers/truebit.pdf.

The iEx.ec Project. 2017. “IEx.ec White Paper.” http://iex.ec/wp-content/uploads/2017/04/iExec-WPv2.0-English.pdf.

Wood, Gavin. 2014. “Ethereum: A Secure Decentralised Generalised Transaction Ledger.” *Ethereum Project Yellow Paper*. https://github.com/ethereum/yellowpaper.