A Blockchain Computer [DRAFT/NOTES]

Rasmus Erik Voel Jensen

2017

Abstract

A proposal a design for a new kind of decentralised trustless computer. The shared state is stored in a blockchain. This allows computations to be distributed and verified safely across without trusting individual nodes. It also allows proofs of work, and thus crediting nodes for their commputations.

Outline:

- Introduction
  - Motivation
  - Related work
- Architecture
  - State
  - Computation
  - Scheduling of computation
  - General tasks
- Future work
  - Actual implementation (in progress)
  - Stakes in addition to proof of work for better security

The goal is to make it possible to build web-apps with backend-like functionality, but without needing to host a backend.

For this to happen, you need a system, where you can can save up computations/storage when your device is online, - which can then be spend on computation/storage, when your device is offline.

Every time you do computation/storage for the network, you get credits, which can be used to execute computation.

Devices/peers can make a single distributed network, via WebRTC peer data connections. It is safe to run computations for others, due to the sandboxing of WebAssembly.

You cannot necessarily trust other peers. The way to make sure that the result a computation is correct, is to schedule the computation on several random nodes in the network, and verify that they yield the same result.

The underlying datastructure for assigning computational tasks, and making the proofs needed for assigning the credits, etc., turns out to be a blockchain.

TODO Introduction

This paper describes the design of a decentralised trustless network computer. When a node in the network does computation for the computer, it saves up currency. Currency can be used to run computations in the computer. The computer has a distributed state, whoose merkle tree is stored in a blockchain. This makes it possible to schedule and verify computation in a safe manner. Each node only stores a small part of the blockchain. It is designed such that it is easy to boot up a full node within modern web browsers.

The motivation comes from web apps: The blockchain computer could be used instead of a backend. Apps could save up computation, to allow others to interact with user data, even when the users app is offline. There is no server maintenance/scaleability (clients bring their own "server"-resources). If an app is not supported anymore, it can still continue to run, as it does not depend on central services.

Related Work

TODO: explore these in more details, and document differences to our approach

Ethereum …

Golem …

Computes.io https://blog.computes.io/distributed-computed-centralized-vs-decentralized-c1d2120

TrueBit https://people.cs.uchicago.edu/~teutsch/papers/truebit.pdf

iEx.co ??

The motivation is to make it possible to built webapps that does not need a backend.

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. 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.

Memory

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.

Computation

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.

TODO: document origin. TODO: document scheduling

What is a "task":

provable results

- 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

TODO Distributed ledger

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 is used to

TODO 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.

TODO Bibliography