A Blockchain Computer [DRAFT/NOTES]

Rasmus Erik Voel Jensen

2017

Abstract

We propose a design for a new kind of decentralised trustless computer. The shared state is stored in a blockchain. This allows computations to be distributed safely across without trusting indivudual nodes. It also allows proofs of work, and thus crediting nodes for their commputations. Computations are securely distributed and verified, through a blockchain containing the state By storing shared state in the blockchain, it is possible to By storing the machine state in the blockchain, it is possible to securely run distributed computations, without trusting the individual nodes. Individual nodes only need to know/store the small subset of the blockchain that they need for their computation.

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

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 ??

Architecture

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

A node is any device/browser/app/client 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, and timestamped.

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 references to a snapshot of the state. Both references are the hash of binary merkle trees to minimise lengths of proofs.

The merkle tree *snapshot* of the state, has branching based on the bit-pattern of the address. It can be calculated in divide-and-conquor fashion, where the nodes are responsible for verifying and storing the path through the tree, that their address lay on (for the most recent couple of blocks).

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

Protection against evil nodes halting the computation 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

TODO: document origin. TODO: document scheduling

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

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

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.