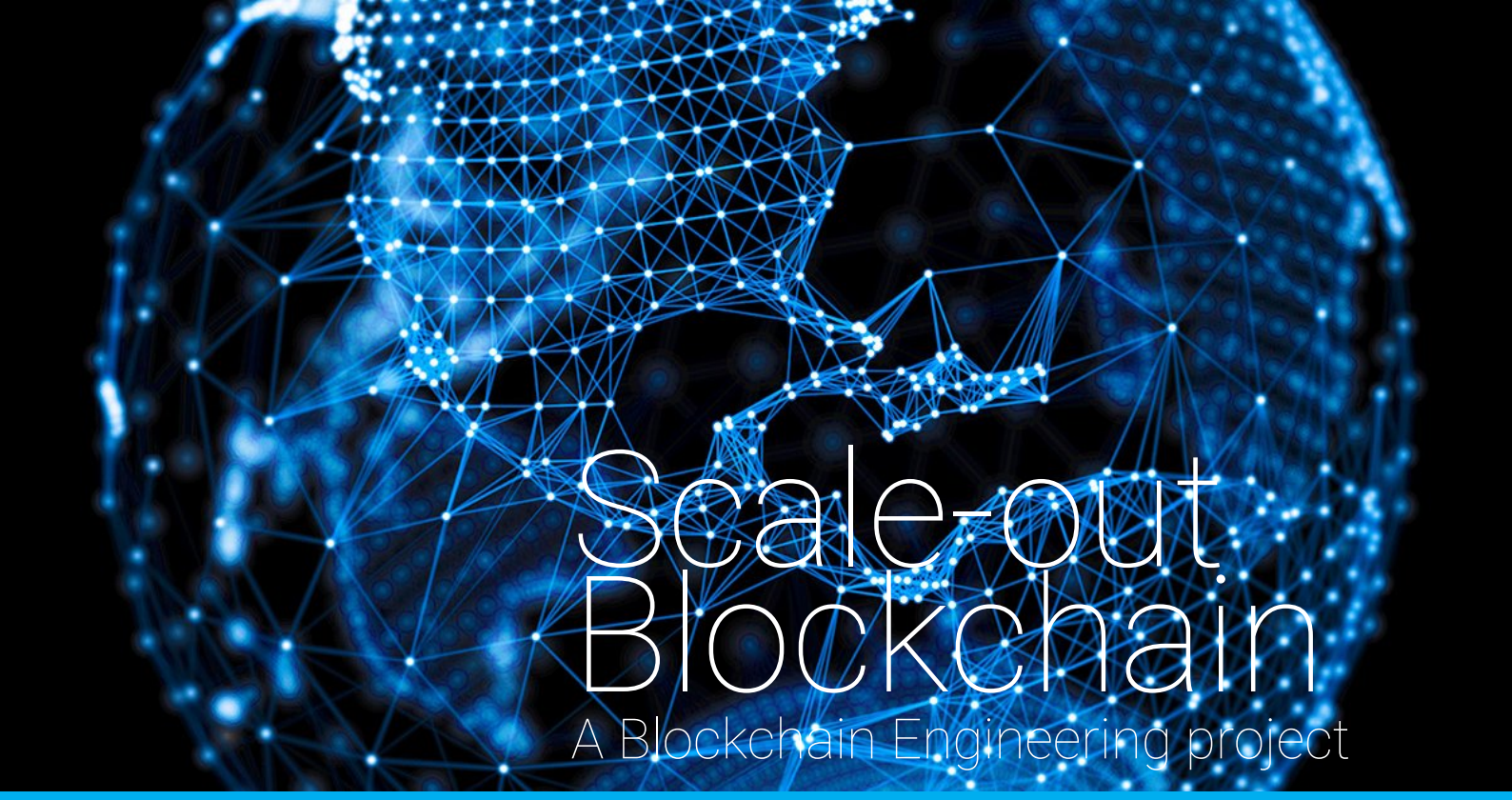

Scale-out Blockchain

A Blockchain Engineering project

Taico Aerts
Chiel Bruin
Bart de Jonge
Karol Jurasinski
Alex Morais

Technische Universiteit Delft

TU Delft

Scale-out Blockchain

A Blockchain Engineering project

by

Taico Aerts
Chiel Bruin
Bart de Jonge
Karol Jurasinski
Alex Morais

Taico Aerts	XXXXXXX
Chiel Bruin	4368436
Bart de Jonge	XXXXXXX
KarolJurasinski	XXXXXXX
AlexMorais	XXXXXXX
Some title:	ir. J. Pouwelse
	Z. Erkin
	Z. Ren

An electronic version of this report is available at
`https://github.com/bartdejonge1996/ScaleOutDistributedLedger`.

TU Delft

Contents

1 Introduction

Introduction with proper references

2

Problem Statement

3

System Overview

High level of the system, and reason for certain critical choices made in the design. Details should be in the appendices.

4

Test Results

A

Detailed system overview

A.1. Algorithm 1
[SOMETHING SOMETHING]

A.2. Algorithm 2
[SOMETHING SOMETHING]

A.3. Algorithm 3: Non-interactive Smart Transacting Algorithm
The algorithm in the paper basically requires the use of a powerset of all unspent transactions. Since this requires $O(2^n)$ sets, this is only feasible for small numbers of n.

The algorithm implemented is smarter and eliminates bad combinations by keeping track of a best-so-far. It has the same worst case performance, $O(2^n)$, e.g. if all unspent transactions are required for the transaction. If this is not the case however (you send a part of your money), then the performance of the implemented algorithm is much better. The less transactions are required as sources, the better this algorithm performs. The algorithm also benefits from sharding, as transactions with the same requirements (e.g. received from the same source) will be considered as a group instead of individually.

A.3.1. Step 1
First we collect all unspent transactions (these are tracked in application). We then create a TransactionTuple for each transaction, which will calculate which chains would need to be sent based on what the receiver already knows. We then merge the tuples that have the same chain requirements in O(|transactions|) by hashing them to buckets.

A.3.2. Step 2
Check if there are single transactions / single groups that have a large enough amount to be used as the only source. If we find at least one of these, then all transactions that require more chains than the best single source are directly eliminated. (Combining those transactions with other transactions would only make the required set of chains bigger, so they will never be in the best choice)

We call the resulting set ‘candidates‘.

If there are less than two candidates we stop and return the best-so-far.

A.3.3. Step 3
We have a set of TransactionTuples ‘currentRound‘, which is initially set to ‘candidates‘. We have a set of TransactionTuples ‘nextRound‘, which is initially empty.

For at most `candidates.size() - 1` rounds, we do the following:

- We iterate over the candidates. We try to combine each candidate tuple with each element of ‘currentRound‘.

  - If the combination requires more than or the same number of chains than the best-so-far, then it is eliminated.
  - Otherwise:
    - ◇ If it is able to cover the costs, we set it as best-so-far.
    - ◇ Otherwise, we add this combination to 'nextRound'.

- If there are less than two combinations selected for the next round, then the algorithm returns the best-so-far.
- Otherwise, `currentRound := nextRound' and 'nextRound := []` .

A.3.4. Result

The final combination of transactions returned is the set of sources with minimum amount of chains required. The set found is the largest possible set for its chain requirements, but not necessarily the largest possible set.

A.3.5. Performance

The performance of the algorithm is still $O(2^n)$ in the worst case. The worst case would be if all transactions are needed to get the amount of money required and none of them can be grouped. Then the algorithm will consider all possible combinations.

A.3.6. Future optimizations

- It would probably be useful to keep track of how much money we have, to prevent the costly case where we don't have enough money.
- There could be 2 algorithms, one for when we expect many transactions will be required, one for when we expect only a few transactions will be required. The current algorithm works bottom up, the other algorithm could work top down. (How many transactions do we need at minimum to cover the transaction?)

A.3.7. Implementation Details

BitSets are used as a convenient and efficient way to represent the chains that are required. They represent a sequence of bits of a particular size (e.g. 1000 bits).

- The bit at index 0 represents chain 0, the bit at index 1 represents chain 1, etc.
- A bit with a value of 1 means that the corresponding chain is required.
- Combining the chains required of 2 tuples is done with a bitwise or.
- The number of chains required is the cardinality of a bitset.
- Removing the chains that the receiver already knows is done with a bitwise andNot.

A.4. Simulation

A.5. Tracker Server

The tracker server can be installed by installing NodeJS and running 'npm install' in the tracker-server folder. After this it can be run by running 'npm start' in that same folder.

A.6. Main Chain

For the main chain we use [Tendermint](https://tendermint.com/). We can communicate to this chain using the provided ABCI interface.

A.6.1. Running tendermint

Tendermint process can be spawned using TendermintHelper class, which takes care of everything that is needed to start it up. The steps are usually:

1. Generate priv_validator.json file for each node.
2. Generate genesis block, which is then passed to calculate the initial application hash.
3. Generate genesis.json file for each node.
4. Run tendermint process for each of the nodes (this means there is a tendermint processes running at the same time, one for each node).

When running a simulation, these steps are all automatically taken by the simulation initialization. Alternatively, running tendermint can also be done with the following command: (the use of which is discouraged though) `./tendermint node -consensus.create_empty_blocks=false` (Note that Tendermint creates empty blocks by default) Note: For this method to work, you need to have a config.toml file, genesis.json and priv_validator.json for each node, and/or specify more parameters.

See the [tendermint documentation](https://tendermint.readthedocs.io/en/master/using-tendermint.html) for more information.

Bibliography