Modelling Read Cache Solutions for the Blockchain

Jonas Vanden Branden
jonas.vandenbranden@student.uantwerpen.be

December 15, 2017

1 Introduction

In the current landscape of blockchain development, we are now confronted with many boundaries of blockchain distributed applications (DApps). Among these, there are the (R)DBMS-capabilities of a permissioned blockchain which are desirable when developing data-intensive applications.

Tools

The tools currently used are:

- **Tendermint**: *the private blockchain platform with a proof-of-stake pBFT consensus algorithm*
- **Solidity**: *the smart contract language, compiled by the Ethereum Virtual Machine (EVM)*

Smart Contracts

Distributed Applications or 'DApps' come in the form of *smart contracts*. In essence, they are no more than deterministic state machines which manipulate the state of the blockchain. Because of their deterministic nature, you can run them multiple times from an equal start state and always expect the same outcome, by definition. That is exactly what is done when executing function calls or transactions on these smart contracts. All validator nodes on the network execute this code and the Byzantine-fault-tolerant consensus-algorithm will coördinate the messages in such a way that all peers in the network are working with consistent data. That is the strength of blockchain, multiple actors can work with common data without necessarily having to trust one party for managing this data.

The problem arises when large amounts of data are needed in read, write, update or delete operations. Smart contracts working with this data need to execute on all nodes before completing and returning any information. Advanced queries are not possible, querying lists is impossible, the storage capacity of a smart contract is limited and so on. There are shortcomings in both performance and expressibility. Where a blockchain is obviously preferred when handling raw transactional types of data, its capabilities as a database are questionable.

Current Workaround

There is a frequently used *workaround* that deals with the read-performance shortcomings. Smart contracts can emit events, and local (non-blockchain) applications can subscribe to these events for specific contracts. Local databases, functioning as caches, are often maintained to mirror the blockchain state and enable fast and performant reads.

2 The Problem

Blockchain applications are suitable in the domain of logistics, where transactions can take many forms. From container-handovers to commercial sales, and everything in between on the supply chain. But requesting a lot of data from the blockchain forms a bottleneck.
From personal experience I can say that the Solidity smart contracts (used by Ethereum) are fairly limited in their data-aspect. They can handle simple logic, but are literally restricted in data storage and handling.

The reasons for these shortcomings have two aspects:

- **Data storage:** The data stored in a blockchain is immutable, data in the chain cannot be altered or changed, only by controlled transactions, which are part of the protocol. This is one of the core strengths of the architecture, but also a weakness to take into account. Changes to data models are limited by this property as well. Smart contracts are not change-friendly and upgrading existing entries to a new data-model is not supported.
- **Data handling:** Reading of data of the blockchain is done as a smart contract function-call , essentially a transaction. To ensure you ready the true data, the transaction must pass the consensus mechanism, which takes time. Large reads, like lists of all objects or more advanced queries, suffer from this limited throughput and large latencies.

The handling of data can be approached by working with a performant cache, but the storage is something more complex.

3 A Solution

As previously mentioned, a possible solution is to add an extra caching 'layer' to the platform where data can be accessed easier, at the cost of some possible inconsistencies. This is interesting when clients are requesting large lists of entries in the system.

Current solutions handle this by creating a small database, which is kept up-to-date by subscribing to certain events from the blockchain. This is not really a structural solution, so it might be beneficial to incorporate this into the

platform. With calculated guarantees about the performance and consistency of the data, instead of a handmade solution.
This will mainly handle the read-operations of the database, as writes, updates and deletions of the data is something that should pass the consensus and happen inside the blockchain. Moving the data off-chain and just keepin the (business-)logic on the blockchain is also a possible course of action, but this is out of the scope of this research.

4 Why Modelling?

Different (distributed) caching implementations will have different characteristics regarding latency, consistency, throughput,... Choosing between different approaches can be supported by developing a domain specific language and modelling simulations.

Saving Time & Resources

Modelling the implementations will have the advantage that time and resources can be saved by working with a simulation model.

Abstraction & Control

Another important reason is that the network layer, upon which the system will be running, is a complex environment which should be taken into account. Modelling this system is advantageous in comparison with a real-life setup of the system; a developer/analyst has more control over the environment, it is more cost-efficient as well.
Reasoning can be done on higher level, without having to spend too much time on implementation.

5 Conclusion

There are some challenges which should be tackled in the next phase of this project:

Without knowing how the caching algorithms will be implemented, we should find a way to model their characteristics in a way that they become comparable. This could be 'top-down', by analysing a few algorithms and extracting their characteristics, or 'bottom-up', by abstracting the properties without necessarily having concrete algorithms.

The network influence should be modelled or borrowed from a existing network-model. Co-simulation could be an interesting approach.

How can the blockchain consensus protocol be modelled? By constructing Petri Nets, or are there better alternative solutions?

Suggestions to tackle these challenges are very welcome.