Parsec Network

A state channel for the Internet of Value

Josep Casals <joscas@gmail.com> | @jcasals
V 0.2

Abstract

Parsec is a web-scale state channel for the Internet of Value. State channels can be used to solve speed, scalability and confidentiality problems of public distributed ledgers. A web-scale solution for State Channels is needed to enable a layer of value transfer to the internet.

Other proposals for State Channels include Raiden for Ethereum and Lightning Network for Bitcoin. However, we intend to leverage existing web-scale technologies used by large Internet companies like Uber, LinkedIn or Netflix. We'll use Apache Kafka that has been proven to scale to trillions of operations per day.

Introduction

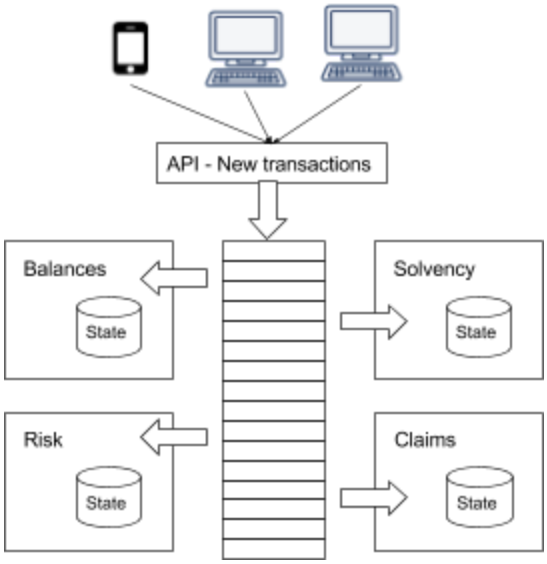
The impact of data systems in finance has been so far focused on cutting administrative costs. Now however, technology could transform business models in finance. With the help of cryptography on distributed data systems, we can now design infrastructure that not only keeps track of transactions and balances but also allows transfer of assets and enforcement of contracts without the need of clearinghouses, middle men, escrows and even some tasks performed by lawyers. On these systems settlements are final and there's no possibility of repudiation while contracts execute deterministically according to predefined inputs.

Institutions that embrace this vision will have a competitive advantage. Investing in distributed cryptographic technology and regulatory oriented technology (regtech) will pay off in the form of less intermediation, less legal fees or litigation and ultimately an accelerated innovation cycle where new financial services can be provided with much less friction and where these services are much better tailored to smaller market segments. These institutions will be able to compete

in the long tail of financial services and offer a much more competitive service for the mainstream.

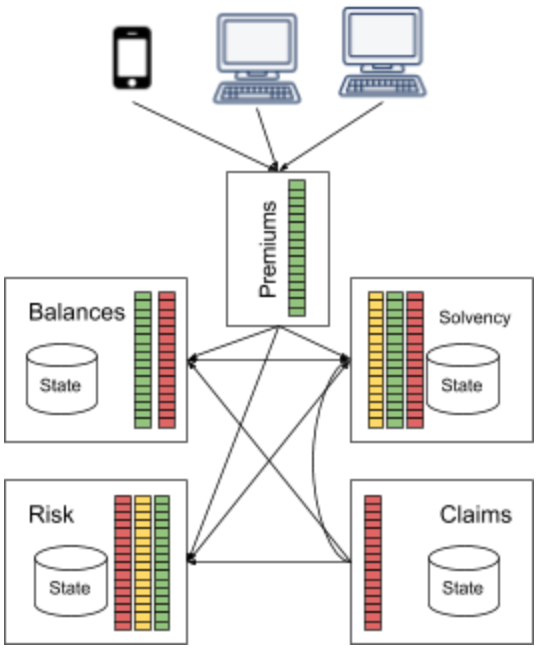
However, for the Internet of Value vision to happen, there's a need for a whole redesign of the architectural IT principles used to build new services.

Architectural discussion



With that in mind, we should come up with an architecture that makes possible to quickly innovate and create new services. In order to achieve this at web scale, it is convenient to look at the Event Sourcing pattern. In this kind of architecture, instead of having a centralised state for all the applications in a centralised database or a set of disconnected state holding databases, we rely on a centralised log of state changes. This allows building new applications very quickly because it is only necessary to replay the log of changes in order to build new services with new states that will be consistent with the state being held by other services in the same company.

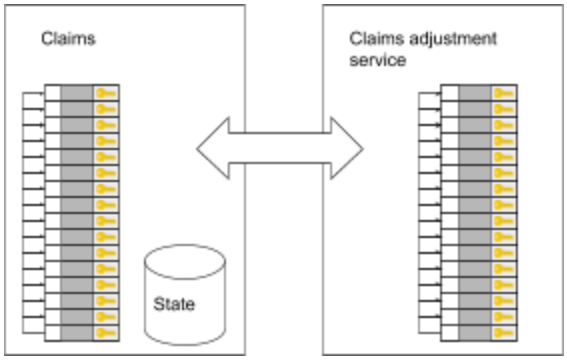
Event Sourcing is very powerful but has a coupling problem. In big organisations, a central team needs to own the event log. Usually, many different teams want to access the log and they all need to agree on the technology and permissions to write and read from it. Also teams reading or writing intensively from/to the central infrastructure risk causing performance problems to the whole.



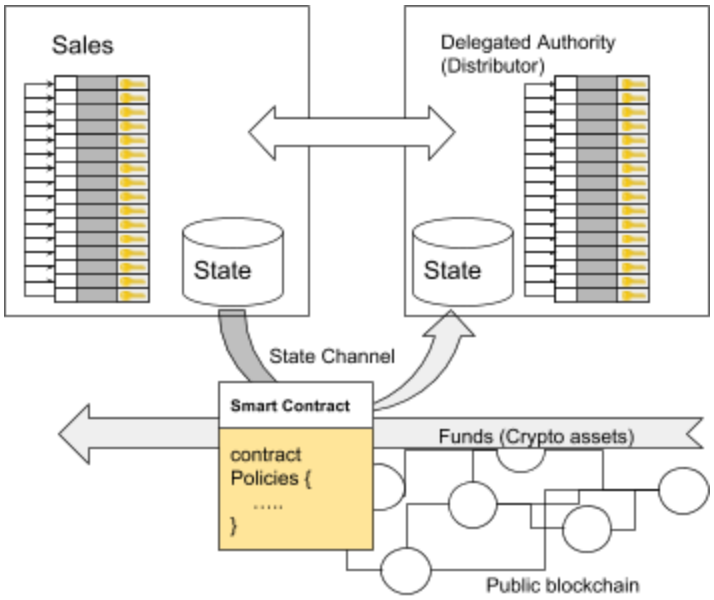
Also, this kind of architecture is not suitable for collaborating between different organisations or groups of companies.
For that reason, an architecture where each business unit or application would hold a copy of the parts the log relevant to it would be better. If the stream of data is composed of different topics (for example, premium payments or claims), then each business unit would be able to hold a copy of the topics relevant to it

The above architecture would be feasible in a trusted environment but in a collaborative environment, all transactions would have to be signed by the different parties and the order of the transactions should be guaranteed.

We could achieve that in a manner similar to blockchain by means of a linked stream of events using hash pointers to the previous events and by embedding signatures of both sides of a transaction in the message itself. This would imply that some business units would have to act as signing authorities to their counterparties in external organisations. By acting like that we would achieve the benefits of a distributed ledger but without being able to transfer assets.

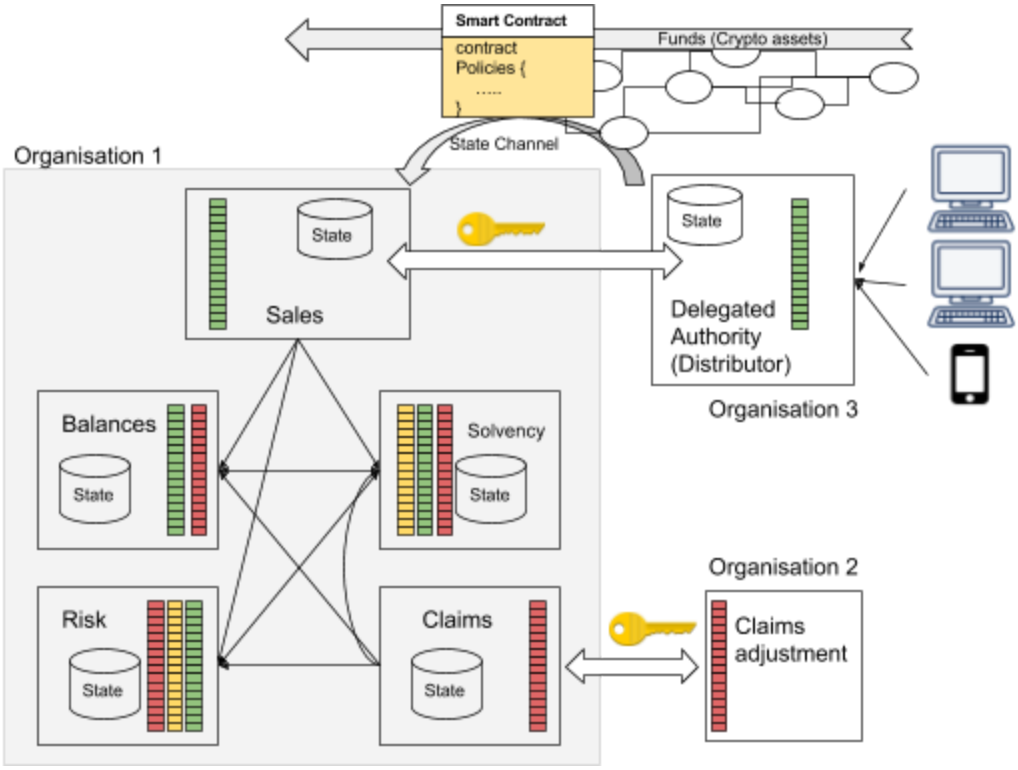


Asserting that assets can't be transferred with only a distributed ledger can be contentious. Indeed if both parties sign a transaction on a distributed ledger, and the order of transactions is impossible to alter then we could agree that both parties have proof that the transaction has been agreed. However, even in the case where the transaction implied a transfer of assets, for example, in the case of signing an insurance policy, this wouldn't imply that any of the parts would pay the stipulated amounts. We would have no guarantee that policy takers would pay the primes or insurers would pay the claims. For that purpose we would need to resort to a third party escrow or even better to a Smart Contract running in a blockchain with distributed consensus. It's arguable whether this blockchain would have to be public or if a permissioned distributed ledged would be able to achieve this purpose. Here we'll assume it's a public blockchain capable of running Smart Contracts. In fact the structure we're describing here is not new. It's known as a State Channel and it allows untrusted parties to exchange transactions with the possibility to resort to an independent escrow held in a Smart Contract operating on a public blockchain in case that one of the parties doesn't fulfill its commitment.



Finally get the big picture on how a state of the art architecture for a financial system should look like. It would have to:

A. Be based on transaction logs
B. It would have to include a replication mechanism of such logs between business units
C. Logs would have to be enhanced to become shared ledgers when dealing with external untrusted parties.
D. State channel support would have to be added for asset transfer.



Implementation using Apache Kafka

Apache Kafka is a distributed data system that can be used as an event sourcing unified log tool. It's can be seen both as a queue or as a data store.

We intend to implement only the very minimum additions to the already existing functionality so that it can work as a State Channel. For that purpose a fundamental tool of the Apache Kafka ecosystem will be Kafka Streams as it already offers many of the functionalities we need to implement.

The main functions we need for a state channel to work are:

- Scalability -> Provided by Apache Kafka

- Replicable -> Provided by tools like Kafka Mirror Maker
- Able to maintain state at scale -> Provided by Kafka Streams
- Able to interact with Smart Contracts -> Provided by Parsec on top of Kafka Streams
- Able to guarantee an exact sequence of hash pointers -> Provided by Parsec on top of Kafka Streams
- Able to verify and provide cryptographic signatures of every transaction -> Provided by Parsec

For that, the intention is to provide a library on top of Kafka Streams that guarantees the last 3 points.

Parsec Nodes

Parsec nodes will be job instances of the Parsec Library implementing the Parsec SDK. The Parsec Library will be built on top of the Kafka Streams library with the following added functionality:

- Support for a catalogue of different kinds of transactions
- Keeping an up to date state for all unique Ids of the transactions topic
- Interacting with Ethereum Smart contracts (e.g. Hashed Time Locks) for the topic partitions under the node's supervision for scheduled settlements or under the request of the control topic.

All functionality for each Parsec job will relate only to the partitions under it's control.

Parsec Transactions

A basic catalogue of predefined transactions will be implemented. It should be simple to add additional custom transactions.

Micro-payment

This transaction will perform the following actions:

- Keep a sliding array of the latest ***n*** transactions in order to reconstruct the exact order of the transactions from the transaction hash pointers. This is needed in order to cope with our of order messages or repeated messages.
- Calculate a balance after each transaction.
- Commit the balance to the Smart Contract in the Ethereum network at predefined intervals. Intervals will be defined by the modulo ***m*** of the incremental transaction sequence number.

- Commit the balance to the Smart Contract in the Ethereum network at predefined timeouts. The timeout duration will be specified by the ***checkpoint_timeout*** parameter.
- Under the request of the control topic, perform an unscheduled settlement request to the Smart Contract.
- Under the request of the control topic, perform a disputed settlement request to the Smart Contract by means of transferring the signed transactions since the last settlement.

Initialisation and Control

Smart Contracts