Coinweb: Cross-Chain Computation Platform

Alexander Kjeldaas, Alejandro Duran-Pallares, Knut Arne Vinger

May 15, 2021

Abstract

In this white paper, we introduce Inchain architecture: a novel distributed ledger technology (DLT) architecture to be embedded inside existing self-sovereign blockchains. Inchain utilizes existing availability and consensus mechanisms, but operates independently of the data integrity rules of the blockchain, reducing the scalability portion of the "Blockchain Trilemma" to a question of data throughput. By sharing availability and consensus, the Inchain architecture enables tighter coupling than the existing art. Central to the Inchain architecture is the more recent proof theory related to verified outsourced computations.

We introduce Coinweb, a public DLT that spans multiple self-sovereign blockchains. Multiple cooperating Inchain instances create tight semantic coupling safely between these blockchains, by delaying information propagation. Self-sovereign blockchains can join and leave Coinweb safely, while containing a catastrophic failure of a blockchain's consensus. Coinweb also provides causal consistency across all member blockchains.

Finally, we demonstrate that market mechanisms for immediate information propagation between blockchains can be created on top of Inchain, utilizing the semantic coupling between the blockchains. This reduces the risk for market participants and provides efficient elimination of information propagation delays.

Keywords: Blockchain, interoperability, smart contract, scalability

1 Overview

To introduce the technology we provide a brief introduction and social context in section 2, we define and discuss the Inchain approach and related concepts in section 3, and contrast these concepts against common alternatives and well-known projects in section 4. We introduce Coinweb in section 5, a proposed cryptocurrency project following the Inchain architecture, and detail its computation model in section 6. Finally, we conclude all details in section 7.

2 Introduction

If money is viewed as the accepted reward system of modern civilization, cryptocurrency can be considered as the automated, decentralized reward system of our future civilization[1]. The next generation of decentralized computations promise to act as a decentralizing force on the core social and systemic building blocks of that civilization.

Leading the curve at present is DeFi, but following the trend established by decentralized computations it can be safely predicted that all trust-based societal processes will operate within DLT-based infrastructure at some point. However, as shown by the history of infrastructure, from railways to the internet, large-scale use comes with standardisation and interoperability, after lock-in effects have been eliminated. Infrastructure and business models for mainstream usage must be able to adapt and scale with new innovations as they become available, while maintaining backwards compatibility to preserve and take advantage of existing networks. This is why creating computations at layer 2, across multiple blockchains and independent of their underlying consensus mechanisms—thus enabling access to a continuously evolving Pareto optimal solution space—is key to achieving mainstream adoption.

3 Inchain

Throughout the history of technology, innovations typically start out as vertically-integrated solutions. Broad uptake happens when a revised, layered approach is invented. Decoupling layers in the technology stack enables faster innovation by reducing lock-in effects[2].

The nascent Blockchain space is following a similar history. Several vertically-integrated blockchain projects have been launched[3][8][12][9], but strong network effects hinder the uptake of technical improvements in layer 1 blockchain technologies. Ethereum continues to dominate this space.

As a result of this fragmentation, various interoperability systems have been proposed. These systems move data in, out, and between chains, but are not computers themselves. While movement of data is good, these systems do not provide a robust alternative to the vertically-integrated blockchain projects.

3.1 Inchain data

We refer to a piece of information or data as *Inchain* when it is commonly available within the discussed universe, and there is a consensus mechanism that allows any participant to agree upon it. For example, we consider that the amount of Bitcoin a Bitcoin address holds, the number of Ethereum smart contracts executed on the eth-block 1432, or what the Litecoin genesis block-id is, to be instances of *Inchain* data.

We consider a computation to be an *Inchain* computation when both the input to the computation, and the computation result, are *Inchain* data. The execution of a smart contract, or

[1]At Bitcoin block zero, Bitcoin's creator Satoshi Nakamoto embedded the following message: "The Times 03/Jan/2009 Chancellor on brink of second bailout for banks" The message indicates a breach of trust between the public and the governors of the monetary system.

[2]For example, defining IP as layer 3 on top of layer 2 such as Ethernet, Token Ring, and ATM was critical to the Internet's success and longevity. Now many of these initial layer 2 protocols are not used, but the Internet endures.

a specific hash difficulty estimation (as long as it follows a known formula), are examples of Inchain computation. Not every value defined over Inchain data is an Inchain data, and therefore not produced by an Inchain computation; such as finding a non-published preimage of an Inchain hash, because even though there is consensus over the result, it lacks availability.

We define data or a computation as *offchain*, when it is not *Inchain*.

3.2 Inchain architecture

An Inchain architecture is a DLT in which the execution layer is separated from the availability and consensus systems. Looking more closely, an Inchain architecture reveals multiple conditions: Inchain computations over multiple independent availability and consensus systems; a consistency model for composing these computations; and mechanisms to make computational results provable and useful for offchain participants.

Blockchains and para-chains have Inchain computations, but the results of these computations affect their consensus, making the computation and consensus layer mutually dependent. Therefore they are not examples of Inchain architecture.

In blockchain design, there has been a fundamental conflict between building a powerful public computer by exposing powerful computations, and obtaining decentralized consensus. When participants in the consensus layer are required to undertake more and more computations, fewer actors can participate, and power is concentrated in fewer hands, making the system less decentralized.

This conflict has a name, the Blockchain Trilemma, which posits that decentralization, scalability, and security cannot be simultaneously obtained[3]. However, just as blockchain technology itself refuted Zooko's triangle[4], we believe Inchain partially refutes the Blockchain Trilemma.

Inchain recognizes that computation, as a deterministic process[5],does not need consensus, and that consensus need only focus on ordering data. A blockchain needs an incentive system, typically based on a combination of payment, mining, and staking, but the consensus layer of a blockchain does not need to perform complex computations[6].

By making it possible to lift complex computations out of the layer 1 blockchain, we can try to quantify how much computation the resulting layer 1 blockchain should perform. We do this by noting that a maximally-decentralized layer 1 blockchain, whose security depends on participants running full nodes, should be constrained by a participant's available bandwidth. It is available bandwidth, not computation, that is ultimately the limiting factor for transaction processing. Depending on security parameters such as block rate and block size, the client requirements (and thus how decentralized the blockchain can be) differs, but unless it is bandwidth constrained it is artificially constrained. This is because either: the network is not processing enough transactions because processing transactions requires computational resources and thus limiting transaction throughput enables more full nodes to be part of the network; or it is excluding potential full node participants due to the high cost of participation. As an example, Bitcoin-derived blockchains are generally bandwidth-constrained, while a smart contract-based

[3]For example, EOS is well known as a high throughput blockchain, while Ethereum has lower throughput. At the same time EOS is considered less decentralized than Ethereum as they make different tradeoffs in the trilemma.

[4]The conjecture that names could not at the same time be decentralized, secure, and human-meaningful.

[5]A deterministic process is one where given the same inputs, the output is always the same.

[6]An alternative, tongue in cheek, title for this paper could be "Smart contracts considered harmful"

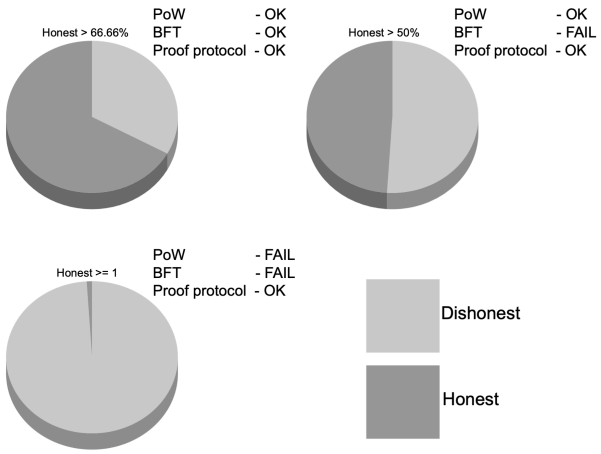

Figure 1: Options for verifying deterministic computations

layer 1 blockchain such as Ethereum is currently artificially compute-constrained up to a decade in the future.[7]

The Blockchain Trilemma is partially resolved by using the Inchain architecture, as it lifts computational and transactional scaling from the trilemma and leaves the simpler trilemma between decentralization, data throughput, and security.

3.3 Lifting computations to layer 2

As noted in the previous section, lifting complex computations out from the layer 1 blockchain enables better-optimized tradeoffs in the Blockchain Trilemma. Shortly following the publishing of the Bitcoin paper[14], research into proofs for verifiable computations were published[11], and work in this area continues to be published today.[13][4]

These proof protocols attempt to solve the overall problem of outsourcing a general, deterministic computation to one or more workers that are untrusted. For example, a client could use such a protocol when outsourcing a computation to a cloud computing service.

As an illustration of how this approach differs from the traditional blockchain approach, a crucial aspect of the blockchain system is that a block includes a Merkelized version of the state of the system, providing for any layer 1 computation to form part of what the network agrees on in the consensus protocol.

Based on the consensus protocol of each blockchain, the consensus breaks down when a predetermined mass of actors do not adhere to the protocol when controlling a finite resource—either half or two-thirds of actors. By contrast, a proof protocol for verifiable computing only requires a single honest actor adhering to the protocol.

[7] As bitcoind could sync the full blockchain, 250GB in around 10 hours, or at around 50Mbps[18] in 2019, this blockchain is not compute- constrained, but rather bandwidth-constrained, and will remain so given the existing protocol. As the growth rate for the Bitcoin blockchain is less than the growth in available bandwidth (50% per year[16]) it becomes easier to participate as a full node in the network over time. The opposite is true with Ethereum. Current implementations are strongly compute-constrained with a sync speed of only 0.3Mbps[2, 17, 20]. Assuming no further optimizations, and growth in available compute power (60% per year[23]) it will take 12 years for the network to become bandwidth-constrained to a level observed in the Bitcoin network today.

In short, consensus protocols solve a more general problem by agreeing on an arbitrary value. A proof protocol solves a more constrained problem in which there can only be one valid result.[8] Using a consensus protocol to prove "too complex" computations employs the wrong tool for the job, and weakens the trilemma parameters of the layer 1 blockchain. On the other hand, computations verified through proof protocols can be more complex at the same security budget, as only a single honest participant is needed.

3.4 Proving state

The Merkelized state under consensus in a blockchain makes it simpler for thin clients to verify the current state. Bitcoin introduced using merkle proofs in thin clients, and most other blockchains have continued using the same system to prove state.

Inchain retains the use of Merkle proofs to prove any state that is under layer 1 consensus, but proving layer 2 states cannot be achieved using Merkle proofs and must be proved using verifiable computations. Computing the correct state for layer 2 can be seen as outsourcing a computation, where the input to the computation is the data from the layer 1 blockchain, and the output is the layer 2 state. This is a deterministic computation, and proof protocols from verifiable computations can be used for this computation.

A sketch proof protocol for refereed verified computations would require asking n computers to compute the answer (Merkelized), and if there is any disagreement, execute a binary search through the steps of the computation to find the disagreement, finally executing the culprit step yourself.[13][4][22]

.

3.5 Consensus scalability challenges:

In principle, a system following an Inchain architecture could choose any existing consensus mechanism as its consensus layer. It could even opt for a combination of several unrelated consensus mechanisms as its consensus layer. For example, it could naively combine Bitcoin, Litecoin and Ethereum, and consider any piece of information to be both available and under consensus if it appeared on any of those blockchains. With a careful implementation to take care of reorganizations, these could be extended to more and more blockchains, scaling the consensuses to a broader universe. However, as the consensus scales into something broader and more complex, several issues will arise to eventually render the system infeasible:

- Consistency stability and delay: When several different consistency and availability mechanisms are combined in such a way that all are depended on for the system to work, the resulting combination becomes weaker than the sum of its parts: for eventually consistent mechanisms, reorganizations become more frequent and deeper. For final mechanisms, the percentage of the network we rely on for the continuation of the system grows.

- Computational requirements: in addition to handling a more unstable consistency and availability mechanism (which implies more reorganizations and more wasted computation), the combined result has a more complex and bigger state, increasing the required computational resources to handle it.

[8] A deterministic computation

- System fragility: relying on multiple pieces increases the chance that some might not work as expected. If nothing is done to mitigate this, the overall system becomes fragile.

In section 5 we explore an approach which will tackle these challenges and minimize their effects, allowing for an Inchain system consensus which could grow to embrace a greater number of underlying blockchains.

4 Background , existing art

4.1 Ethereum

Ethereum is the largest dApp platform and provides the ability to run Turing complete application code through its execution layer EVM. Execution of code requires a gas fee paid in Ethereum's native token, Ether. To verify the validity of a block, each node must make sure that the provided gas fee is enough to perform any execution of code that the transactions will trigger. This means that every node in the network must calculate the whole state of the Ethereum network for every block that the network produces. This is a bottleneck for the capacity of the Ethereum network, as the aggregated computation from all the nodes adds to the overhead of the network, increasing gas prices and reducing overall bandwidth. By decoupling the calculation of the gas fee from the block composition process, it is possible to dramatically reduce the amount of aggregated computation to therefore lower computation cost and increase useful execution bandwidth by orders of magnitude.

4.2 The Graph

The computations done by The Graph[21] are based on data from Ethereum. These computations are independent of the Ethereum consensus mechanism. The Graph executes computations described by manifests read from IPFS[1]. The project has discussed plans to work across multiple chains[10], but how computations are composed in the system remains unclear. For example, implementing a token spanning multiple chains requires composing computations.

4.3 Polkadot

Polkadot[24] is a blockchain with two levels of consensus systems. The relay chain is the primary consensus system, and each member chain has their own consensus system. Similar to a traditional blockchain, execution of smart contracts is tied in with the consensus mechanism. Polkadot needs special gateway constructions in order to read and write to self-sovereign blockchains, and these existing blockchains cannot be part of Polkadot. If a blockchain chooses to be part of Polkadot, then it cannot be self-sovereign as the relay chain consensus must dominate the local consensus.

4.4 Uniswap

Uniswap is a popular automatic market maker (AMM) project running on the Ethereum blockchain. Uniswap uses constant product liquidity pools that can be created for arbitrary token pairs as long as the tokens are available on Ethereum. However, 74% of the top 50 crypto assets are not

on the Ethereum blockchain, and 63% (16% excluding Bitcoin) are not on a smart contract-enabled chain[9]. Wrapped tokens are used as substitutes for tokens from blockchains without smart contracts, but this requires a substantial number of transactions and overhead. Still, wrapped BTC is the top non-stablecoin in terms of volume on Uniswap[10]. This indicates that AMMs that can connect a larger portion of the blockchain space is a market need.

4.5 Tether

Tether is a stable token pegged to USD. The Tether token is available on multiple platforms[11], including Omni and on Ethereum as an ERC20 token. Movement of Tether between the platforms is mostly done through centralized exchanges but requires the issuer, Tether Limited, or designated exchanges to manage liquidity between the different platforms. A Coinweb issued Tether or similar stable token enables trustless movement of the token between platforms.

4.6 Crypto payment gateways

There are several companies that offer crypto payment gateways[12]. Some gateways support more than 50 different crypto currencies. Their business model involves providing a payment interface for merchants to integrate on their webpages. The payment gateway can then accept crypto payments on behalf of the merchant and settle them—less a fee. The ability to accept payments in multiple cryptocurrencies enables the merchant to accept a wider range of customers holding different cryptocurrencies. It also lowers the payment threshold for the customers, as they are able to transact in the cryptocurrency of their choice. Current payment gateways are built as a centralized system wherein payment processing and outpayments to merchants are handled by the gateway as a trusted party. A similar payment gateway system could be implemented on Coinweb as a dApp, where stakeholders interests would be handled in a fully trustless manner.

5 Coinweb

An Inchain architecture can be extended to cover several consensuses; this means a DLT following this approach could be built on top of different DLTs, operating as a unifying layer over them to benefit from their combined network effect and different properties. Coinweb is an instance of Inchain architecture that tries to maximise the number of blockchains that can be covered this way, unifying them through a common currency, CWEB. In the following subsections, we'll explore the different challenges that arise from this, and how best to approach them.

[9]Adjusted for market capitalization, from coinmarketcap.com 2021-05-06. The top 50 assets have a combined market capitalization of $2.228M. The top 50 assets that are a) not on Ethereum have a market capitalization of $1.641M, b) not on a smart contract-enabled chain have a market capitalization of $1.402M, and $332M excluding Bitcoin

[10]Excluding ETH which is the base token for Uniswap

[11]Tether is available on 8 different blockchains as of today

[12]Such as Coingate, Bitpay and similar

5.1 Causally consistent views

One of the first problems found when attempting to increase the number of covered blockchains, is the linear increase in computation requirements and the worsening impact from reorganizations of underlying blockchains.

If we assume a simplified reorganization model for a single blockchain under which the chances for a block to eventually be substituted is p^h, where p is a constant and h is the depth of the block, it becomes clear that the combined chances p' that on at least one blockchain, a block with depth h will be substituted due to a reorganisations, approximates to 1 as the number of blockchains covered, n, increases.

$$p' = 1 - (1 - p^h)^n$$

This means that as more and more blockchains are added to be used on the consensus layer, the confirmation time apps will need to wait for will increase. Regarding the increase in computation resources, such as network bandwidth, CPU or memory, these will presumably increase when the amount of data to be processed increases. The problem presents such that if each single machine running as a node is forced to process everything, it will reach a point at which most conventional machines will not be able to run as a node, which will eventually lead to increased centralization. [25].

Our solution: relax the uniform view of the consensus layer into many different but causally consistent views, one per each covered blockchain, and reuse computations in a secure manner. The idea is, instead of enforcing every Coinweb client to hear and react to every block on every blockchain as soon as they are found, to introduce some artificial delay, such that when a new block is found on some specific blockchain, i.e Bitcoin, some Coinweb clients will immediately pick it up and start processing it, but most Coinweb clients will instead wait a short time before accepting it.

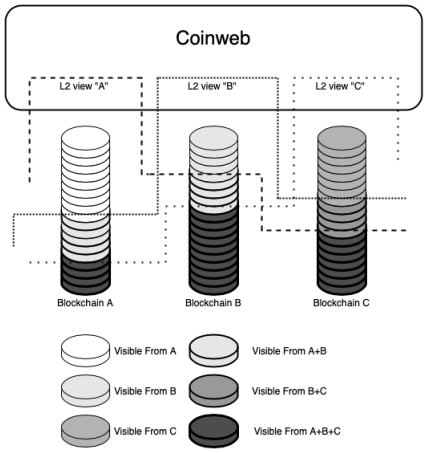

Figure 2: Inter-shard visibility

More specifically, we can define a directed, weighted graph wherein each vertex would represent a blockchain and each weight would represent a delay. Each Coinweb client may pick

an arbitrary vertex from this graph on which to focus its attention; every time the blockchain related to this vertex yields a block, the client will immediately process it, but if some other blockchain yields another block, it will wait for d confirmations[13] before beginning to process it, where d is the weighted distance between the graph vertex representing the blockchain that produced the block, and the vertex representing the blockchain the Coinweb client is focused on[14].

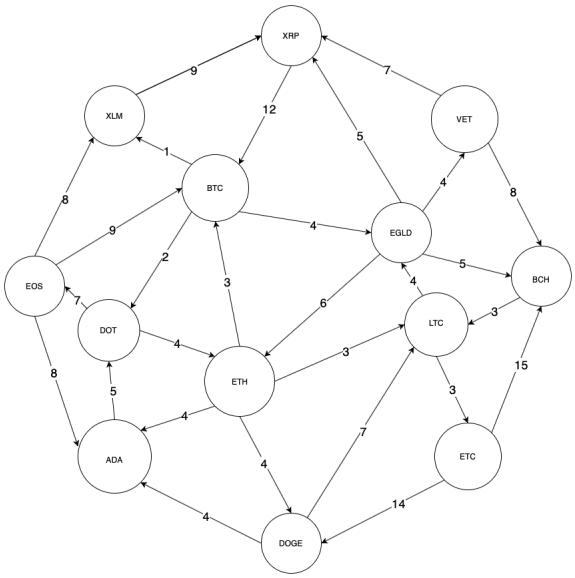

Figure 3: Example delay graph between blockchains

If we design such a graph to be sparse, limiting the maximum number of vertices of which each vertex can be connected to a constant g, and then set the minimum edge weight to d, we can see, under the usual assumptions and the proper delay values, that the reorganization instability issues disappear, as the reorganization probability p' at vertex A remains constant no matter the size of the graph:

[13]Understanding a confirmation as the number of blocks minted on top of a given block, regardless of the blockchain having finality or not.

[14]The reader might assume that a consistent definition of "d confirmations" requires anchors to create a deterministic ordering between the blockchains, but also a header timestamp in a block can be used for this purpose as long as keeping the timestamp drift within some bounds is part of the consensus protocol for the blockchain

$$
\begin{aligned}
p' &= 1 - \prod_{B}(1 - p^{1+distance_{AB}}) \\
&\leq 1 - \prod_{i=0}^{\infty} \prod_{B|hops_{AB}=i} (1 - p^{1+i \cdot d}) \\
&\leq \sum_{i=0}^{\infty} \sum_{B|hops_{AB}=i} p^{1+i \cdot d} \\
&\leq \sum_{i=0}^{\infty} p^{1+i \cdot d} g^i \\
&= p\left(\sum_{i=0}^{\infty} (p^d \cdot g)^i \right) \\
&= k
\end{aligned}
$$

Assuming d is big enough such that $p^d \cdot g < 1$.

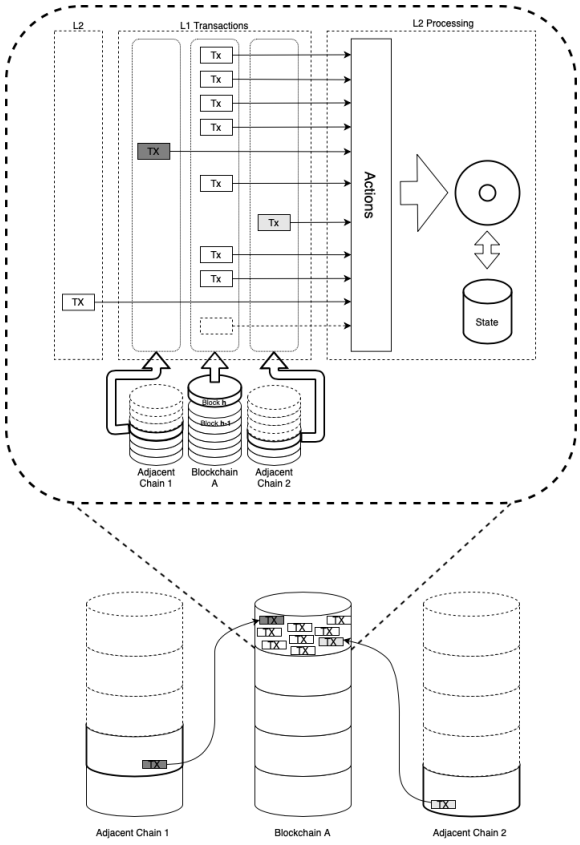

Figure 4: Delayed transaction processing

5.2 Reusing computation

Once the reorganization instability issue has been addressed, it's time to focus attention on the issue of computation resources. The key idea is to notice that, given there is at least one client focused on each blockchain and because of the mentioned delay, by the time a client focused on a blockchain A needs to react to a block w from another blockchain W, clients focused on some blockchain B, whose vertex is adjacent to W's vertex on the delay graph, have already reacted to block w. This means that if we can compute the reaction a client focused on A will have to w, based on the combination of these reactions, then we will be able to reuse most computations.

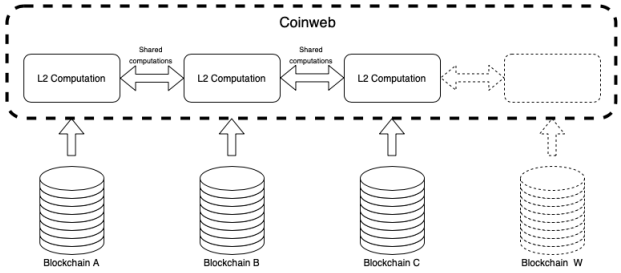

Figure 5: Computation sharing

5.3 Sharding

The global Coinweb state needs to represent information related to every covered blockchain, thus this state will need to be split into different shards in such a way that each Coinweb client will maintain a partial view; otherwise the requirements imposed by the size of this state would eventually limit the amount of blockchains Coinweb is able to cover. As the amount of shards required depends on the number of blockchains covered, and as we can assume that the information extracted from one blockchain is more related to itself than to information from another blockchain, a natural way of doing this is to associate a blockchain with a shard. This way, we will have a shard for information extracted from the Bitcoin blockchain, one for the information extracted from the Litecoin blockchain, and so on. For ease, each of these will be called shards, the level-2 blockchains (or l2-), and named as l2-bitcoin, l2-litecoin, etc. Two shards (i.e l2-blockchains) will be described as being adjacent, if their related blockchains are adjacent to each other on the delay graph.

Taking these points into consideration, the following schema for state composition and computation reuse can be sketched:

- We associate each shard to a covered blockchain, and state that this blockchain is the shard's underlying blockchain.
- Every time a new block from the underlying blockchain is found, we compute a new state for the shard; this way each block can be associated to a shard state. We can state that it is the shard at height h to that shard state computed after the block at height h.

- To compute the next shard state, we will use: its previous state, the new block from the underlying chain, and the set of the shard states from the adjacent shards that has not been used previously and have at least $delay_A B$ confirmations. Where A is the shard being computed's underlying blockchain, B is an adjacent shard's underlying blockchain, and $delay_A B$ is the weight of the edge from vertex B to A on the delay graph.

$$prev_{AB}(i) = \max_j \left\{ tstmp\Big(block_B(j)\Big) < tstmp\Big(block_A(i)\Big) \right\}$$

$$uncles_A(i) = \bigcup_{B|delay_{AB}<\infty} \bigcup_{j=prev_{AB}(i-1)}^{prev_{AB}(i)} shard_B\Big(j - delay_{AB}\Big)$$

$$shard_A(t) = eval\left(shard_A\Big(t-1\Big), block_A(i), uncles_A(t) \right)$$

This way, the required computational resources (i.e, network, CPU, memory) a Coinweb client requires could be kept constant regardless of the number of covered blockchains; However, as observant readers will have noticed, naive sharding in this way requires trust between the clients. For a relatively small amount of covered blockchains this would not be a problem, as each actor involved could simply run a small cluster of Coinweb clients in such a way that each of these clients would trust each other, but distrust any other Coinweb client outside of this cluster. But in order to scale to a large number of blockchains, this issue must be solved, as asking every actor to run a large computing cluster would become an unrealistic assumption.

5.4 Reusing computation between untrusted clients

As a way to prove state either for PoW blockchains or PoS blockchains already exists[15], we need only prove that the computation based on their state was done correctly. Given that this computation is deterministic, we can use a referee check of computation [4][13]: In order to verify a given foreign shard state is correct, the client would connect to multiple other peers and query for the hash representing the root of computation. If the client finds any discrepancy, it will initiate the referee check. As long as one single peer is honest, the client will be able to identify which it is.

The biggest risk with referee check of computations protocols is not finding an honest referee (in our case, a peer computing an adjacent shard). This could occur if, after a Sybil attack, the number of malicious peers is so high that it becomes difficult to locate an honest peer. To protect against this attack, we would require a percentage of the peers a Coinweb client connects to, to sign their messages with a cryptographic key associated with an old-enough proof of CWEB burn; this way, spawning many malicious peers—each one with a different proof of burn—would become economically unfeasible. On top of that, because signed messages would become a cryptographic proof that a client lied, using a gossip protocol would make it possible

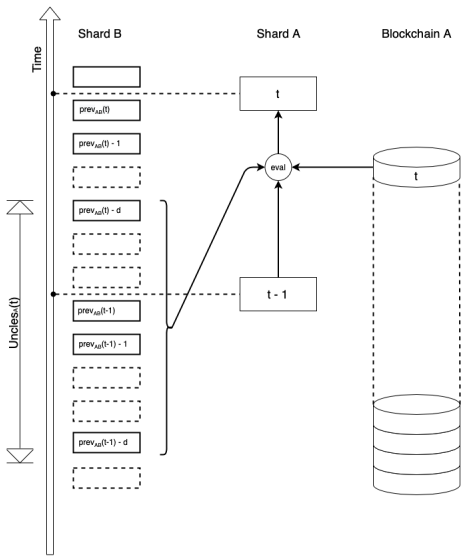

Figure 6: Visual representation of the shard state composition $eval$ function

to keep a distributed blacklist of malicious peers; These malicious peers would be ignored unless they change their identity (based on their cryptographic public key), but doing so would require paying the proof of burn again and waiting for enough time to elapse that the proof becomes old.

Because a shard will only consume the state of an adjacent shard after a delay (measured in number of block confirmations over the adjacent shard), it is possible to verify a shard state before it needs to be to consumed, such that the overall delay will not suffer—even if the referee check of computation protocol is time-consuming. For example, if we think of an adjacent shard over blockchain A with a delay of $2 \cdot d$, then when a new block t is found on A, we will be verifying the shard's state corresponding to block $t - d$ while consuming the shard's state corresponding to block $t - 2 \cdot d$. To further reduce the impact the check protocol could take [15], we can focus only on shard states over blocks with a height multiple of half the delay (in our example case this would be d), as verifying one implies verifying the ones before it.

5.5 Strong semantic coupling, loose timely coupling

Thus far, blockchains have been argued over as if they were systems whose behaviour could be modelled and approximated; but when long term scenarios come to play, the reality becomes more complex. We can think of extremely long, unexpected reorganizations due to bugs as happened with Bitcoin [16] and again [17]; or even due to political decisions as happened during Ethereum's DAO incident [18]. We can recall blockchains that forked into two or more branches where, from the old code point of view, neither was the original. And we can think of many

[15]Though each step of the protocol itself is relatively simple, because it is interactive and there are many peers could be involved, it becomes potentially time consuming

[16]https://news.bitcoin.com/bitcoin-history-part-10-the-184-billion-btc-bug

[17]https://siliconangle.com/2013/03/11/breaking-disagreement-between-bitcoin-clients-generates-fork/

[18]https://www.coindesk.com/ethereum-executes-blockchain-hard-fork-return-dao-investor-funds

promising blockchain projects that died after being abandoned, compromised, or attacked [19].

If Coinweb is determined to cover as many blockchains as possible, some of those covered blockchains can be expected to became unstable or even suddenly die. An obvious solution would be to curate a list of trusted stable blockchain projects, and to focus only on those—but that would mean missing the opportunity to cover interesting new experimental blockchains. Instead, we approach this challenge by taking the time-decoupling solution to the extreme:

Consider one of the worst case scenarios: on blockchain A, a critical vulnerability is uncovered and the project gets delisted and abandoned. Mining on blockchain A stops being profitable and, after a sharp drop of hash power, what would become A's last block is minted. Worse still, A dying would mean the eventual consistency of its state over which Coinweb's consensus is built will never become consistent. Coinweb's shard computed over A will then become inconsistent, followed by the adjacent shards, then the shards adjacent to those adjacent, and so on. In a matter of time, the whole Coinweb system will become inconsistent. In such an extreme scenario, expect the community to take aggressive action to save the project. However, once they do the harm suffered may be too great. The question becomes: how long does the community require to solve a situation like this? We've learned from previous experience that a proper reaction could take a matter of days or even longer, but for the sake of argument if we could predict A's fate with some weeks of advance notice then the solution would be rather simple: the Coinweb community could gather and, after some voting mechanism (i.e proof of stake, proof of burn), decide to stop covering A before it dies. User funds held on A shard would be lost, but the rest of the system would escape mostly unscathed.

The idea to make Coinweb robust against such scenarios is to develop an alternative, off-chain communication mechanism between shards, based on a combination of market mechanisms and the original Inchain communication. With that alternative mechanism, we expect to be able to extend delay from the magnitude order of minutes, to the magnitude order of weeks, without slowing down transactions or smart contracts interacting between the shards. This will be detailed further in the next subsection.

5.6 Information markets

We call an off-chain communication mechanism between shards an 'information market'.

An information market provides a service to an actor wherein a guarantee for information, including tokens, can be provided. This guarantee is manifested as a stake or similar, and remains in effect until the information is available. Requiring the use of an information market ensures a financial penalty can be levied against it for not providing correct service.

An information market focused on token transfers could be created by positioning token pools on different shards and releasing tokens on shard A upon deposit on shard B. This can be made trustless, since the transaction that deposits tokens on shard B is eventually visible by shard A. Smart contracts on shard A can penalise a market maker that did not release tokens, and also release the tokens back to the user. Similar action can be taken by a smart contract on shard B by observing whether tokens have been released on shard A.

For computations, the situation is a bit more complex, but a similar mechanism can be constructed. A smart contract on shard A can ask an information market maker to do a computation

[19] https://www.coinopsy.com/dead-coins/abandoned/

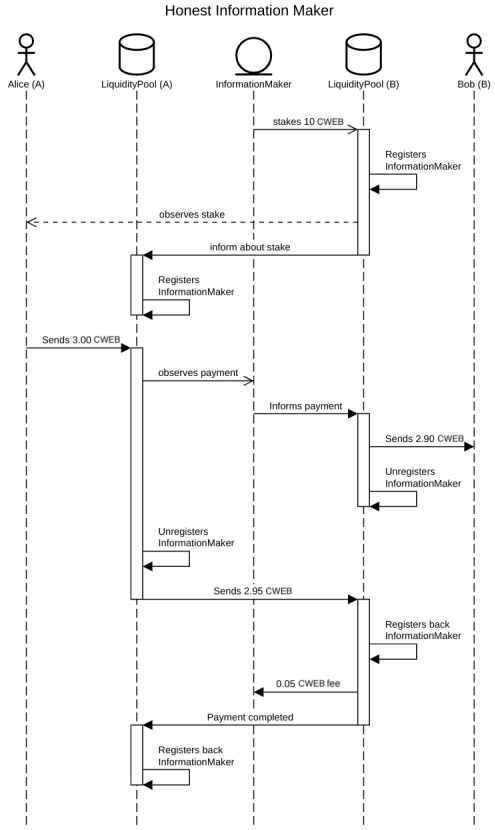

Figure 7: Information market maker example: Alice sends CWEB stored on blockchain A, to Bob on blockchain B. To expedite the process and avoid the long delay between blockchains A and B, Alice uses an information market maker.

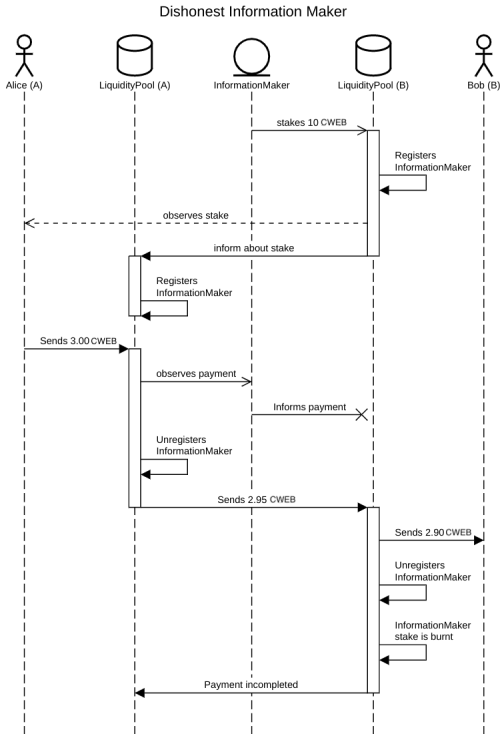

Figure 8: Same case as example 7, but on this case the information market maker, either intentionally or accidentally, misbehaves. We can see that regardless of this, Bob will get the same funds, though later than expected.

on shard B. This, effectively, is similar to seeing into the future from the point of view of shard A. The information market maker would give the smart contract the results of the computation, and this result would be associated with a financial penalty so that if it ends up being wrong, there will be a payout.

As only one honest actor is required to reveal a bad actor, and a bad actor is nevertheless revealed after an information propagation delay between the shards, there exist plenty of opportunities to construct mechanisms and incentives for honest actors to alert the contracts.

This off-chain communication mechanism between two shards separated by a long delay is based on two liquidity pools, one on each shard, controlled by smart contracts, and an information market maker feeding information to each liquidity pool. When a user (or smart-contract) on shard A want to send funds (or a message) to a user on shard B:

- it will instead send the message or funds directly to the A side of the liquidity pool;

- The contract on A's side will send the funds (or message) to B's side. This action will take some time;

- Meanwhile, a forwarder will send a message to B's side to immediately release the funds (or message), minus an applicable fee.

- B's side will immediately release the funds from its own pool, as it has not yet received the funds from A's side.

- Eventually, B's side will receive the funds from A's side, compensating back the balance.

Notice that even in the case that the information market maker misbehaves, the receiver

is guaranteed to eventually get the funds, as the B smart contract will automatically send the fund once it is received, if it has not done it before, plus a potential compensation; therefore a malicious actor could, at most, delay transfers. On the other hand, any misbehavior from information market makers would be detected and stake-penalized. As the liquidity pool risks losing its funds in case some of the underlying blockchain collapses, a charged fee is to be expected.

6 Coinweb's computation model

Having established a way to intercommunicate between a large number of blockchains, it becomes increasingly interesting to develop a computation model able to support all communication and smart contract opportunities that will arise, while keeping low fees and realistic requirements for those machines running as clients. Most computation models inspired by that of Ethereum are conceptually sequential, in the sense that transaction execution order matters. Even though conceptually sequential doesn't imply an actual sequential implementation, developing a convenient parallel implementation has so far proven to be an exceptionally challenging problem [19]. Instead, we have chosen to aim for an intrinsically parallel model on which, within a block, it would not be possible for smart contracts to be influenced by the order the transactions were issued. In this way, implementation would easily port to parallel frameworks such Map-Reduce or GPU and this, in addition, would remove or reduce some malicious incentives based on transaction reordering such as [7] or [5].

Following a BSP model [6], we split a transaction's computations from a transaction's actions, in such a way that each computation can run isolated from others (hence trivial to parallelize [map]), and each action will be a simple, reduce-like operation. The challenge with this approach lies on allowing smart contracts to interact with each other, and establishing how to prevent a single computation becoming the bottleneck of the whole map-reduce operation. Our solution is simple, but with deep implications: Allow smart contracts to issue their own transactions as if they were external actors. On the following subsections, we develop a computation model based on these ideas:

- External and internal actors (i.e smart contracts) are treated the same way. That is, the only way smart contracts can modify the state of the system is issuing transactions, and these transactions are processed the same way as transactions from external actors.
- Smart contracts can not observe the order of those transactions that are issued at the same batch.
- Transactions embedded at the same block belong to the same batch. A transaction issued by a smart contract triggered by a batch belongs to the next batch. Batches are executed sequentially, one after another. There's a constant number of batches per block; the batch after a block's last batch, is the next block's first batch. This means that a **chain of reactions between smart contracts may span several blocks**.

6.1 Claims

Because private keys used inside smart contracts can not be hidden, we generalize the concept of signature to that of $claim$. A claim is a combination of a unique nonce; a message, called the *claim-payload*; and an actor, such that it can proved the message at the given nonce was issued by said actor. In the case of external actors, a claim is proved by a cryptographic signature, in case the sender is a smart contract (i.e an internal actor), the claim is proved by keeping track of which messages were issued by which smart contract.

In Coinweb, when an actor creates a claim it pays a fee in the form of CWEB. The claim then becomes visible to any actor until, at any later time in a future transaction, the actor decides to delete it, partially recovering the CWEB paid. Only the actor that originally created the claim, whom will be called the claim owner, is allowed to delete it. In this way, claims on Coinweb fulfill a similar role as UTXOs do for Bitcoin-like blockchains [20].

6.2 Smart contracts

On Coinweb, smart contracts are pure functions—stateless functions without side effects which can be modelled as mathematical applications—taking as input a payload, and returning as output either a Coinweb transaction or an error. They are encoded as a combination of a piece of web-assembly code, plus a set of parameters, represented as a raw bytestring. They are referenced either explicitly (i.e, as the web-assembly code and parameter itself), as the hash of their explicit representation, or as a reference to a previous execution. Two smart contracts are considered to be the same smart contract if they can be represented by the same hash.

Coinweb smart contracts are considered to be system actors, and the system keeps track of any claim made by them.

As smart contracts are stateless, they are neither created nor destroyed, in some way you can consider that every possible smart contract already exists, waiting to be called for the first time.

6.3 Actions and transactions

On Coinweb, $actions$ are the set of primitives that enable smart contracts and users to interact with each other. Each type of action is associated with a fee (issued in CWEB), which would be negative for $delete - claim$ actions, and positive for any other type of action. Actions are embedded in Coinweb transactions. The combination of the transaction ID (txid) plus the action position within its transaction will become the **action-id**. Because it is not possible for two different transactions to have the same txid, no two actions will share the same action-id.

Actions are not issued on their own, but as a part of a transaction which, on Coinweb, is no more than a set of actions, an initial CWEB balance, and a reference to how it was issued. They are termed in this way because when an action within a transaction fails, it propagates the failure to every other action from the same transaction, resembling some properties of the concept of a transaction in a database or in other cryptocurrencies. However, Coinweb transaction are not "transactions" in the formal sense, as they are neither atomic (when they do fail, they do not rollback any state change they might have already generated) nor isolated (two different

[20]For a brief introduction to the UTXO concept: https://river.com/learn/terms/u/unspent-transaction-output-utxo

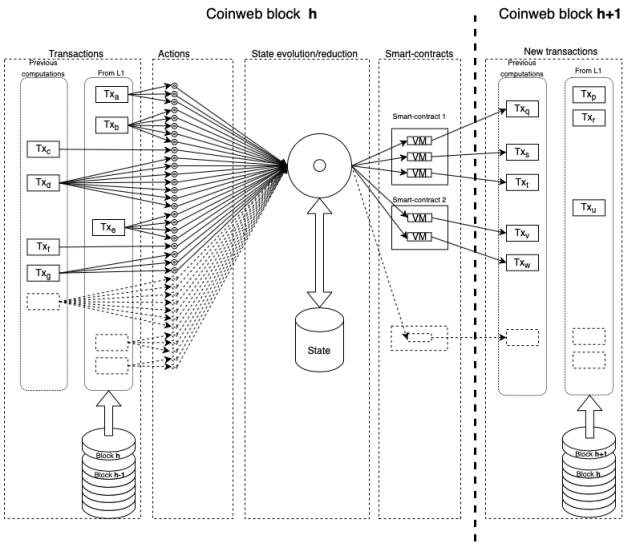

Figure 9: Parallel execution of smart contracts

actions being executed can sometimes influence each other even when they belong to different, simultaneously executed, transactions).

Transactions parsed from the underlying blockchain have an initial balance of 0 CWEB, and are considered to be issued or owned by the external actor who signed them; for any other transaction, the initial balance and ownership depends on the action after which they were created.

The different types of actions are:

- **create-claim**: takes a claim-payload as argument and stores it as a claim made by the issuer of the transaction within which this action is contained. The fee for this action is determined by the user, but there will be a minimum fee value determined by the governance parameters based on the claim-payload size.

- **delete-claim**: takes a reference to a claim and deletes it. The fee for this action is determined by the governance parameters based on the claim-payload size and age, minus the original fee paid to store the claim. It is expected for this fee to be negative.

- **lookup-claim**: takes a query as argument and returns its result. The fee will be determined by the governance parameters based on the query complexity.

- **send-message**: generates a payload and uses it as an input for a smart contract, transferring to it both CWEB and gas. This action takes as argument the amount of CWEB to transfer, the amount of CWEB to use as gas, the target smart contract, and an aggregation-expression. Aggregation-expressions are explained on subsection 6.5 and define how to compute the payload based on the result from lookup-claim actions within the same transaction to which the actions belong, and aggregated values from other send-message payloads sent by the same issuer at the same execution stage step. The resulting transaction of the smart contract invocation establishes the smart contract as owner, and

transfers the field amount as the initial balance. The action fee would total the sum of the transferred CWEB amount, plus the gas CWEB amount, plus a fee determined by the governance parameters based on the aggregation-expression complexity.

- **send-transaction**: similar to ”send-message”, but targeting a shard as destination rather than a local smart contract. This action takes as argument the amount of CWEB to transfer, the target shard, and an aggregation-expression. The result of the aggregation-expression is a new transaction that will be processed on the target shard. The issuer of the new transaction is considered to be the issuer of the previous transaction containing the send-transaction action. The initial balance of the new transaction is the same as the transfer field. The action fee would total the sum of the the transferred CWEB amount, plus a fee determined by the governance parameters based on the aggregation-expression complexity and the target shard.

6.4 The execution pipeline

On Coinweb, actions are executed in batches following a static pipeline; in turn, each stage of this pipeline is executed following a parallel map-reduce process. When an action passes through one of these stages it will either succeed or fail; if it succeeds, it passes to the next stage. If it fails, it is discharged from the pipeline.

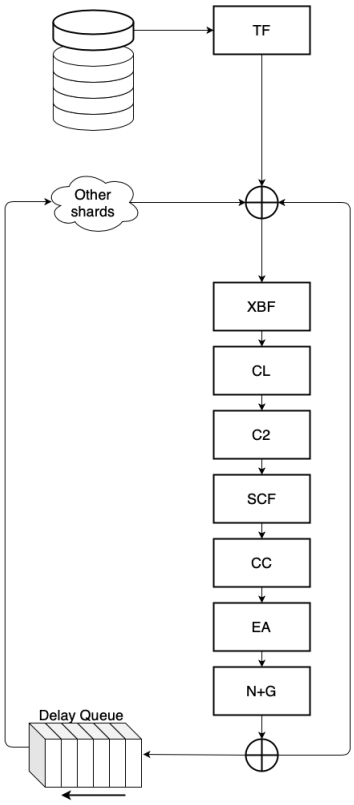

Figure 10: Execution pipeline

The different stages are:

- Transaction Fetch (**TF**): receives transactions embedded from the underlying blockchain and transactions sent from the adjacent shards, and unpacks them as actions.
- CWEB Balance Filter (**XBF**): computes transactions' required fees as the sum of the fee of each of their actions, and removes all actions belonging to transactions with a fee greater than their balance.
- Claim Lookup (**CL**): executes the lookup-claim action queries, retaining the results for later stages.
- Claim Delete (**CD**): executes the delete-claim actions. For each of these actions,the referenced claim is marked as deleted only if the action issuer and the claim owner were the same; otherwise, or if the referenced claim did not exist, the delete-claim action will have no effect.
- Spent Clash Filter (**SCF**): removes all actions that belong to transactions containing a delete-claim referencing a claim that either didn't exist, didn't belong to the issuer, was referenced by another delete-claim on the pipeline from the same issuer, or was already marked as deleted. The role of this stage is to prevent double-spends, but **if there are two competing transactions for the same claim, there will be no winner and both will fail**. Furthermore, the claim will remain deleted (assuming the action issuer was the owner) and effectively burnt. This disincentivises double-spend attempts at the application level.
- Claim Create (**CC**): adds entries to the claim DB.
- Expression Aggregation (**EA**): carries out several reduce rounds until every aggregation is done. Because the number of reduce rounds is equal to the maximum height of these expressions, there is a limit on this height set as governance parameters.
- New Transaction Generation (**NTG**): for send-message actions, now that the payload is computed, we load the smart contract WebAssembly code and execute them together on their own isolated VM. Note that there would be 1 VM spawned for each send-message action, even if all send-message actions target the same smart contract. The result of the smart contract is parsed and interpreted as a new transaction, then is fed back into the pipeline. The maximum memory size and number of steps the VM would be able to use would be limited, based on the governance parameters and the gas the send-message action declared. If the result could not be parsed as a transaction, the maximum memory or steps was reached, or an exception halted the result, then the action will be considered to have failed and the result will be discharged. For send-transaction actions, this stage is very similar, but its VM invocation is omitted and the output will be directly parsed from the input without any intermediate computation.

When a new block on the shard's underlying blockchain is found:

- The delay queue is popped, any transaction that has been waiting long enough is sent to the target shard where it will eventually be processed.

- The TF stage is called. It parses the new block, extracting any embedded Coinweb transaction.
- We combine into a single batch, all these new transactions plus the transactions issued on the previous NTG-stage execution (made on the previous block) and transactions coming from any of the adjacent shards.
- Starting with this transaction batch as an input, we execute the CL, CD, SCF, CC, EA, and NTG stages in a row, pipelining the result from one stage into the next. The result from the last stage, NTG, will be a new set of transactions. Those targeting a different shard will be sent to the delay queue, those targeting back the same shard will be grouped together to form the next batch.
- The next batch will be sent back to the CD, SCF, CC, EA and NTG stages to be executed the same way as its predecessor, creating a newer batch. This way, the stages will be iterated N times, where N is a governance parameter.
- After the N loops, the system saves the last result from the NTG execution, and halts waiting for a new block from the underlaying blockchain.

6.5 Aggregation-expressions

Because several messages can be sent simultaneously to a smart contract, but for each of these messages a different isolated VM is spawned, a mechanism to enhance the information a VM can observe is needed, so it not only sees a single message, but also the relevant information related to the combination of every message sent as a whole. Think, for example, of an auction smart contract that needs to tell apart the biggest bid from any other message, or a token smart contract that needs to compose a transaction based on the combination of several messages, or a lending smart contract that needs to know if it has enough funds to lend to every message being sent. The goal of the aggregation-expressions is to cover this need.

Aggregation-expressions are very similar to the concept of windows functions in query languages such SQL. At stage **EA**, every aggregation-expression from the same issuer (either external actor or smart contract) defines a context or *window* on which to run the collective operation defined by the aggregation-expressions.

Aggregation-expressions are modelled as Abstract Syntax Trees (ASTs), wherein each internal branch node represents an operation primitive (i.e max, $first$, $average$) and each leaf represents either a literal, or a reference to a *claim-lookup* action from the same transaction. Aggregation-expressions are computed over a given context recursively:

- Different aggregation-expressions are regrouped into several sub-contexts, so that in every sub-context each aggregation-expression will have the same root node.
- For every sub-context:
  - If the root node is a leaf node:
    * If it is a reference to a *claim-lookup*, return the claim content; otherwise, if it is a literal, return the value of the literal

  - Otherwise, if it is a branch node:
    * Recursively execute every child as an aggregation-expression.
    * Over the set of returned values, run the collective operation represented by the branch-node.

- Return the union of the value sets returned from each sub-context.

Following the above pattern, not only can a set of aggregation-expressions be computed in parallel as map-reduce iterations, but the number of these iterations can be estimated based on the number of iterations it will take to compute the most complex of these expressions. This complexity estimation, measured as the number of map-reduce iterations, is what will be called the expression height. Any transaction containing an aggregation-expression with a higher height than allowed by the governance parameters will be considered invalid.

Primitives will be made out of collective operations that have the following properties:

- Their height can be statically estimated.
- The amount of computation or data to transfer grows log-linear with the size of the set over which they are run.
- The output set size is the same as the input set size.

For example a primitive to $concatenate$, where every returned result will be the concatenation of every input, would not be allowed, because the amount of data to transfer will grow quadratically — but a parameterized primitive $concatenate_n$, where n is a value known at the time the transaction was issued, and that for every input it will return the concatenation over a fixed window of size n around it, would be allowed. To keep the overall computation agnostic to the order in which the expression was issued, any non-commutative primitive, such as $first$, $last$, $atPosition_n$, or $concatenate_n$ will only be allowed after an explicit $sort$ operation.

Note that because the final goal for an aggregation expression is to produce a set of structured data to work as smart contract input or to represent a transaction to be issued, most of these primitives will be scalar functions (height 0); either commonly-used math functions (scalar $+$, scalar and, $round$, etc), functions to construct or deconstruct hierarchical data (i.e json, xml, dictionaries, trees, etc) or functions to work as flow control (i.e $ifThenElse$, $coalesce$, $nullif$, etc).

6.6 Overcoming computation limits

Similar to other cryptocurrencies, fees in the native currency, CWEB, will be used as an anti-spam mechanism. Specifically in the case of Coinweb, fees will control how many actions a transaction can contain, and how complex these actions can be. So, in principle, as long as one can afford it, a computation could be as complex as required. Considering a practical implementation, if this would be allowed directly, a single computation could become a bottleneck, holding up the whole pipeline execution. This is why some limits are fixed no matter the size of the fee the issuer is willing to pay. For aggregation-expression this limit is a maximum height. For smart contract VM invocations this limit is a maximum number of steps to be run, and the

maximum of memory to be used. So what happens when some computation does not fit within these limits?

The observant reader might have already recognised that these constraints are not too tight if we take into consideration that a smart contract can issue a transaction containing messages back to itself, or that an aggregation-expression could return a transaction containing a *send-transaction* with an aggregation-expression based on the previous aggregation-expression. This means that if a computation is too complex to be computed within a batch, it could be spanned into several computations in order to be gradually computed. For example, if a computation were so big that it would require 10,000 steps of the wasm machine, but the maximum allowed is 6,000: then the smart contract could execute 5,000 steps of the computation, use another 1000 steps to serialise a transaction with a message back to itself containing the state of the computation, and finish. Then, on the next batch, it will receive the message, deserialise it, and continue with the remaining 5000 steps.

Because a smart contract can continue sending messages back to itself, then, considering the most extreme case, there is no reason why a single smart contract computation could not span into several blocks. This opens an interesting possibility: As long as a smart contracts can continue earning CWEB, it could use the accumulated CWEB to keep sending messages back to itself indefinitely in what would become a never-ending execution, **effectively becoming a daemon**. Furthermore, a smart contract daemon could be used to register callbacks to be triggered to other smart contracts after specific events, **enabling reactive smart contract execution**.

7 Conclusion

We introduced and demonstrated the benefits that the *Inchain* architecture could bring to the crypto space by interconnecting different self-sovereign blockchains without compromising their underlying consensus. We also introduced *Coinweb*, a proposed implementation of *Inchain* architecture. We showed that Coinweb, as a multichain Inchain implementation, would significantly increase the solution space for dApps compared to what is currently available.

References

[1] Juan Benet. IPFS - content addressed, versioned, P2P file system. *CoRR*, abs/1407.3561, 2014.

[2] Blockchair. Ethereum blockchain size. `https://blockchair.com/ethereum/charts/blockchain-size`, jul 2019.

[3] Vitalik Buterin. Ethereum: A next-generation smart contract and decentralized application platform. `https://github.com/ethereum/wiki/wiki/White-Paper`, 2013.

[4] Ran Canetti, Ben Riva, and Guy N. Rothblum. Refereed delegation of computation. 2011.

[5] Yixin Cao, Chuanwei Zou, and Xianfeng Cheng. Flashot: A snapshot of flash loan attack on defi ecosystem. 2021.

[6] Daniel Cordeiro, Alfredo Goldman, Alessandro Kraemer, and Francisco Pereira Junior. Using the bsp model on clouds. 2013.

[7] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges. 2019.

[8] D.Larimer, J.Lavin, N.Hourt, Q.Ma, and W.Prioriello. Eos.io technical white paper v2. https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md, 2017.

[9] Team Elrond. Elrond: A highly scalable public blockchain via adaptive state shardingand secure proof of stake. https://elrond.com/assets/files/elrond-whitepaper.pdf, 2019.

[10] The Graph Foundation. The graph - evaluating multiblockchain. https://thegraph.com/blog/evaluating-multiblockchain, 2021.

[11] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, pages 465–482, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[12] L.M Goodman. Tezos — a self-amending crypto-ledger. https://whitepaper.io/document/376/tezos-whitepaper, 2014.

[13] Yuqing Kong, Chris Peikert, Grant Schoenebeck, and Biaoshuai Tao. Outsourcing computation: the minimal refereed mechanism. *CoRR*, abs/1910.14269, 2019.

[14] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf, 2008.

[15] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009.

[16] Jakob Nielsen. Nielsen's law of internet bandwidth. https://en.wikipedia.org/wiki/Jakob_Nielsen_(usability_consultant)#Nielsen's_law_of_internet_bandwidth.

[17] Poma. How to find bottleneck in geth sync? https://ethereum.stackexchange.com/questions/79602/how-to-find-bottleneck-in-geth-sync.

[18] BitMEX Research. Bitcoin's initial block download. https://blog.bitmex.com/bitcoins-initial-block-download/, nov 2019.

[19] Vikram Saraph and Maurice Herlihy. An empirical study of speculative concurrency in ethereum smart contracts. 2019.

[20] Péter Szilágyi. Geth v1.9.0 - performance. https://blog.ethereum.org/2019/07/10/geth-v1-9-0/, jul 2019.

[21] Yaniv Tal, Brandon Ramirez, and Jannis Pohlmann. Thegraph: A decentralized query protocol for blockchains. 2018.

[22] Jason Teustch and Christian Reitwießner. A scalable verification solution for blockchains, 03 2017.

[23] Wikipedia. Moore's law. `https://en.wikipedia.org/wiki/Moores_law`.

[24] Gavin Wood. Polkadot: Vision for a heterogeneous multi-chain framework. 2016.

[25] Brent Xu, Dhruv Luthra, and Zak Cole andNate Blakely. Eos: An architectural, performance, and economic analysis. 2018.

8 Appendices

8.1 Space analogy

The multi-chain communication system in Coinweb can be explained by way of a universe. In this universe, many blockchains can be observed as stars. Some stars shine brighter than others, and floating around each of the stars are space rocks and planets attracted to the gravitational field of the stars. The rocks and planets can be viewed as transactions and smart contracts.

In this universe, the laws of relativity holds between the solar systems, and the speed of light is constant in all referrence frames.

In our universe, one blockchain is called Alpha Centauri A, or A. Another blockchain is called Alpha Centauri B, or B. A third blockchain is called Proxima Centauri, or C.

The different planets and stars can see each other, but they cannot easily communicate[21]. Stars are typically five light years apart, so a smart contract close to one star would observe information from another star that is 5 years old. Sending a message to a smart contract at another star would take 5 years.

Some stars are close to each other, such as Alpha Centauri A and B. The designers of this universe might see that these stars' stability and security are inherently linked, such as the case might be if a smaller blockchain makes use of merge mining to make its security depend on another blockchain. Communication between these stars is quick.

Unbeknownst to the smart contracts themselves, the stars have been deliberately placed far apart in the universe. An exploding star destroys everything around it, and the blockchain it inhabits. Stars that are close to each other would consume each other in a disastrous supernova, or the creation of a black hole. On the positive side, this universe is inhabited by, from the point of view of the smart contracts, magical creatures called external actors. These external actors are not bound by the theory of relativity, and can see actions as they happen across the whole universe.

A smart contract can ask these external actors for help. One external actor that is particularly popular is a fortune teller who, for the sum of just one GOAT, can tell a smart contract what is happening at a nearby star. Smart contracts are, by nature, suspicious of all external actors, including fortune tellers, and do not trust them to tell the truth. They are not willing to sacrifice 1 GOAT for a fortune teller that could potentially lie to them. Smart contracts, on the other

[21]The stars and planets related to each other is treated as a point in space in this case, as each blockchain has a defined order of events. Thus while relativity of simultaneity holds between stars, the order of events within a solar system is the same for all reference frames

hand, can be trusted to always do as they are programmed. They are bound by laws of nature and the computer instructions they are made of.

The fortune teller and the smart contract make an agreement:

- The smart contract agrees to pay 1 GOAT to the fortune teller,
- The fortune teller gives the smart contract 2 GOLD (the equivalent to 10,000 GOATs or more), as well as the results of the observations of the other star,
- If the fortune teller lies, the smart contract will be able to keep 2 GOLD.
- If the fortune teller is honest, their 2 GOLD will be returned.

The smart contract simultaneously announces to the world that if anyone has reason to believe that the fortune teller is not being truthful, they will receive 1 GOLD in compensation. However, any public accuser must temporarily hand over 2 GOLD, which they also will get back if the accusation turns out to be true.

Now if just **one single truthful fortune teller** exists in the system, the correctly observed result will be given to the smart contract, or the observation will fail (more complex approaches exist that will always succeed—as in Coinweb). In its further calculations, the smart contract now uses the fortune teller's results as if it was true. The smart contract might reinsure its own results in case it was wrong by using some of its 2 GOLD, or a client to the smart contract can ask a fortune teller to insure it by paying, for example, 1 GOAT. When light from the close star reaches the smart contract, after almost 5 years, the 2 GOLD is released to the fortune teller if the prediction was truthful.