Proof of Claim: A Fast, Fair & Collision-Proof Decentralized Node Election Algorithm

By: Andrew Nicholas Smith
Title: Founder
Organization: Versatus Labs
Email: as@versatus.io
Organization Email: info@versatus.io
Website: versatus.io

Abstract

Distributed and decentralized systems necessitate robust and unbiased node election mechanisms to ensure their efficacy and fairness. This paper introduces the Proof of Claim (PoC) algorithm, a novel approach to node election that offers a rapid, equitable, and collision-resistant solution adaptable to both synchronous and asynchronous configurations. Contrasted with prevalent election algorithms, PoC emerges as a superior mechanism, underpinned by a verifiably random seed, a unique claim hash for node identification, and the XOR operation. The algorithm's design inherently precludes biases favoring capital-rich node operators, ensuring elections remain resistant to undue influence from hardware or monetary accumulation. Empirical benchmarks further underscore PoC's scalability and speed, even in networks with extensive node participation. In summation, the Proof of Claim algorithm presents a pioneering solution for provably fair, randomly distributed, and unbiased node elections, marking a significant advancement in the realm of distributed and decentralized systems.

Introduction

In decentralized and distributed systems, the notion of "elections" plays a pivotal role[1]. Election algorithms serve various purposes, including leader selection, quorum formation, and task allocation[2]. Historically, these mechanisms have been plagued by challenges such as reliance on trusted nodes, potential for collisions, non-uniform distribution, and biases towards nodes with superior hardware or monetary resources[3]. As global systems gravitate towards decentralization, there's an imperative need for election mechanisms that ensure fairness, obviate trust dependencies, and operate efficiently[4]. This paper delves into prevalent leader election techniques and introduces the Proof-of-Claim (PoC) algorithm as a solution to the aforementioned challenges.

Consistent Hashing

Consistent hashing, primarily employed in distributed systems, facilitates efficient data distribution across nodes[5]. Unlike conventional hashing, which necessitates rehashing upon node alterations, consistent hashing minimizes key relocations by mapping nodes and keys onto a circular space. While its primary application lies in load-balancing, this paper emphasizes its role in leader election.[6] Despite its advantages, consistent hashing presents several limitations:

- **Non-Uniform Distribution:** Nodes may experience uneven leadership probabilities, exacerbated by network growth and node fluctuations.

- **Implementation Complexity:** Precise implementation is paramount to avoid imbalances, necessitating meticulous hashing function selection and virtual node mechanisms.

- **Latency in Leader Transition:** Rapid leader transitions can be hindered by identification and propagation delays.

- **Stable Node Identification Dependency:** Unstable node identifiers can induce frequent leader changes, introducing system latency.

- **Partition Handling Challenges:** Network partitions can result in multiple leader elections, culminating in "split-brain" scenarios that are arduous to rectify.

Proof-of-Work (PoW)

Though PoW is fundamentally a Byzantine Fault Tolerant consensus mechanism, it inherently encompasses a decentralized leader election component[7]. Nodes compete to find suitable hashes for specific problem sets. While PoW offers trustless advantages, it is not devoid of limitations:

- **Collision Vulnerability:** Multiple proofs can meet criteria simultaneously, leading to "Orphaned Blocks" in networks like Bitcoin.

- **Resource Intensiveness:** The algorithm's nature demands significant computational power and energy.

- **Bias Towards Resource-rich Nodes:** Nodes with superior computational capabilities are favored.
- **Susceptibility to Centralization:** Over time, nodes with superior resources can dominate the network.

Proof-of-Stake (PoS)

Proof-of-Stake (PoS) operates as a comprehensive Byzantine Fault Tolerant consensus mechanism tailored for decentralized networks. At its core, PoS employs economic incentives, compelling nodes to align with the overarching objectives of the network[8]. This alignment is achieved by mandating nodes to commit a substantial monetary stake as collateral, thereby ensuring their vested interest in the network's well-being. Intriguingly, beyond its consensus role, PoS also functions as an election algorithm. A salient feature of most PoS implementations is their reliance on a designated "protocol" for leader election during state transitions[9]. The probability of a node's election can be influenced by the magnitude of its stake in some implementations, while in others, the selection process is entirely randomized[9]. This variability in implementation introduces a spectrum of challenges. While PoS inherently prevents collisions, it can be predisposed to issues of centralization and resource accumulation, reminiscent of the challenges observed in the PoW mechanism. Alternatively, it might necessitate unwavering trust in a "protocol" to ensure unbiased and randomized node elections[8].

Proof-of-Claim (PoC)

Proof-of-Claim (PoC) stands out as a specialized election algorithm, distinctly segregating node election from overarching consensus mechanisms. Its objectives resonate closely with those of Consistent Hashing. However, drawing parallels with PoS, PoC hinges on verifiable data, specifically an object retained within the global state of a system. This paper posits that PoC adeptly circumvents the limitations inherent to consistent hashing. It guarantees a verifiably random distribution, dispels latency apprehensions, ensures unwavering node identification, and champions simplicity in its architectural design. Contrasting with PoW, the PoC algorithm is inherently immune to collisions, broadening its applicability across diverse system functionalities. Mirroring the energy efficiency of PoS, PoC further distinguishes itself by obviating the need for centralized "protocol" trust or the dominance of resource-abundant participants. Its foundational principles, rooted in verifiability, provable randomness, and fairness, combined with the eventual transparency of its core inputs to the entire network, render PoC as synchrony-agnostic. This flexibility allows for its integration into broader systems, irrespective of whether they mandate unanimous node agreement within a stipulated timeframe or permit staggered conclusions.

Verifiable Random Seed in PoC Algorithm

The efficacy of the Proof-of-Claim (PoC) election algorithm hinges on the incorporation of a Verifiable Random Function (VRF). A VRF is a cryptographic construct that yields a randomized output for a given input, analogous to a hash function. Two salient properties characterize VRFs:

1. Uniqueness: For a designated input and associated private key, the VRF consistently generates an identical random output, ensuring the reproducibility of the randomness.

2. Verifiability: Concurrently with the random output, the VRF furnishes a proof. Possessors of the pertinent public key can employ this proof to ascertain that the random output was indeed derived by the private key holder for the stipulated input, all the while remaining oblivious to the private key itself[10].

Ensuring Fair Input

For the VRF to yield an equitable output, it necessitates an unbiased input. Consequently, meticulous deliberation is imperative when selecting the data serving as the input to engender an election seed[11]. The nature of data harnessed as input to the VRF may vary contingent on the system's objectives and configuration. Implementations of the PoC algorithm must judiciously discern the data types to be utilized, ensuring they preclude potential system manipulations. Optimally, the input data should be beyond the influence of the Verifiable Random Seed's producer. In Byzantine environments, where nodes might conspire deceitfully, the input data should emanate from a Byzantine Fault Tolerant process[11]. This implies that a consortium of nodes, provided a minimum subset remains trustworthy, should shoulder the responsibility of generating the data serving as the VRF input.

VRF Selection Considerations

While an exhaustive exploration of VRF implementation exceeds the purview of this discourse, it remains paramount for PoC algorithm implementations to exercise prudence in VRF selection. For bespoke implementations, rigorous evaluations and audits are indispensable. However, leveraging vetted, open-source VRF implementations is often the more judicious choice[12].

Several reputable, peer-reviewed, and audited open-source VRFs implementations are available across a variety of ecosystems[1].

Generation of the Verifiable Random Seed

Within the context of this exposition, the PoC algorithm employs a random seed, constrained between the maximum unsigned 32-bit integer and the maximum unsigned 64-bit integer, coupled with a 256-bit claim hash. Although the seed could theoretically span a minimum of 16 bits and the claim hash could exceed this, the algorithm's fairness and collision-proof nature remain intact provided the network's node count remains below the seed's maximum value and the claim's hashing algorithm is robust. However, should the seed be restricted to 8 bits with a node count exceeding 255, certain nodes would invariably be excluded from election victories. Given the virtually collision-proof nature of the SHA256 algorithm and the expansive range between the 32-bit and 64-bit unsigned integer maxima, it is aptly suited for extensive node networks, ensuring an equitable election probability for each node.

XOR Metric in PoC Algorithm

The Exclusive OR (XOR) operation is a fundamental binary operation that accepts two binary inputs and yields a binary output[13]. The operation returns true (or 1) if, and only if, precisely one of its inputs is true (or 1). If both inputs are identical (either both 0 or both 1), the XOR operation outputs false (or 0). This characteristic ensures that when two distinct inputs undergo XOR against a common value, provided they differ by a single bit, their results will also differ[14].

Truth Table for XOR

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

[1] For the Rust language, which the benchmark at the bottom of this paper was written in, crates.io/search?q=VRF will return a handful of implementations, with statistics on their use. Though this is not an alternative to audit, it likely indicates the developer community's review of the specific implementation in question.

The XOR operation holds significance in Boolean algebra and cryptographic systems[13]. In the system under consideration, the claim hashes of nodes are XORed against the verifiable random seed. Given the 64-bit nature of the random seed and the 256-bit stature of the claim hash, a direct XOR operation would only engage the lower 64 bits of the claim hash. This could potentially induce collisions, as the likelihood of two distinct SHA256 hashes sharing identical lower 64 bits is considerably higher than the probability of two SHA256 hashes being completely identical[15]. To preserve the collision-proof attributes of the PoC election, the 256-bit SHA256 representation is segmented into four distinct 64-bit values. Each segment undergoes XOR against the 64-bit seed and is subsequently recompiled into a new 256-bit unsigned integer.

```
                 Claim Hash (SHA256 - 256 bits)
+----------------------------------------------------------------+
|                                                                |
|   +--------+   +--------+   +--------+   +--------+            |
|   | 64-bit |   | 64-bit |   | 64-bit |   | 64-bit |            |
|   |   1    |   |   2    |   |   3    |   |   4    |            |
|   +--------+   +--------+   +--------+   +--------+            |
|                                                                |
+----------------------------------------------------------------+


                Verifiable Random Seed (64 bits)
                            +--------+
                            |        |
                            | 64-bit |
                            |        |
                            +--------+
    XOR Operation (For each 64-bit segment of the Claim Hash)
        +--------+   +--------+   +--------+   +--------+
        | 64-bit |   | 64-bit |   | 64-bit |   | 64-bit |
        |   1    |   |   2    |   |   3    |   |   4    |
        +--------+   +--------+   +--------+   +--------+
               |            |            |            |
             | XOR        | XOR        | XOR        | XOR
               v            v            v            v
        +--------+   +--------+   +--------+   +--------+
        | 64-bit |   | 64-bit |   | 64-bit |   | 64-bit |
        | Result |   | Result |   | Result |   | Result |
        +--------+   +--------+   +--------+   +--------+

                  Reassembled Result (256 bits)
+----------------------------------------------------------------+
|                                                                |
|   +--------+   +--------+   +--------+   +--------+            |
|   | 64-bit |   | 64-bit |   | 64-bit |   | 64-bit |            |
|   | Result |   | Result |   | Result |   | Result |            |
|   +--------+   +--------+   +--------+   +--------+            |
|                                                                |
+----------------------------------------------------------------+
```

Proof of Probability of Collision

Under the assumption that a tested, audited Verifiably Random Function is used to generate a random seed, the probability of collision of Proof-of-Claim election algorithm is 2^{-256}, or the same as the SHA256 algorithm.

1. $SHA256_{P_{collision}} = 2^{-256}$ due to the secure collision resistance nature of the SHA256 algorithm, and therefore it's resistance to birthday attacks, and the inapplicability of the birthday paradox[16].
2. $64bit_{P_{collision}} = 2^{-64}$
3. Given that the XOR operation is deterministic, this means that the probability of collision for the XOR operation is the same as the probability of collision for the 64-bit segment[17].
4. Since the 256-bit hash is broken down into four 64-bit segments, and each segment is XORed against the seed, the probability of all four segments colliding is the produce of their individual collision probabilities, or
5. $P_{collision} = (2^{-64})^4 = 2^{-256}$

So long as an audited, tested, and provably collision resistant SHA256 hashing algorithm is used to produce the claim hash, given the deterministic nature of the XOR operation, and the collision probability of each of the four 64-bit segments the claim hash is broken down into, the probability of 2 nodes having the same result in a given election is equal to the probability of 2 nodes having the same claim hash, as that is the only way in which the results would collide.

Proof of Fair Election Distribution

To prove that the algorithm leads to a fair, random distribution of elected nodes, we show that given a verifiably random seed, the result of the XOR operation on each node's unique 256-bit claim hash is also uniformly distributed across the 256-bit space. If this is true, then each node has an equal probability of being elected ensuring fairness and randomness.
Properties of XOR:

If A is a random variable uniformly distributed over a bit space, and B is any bit string of the same length, then $A \oplus B$ (where $\oplus$ denotes the XOR operation) is also uniformly distributed over that bit space[18]. This property holds because XOR with a fixed value (in this case, B) simply flips specific bits in A but does not change the randomness or distribution of A.

Given:

- Each node has a unique 256-bit claim hash.
- The seed is verifiably random and uniformly distributed across the 64-bit space.

Application of XOR:

- When the 256-bit claim hash is broken down into 64-bit segments and each segment is XORed with the 64-bit seed, each segment's result is uniformly distributed across the 64-bit space (due to the property mentioned above).
- The reassembled 256-bit result, which is a combination of the four XORed segments, is also uniformly distributed across the 256-bit space.

Proof Conclusion:

Since the result of the algorithm is uniformly distributed across the 256-bit space, and each node has a unique 256-bit claim hash, each node has an equal probability of producing any specific 256-bit result. Therefore, each node has an equal probability of being elected, ensuring a fair and random distribution of elected nodes.

This proof assumes that the seed is verifiably random, and that the 256-bit claim hashes are not biased in a way that would favor certain outcomes when XORed with the seed. Given these assumptions, the algorithm provides a provably fair and random election process.

Under this scenario, the network configuration can choose the smallest resulting value, the largest resulting value, the value closest to the mean, the median value, or any other fixed ranking of the results and ensure provably fair and random elections. Due to the fact that every node in the network, eligible for election, will produce a valid result, the algorithm can be used to elect single nodes, can be used to elect quorums of nodes, and can be used for any other scenario in which a single or subset of node(s) needs to be elected. This makes the PoC algorithm versatile as an election algorithm in distributed and decentralized systems.

Further, since the XOR metric is deterministic, and the seed is known and verifiably random, and that claim hashes are known to all participants as a function of the eligible subset being stored in global state, every node in the network can conduct the operation for elections locally without any trusted party or single point of failure, and reach the same conclusion as to the "winner(s)" of a given election.

Speed at Scale

Speed of election, particularly a decentralized election with no trusted parties involved, is important if building a scalable, high throughput distributed system. One of the additional advantages of the Proof-of-Claim election algorithm is that it is fast at scale. There are 2 core properties that make PoC election operations fast:

1. **Non-mutation of Inputs** - Given that the inputs are not mutated in order to produce outputs, the ability to concurrently run the operation on many claim hashes at the same time without race conditions exists. This means that each individual node, when conducting the election operation, can execute it across a subset, or the entire set of eligible claims at the same time, speeding up the process of selecting the winner(s).

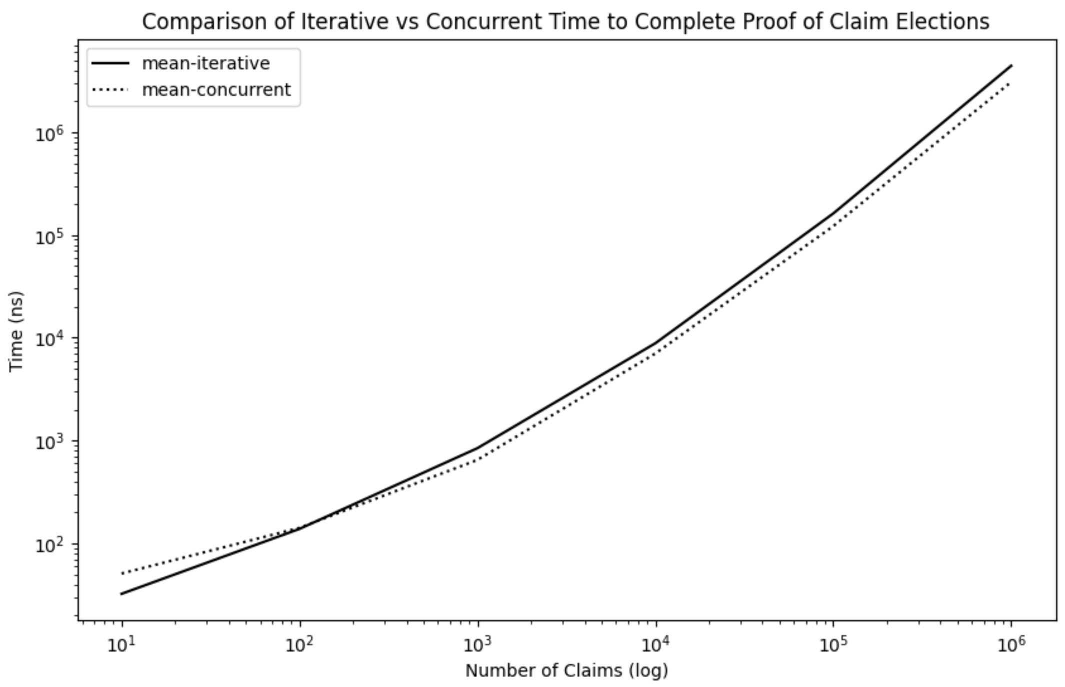
2. **Time-Complexity of XOR Operation** - The time complexity of the XOR operation in modern implementations is O(1) as the bits can all be processed in parallel[19][20]. In our scenario, there are 4 separate operations, so technically speaking it, in the Proof-of-Claim implementation, the total time complexity is O(n) where $n = 4$.

As a result of the number of elements always being 4, and the fact that chunks of eligible claims, if not all eligible claims, can be processed concurrently, it is possible for each individual node to run an election in a network of thousands to hundreds of thousands of nodes in microseconds to milliseconds.

If the implementation of the PoC election algorithm requires synchronization of nodes and agreement on the winner(s) before moving forward, adjusting for latency, it is possible to elect winner(s) in less than 1 second. If the implementation allows for asynchronous consensus around the winner(s) then, the network may be able to move forward even faster, allowing latent nodes to catch up down the line.

Benchmark

The following benchmark measures the time, in nanoseconds, to run a Proof of Claim election across different number of node claims, in a purely iterative model, and a model in which each claim is processed concurrently. As can be seen, the algorithm calculates results extremely fast, and is able to select winner(s) in nanoseconds to microseconds even as the network scales to over 100,000 nodes. Even at 1 million node claims, the algorithm completes the calculation of results in under 4.5 milliseconds in the iterative version, and in 3.07 milliseconds in the concurrent version.



This benchmark was run on an machine with a x86_64 AMD Ryzen Threadripper 2970WX 24-Core Processor and can be replicated by cloning the repo found at https://github.com/versatus/versatus and running:

```
cd path/to/repo/
cargo bench[2]
```

Conclusion

This paper introduced the Proof of Claim (PoC) algorithm, a novel approach to node election in distributed and decentralized systems. Upon comparative analysis with prevalent election algorithms in analogous contexts, the PoC algorithm emerges as a robust mechanism that ensures rapid, equitable, and collision-resistant node elections, adaptable to both synchronous and asynchronous configurations.

Central to the PoC algorithm's efficacy are several foundational components:

1. A verifiably random seed derived from fair inputs.
2. A unique claim hash serving as an identifier for eligible nodes.
3. The application of the XOR operation between the seed and the claim hash.

[2] This assumes you have Rust, Cargo and dependencies installed on your local machine. If not, follow this guide: https://doc.rust-lang.org/cargo/getting-started/installation.html to get your local Rust environment installed

These elements collectively guarantee elections that are not only provably fair and randomly distributed but also devoid of any central trusted entity's involvement. A salient feature of the PoC algorithm is its resistance to potential biases favoring capital-rich node operators. By design, the algorithm precludes the possibility of unduly influencing elections through the mere accumulation of hardware or monetary assets.

In summation, the Proof of Claim election algorithm stands out as a pioneering solution, offering provably fair, randomly distributed, and unbiased elections in the realm of distributed and decentralized systems. Its attributes underscore its potential as a preferred choice for systems necessitating impartial and efficient node elections.

[1] Madisetti & Panda, "A Dynamic Leader Election Algorithm for Decentralized Networks", 2021
[2] Santoro, "Design and analysis of distributed algorithms", 2007
[3] Byrenheid, Strife & Roos, "Attack resistant Leader Election in Social Overlay Networks by Leveraging Local Voting", 2020
[4] Zhang & Chow, "The leader election criterion for decentralized economic dispatch using incremental cost consensus algorithm", 2011
[5] Paznikov, Gurin & Kupriyanov, "Implementation in Actor Model of Leaderless Decentralized Atomic Broadcast", 2020
[6] Xie & Chen, "Elastic Consistent Hashing for Distributed Storage Systems", 2017
[7] Zhang, Wu & Want, "Overview of Blockchain Consensus Mechanism", 2020
[8] Alrowaily, et. al., "Modeling and Analysis of Proof-Based Strategies for Distributed Consensus in Blockchain-Based Peer-to-Peer Networks", 2023
[9] Supreet, et. al., "Performance Evaluation of Consensus Algorithms in Private Blockchain Networks", 2020
[11] Wang & Tan, "Block Proposer Election Method Based on Verifiable Random Function in Consensus Mechanism", 2020
[11] Galindo, et. al., "Fully Distributed Verifiable Random Functions and their Application to Decentralized Random Beacons", 2020
[12] Abraham, "Post-quantum verifiable random functions from ring signatures", 2018
[13] Trimoska, Ionica & Dequen, "Parity (XOR) Reasoning for the Index Calculus Attack", 2020
[14] Thorwart & Hanggi, "Decoherence and dissipation during a quantum XOR gate operation", 2001
[15] Wang, Xu & Jia, "SXVCS: An XOR-based Visual Cryptography Scheme without Noise via Linear Algebra", 2023
[16] Ahle, "On the Problem of p^{-1}_{1} in Locality-Sensitive Hashing", 2020
[17] Park & Manocha, "Efficient Probabilistic Collision Detection for Non-Gaussian Noise Distributions", 2019
[18] Gaoming, et. al, "Adaptive Analysis with HD Model on XOR Operation in Cipher Chips", 2012
[19] Dodmane, Aithal & Shetty, "Construction of vector space and its applications to facilitate bitwise XOR - Free operation to minimize the time complexity", 2022
[20] Dodmane, Aithal & Shetty, "Time Complexity Reduction for the Application of Stream Cipher System Based XOR Free Operation", 2019