Enhancing DAG-Based Consensus Protocols with Weighted Voting: A Performance Analysis of Narwhal and Tusk

Vian Robotin

Supervisor(s): Jérémie Decouchant, Rowdy Chotkan

[1]**EEMCS, Delft University of Technology, The Netherlands**

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 23, 2024

Name of the student: Vian Robotin
Final project course: CSE3000 Research Project
Thesis committee: Jérémie Decouchant, Rowdy Chotkan, Kaitai Liang

Abstract

This paper explores the integration of weighted voting mechanisms into DAG-based consensus protocols, such as Tusk [EuroSys'22], which promise high throughput and low latency. Weighted voting, pioneered by protocols like WHEAT [SRDS'15] and AWARE [TDSC'20], aims to optimize performance metrics such as latency and throughput by assigning weights to nodes based on their latency with other nodes. We employ real-world latency data from CloudPing to evaluate the impact of weighted voting on Tusk, comparing them against its unweighted counterpart. Our results demonstrate significant performance improvements, with the weighted voting mechanism achieving up to 37% lower consensus latency compared to unweighted approaches.

1 Introduction

Consensus algorithms lie at the heart of many distributed systems, such as blockchains and cryptocurrency systems. They play a critical role in ensuring data consistency across multiple, independent nodes. This is accomplished by allowing the nodes to agree on a single value or a sequence of actions, even in the presence of failures or network issues. In blockchains, it allows participants to reach an agreement on a shared ledger of transactions and prevent double-spending.

Blockchain consensus algorithms fall into two categories: permissionless, where participation is open to all, and permissioned, where a predetermined set of nodes governs the network. The latter system is sometimes preferred as it demonstrates superior throughput, lower latency, and immediate finality. Multiple strategies have been explored to improve the performance of permissioned systems further, such as minimizing the system size [12] or leader rotation schemes [3].

The PBFT [7] protocol is a Byzantine Fault Tolerant consensus algorithm designed to withstand f Byzantine nodes with arbitrary behaviour in a system with $n = 3f + 1$ total nodes. This enhances the resilience of distributed systems, however, it comes with the cost of high message complexity and heavy view change procedure. In the optimistic case, when the leader is honest, performance is stable, but it degrades significantly when the leader is not trustworthy.

These disadvantages have been supporting the research for better alternatives. Notably, there has been notable progress in the development of DAG-based consensus algorithms [11], showcasing superior time complexity and throughput compared to traditional consensus approaches. Among these advancements are Narwhal and Tusk [8]. Narwhal is a mempool protocol that obtains high throughput even amidst faults and asynchrony by using a reliable broadcast protocol, and Tusk is a zero-message overhead consensus algorithm designed to run on top of Narwhal, ensuring security under full asynchrony. Further refining this innovation, Bullshark [15] represents a significant leap forward from Tusk. It combines the strengths of both partially synchronous and asynchronous protocols, achieving a remarkable balance. This allows Bullshark to leverage the efficiency of synchronous communication when possible, while maintaining functionality and fault tolerance even in asynchronous environments.

In parallel, the idea of incorporating weighted voting in consensus algorithms has gained traction. This approach led to improved performance in state machine replication across geographically dispersed networks, exemplified by WHEAT [14]. Subsequently, researchers combined the weighted voting mechanism from WHEAT with BFT-SMaRt [6], a recent implementation of PBFT [7], to develop AWARE [5]. This led to substantial latency and throughput improvement over PBFT.

While the benefits of weighted voting have been primarily studied in algorithms like PBFT, there is potential for broader application. This research aims to extend the investigation of weighted voting to DAG-based consensus protocols such as Narwhal and Tusk.

To this end, this paper aims to:

- Adapt AWARE's algorithms to find the optimal weight for a Narwhal node given the network latencies.
- Incorporate weighted voting in Narwhal using WHEAT's weight assignment scheme.
- Evaluate the impact of weighted voting on the performance of Narwhal and Tusk.

The rest of this paper is organized as follows: Section 2 discusses previous research on weighted voting in consensus algorithms and other methods to improve consensus latency. Section 3 provides an overview of the protocols and mechanisms that are the most relevant to this work, including WHEAT, Narwhal, Tusk, and AWARE. Section 4 outlines the integration of weighted voting into Narwhal and modifications to Narwhal and Tusk for weighted voting. Section 5 presents our experimental setup, including latency data collection, and our experiment results, comparing the performance of our weighted and the unweighted versions of Tusk. Section 6 addresses ethical considerations, including reproducibility and objectivity, ensuring adherence to research integrity standards. Section 7 discusses the limitations of this research and suggests directions for future work. Finally, Section 8 concludes the paper with an overview of the impact of studying weighted voting on Narwhal and Tusk.

2 Related Work

The idea of weighted voting in consensus algorithms has been studied in the past. Swiper and Dora [17] discusses converting protocols designed for a nominal setting (where all parties are equal) to a weighted setting (where parties have weights). The authors propose three weight reduction problems to achieve this conversion. For Narwhal, the Dora algorithm is applicable, which is the solution to the proposed Weight Qualification problem. However, this research considers a different approach, namely using WHEAT's weighting scheme.

Other methods to improve the performance of consensus algorithms have been explored. Damysus [9] and OneShot [10] introduce trusted components and leverage them to tolerate a minority of faulty nodes and use a reduced

number of communication rounds. ThreatAdapt [13] established the conditions for a BFT-SMR protocol to safely reconfigure itself based on a synchronous threat detector, allowing for dynamic optimization by reducing the number of active replicas and achieving reconfiguration 30% faster than previous methods. Mir [16] presents a novel protocol mechanism that allows a set of leaders to propose request batches independently and in parallel, while rotating the assignment of a partitioned request hash space to leaders.

All works presented above concern PBFT or HotStuff-like protocols, which are vastly different from the approach introduced by Narwhal. To the best of our knowledge, this constitutes the first paper to improve on the Narwhal protocol.

3 Background

WHEAT

WHEAT [14] is a WAN-optimized State Machine Replication (SMR) protocol built on top of BFT-SMART. The protocol first employs WAN optimizations, such as reducing communication steps and the number of replies clients wait for, and decreasing the ratio between quorum size and total replicas. It favors tentative executions over fast executions to handle network unpredictability effectively.

The weight assignment scheme introduces the concept of weighted replication to SMR protocols. The goal is to primarily rely on the fastest replicas in the system while ensuring safety and liveness. To this end, the authors introduce Δ extra replicas in the system and prove that their weight assignment scheme progresses with fewer replicas while still upholding the requirements of quorum formation. More specifically, in CFT mode, where the total number of nodes is $n = 2f+1+\Delta$ total nodes, the scheme assigns a weight of $V_{max} = 1 + \frac{\Delta}{f}$ to the fastest f replicas in the system, while all other replicas are assigned a weight of 1. Additionaly, the minimum number of votes becomes $f + \Delta + 1$. Similarly, in BFT mode, where $n = 3f + 1 + \Delta$, the weight V_{max} is assigned to the $2f$ fastest replicas in the system and a quorum needs a minimum of $2(f + \Delta) + 1$ weights to form.

This weight distribution is shown to improve the system's latency and fault tolerance by allowing more flexibility in selecting optimal quorums based on the performance of individual replicas. Additionally, adjusting the replica group and reassigning votes in response to detected faults enhances fault tolerance, enabling the system to withstand up to $f+\Delta$ faulty replicas in certain scenarios.

Narwhal

The Narwhal [8] protocol presents a novel approach to building a mempool and achieving high-throughput consensus. It combines concepts from reliable broadcast, reliable storage, and Byzantine fault-tolerant Threshold clocks. Narwhal's core innovation lies in its structured Mempool, which leverages a DAG-based approach to ensure reliable broadcast of transaction blocks in causal order. An illustration of Narwhal operation, forming a block DAG, can be seen in Figure 1.

The protocol operates by maintaining local rounds for validators, who continuously receive transactions from clients

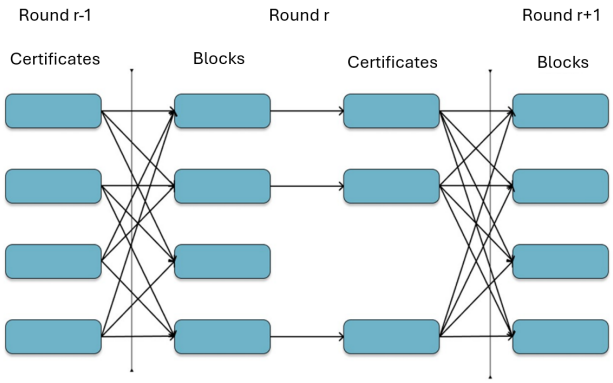

Figure 1: Three rounds of Narwhal.

and accumulate them into transaction lists. Validators also accumulate certificates of availability for blocks from previous rounds. Once a validator accumulates $2f+1$ certificates from the previous round, it advances the local round and creates a new block, broadcasting it to other validators. Validators reliably broadcast blocks they create and acknowledge valid blocks by signing their digests, round numbers, and creators' identities. A block is considered valid if it meets certain criteria, including containing valid signatures, being at the correct local round, and having the necessary certificates from the previous round. After a validator accumulates $2f + 1$ signatures on its proposed block, it generates a certificate of availability for that block, which can be utilized in the next round. The system initializes with validators creating and certifying empty blocks for round zero.

The security of the protocol is based on the presence of a sufficient number of honest validators signing certificates of availability, ensuring block availability and integrity. Additionally, quorum intersection ensures the integrity and availability of each block, and the protocol maintains causality by certifying and making available all blocks in the causal history.

Tusk

Tusk [8] is an asynchronous consensus protocol that runs on top of Narwhal and remains live even under conditions of asynchrony or DDoS attacks.

Tusk validators manage a Narwhal mempool and include in their blocks information necessary for generating a perfect global random coin. This random coin is critical for the consensus process and is derived using an adaptively secure threshold signature scheme. Notably, this key setup is robust enough to be performed under full asynchrony, ensuring the system's resilience and security even in the face of network delays or adversarial conditions.

A core innovation in Tusk is its approach to block ordering, leveraging the causally ordered Directed Acyclic Graph (DAG) constructed by Narwhal. Unlike traditional consensus algorithms that may require extensive communication overhead, Tusk achieves total block ordering with zero additional communication. Validators independently interpret their local views of the DAG and utilize shared randomness to deter-

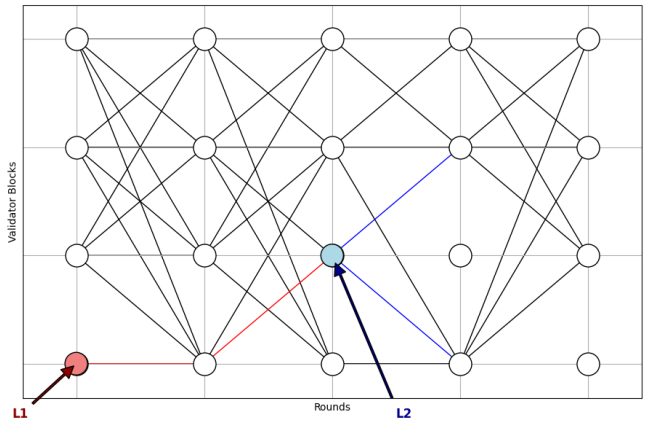

Figure 2: Example of commit rule in Tusk. Every odd round has a coin value that selects a leader of round $r-2$. If the leader has less than $f+1$ support (red) they are ignored, otherwise (blue) the algorithm searches the causal DAG to commit all preceding leaders (including red) and totally orders the rest of the DAG afterward.

mine the global order of blocks. This process involves dividing the DAG into waves, each comprising three consecutive rounds. In the first round, validators propose their blocks; in the second round, they vote on these proposals by including them in their blocks; and in the third round, validators produce randomness to retrospectively elect a random leader's block.

The mechanism for committing and ordering blocks in Tusk ensures that all honest validators eventually reach consensus on the same block leaders, despite potential discrepancies in their local views of the DAG. After a validator commits a leader block in a wave, it recursively examines previous waves to ensure all intermediary blocks are correctly ordered. This recursive verification process involves checking paths between candidate leaders and previously committed blocks, thereby ensuring a consistent and agreed-upon order of blocks. The safety of this mechanism is analogous to that of DAG-Rider [11], providing a robust guarantee that all honest validators will converge on the same set of block leaders over time.

Figure 2 shows two waves of Tusk: the first wave covers rounds 1-3, and the second wave covers rounds 3-5. There are 4 validators, and $f=1$. Leaders for waves 1 and 2 are chosen at rounds 3 and 5, named $L1$ and $L2$ respectively. In round 2, there are not enough blocks (fewer than $f+1$) that vote for $L1$, so $L1$ is not committed when round 3 is evaluated. However, in round 4, $f+1=2$ blocks vote for $L2$, leading to $L2$'s eventual commitment. Since there is a path between $L2$ and $L1$, $L1$ is ordered before $L2$. This means that the sub-DAG dependent on $L1$ is ordered first using a deterministic rule, and the same rule is then applied to the sub-DAG dependent on $L2$.

AWARE

AWARE [5] presents a comprehensive framework for self-optimization in Byzantine fault-tolerant (BFT) consensus protocols within distributed systems. The system addresses the critical requirement for robust self-monitoring capabilities, ensuring accurate measurements to facilitate optimization processes effectively.

Self-optimization in AWARE is achieved through predictive modeling of optimal configurations, leveraging simulated protocol runs to minimize consensus latency. This is based on the following two algorithms.

The first algorithm, *formQuorum*, determines the time at which each replica can progress to the next protocol stage by forming a weighted quorum of votes. Initially, the algorithm calculates the time when each replica receives WRITE messages from all other replicas. Subsequently, it computes the time required for each replica to accumulate sufficient voting weights to proceed to the next phase, such as moving from WRITE to ACCEPT. This is achieved by leveraging a priority queue that sorts incoming messages by their arrival times. The time at which the last necessary message arrives, completing the quorum, dictates when the replica can advance to the subsequent protocol stage.

The second, *predictLatency*, calculates the consensus latency by iteratively simulating multiple consensus rounds. It determines when replicas receive the leader's proposal and complete the WRITE and ACCEPT stages using *formQuorum* as a building block. By averaging the leader's consensus latency over multiple rounds, this method provides an accurate prediction of the system's consensus performance under a specific weight configuration.

To find the optimal weight configuration that minimizes predicted consensus latency, AWARE employs *Simulated Annealing* [2], which is a search algorithm that explores the configuration space and gradually converges towards configurations with lower predicted latency.

Finally, once the optimal weight configuration is identified, all replicas are informed, and a reconfiguration process is initiated to adjust voting weights accordingly. This allows AWARE to dynamically adapt to network conditions and optimize performance.

4 Weighted voting in Narwhal and Tusk

This section presents the methodology used to perform the experiments. Section 4.1 showcases how to incorporate weighted voting in Narhwal. Section 4.2 describes the changes made to the original Narwhal and Tusk implementation to allow weighted voting.

4.1 Weighted Narwhal

A step-by-step analysis of the Narwhal protocol reveals that each validator forms 2 quorums per round: first to receive signatures on its own block from other validators and create the certificate of availability and second to receive the certificates of other validators. In each case, the threshold is $2f+1$ out of the $3f+1$ validators.

Given the required information on the network latencies between validators and the weight assignment, we created a latency prediction model for Narwhal. This model computes the quorum formation times for each node, resulting in the completion time of a round for each validator.

The model leverages the voting power of a node during the quorum formation process. It considers messages that arrive

faster and sums the weights of the senders until the threshold is reached. This idea comes from AWARE's *formQuorum* algorithm [5].

In line with WHEAT's weight assignment, the Narwhal consensus protocol was adapted as follows:

- Introduce Δ extra validators in the system
- Each validator is assigned weight $V_{min} = 1$ or $V_{max} = 1 + \frac{\Delta}{f}$
- In total, $2f$ out of n validators are assigned weight $V_{max} = 1 + \frac{\Delta}{f}$
- Validators form quorums based on the weights of the participants
- The threshold for quorum formation becomes $2(f + \Delta) + 1$ for the weighted protocol and $\lceil \frac{N+f+1}{2} \rceil$ for the unweighted one.

The model returns the average completion time which is used as a performance metric. Alongside this, it returns the objective value that it tries to minimize. We implemented 4 objective functions:

- Average consensus latency
- Maximum consensus latency of a validator
- Standard deviation of the consensus latency
- Average consensus latency, using standard deviation as a tiebraker

Throughout the remainder of the paper, we will refer to these objective functions as Mean, Max, Stddev and Mean+Stddev respectively. Mean+Stddev behaves the same as the Mean objective function, however, it accepts a configuration with a slightly higher avearage if the standard deviation is lower than the previous best configuration.

The model supports two types of searching for the optimal weights. *Exhaustive search* iterates through every possible weight assignment and evaluates it. This method is simple to implement and is suitable for systems with few nodes. *Simulated Annealing* [2] starts with an initial configuration and changes it slightly, accepting the change with a certain probability. The probability of accepting a worse solution is initially high and gradually decreases as the number of iterations increases. This allows the algorithm to escape from local minima and converge to a global minimum. For systems with a high number of nodes, employing *Simulated Annealing* is advantageous because even if the algorithm is halted prematurely, it will still produce a viable solution.

4.2 Weighted Narwhal and Tusk

For evaluating the combination of Narwhal and Tusk, we opted to modify the original code presented in [8]. We first introduced artificial latencies in order to use real-world latencies between nodes. We achieved this using the **sleep** functionality, which interrupts a node for a specified duration. More specifically, each node, before sending a message to another node, would wait for a duration equal to the latency specified in the matrix.

Secondly, we appropriately adjusted the threshold for quorum formation as previously mentioned: $2(f + \Delta) + 1$ for the weighted protocol and $\lceil \frac{N+f+1}{2} \rceil$ for the unweighted one.

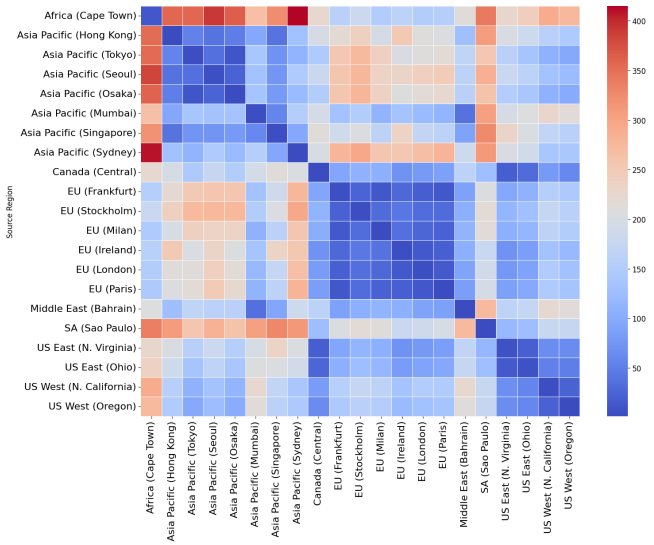

Figure 3: Heat map of latency measures gathered from CloudPing.

Lastly, using the simulation for Narwhal, we were able to inspect the weight assignment found for the chosen objective function. Thus, we were able to assign the weights in the original code so that we can evaluate Narwhal and Tusk according to the same 4 objective functions.

5 Experimental Setup and Results

Latency gathering. To ensure our results reflected real-world conditions, we utilized latency data gathered from CloudPing [1]. This website reports on the latencies between servers located across the globe within the AWS infrastructure. Figure 3 showcases the latency matrix used in our experiments as a heat map. This method is superior to a random matrix, as it incorporates real-world connections between regions. It is important to acknowledge that CloudPing data represents a snapshot in time, and latency can fluctuate due to network congestion and server load.

Narwhal. The experiment was conducted using a simulation of the described variant of Narwhal, written in Python. The results were gathered by running the weighted protocol for two rounds and comparing it with the performance of the unweighted variant. In both variants of the algorithm, the latencies presented in Figure 3 were used. Note that when running the simulation with N nodes, the latency matrix consists of the first N servers showcased in Figure 3. For example, if $n = 5$, the simulation uses the 5x5 matrix consisting of Africa (Cape Town), Asia Pacific (Hong Kong), Asia Pacific (Tokyo), Asia Pacific (Seoul), Asia Pacific (Osaka).

Our Narwhal latency prediction model is released on Gitlab[1]. It provides a *Validator* class, which encapsulates the logic of the protocol. Additionally, it features *Block* and *Certificate* classes, representing the blocks that validators broadcast and sign, as well as the certificates used to demonstrate the availability of the blocks.

[1] https://gitlab.com/vianrobotin/cse3000-weighted-voting

Table 1: Narwhal and Tusk evaluation metrics with different weight assignments.

	Unweighted	Mean	Max	Stddev
Consensus TPS	36,296 tx/s	27,330 tx/s	29,394 tx/s	21,572 tx/s
Consensus BPS	18,583,673 B/s	13,993,181 B/s	15,049,521 B/s	11,044,740 B/s
Consensus latency	2,607 ms	1,673 ms	2,008 ms	2,245 ms
End-to-end TPS	35,977 tx/s	27,086 tx/s	29,151 tx/s	20,675 tx/s
End-to-end BPS	18,420,032 B/s	13,868,152 B/s	14,925,111 B/s	10,585,449 B/s
End-to-end latency	2,982 ms	1,961 ms	2,643 ms	2,788 ms

Figure 4 showcases the performance of the weighted protocol in accordance with several objective functions. The results were produced by running the simulation with $n = 5$, $f = 1$ and $\Delta = 0$ for unweighted or $\Delta = 1$ for the weighted algorithm. For each objective function, the first bar highlights the latency decrease achieved by the weighted protocol in comparison with the unweighted one. The second presents the mean consensus latency in the ideal case - optimal weights that minimize the objective function and no replica is faulty. Lastly, the final bar provides insight into the effect that a faulty replica has on the system. The replica to be made faulty was chosen so that it would cause a significant increase in the consensus latency. Analysing the latencies in Figure 3 reveals that the 5th node, Asia Pacific (Osaka), is the best connected node in the first 5 nodes. Therefore, this node is always the faulty node in our experiments.

As seen in Figure 4, finding weights that minimize the average consensus latency of the validators achieves the largest latency decrease. However, depending on the needs of the system, the Mean+Stddev objective function may be more helpful, as validators will complete the rounds at closer time intervals. Figure 4 showcases this behaviour and achieves comparable performance with plain average minimization.

With only 5 nodes, the number of possible configurations is relatively small. Figures 5 and 6 explore the behaviour for 8 and 11 nodes, respectively. Across the different parameter settings, the Stddev objective function yields the lowest latency decrease, while Mean and Max have similar performances.

Narwhal and Tusk. Using the steps described in Section 4.2, the performance of Narwhal and Tusk with weighted voting was evaluated. The experiment was conducted with parameters $n = 5$ and $f = 1$, using $\Delta = 0$ for the unweighted algorithm and $\Delta = 1$ for the weighted algorithm.

The metrics used in Table 1 are the same ones used to evaluate Narwhal and Tusk in the original paper. The 'Consensus TPS' and 'Consensus latency' respectively report the average throughput and latency without considering the client. The consensus latency thus refers to the time elapsed between the block's creation and its commit. In contrast, 'End-to-end TPS' and 'End-to-end latency' report the performance of the whole system, starting from when the client submits the transaction.

Table 1 highlights the result of the evaluation. The unweighted variant dominates both consensus and end-to-end transactions per second. However, the weighted algorithm with the Mean objective achieves the lowest consensus and

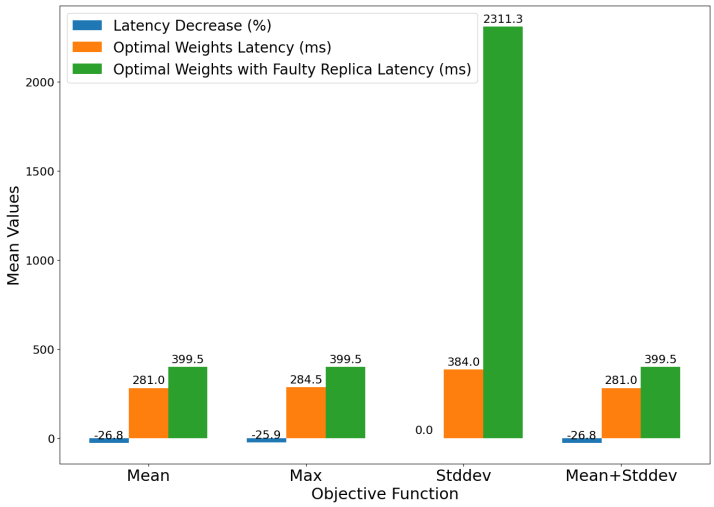

Figure 4: Narwhal performance metrics for different objective functions ($n = 5, f = 1$)

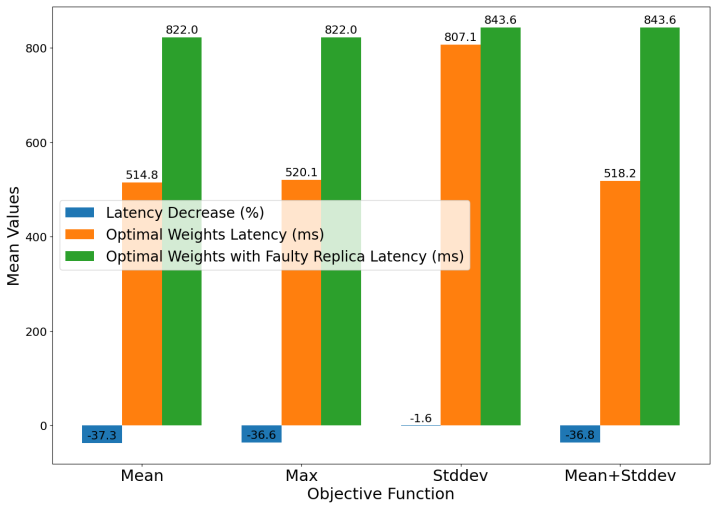

Figure 5: Narwhal performance metrics for different objective functions ($n = 8, f = 2$)

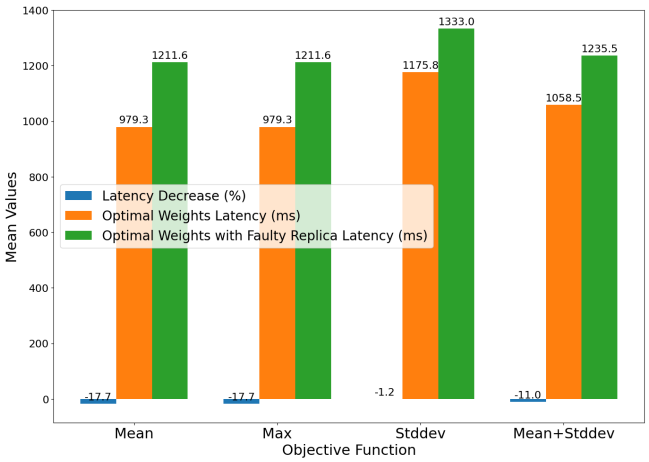

Figure 6: Narwhal performance metrics for different objective functions ($n = 11, f = 3$)

end-to-end latency, which is the focus of this research. Moreover, the other objective functions have better performance than the unweighted algorithm, solidifying their place as reasonable options. The Mean+Stddev objective function is not present here because it utilises the same weight assignment as the Mean objective function, resulting in equivalent performance.

6 Responsible Research

This section reflects upon the ethical aspects of the research, adhering to the Netherlands Code of Conduct for Research Integrity [4]. The discussion will cover the reproducibility of the methods, the objectivity of the results, and the ethical conduct throughout the research process.

Reproducibility. To ensure the reliability and reproducibility of the results, real-world latency data gathered from CloudPing was employed. This data represents latencies between AWS servers across the globe, providing a realistic basis for the experiments. By using real-world data instead of synthetic or randomly generated values, the credibility of the findings is enhanced. The latency matrix used in the experiments is publicly available, allowing other researchers to replicate the study under similar conditions. Furthermore, a GitLab repository includes the complete codebase and detailed documentation, enabling others to reproduce the experiments by running the provided scripts. This transparency ensures that the research adheres to the FAIR principles (Findable, Accessible, Interoperable, and Reusable).

Objectivity and Ethical Conduct. The interpretation of results was conducted with strict adherence to objectivity, avoiding any manipulation of data to favor desired outcomes. A thorough analysis of the results was provided, highlighting both strengths and weaknesses of the weighted and unweighted algorithms without bias. The design choices and modifications to the Narwhal and Tusk protocols were meticulously documented, ensuring that all steps are transparent and justified. Ethical integrity was maintained throughout the research process by conducting a comprehensive literature review, ensuring that the methodologies were grounded in established research, and by critically evaluating the results. This study aims to contribute to the academic community with accurate, reliable, and ethically sound research, offering a solid foundation for future studies in the field of blockchain algorithms.

7 Discussion

The results presented in Section 5 illustrate the improved performance of Narwhal and Tusk, achieved using weighted voting. By developing a model to optimise Narwhal and Tusk according to different objective functions, this research reinforces the benefits of weighted voting for DAG-based consensus algorithms.

The current research is made on latencies gathered from CloudPing. While it provides reliable insights into the effect of weighted voting on Narwhal and Tusk, a more diverse set of latencies could reveal situations in which weighted Narwhal performs better or worse than expected. Furthermore, both approaches to finding the optimal weights, *Exhaustive Search* and *Simulated Annealing*, are computationally expensive, rendering simulations with $n > 20$ infeasible. For *Simulated Annealing*, the parameters of simulated annealing can be changed to return a result faster, but it might be suboptimal.

We propose two recommendations for future work on the research presented in this paper. First, future work should evaluate the performance of other consensus algorithms compatible with Narwhal, such as HotStuff [18] and Bullshark [15], when used with the weighted Narwhal protocol. Second, the introduction of artificial latencies in Narwhal and Tusk is not optimal, as using **sleep** could result in a node dismissing a message that it would otherwise receive and acknowledge. A better alternative would be to directly add delays to messages in the communication channel.

8 Conclusion

In this study, the incorporation of weighted voting in the Narwhal and Tusk consensus protocols was explored, aiming to enhance their performance metrics. By leveraging WHEAT's weight assignment scheme, the use of weighted voting was extended beyond first-generation algorithms like PBFT to DAG-based consensus mechanisms. Modifications to the Narwhal and Tusk implementations, along with the incorporation of real-world latency data, allowed for simulation and evaluation of the impact of different weight configurations on system performance.

Findings demonstrate that weighted voting can significantly reduce consensus latency in both Narwhal and Tusk protocols. Specifically, the Mean objective function achieved the lowest consensus and end-to-end latency, showcasing its potential for optimizing system performance. Other objective functions, such as Max and Stddev, also contributed to performance improvements, highlighting the flexibility of weighted voting in addressing various optimization goals.

While the unweighted algorithm exhibited superior throughput, the weighted approach excelled in minimizing latency, which is crucial for time-sensitive applications. The results affirm the potential of integrating weighted voting into

advanced consensus protocols, providing a pathway for further research and development in this area.

References

[1] Cloudping. https://www.cloudping.co/grid/latency/timeframe/1D. Accessed on 04-06-2024.

[2] Emile Aarts, Jan Korst, and Wil Michiels. *Simulated Annealing*, pages 187–210. Springer US, Boston, MA, 2005.

[3] Yair Amir, Brian Coan, Jonathan Kirsch, and John Lane. Prime: Byzantine replication under attack. *IEEE Transactions on Dependable and Secure Computing*, 8(4):564–577, 2011.

[4] Association of Universities in the Netherlands (VSNU). Netherlands Code of Conduct for Research Integrity, 2018.

[5] Christian Berger, Hans P Reiser, João Sousa, and Alysson Bessani. Aware: Adaptive wide-area replication for fast and resilient byzantine consensus. *IEEE Transactions on Dependable and Secure Computing*, 19(3):1605–1620, 2020.

[6] Alysson Bessani, João Sousa, and Eduardo E. P. Alchieri. State machine replication for the masses with bft-smart. In *Proceedings of the 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, DSN '14, page 355–362, USA, 2014. IEEE Computer Society.

[7] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *3rd Symposium on Operating Systems Design and Implementation (OSDI 99)*, New Orleans, LA, February 1999. USENIX Association.

[8] George Danezis, Eleftherios Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and tusk: a dag-based mempool and efficient bft consensus. *Proceedings of the Seventeenth European Conference on Computer Systems*, 2021.

[9] Jérémie Decouchant, David Kozhaya, Vincent Rahli, and Jiangshan Yu. Damysus: streamlined bft consensus leveraging trusted components. In *Proceedings of the Seventeenth European Conference on Computer Systems*, EuroSys '22, page 1–16, New York, NY, USA, 2022. Association for Computing Machinery.

[10] Jérémie Decouchant, David Kozhaya, Vincent Rahli, and Jiangshan Yu. Oneshot: View-adapting streamlined bft protocols with trusted execution environments. In *IPDPS 2024*. 2024.

[11] Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. All you need is dag. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, PODC'21, page 165–175, New York, NY, USA, 2021. Association for Computing Machinery.

[12] Douglas Simões Silva, Rafal Graczyk, Jérémie Decouchant, Marcus Völp, and Paulo Esteves-Verissimo. Threat adaptive byzantine fault tolerant state-machine replication. In *2021 40th International Symposium on Reliable Distributed Systems (SRDS)*, pages 78–87, 2021.

[13] Douglas Simoes Silva, Rafal Graczyk, Jérémie Decouchant, Marcus Volp, and Paulo Esteves-Verissimo. Threat adaptive byzantine fault tolerant state-machine replication. 2021.

[14] João Sousa and Alysson Bessani. Separating the wheat from the chaff: An empirical design for geo-replicated state machines. In *2015 IEEE 34th Symposium on Reliable Distributed Systems (SRDS)*, pages 146–155, 2015.

[15] Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. Bullshark: Dag bft protocols made practical. page 2705 – 2718, 2022. Cited by: 31.

[16] Chrysoula Stathakopoulou, Tudor David, and Marko Vukolic. Mir-bft: High-throughput bft for blockchains. *arXiv preprint arXiv:1906.05552*, 92, 2019.

[17] Andrei Tonkikh and Luciano Freitas. Swiper: a new paradigm for efficient weighted distributed protocols. Cryptology ePrint Archive, Paper 2023/1164, 2023. https://eprint.iacr.org/2023/1164.

[18] Maofan Yin, Dahlia Malkhi, Michael Reiter, Guy Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. pages 347–356, 07 2019.