Farmer-Harvester: LLMQ Driven Scalable BFT Consensus for Permissionless P2P Networks

By: Andrew Nicholas Smith & Vinay Sawant
Title: [Andrew Nicholas Smith <Founder>, Vinay Sawant <Senior Systems Engineer & Director of Protocol Research>]
Organization: Versatus Labs
Email: [Andrew Nicholas Smith <as@versatus.io>, Vinay Sawant <vs@versatus.io>]
Organization Email: info@versatus.io
Website: versatus.io

Abstract

Scaling a blockchain network to greater than 1 million transactions per second, without sacrificing security or decentralization, previously referred to as “The Blockchain Trilemma”, is a difficult, though not impossible task. In order to scale throughput of transactions, while maintaining the highest degree of security, fault tolerance, and the lowest time to finality possible, a new approach to system design and architecture is necessary. In existing blockchain networks, nodes propose blocks of transactions from the mempool, which other nodes then have to iterate through and verify, one by one, before updating their local copy of the global state of the network. This process is time consuming, and computationally expensive. It also makes it difficult to scale transaction throughput in a secure fashion.

We propose a new design, in which transactions are processed and validated concurrently locally, without the need for locks, and parallel across the network by Long Lived Masternode Quorums (LLMQ) whose participants are elected randomly, at fixed intervals, and have dynamic staking requirements derived from their reputation to enhance security. Blocks are then built from these pre-processed/validated transactions, and certified by an LLMQ with a separate role in the network. Under this model, nodes can scale local throughput, the network can scale network throughput, while maintaining the highest degree of security possible, and a time to finality that competes with the fastest modern networks available today. This model also ensures that the network can scale with additional staking nodes, while also scaling as hardware improves.

Introduction

Achieving scalable consensus in a peer-to-peer network, especially when faced with a significant number of potentially malicious peers, remains a complex challenge[1]. Despite numerous efforts in this domain, a truly successful solution has remained elusive[2]. In this paper, we introduce a novel model designed to achieve decentralized Byzantine fault-tolerant consensus at scale[3]. This model synergistically integrates vertical scalability mechanisms with a horizontally scaling approach, setting new benchmarks in throughput. Our approach marries the efficiency of left-right wrapped data structures with an innovative, optimistic strategy for transaction validation. Furthermore, we incorporate a conflict-resolving, converging directed acyclic graph within a network architecture that distinctly separates long-lived masternode quorums from permissionless, stakeless miner nodes[4].

The protocol is structured around specialized subsets: transaction execution/processing quorums, block proposal & certification quorums, and conflict resolution & block consolidation nodes. Each subset plays a pivotal role in maintaining the network's security, resistance to censorship, and consistency, even while processing an impressive range of hundreds of thousands to millions of transactions per second[5].

The transaction's journey begins with its allocation to an entry quorum, leveraging Maglev's Hash Ring to ensure consistent allocation based on the signing account. Depending on various factors like data locality and the quorum's current workload, transactions might be directed to a processing quorum[6].

Within this framework, pending transactions are housed in a left-right wrapped mempool, optimizing internal scheduling based on the validator processor/core's workload. Transactions dependent on others remain in the mempool until their preceding transactions are validated. Post-validation, nodes cast their votes to the active Harvester quorum. Upon achieving a consensus (a minimum of 60% agreement within the responsible quorum), the transaction is integrated into a proposal block, which is subsequently appended to the DAG.

Miners play a crucial role in consolidating proposal blocks into a unified reference point termed the "Convergence Block," marking the round's conclusion[4]. This block, once validated by a minimum of 60% of the active harvesters, is certified and becomes an immutable part of the chain's history. The certification of a convergence block also signals a network-wide state transition, reflecting the transactions within that block.

Our optimistic approach to transaction processing and validation means that nodes outside a specific farmer quorum aren't burdened with validating every transaction in a block. Instead, the responsibility of transaction validation is delegated to Farmer Quorums comprised of staking nodes, with the Harvester Quorum acting as a secondary validation layer. This streamlined

process significantly reduces the consensus workload across a decentralized network, ensuring consistent and highly available network state data[3].

Masternode Eligibility

For the optimistic transaction processing approach delineated in this paper to be effective, a rigorous eligibility protocol is paramount. We advocate for masternodes to be qualified based on a staking protocol. Historically, staking protocols have been instrumental in aligning economic incentives, thereby fortifying peer-to-peer networks with robust security measures. While there exists a plethora of proven staking protocols within Proof of Stake networks, the detailed selection criteria for a specific staking protocol fall outside the purview of this discussion[7][8]. Equally critical is the establishment of a protocol for the fair, randomized, and unpredictable election or assignment of nodes to quorums[9]. Such a protocol is essential to preclude malicious entities from exploiting the system, potentially compromising the network's integrity. If malevolent actors could anticipate quorum memberships for specific durations, they might strategically position themselves to dominate the quorums, jeopardizing the entire network[10].

By instituting a decentralized, randomized election mechanism, the resource commitment required from malicious actors to launch a successful attack becomes prohibitively high, rendering such endeavors economically unfeasible[7]. While there are myriad methodologies to orchestrate fair, randomized elections of eligible nodes, the specifics of such methods are beyond this paper's scope. It is imperative for implementers adhering to this paper's guidelines to judiciously select both a staking protocol for eligibility and an election protocol.

The frequency of election intervals is another pivotal aspect when constituting long-lived masternode quorums. Overly frequent elections might engender consensus disarray, whereas rare elections could expose the network to potential threats and amplify attack avenues. The optimal balance for election intervals, while crucial, is not explored in this paper but remains a salient security consideration for any protocol developed in alignment with this paper's specifications.

When the aforementioned considerations are meticulously addressed, the consensus mechanism delineated in this paper surpasses the Byzantine Fault Tolerance of conventional Proof of Work and Proof of Stake networks, especially when the network encompasses two or more Farmer Quorums. The extent of the consensus mechanism's BFT is intrinsically linked to the number of operational Farmer Quorums[11].

Maglevs Hash Ring: A Consistent Account-Based Entrypoint for Transactions

The consensus mechanism delineated in this paper commences with a pivotal step: ensuring consistent initial allocation of transactions based on their origin[12]. Specifically, transactions emanating from the same signer must invariably be allocated to an identical initial quorum. However, this does not imply that the designated quorum is solely responsible for the transaction's validation or execution.
Given the optimistic nature of our mechanism—where validation is delegated to a subset of validators constituting an LLMQ and subsequent validation stages are streamlined—it becomes imperative to maintain transaction sequencing[13]. Especially for transactions contingent on the validity of their predecessors, it's crucial to validate them only post the validation of their dependencies. By designating a consistent quorum as the entry point for all transactions from a particular signer, we can efficiently sequence interdependent transactions with minimal interquorum communication[14].

Within the entrypoint quorum, nodes construct a dependency graph, which is subsequently allocated to the Decentralized Task Scheduler[15]. Each vertex in this graph encapsulates the status of its corresponding dependency, ensuring streamlined processing by eliminating redundant processing of the same dependency. This meticulous approach guarantees that transactions (and their respective dependencies) are processed sequentially.

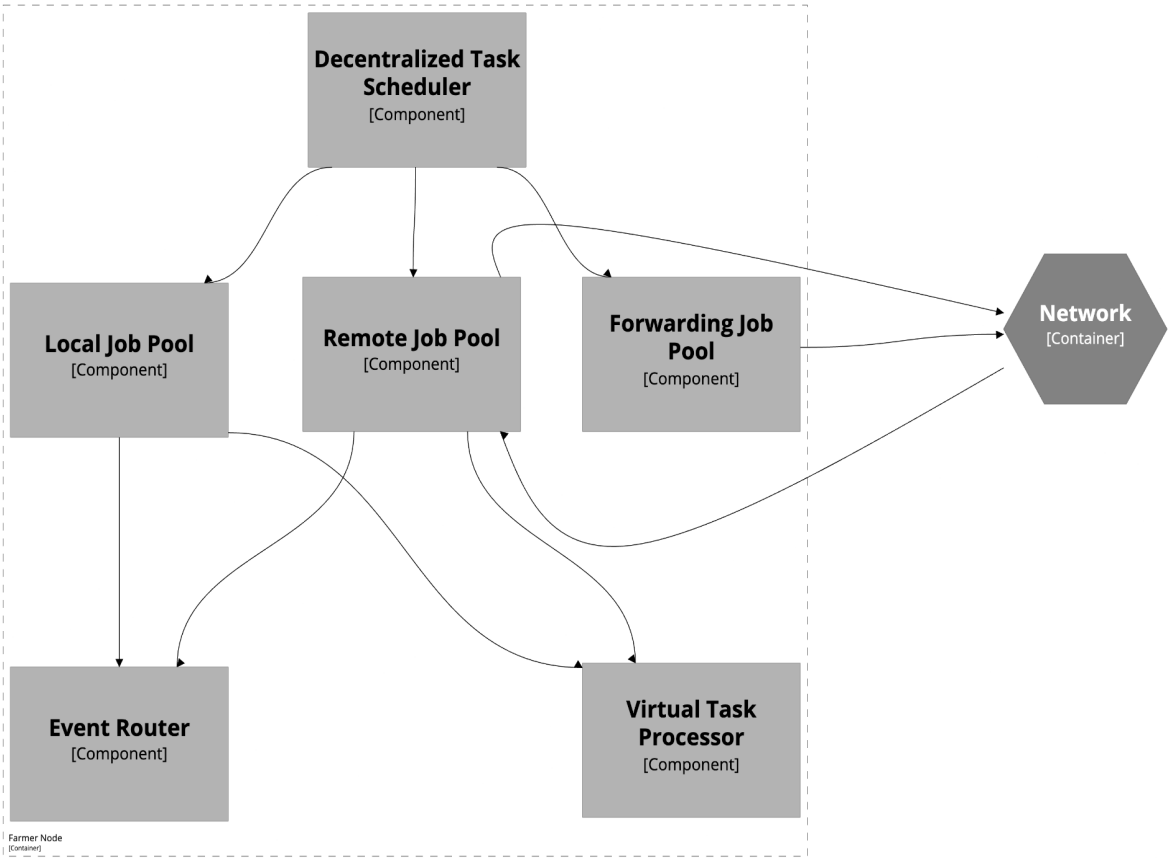
Moreover, they are allocated to the quorum equipped with the requisite data and computational resources, optimizing both execution and validation[16]. To realize this consistent allocation, we advocate for the Maglevs Hash Ring algorithm—a renowned consistent hashing algorithm extensively employed in distributed systems for task distribution among worker nodes[17]. Our preference for Maglevs Hash Ring stems from its robust open-source library, which has been rigorously tested and validated. We firmly believe in leveraging established, proven algorithms over crafting new ones from the ground up.

Decentralized Task Scheduler

A paramount objective of our proposed system is to eliminate single points of failure while optimizing efficiency and throughput. To this end, we introduce a decentralized task scheduler predicated on data locality and quorum backpressure[18]. This scheduler is adept at allocating transactions within a specific farmer quorum and can also delegate transactions to other quorums. Notably, the task scheduler employs a multifaceted allocation strategy. Initially, it assesses data locality. In scenarios where requisite data for transaction processing is non-local—indicating dependencies on transactions processed by other quorums—the scheduler prioritizes the quorum responsible for those dependencies[19].

However, if the data is local but the quorum is experiencing significant backpressure, the scheduler might still delegate the transaction (and potentially its dependencies) to other quorums[20]. Furthermore, the scheduler can opt to retain dependencies locally, transmitting only the processed data to the designated quorum[21]. This dynamic, decentralized task allocation mechanism provides the network with a level of adaptability that is often overlooked. By facilitating parallel transaction processing with maximal efficiency, the network's transactional capacity is significantly enhanced[22].

The scheduler also possesses insights into the backpressure of other quorums, enabling it to judiciously determine transaction allocation. This ensures that transactions aren't relegated to quorums lacking immediate processing capacity. Importantly, this mechanism operates asynchronously, ensuring backpressure metrics are updated as needed without excessive resource consumption[23].

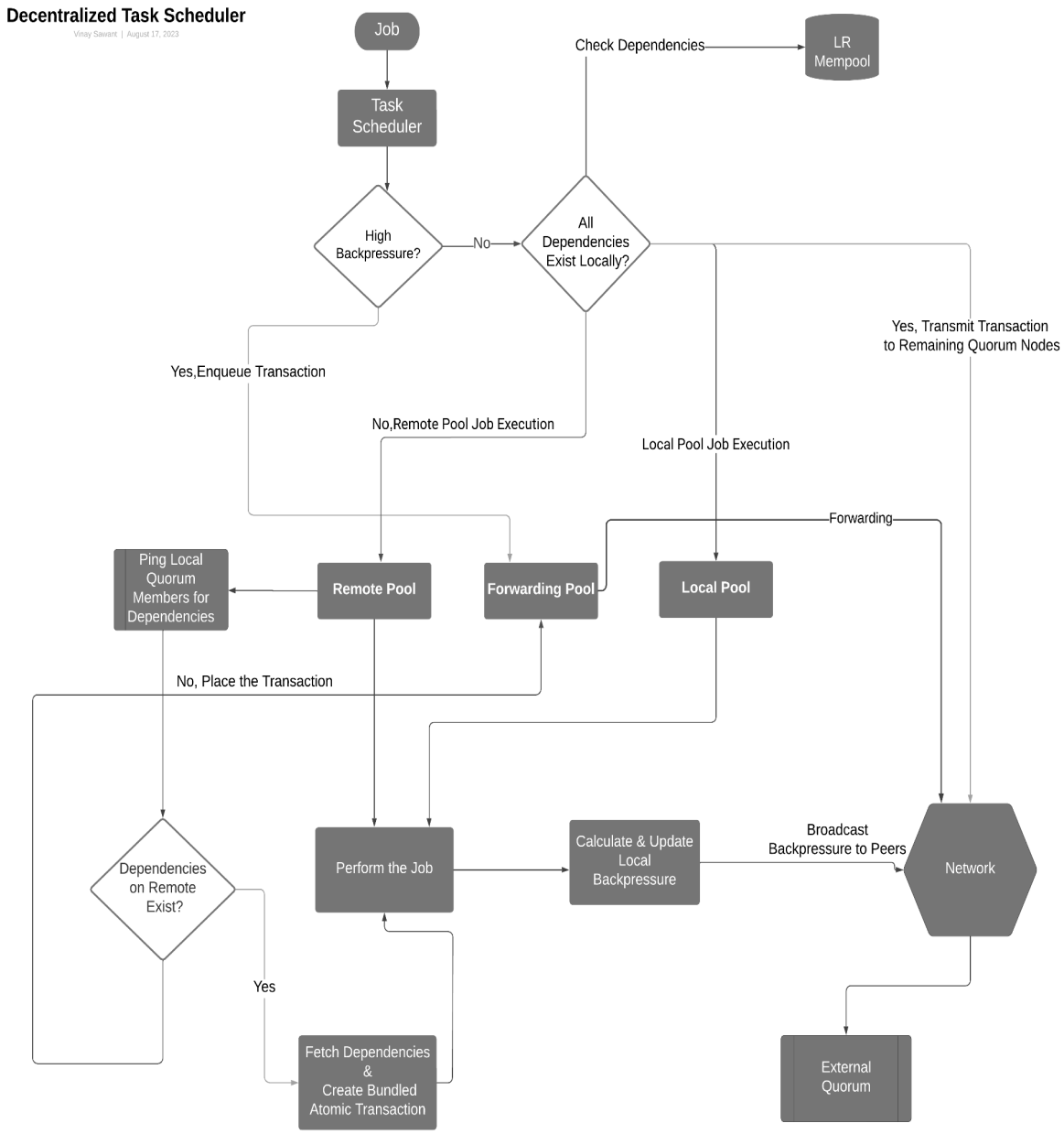


[Component] Decentralized Task Scheduler - Farmer Node
Wednesday, 16 August 2023 at 21:29 India Standard Time

The decentralized task scheduler's efficacy hinges on a sophisticated algorithm, underpinned by an efficient network transport layer facilitating both intra- and inter-quorum communication[24]. The scheduler contains three distinct queues: a local execution queue, an intra-quorum remote

execution queue, and an inter-quorum forwarding queue. The local execution queue implies that the transaction will populate the local mempool and be executed by the local node. Conversely, the remote execution queue indicates that while the transaction will be processed within the quorum, the local node might not execute it promptly. The inter-quorum forwarding queue designates the transaction for execution by another quorum[25]. The intricacies of this process are demonstrated by the subsequent decentralized task scheduling decision tree that provides a visual representation of the underlying algorithm:

Scheduler Decision Tree



Backpressure computation is integral to the decentralized task scheduling algorithm, and its formula is delineated below:

$$T_{q_n} = \sum(t_{q_1}, \ldots, t_{q_n})$$
$$\text{avg}_q = \frac{T}{n_q}$$
$$T_m = T_l + T + T_r$$
$$P = 1 + \log_{10}\left((W_l T_{l,m}\text{avg}_{m,l}) + (W_r T_{r,m}\text{avg}_{m,r}) + (W_f T_{f,m})\right)$$
$$L = \{P_1, P_2, P_3, \ldots, P_n\}$$
$$LN_b = \frac{(B_n - \min(L))}{(\max(L) - \min(L))}$$

Where:

T = *Total time to complete all tasks in q*
q = *The queue the formula is applied to*
avg = *Average*
m = *Aggregate of queues*
l = *local queue*
r = *remote queue*
f = *combined forwarding queues*
W = *weighted time to complete all tasks*[1]
P = *weighted backpressure of all queues*
L = *backpressure list* of local node and peers
LN_b = *Log Normalized Backpressure List*
t = *Total time take to complete task* $q1$

Left-Right Mempool for Concurrent Execution of Pending Transactions

A vital component of any scalable system is concurrency management. The prevention of data races, and the guarantee of, at the very least, eventual consistency, is paramount[26]. Furthermore, any system that employs concurrency in a safe manner must consider the potential for bottlenecks. Often, when a concurrency enabled data structure is used to conduct a lot of short order operations, the process of guaranteeing exclusive access, read only access, or other methods for preventing data races becomes the bottleneck itself[26].

In the case of transaction processing in a P2P compute system, verifying and proving that a given transaction is valid is typically an operation that takes micro to milliseconds. The locking and

[1] Weight is a predefined, hardcoded number for each queue

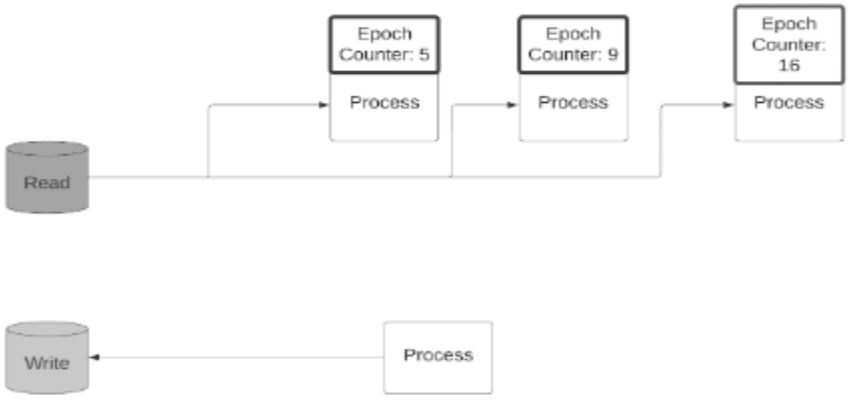
unlocking of the underpinning data structures may take just as long, if not longer, as the processing of the transaction[26].

In this case, acquiring mutually exclusive locks or read-only locks while pending writes continue to pile up can become a significant bottleneck.

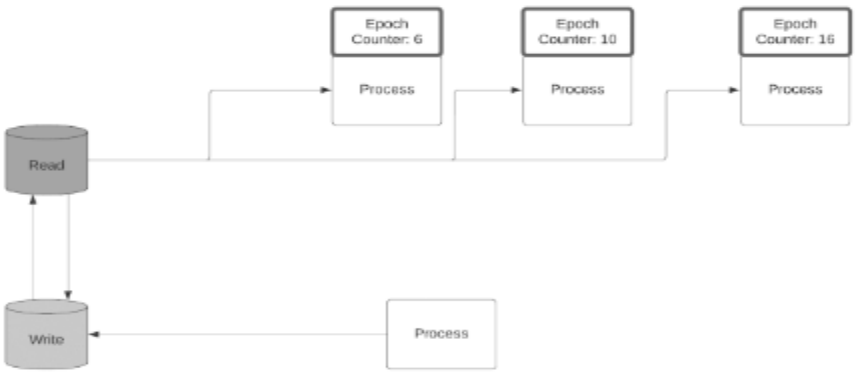
The Left-Right Concurrency Management Model

In the Left-Right concurrency management model, a Left-Right "wrapped" data structure has a "Read Only" and a "Write Only" copy of the data structure. The "Read Only" version issues "read handles" to consuming threads, and can have a theoretical unlimited number of them up to the upper bound of the machine's RAM memory.

The "Write Only" version can only have a single copy that can be accessed by a single thread without locks. If multiple access points for writes are needed, the write handle must be wrapped in an atomic reference counter and mutually exclusive lock. Read handles maintain an epoch counter. When a read begins within a given handle the handle's epoch counter increases by 1, and when it completes it increments by 1 as well.

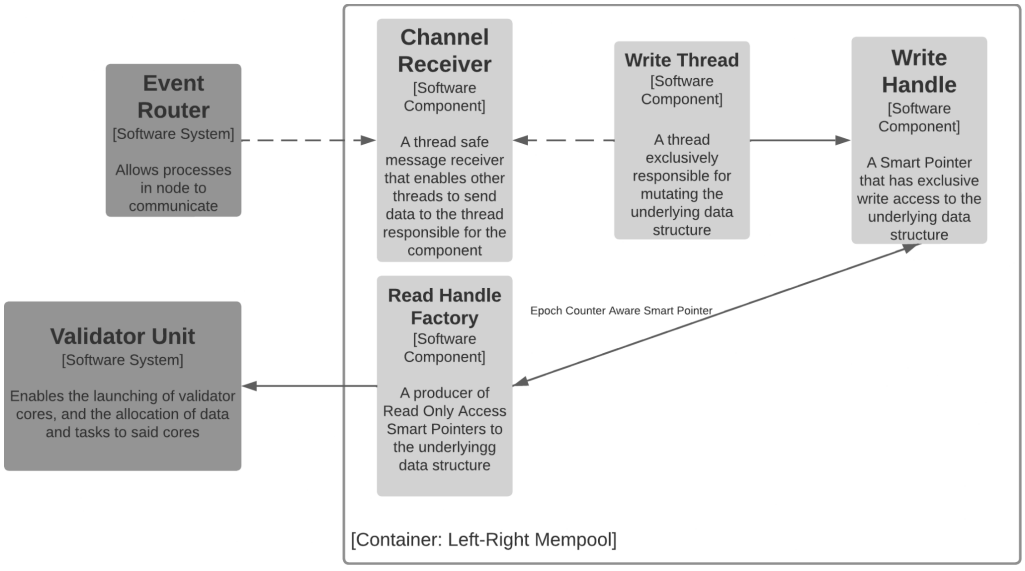


If all read handles have an epoch counter that is sitting on an even number, it is safe for the pointers to the "Read Only" copy and the "Write Only" copy to "switch". This provides processes consuming the read handles access to all of the writes since the previous refresh.



Herein we propose using the left-right concurrency primitive to scale reads within the proposed system without preventing writes to the underlying data structure[27]. Left-Right is a lock-free concurrency enabling mechanism that relies on smart pointers, a counter, and consumes additional RAM memory on the machine using it[27]. In return, any data structure that is left-right enabled is able to scale reads significantly beyond any other concurrency enabled data structures[28][29].

This is a powerful arrow in the quiver of a systems engineer seeking to build a scalable system. For the sake of the BFT consensus mechanism described herein, we explore using the Left-Right mechanism on the local copy of the mempool to enable extremely high throughput of transactions locally[27]. The Left-Right wrapped local mempool copy enables a multi-core approach to the transaction validator unit, wherein validator cores (threads) can access the mempool, access current state and pending transactions via dependency graphs, and efficiently process transactions[29].



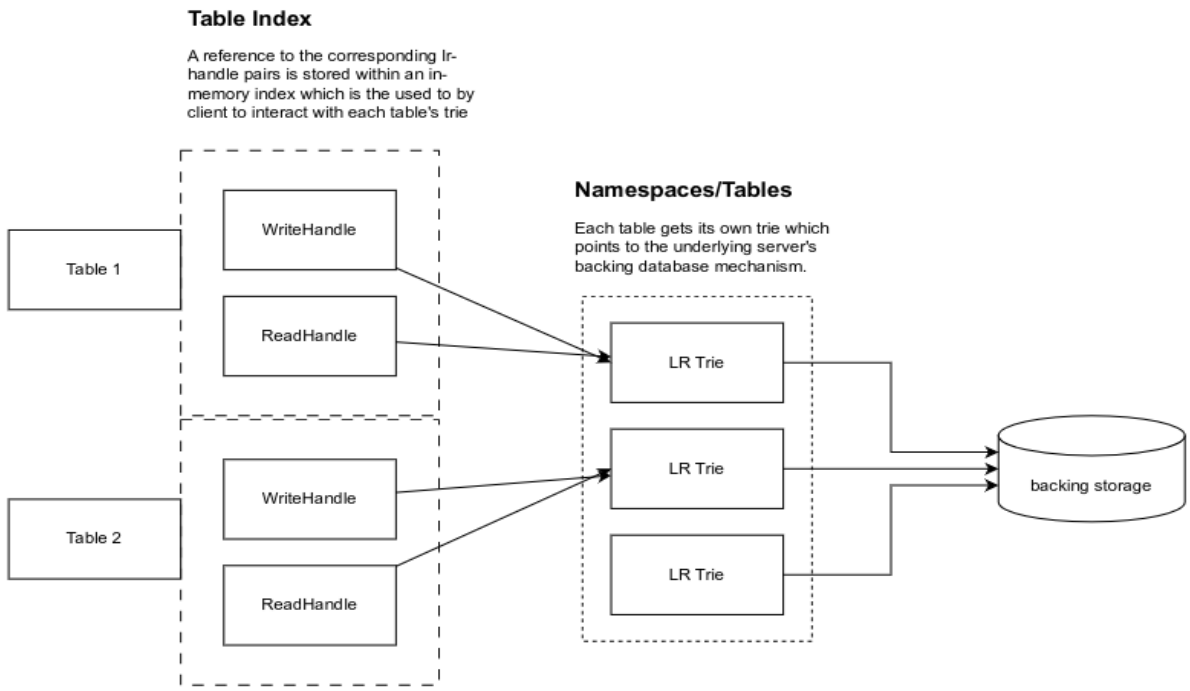
Left-Right Wrapped State Database: Enhancing Concurrent Access to Network State

Efficient access to pending transactions in the mempool is pivotal for enhancing local throughput. However, the validity of these transactions invariably hinges on the current state. Consequently, accessing the current state can emerge as a significant bottleneck, potentially surpassing the challenges posed by accessing pending transactions in the mempool[30]. The intricacies of state storage surpass those of a mempool.

While mempools primarily store pending transactions, their dependency graphs, and maintain a transient cache of processed transactions, state storage demands a more persistent approach. It necessitates the enduring storage of accounts, transactions, program access, and other network dependencies while ensuring data availability, regular state transitions, and network consistency[31].

The Left-Right concurrency model, in this context, is arguably even more apt for facilitating concurrent access to the local copy of the global network state than for the mempool. In tandem, these models forge a formidable data structure duo, propelling transaction processing throughput to unprecedented scales[27]. Presented below is a proposed state store architecture, designed to synergize with the left-right mempool and a concurrent Validator Unit employing a multi-core strategy.

Integral DB - a Left-Right Wrapped Persistent Database with Merkle Proofing:



This architecture champions lock-free access to state. When combined with lock-free access to the mempool within a multi-core validator unit framework, it paves the way for Requests Per Second (RPS) and, by extension, local Transactions Per Second (TPS) to achieve web-scale.

Validator Unit & Validator Cores for Optimistic Transaction Processing

Herein we have made reference to a "Validator Unit" and a "multi-core" architecture without much detail. A multi-core validator unit architecture is one in which multiple transactions, or subsets of transactions, can be processed simultaneously within the same unit[32]. The proposed architecture is intended to be combined with the proposed left-right enabled mempool and state store described above.

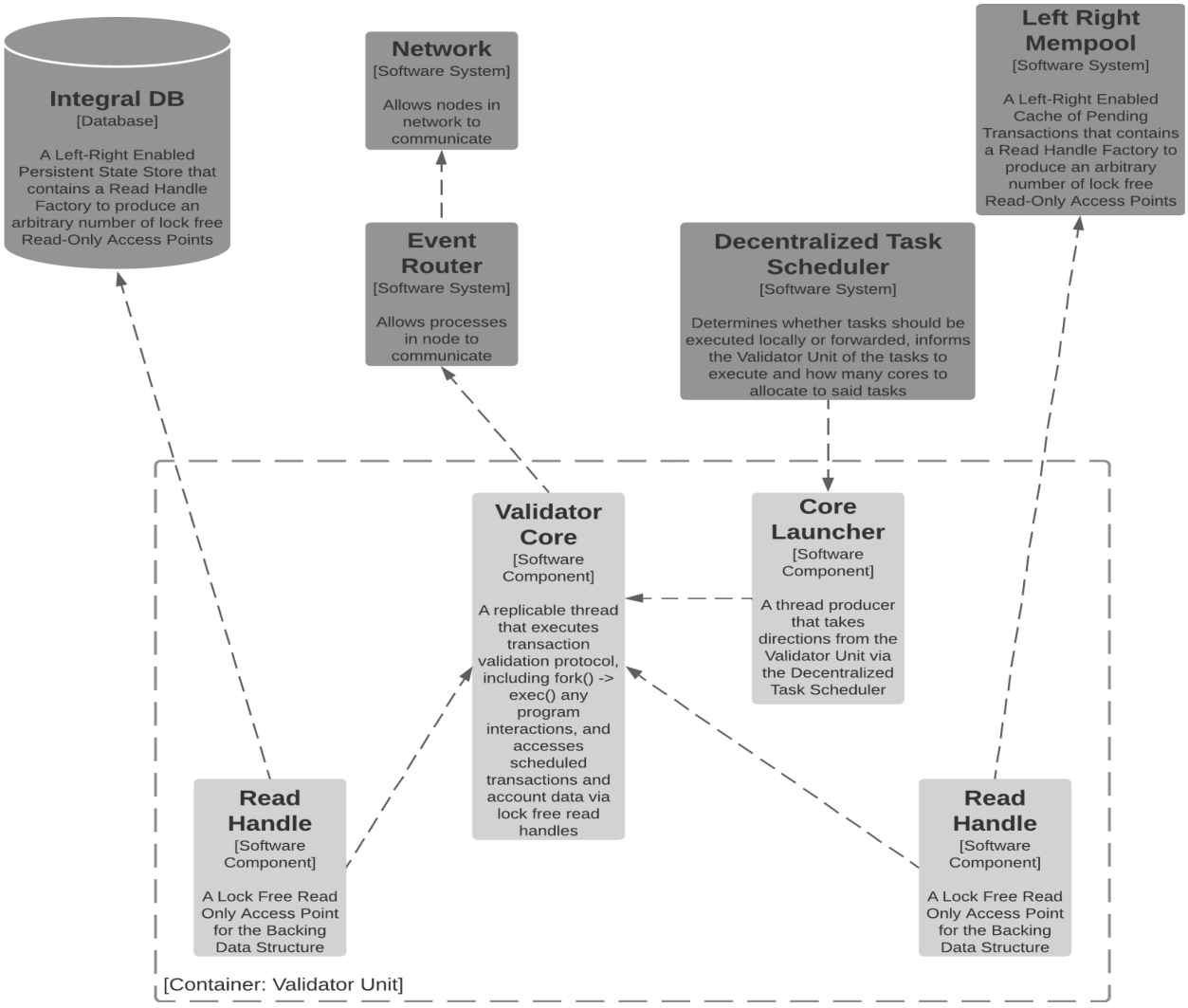
Theoretically, this approach could be applied to mutually exclusive locked and read-write locked data structures as well, albeit with some degree of performance degradation as lock acquisition would potentially become and/or create a bottleneck in performance[33]. In the multi-core validator unit, each core (thread) has access to a read handle of the necessary data structures described herein (mempool and state store), and can concurrently access, in read-only mode, the data stored in each[34].
Since the transaction validation protocol is largely read only, or can be adjusted to be purely read only, i.e. the underpinning data is not changed, and is not written back to the underpinning data structure via the same handle, if at all, this model works well for high throughput transaction processing[35].

Each core streams a subset of pending transactions allocated to the unit's mempool and works on the subset it is allocated, executing the validation/verification protocol. In a network that enables compute as well, any compute can also be handled by the core, if the virtual machine(s) allow parallel processing, or have global locks enabled to prevent race conditions[36]. In models that separate compute from consensus, these cores can fork(); exec(); the program responsible for any compute workloads. These specificities are outside of the scope of this paper, however.

Once the transaction(s) have been processed by the core, the core can use message passing, a queue, or some other mechanism for communicating it's completion of a given transaction or subset of transactions to wherever it needs to communicate such information, i.e. the mempool write thread to remove the pending transaction from the mempool and write it to the LRU cache, the transport/broadcasting mechanism in the local client for communication to the network, etc.

In this case, given that we are proposing a dual-scalability model, we propose that these cores, aside from sending the signal to remove the tx from the mempool to the write handle for the mempool, we also suggest communication to the quorum responsible for the aggregation of votes, in our model called the "harvester" quorum.

These votes inform the harvester quorum that the local node has validated (or invalidated) a given transaction, and allow it to either begin aggregating votes for the given transaction, or add the vote to its ongoing ballot of transaction processing nodes, in our proposed model known as “farmers”.



2

Dealerless Distributed Key Generation for Trustless Secure Threshold Signatures

In the realm of distributed systems, the generation of keys in a decentralized manner is paramount to ensuring robustness against single points of failure and eliminating the reliance on trusted entities. The proposed system herein necessitates the utilization of a Dealerless Distributed Key Generation (DKG) scheme, specifically for the creation of quorum public/private key pairs and the orchestration of a threshold signature scheme. Our choice for

[2] The Core Launcher can produce an arbitrary number of validator cores at any given time to concurrently process non-dependent transactions

this pivotal component is the HoneyBadger BFT Dealerless Distributed Key Generation implementation, which has garnered acclaim for its rigorous auditing, extensive testing, and proven security[37].

The intrinsic security of the threshold signature remains a cornerstone of this proposed system. Rather than venturing into the intricate and risky domain of crafting a DKG scheme from the ground up, it is judicious to lean on implementations that have withstood the test of time and rigorous academic scrutiny. While our system is architected around the HBBFT scheme, it is worth noting that this choice is not immutable.

There exists a plethora of audited, rigorously tested, and academically endorsed Dealerless Distributed Key Generators that could seamlessly integrate into this system. Notable among these are the schemes delineated in "Distributed Key Generation in the Wild"[38] and "Fast Multiparty Threshold ECDSA with Fast Trustless Setup"[39].

Farmer-Harvester Model for Parallel Processing & Scalable Batched State Updates

The model delineated in this paper draws inspiration from the worker-collector paradigm prevalent in parallel stream processing models within distributed systems[40]. This paradigm, an alternative to the conventional Master/Worker model, hinges on the collaboration between "helpers" and "workers" [41]. In classical distributed systems, which prioritize the efficient parallel processing of data streams, an emitter announces a task. Subsequently, the collector disseminates this task to the workers. Upon task completion, the workers relay the results back to the collector for state integration. This paper proposes modifications to this model to accommodate specific characteristics imperative for Peer-to-Peer consensus.

1. Byzantine Fault Tolerance: The model emphasizes redundancy in task completion within a quorum. This ensures that rogue quorum members cannot alter the state without the confirmation of correct and valid computation[42].

2. Decentralized Task Allocation: Instead of a centralized collector node, which could become a single point of failure, the model advocates for decentralized task scheduling[43].

3. Harvester Quorum: This introduces scalable batches of state updates and an additional layer of security and Byzantine fault tolerance[44].

4. Eligibility and Election: While this paper doesn't delve into specific eligibility criteria, it does underscore the importance of stringent eligibility requirements. Coupled with a

randomized, fair election process for quorum membership, this ensures enhanced security against potentially malicious nodes[45].

Byzantine Fault Tolerance

The farmer-harvester model, with its modifications, emerges as a robust framework for the parallel processing of transactions in a Byzantine fault-tolerant peer-to-peer network[42]. The system's design ensures continued operation as long as ((0.6*mf)/nf) + 0.6mh nodes remain non-Byzantine. This means that as long as 60% of a single farmer quorum and 60% of either the active harvester quorum or backup harvester quorum members are honest and active, the network remains secure, while every other farmer quorum can handle as many as 59.9999% malicious and/or faulty nodes.

Comparatively, this model offers a higher Byzantine fault tolerance than traditional PoS implementations like Ethereum's, which stands at ≈ 33% BFT. The network's overall BFT is scalable, increasing with each additional farmer quorum. For instance, with ten farmer quorums, the network can tolerate all but one farmer quorum being entirely faulty or malicious. This results in a network BFT of up to ≈ 60% once the network reaches 100 or more farmer quorums.

$$BFT = N_{m \vee f} \div N_t$$

$Where$:

$$BFT = Byzantine\ Fault\ Tolerance$$
$$N = Number\ of\ Nodes$$
$$m = Malicious$$
$$f = Faulty$$
$$t = Total$$

Given the above formula for Byzantine Fault Tolerant, we can solve for the systems BFT under different conditions with regards to the number of farmer quorums that can form and the size of the quorums:

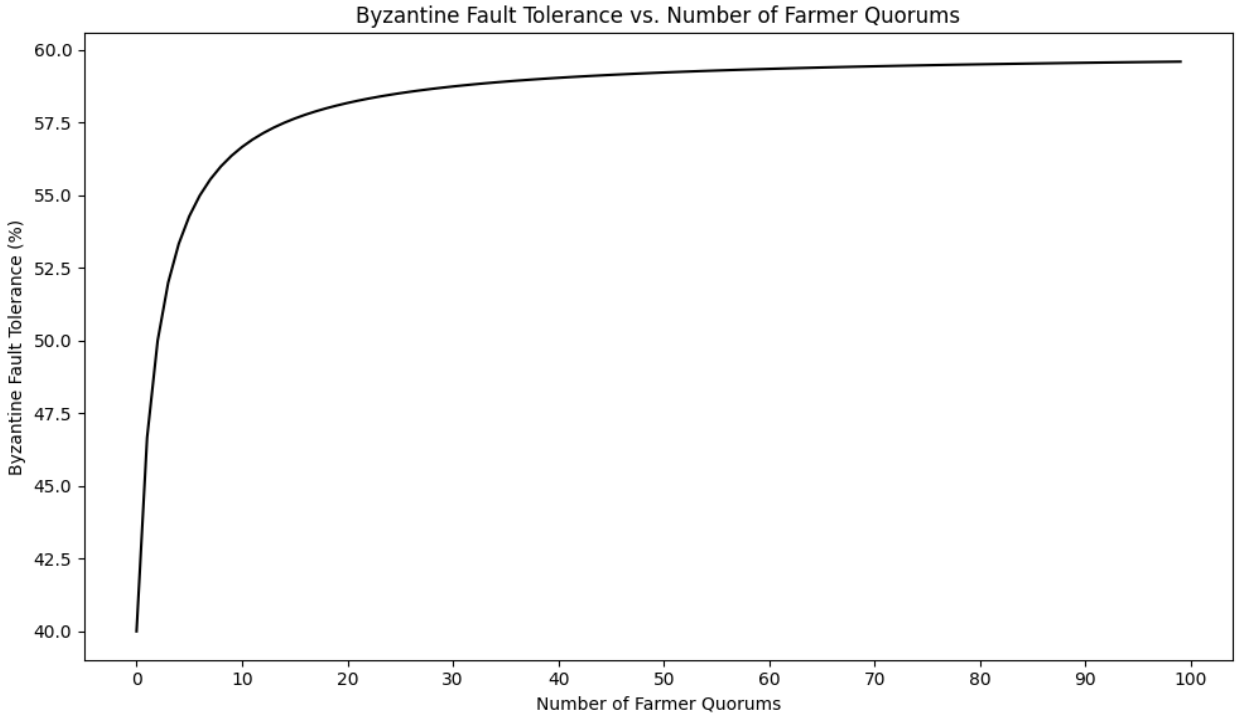
1. Single Farmer Quorum Scenario with 25 nodes in each quorum:
    a. $N_{m \vee f} = 0.4N + 0.4N = 0.8N$

b. $N_t = 2N$

c. $BFT = N_{m \vee f} \div N_t$

d. $BFT = 0.4$

2. Two Farmer Quorum Scenario with 25 nodes in each quorum:
   a. $N_{m \vee f} = 0.5999N + 0.4N + 0.4N = 1.39999N$
   b. $N_t = 3N$
   c. $BFT = N_{m \vee f} \div N_t$
   d. $BFT = 0.46663$
3. Ten Farmer Quorum Scenario with 25 nodes in each quorum:
   a. $N_{m \vee f} = (9 \times 0.59999N) + 0.4N + 0.4N = 6.59991N$
   b. $N_t = 11N$
   c. $BFT = N_{m \vee f} \div N_t$
   d. $BFT \approx 0.5636$
4. One Hundred Farmer Quorum Scenario with 25 nodes in each quorum:
   a. $N_{m \vee f} = (99 \times 0.59999N) + 0.4N + 0.4N = 60.1901N$
   b. $N_t = 101N$
   c. $BFT = N_{m \vee f} \div N_t$
   d. $BFT \approx 0.5959$



Scalability of Throughput

In distributed systems, the scalability of throughput is a paramount concern. The model presented here addresses this by allowing each farmer quorum to work on a distinct subset of transactions, each with its allocated dependency graph. When integrated with the Left-Right Mempool and State Store, an individual farmer node can process an estimated 565,000 native token transfer transactions per second. Furthermore, depending on the computational complexity of the smart contract transactions, this number ranges between 30,000 and 100,000 transactions per second[40].

A notable feature of this model is that only a 60% quorum threshold is required to validate a transaction. This allows subsets of quorum members to concurrently work on different transaction subsets. Assuming optimal task allocation with no redundant transaction processing beyond the 60% threshold, a total of 7 out of 5 quorum-level transactions can be processed. This is relative to the number of transactions a single quorum node can process in a given second. Under a set of assumptions, we can determine the theoretical upper limit:

1. Native Token Transactions take 88.5 microseconds to process[3]
2. Each validator unit has 50 cores that can parallel process transactions
3. A 60% threshold of quorum members are required to reach consensus on the validity of a given transaction
4. A perfect allocation scheme is implemented to ensure no redundancy in the transactions being processed beyond the 60% threshold within the quorum.
5. There are no faulty nodes in the quorum.

Given these assumptions, we can represent the time taken by a single core to process a single native token transfer as T.

$$T = 88.5\mu s$$

Given that there are 50 cores working simultaneously, the number of transactions processed by a single node in time T is:

$$N = 50 \times (1 \div T)$$

Given the 60% validation threshold, if allocated perfectly, for every 5 transactions a given node can process, the quorum can process 2 more, i.e. 7. This provides us with a ratio of 7:5 or 1.4 as a multiplier for quorum level transaction processing. Thus the number of transactions processed by the quorum:

[3] Benchmarks for per core TPS under Left-Right model

$$N' = 1.4 \times N$$

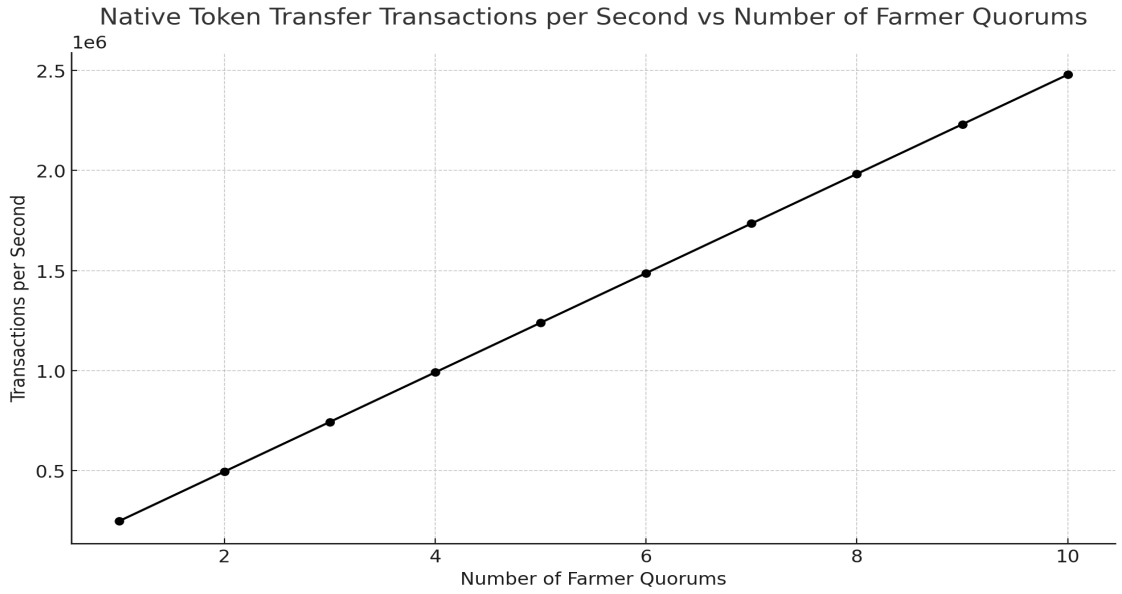
We can now solve for N' as follows:

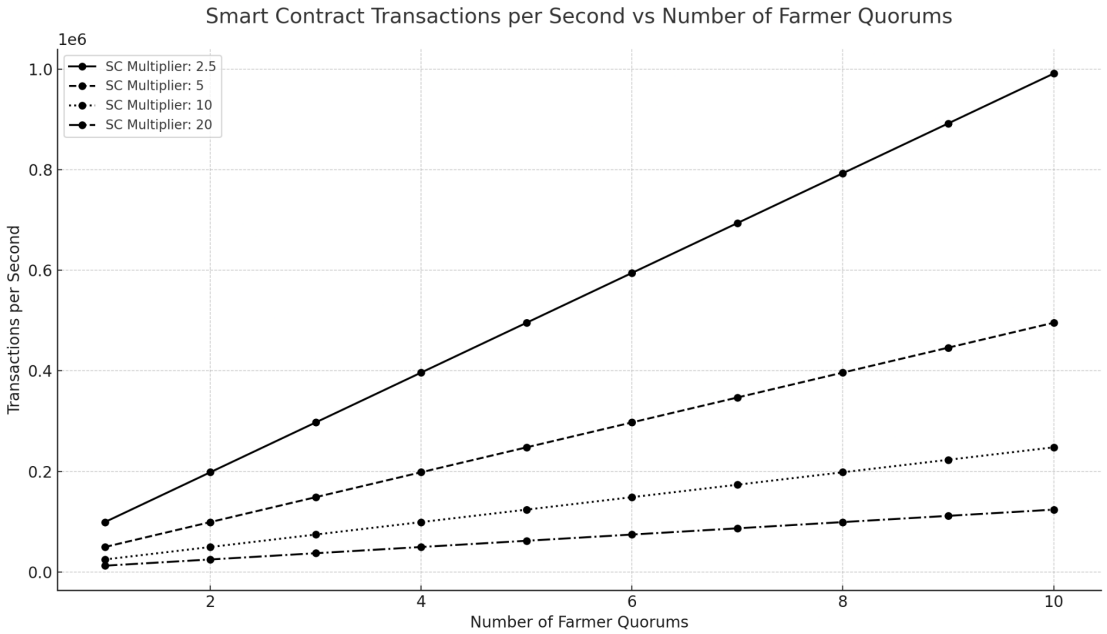
$$N = 50 \times (1 \div 88.5\mu s) = 565,000\ native\ token\ transfer\ tps$$
$$N' = 1.4 \times 565,000 = 791,000\ native\ token\ transfer\ tps\ per\ quorum$$

Adhering to the aforementioned assumptions, we can then apply the multiplier for the number of active Farmer Quorums F to deduce that the theoretical upper limit of transactions per second is $791,000F$. In a live network, due to factors such as latency, imperfect task scheduling, redundancy and bandwidth constraints, these figures may never be fully realized. However, assuming an average latency of 3 seconds across the network, we can estimate that a minimum of one-third of these transactions will be processed in full. Given forward error correction, packet loss leading to transaction drops is unlikely, but will introduce delays when packets require correction. This results in a more practical TPS value of:

$$0.94 \times (791,000F \times (1 \div 3)) \approx 247,846F.$$

We can also determine the practical limit for smart contract transactions by assuming a multiplier of native token transaction processing time, denoted as SC, where we replace N with $N = 50 \times (1 \div (SC \times T))$. Using this revised N definition, we can determine the practical upper limit of Smart Contract transaction processing[41].



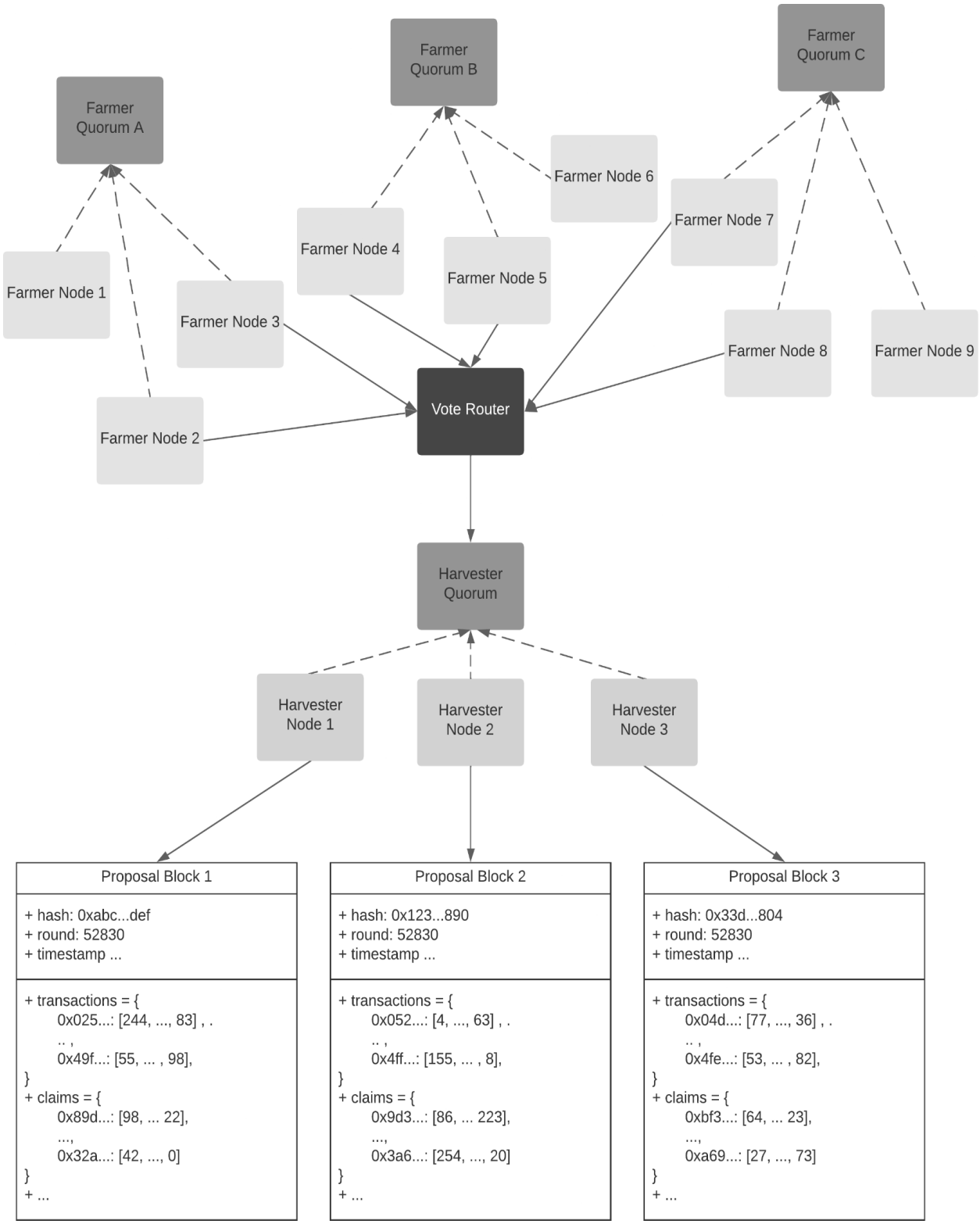


Lastly, because the role of the harvester quorum focuses on 3 non-resource intensive processes:

a. Collecting votes on transactions from farmers asynchronously
b. Building proposal blocks
c. Certifying the validity of Convergence blocks, i.e. state transitions

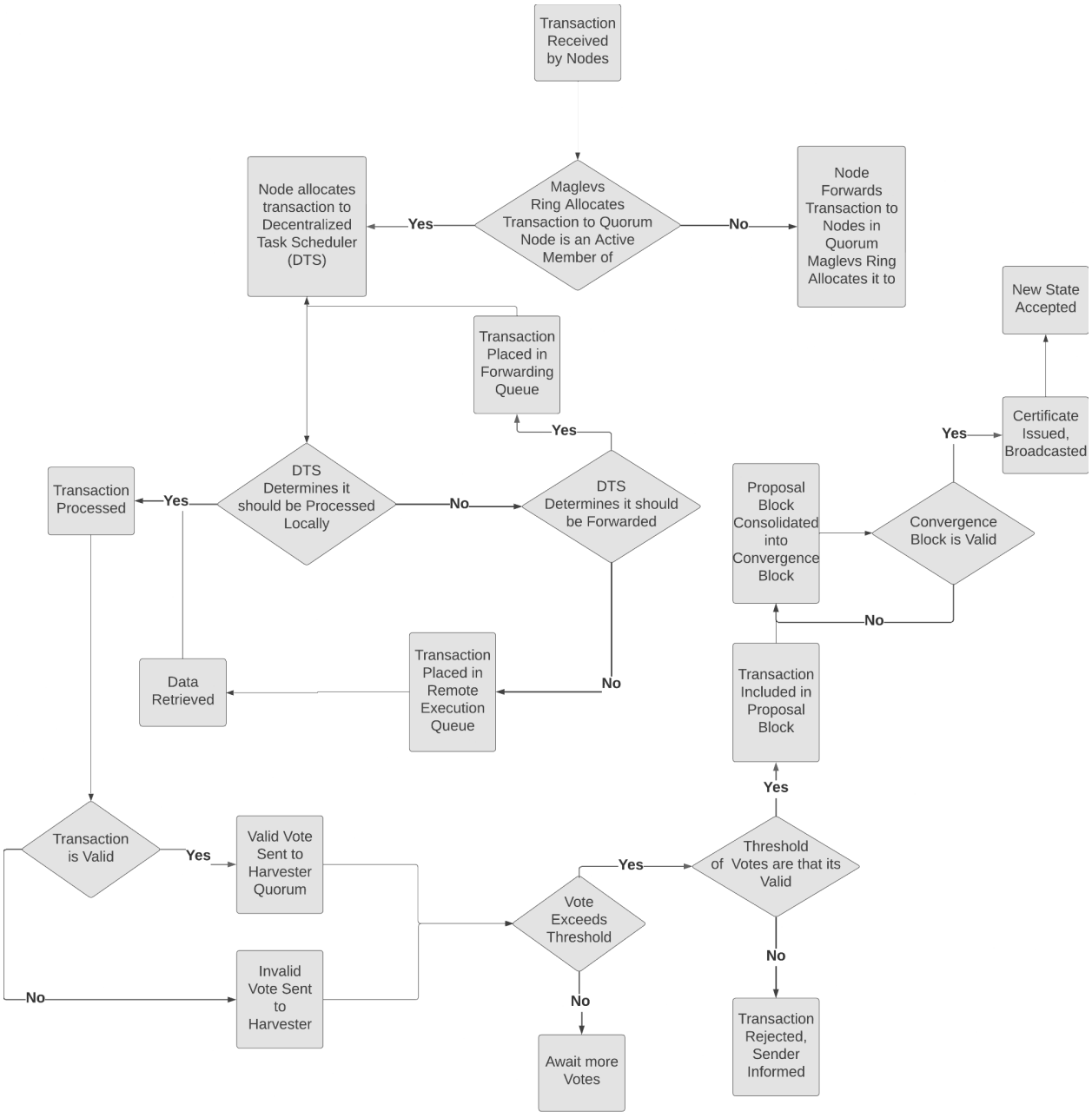
Harvesters can do a lot of what they are supposed to do in a short amount of time. Armed with the right data structures, operations that the harvester nodes engage in are low in time complexity and low in resource consumption, enabling them to scale the number of transactions they can collect votes on, aggregating them into blocks of transactions, and then collecting the consolidated final set of transactions to be included in a proposed state transition and certifying it.

Each quorum has a unique, verifiable distributed secret key, that when each node in the quorum votes on a given transaction, in the case of farmers, or to certify a state transition, in the case of harvesters, a threshold vote can be assembled. Once the threshold suggested, 60% herein, has been reached, the threshold signature can be verified using a public key which is known to all members of the network. Non-quorum members in the network need not verify every single signature for every single transaction, along with the validity of the transaction structure, data and any compute results, they need only verify the threshold signature of the certified block. Further, harvesters need only verify the threshold signature of the transactions they are collecting votes on once the threshold is reached, and need not validate the entire transaction itself. Role specialization, combined with parallel processing, enables the network to move forward fast without wasting resources on redundant tasks.



The lifecycle of a transaction under this model begins with its reception by nodes in the network, and subsequent allocation to the ingestion quorum determined using the Maglevs Ring consistent

hashing algorithm. It is finalized when a Convergence Block, containing a reference to the proposal block responsible for said transaction, is certified, and thus, a new state is accepted by the network.



Using MEV for Positive Network Outcomes

In the system delineated herein, every Harvester node is entitled to propose a single block per round. Given the constraints of the network's maximum block size, and considering the capabilities of contemporary hardware and infrastructure, it's conceivable that each harvester, assuming a consistent view of the confirmed yet unfinalized transactions, would propose blocks that optimize fee revenue[42]. This economic incentive is inherent[43]. While protocols claiming to eliminate MEV entirely remain unconvincing[44], it's essential to recognize that MEV isn't intrinsically detrimental[42]. Its negative implications arise when it leads to transaction censorship, inequitable transaction ordering, or other adverse outcomes[45].

This proposal advocates for harnessing MEV as a positive incentive[46]. When combined with the conflict resolution protocol delineated in subsequent sections, it can mitigate censorship, delays, and unfair transaction ordering. Given the harvester's awareness of their position within the conflict resolution protocol, they can strategically construct blocks to optimize their post-conflict resolution MEV. This approach ensures that low-fee transactions receive the same priority as high-fee transactions, fostering a more equitable transaction environment.

Scenarios Illustrating the MEV Model in Action:

Below we provide 3 arbitrary scenarios demonstrating the benefits of the proposed system, and how through incentive design, MEV can be used to the network's benefit with some limitations depending on network capacity.

Scenario 1: Balanced Transaction Distribution

Network Configuration:

- Number of harvesters: 10
- Maximum block size: 25kb
- Total transactions pending: 180kb across 1,000 transactions
- Fee per byte distribution: Transactions are distributed across 10 buckets, ranging from 0.000001 to 0.01.
- Conflict resolution order: Harvester 8, Harvester 3, Harvester 1, Harvester 9, Harvester 4, …

Outcome:

Given the conflict resolution order, Harvester 8, being first in line, would prioritize transactions from the highest fee-per-byte bucket to maximize its revenue. Assuming each transaction is of equal size, Harvester 8 might select, for instance, 25 transactions from the 0.01 bucket. Harvester 3, next in line, would then prioritize the next highest fee-per-byte bucket, selecting perhaps 25 transactions from the 0.009 bucket, and so on. This ensures that transactions across all fee buckets are processed in a balanced manner, optimizing both throughput and revenue for harvesters.

Scenario 2: High-Volume, Low-Fee Transactions

Network Configuration:

- Number of harvesters: 10
- Maximum block size: 25kb
- Total transactions pending: 180kb, but 70% of the transactions belong to the lowest fee-per-byte bucket.
- Fee per byte distribution: Skewed towards the lower end, with the 0.000001 bucket containing the majority of transactions.
- Conflict resolution order: Harvester 5, Harvester 2, Harvester 10, Harvester 7, Harvester 6, ...

Outcome:

Given the skewed distribution of transactions, Harvester 5, despite being first in the conflict resolution order, might opt to select transactions from the second or third fee-per-byte bucket to maximize its revenue, leaving a significant portion of the lowest fee transactions. Subsequent harvesters, recognizing the abundance of low-fee transactions and their position in the conflict resolution order, would then be incentivized to select from these low-fee transactions, ensuring they are not left unprocessed.

Scenario 3: Near Maximum Capacity

Network Configuration:

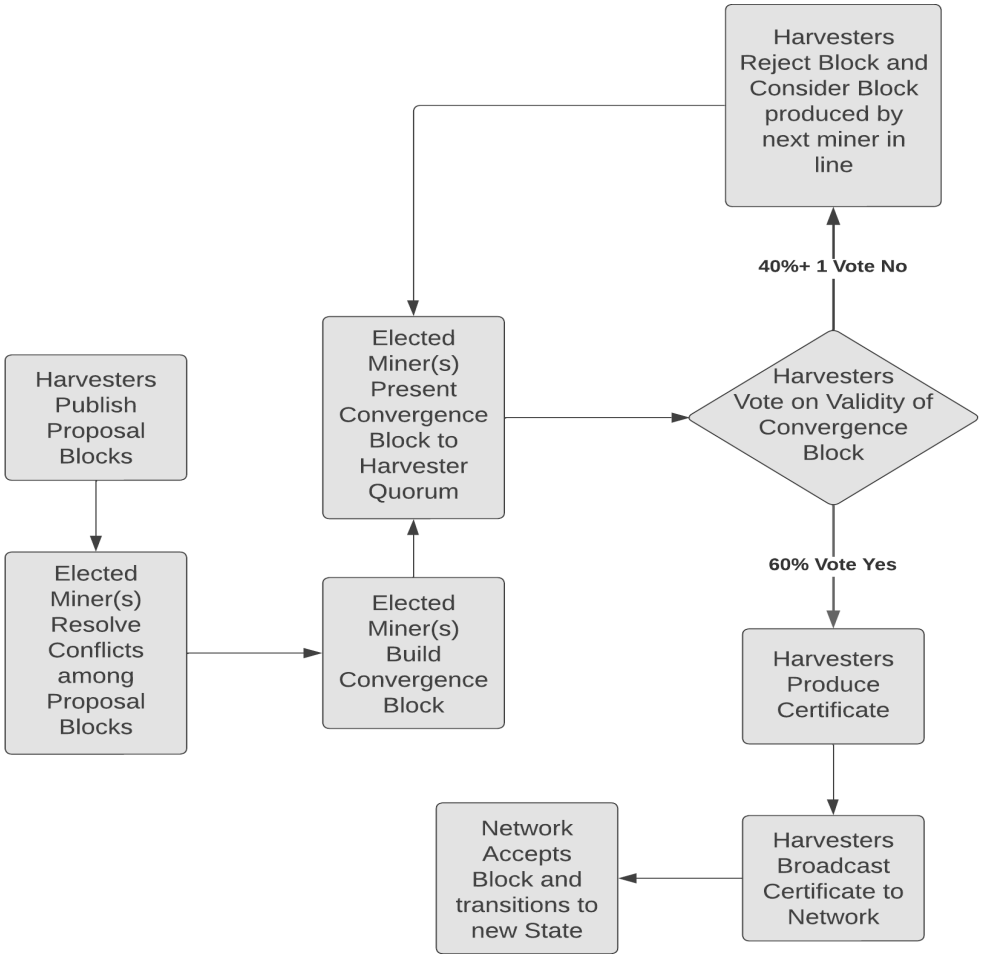
- Number of harvesters: 10
- Maximum block size: 25kb
- Total transactions pending: 240kb, distributed fairly evenly across fee-per-byte buckets.
- Fee per byte distribution: Even distribution across buckets.
- Conflict resolution order: Harvester 4, Harvester 9, Harvester 2, Harvester 6, Harvester 1, ...

Outcome:

With the network nearing its maximum transaction capacity for the round, each harvester would be more strategic in its transaction selection. Harvester 4 might select a mix of high and mid fee-per-byte transactions to fill its block. Harvester 9, recognizing that many high fee-per-byte transactions have already been selected, might opt for a mix of mid and low fee-per-byte transactions. This pattern would continue, ensuring a diverse set of transactions are processed, maximizing both network revenue and transaction throughput.

Certification & Finality

Under the proposed mechanism, a designated group of nodes must concur on the validity of a proposed state transition. We advocate for the Harvester Quorum to serve as this certification committee for convergence blocks, given its comprehensive understanding of the system and its ability to assess the validity of proposed state transitions[47]. Once the convergence block has been produced, the consolidator/conflict resolver sends the proposed convergence block back to the harvesters. Harvesters then, internally, vote on its validity. Upon reaching a 60% threshold of affirmative votes for a given convergence block, a certificate is created. This certificate, inclusive of the Harvester Quorum's threshold signature, is then broadcasted to the network alongside the convergence block[48].



This certification process serves dual purposes. Firstly, it signifies that the convergence block has been validated by an appropriately authorized group of nodes[49]. Secondly, it provides a signal to initiate the transition process from state t to state t'. Additionally, it can act as a finality signal for all transactions referenced within the convergence block. This ensures future network participants can verify the chain/network's history, synchronize with the correct chain, and minimize potential issues like soft forks and chain reorganizations[50]. This process also marks the end of one round and the commencement of the next. As proposed, convergence blocks contain data that aids the network in operating efficiently, such as election information at specified intervals[51].

Conclusion

In this study, we have presented a novel system for achieving Scalable Byzantine Fault Tolerant Consensus. This system uniquely facilitates both vertical and horizontal scalability in terms of throughput, while also ensuring horizontal scalability of fault tolerance. The proposed architecture amalgamates various innovative technologies and designs, aiming to facilitate efficient agreement on the network's current state within Peer to Peer networks. This is achieved

even in the presence of potentially malicious actors, all while processing and updating the network to accommodate hundreds of thousands, if not millions, of transactions every second.

Our assertions regarding the system's performance are not merely theoretical; they are substantiated with rigorous mathematical models and benchmarking. Based on our findings, the system's performance surpasses that of any extant peer-to-peer consensus mechanism. Consequently, it holds the potential to pioneer a web-scale decentralized network that is both permissionless and trustless. When integrated with complementary technological frameworks, this system could pave the way for a resilient internet that is resistant to censorship, ensuring the unfettered flow of information at minimal costs. Furthermore, its capacity to handle workloads comparable to, or even exceeding, the world's most active financial exchanges is noteworthy. Ultimately, such a system could revolutionize global social cooperation, enabling collaborative endeavors that are currently beyond our collective imagination.

[1] Tingling, Middleton & Seron, “Scalability and Fragility in Bounded-Degree Consensus Network”, 2020
[2] Queralta and Westerlund, “Blockchain for Mobile Edge Computing: Consensus Mechanisms and Scalability”, 2020
[3] Yang, et. al, “DispersedLedger: High-Throughput Byzantine Consensus on Variable Bandwidth Networks”, October 2021
[4] Kim, et. al, “Byzantine Fault Tolerance Based Multi-Block Consensus Algorithm for Throughput Scalability”, 2020
[5] Shang, “Scaled Consensus and Reference Tracking in Multiagent Networks with Constraints”, 2022
[6] Wang, et. al, “Consensus-Based Clock Synchronization in Wireless Sensor Networks with Truncated Exponential Delays”, 2020
[7] Hafid, Hafid & Makrakis, “Sharding-Based Proof-of-Stake Blockchain Protocols: Key Components & Probabilistic Security Analysis”, 2023
[8] Barhanpure, Belandor & Das, “Proof of Stack Consensus for Blockchain Networks”, 2020
[9] Wang, et. al., “LRBFT: Improvement of practical Byzantine fault tolerance consensus protocol for blockchains based on Lagrange Interpolation”, 2023
[10] Gans & Holden, “Mechanism Design Approaches to Blockchain Consensus”, 2022
[11] Akbar, et. al. “Distributed Hybrid Double-Spending Attack Prevention Mechanism for Proof-of-Work and Proof-of-Stake Blockchain Consensus”, 2021
[12] Yang & Shen, “Blockchain Consensus Algorithm Design Based on Consensus Hash Algorithm”, 2019
[13] Olszak, “HyCube: A distributed hash table based on a variable metric”, 2017
[14] Zeng, et. al, “AreaHash: A Balanced and fully scalable consistency hashing algorithm”, 2022
[15] Bienkowski, et. al, “Dynamic Load Balancing in Distributed Hash Tables”, 2005
[16] Yin, et. al., “Proof of Continuous Work for Reliable Data Storage Over Permissionless Blockchain”, 2022
[17] Drakatos, et. al., “Rapid Blockchain Scaling with Efficient Transaction Assignment”, 2021
[18] Stavrinides & Karatza, “Scheduling Bag-of-Task-Chains in Distributed Systems”, 2019
[19] Tantitharanukul, Natwichai & Boonma, “A Heuristic Algorithm for Workflow-Based Job Scheduling in Decentralized Distributed Systems with Heterogeneous Resources”, 2015
[20] Jovanovic & Bender, “Task scheduling in distributed systems by work stealing and mugging - a simulation study”, 2002
[21] Pop, “A Fault Tolerant Decentralized Scheduling in Large Scale Distributed Systems”, 2010, pp. 566-588
[22] Tantitharanukul, Natwichai & Boonma, “Workflow-Based Composite Job Scheduling for Decentralized Distributed Systems”, 2013
[23] Wang, Barnard & Ying, “Decentralized scheduling and locality for data-parallel computation on peer-to-peer networks”, 2015
[24] Convolbo, et. al., “GEODIS: towards the optimization of data locality-aware job scheduling in geo-distributed data centers”, 2018
[25] Bu, Rao & Xu, “Interference and locality-aware task scheduling for MapReduce applications in virtual clusters”, 2013
[26] Dice, Marathe & Shavit, “Scalable reader-writer locks”, 2012
[27] Ramalhete & Correia, “Left-Right - A Concurrency Control Technique with Wait-Free Population Oblivious Reads”, 2015
[28] Shapiro, et. al., “A comprehensive study of Convergent and Commutative Replicated Data Types”, 2011
[29] Herlihy & Shavit, “The art of multiprocessor programming”, 2008
[30] Freitag, Kemper & Neumann, “Memory-Optimizaed Multi-Version Concurrency Control for Disk-Based Database Systems”, 2022
[31] Kleppmann, Beresford & Svingen, “Online Event Processing: Achieving Consistency Where Distributed Transactions Have Failed”, 2019
[32] Alperen, et. al, “An Evaluation of Task-Parallel Frameworks for Sparse Solvers on Multicore and Manycore CPU Architectures”, 2021

[33] Wang, et. al., “Meshed Bluetree: Time-Predictable Multimemory Interconnect for Multicore Architectures”, 2020
[34] Gregorio, “A distributed hardware algorithm for scheduling dependent tasks on multicore architectures”, 2009
[35] Zhang, et. al., “Towards Concurrent Stateful Stream Processing on Multicore Processors (Technical Report)”, 2020
[36] Li, et. al., “SwitchTx: Scalable In-Network Coordination for Distributed Transaction Processing”, 2022
[37] Miller, et. al., “The Honey Badger of BFT Protocols”, 2016
[38] Kate, Huang & Goldberg, “Distributed Key Generation in the Wild”, 2012
[39] Gennaro & Goldfeder, “Fast Multiparty Threshold ECDSA with Fast Trustless Setup”, 2019
[40] Cachin, et al., “Scalable Byzantine Fault-Tolerant Agreement in Directed Dynamic Networks”, 2020
[41] Danelutto, Torquati & Kilpatrick, “State access patterns in embarrassingly parallel computations”, 2016
[42] Mazorra, Reynolds & Daza, “Price of MEV: Towards a Game Theoretical Approach to MEV”, 2022
[43] Sakiz & Gencer, “Blockchain Technology and its Impact on the Global Economy”, 2019
[44] Yang, et. al., “SoK: MEV Countermeasures: Theory and Practice”, 2022
[45] Carrillo and Hu, “MEV in fixed gas price blockchains: Terra Classic as a case of study”, 2023
[46] Piet, Nair & Subramanian, “MEVade: An MEV-Resistant Blockchain Design”, 2023
[47] Naserameri, “Improving Privacy in Blockchain by Combining Group Signature and Groth-Sahai Certification System”, 2022
[48] Fu, Du & Li, “Distribution of CA-Role in Block-Chain Systems”, 2018
[49] Azan, et. al., “Proposal for an integrative performance framework based on Distributed Ledger Technology dedicated to higher education students entering the labor market”, 2022
[50] Koutanov, “Strict Serializable Multidatabase Certification with Out-of-Order Updates”, 2021
[51] Zou, et. al., “ArchivesChain: Distributed PKI Archives System”, 2022