Tolerating Disasters with Hierarchical Consensus

Yahyaoui, Wassim; Bruneau-Queyreix, Joachim; Völp, Marcus; Decouchant, Jérémie

DOI
10.1109/INFOCOM52122.2024.10621371

Publication date
2024

Document Version
Final published version

Published in
Proceedings of the IEEE INFOCOM 2024 - IEEE Conference on Computer Communications

Tolerating Disasters with Hierarchical Consensus

Wassim Yahyaoui*, Joachim Bruneau-Queyreix†, Marcus Völp*, Jérémie Decouchant‡
*SnT - University of Luxembourg, †CNRS-LaBRI - Univ. Bordeaux - Bordeaux INP, ‡Delft University of Technology
wassim.yahyaoui@uni.lu, joachim.bruneau-queyreix@u-bordeaux.fr, marcus.voelp@uni.lu, j.decouchant@tudelft.nl

IEEE INFOCOM 2024 - IEEE Conference on Computer Communications | 979-8-3503-8350-8/24/$31.00 ©2024 IEEE | DOI: 10.1109/INFOCOM52122.2024.10621371

***Abstract*—Geo-replication provides disaster recovery after catastrophic accidental failures or attacks, such as fires, blackouts or denial-of-service attacks to a data center or region. Naturally distributed data structures, such as Blockchains, when well designed, are immune against such disruptions, but they also benefit from leveraging locality. In this work, we consolidate the performance of geo-replicated consensus by leveraging novel insights about hierarchical consensus and a construction methodology that allows creating novel protocols from existing building blocks. In particular we show that cluster confirmation, paired with subgroup rotation, allows protocols to safely operate through situations where all members of the global consensus group are Byzantine. We demonstrate our compositional construction by combining the recent HotStuff and Damysus protocols into a hierarchical geo-replicated blockchain with global durability guarantees. We present a compositionality proof and demonstrate the correctness of our protocol, including its ability to tolerate cluster crashes. Our protocol — ORION[1] — achieves a 20% higher throughput than GeoBFT, the latest hierarchical Byzantine Fault-Tolerant (BFT) protocol.**

***Index Terms*—Blockchain consensus, Byzantine fault and intrusion tolerance, clustered protocol**

I. INTRODUCTION

Geo-replication provides disaster recovery by storing important information geographically distributed to tolerate entire sites crashing or becoming partitioned from the network. In addition, cyberattacks and similar accidental or intentionally malicious incidents may compromise individual replicas at a site, causing replicas to act irrationally or possibly even maliciously. To compensate, sites replicate nodes and run Byzantine fault-tolerant consensus protocols to reach agreement about the transactions they commit. In a geo-replicated setting, such protocols are hierarchical with the low-layer groups — called *clusters* — aiming for consensus among the replicas of the same data center or region and a high-layer replica group — the *global group* — striving for agreement among data centers to ensure consistency across locations.

Several hierarchical consensus protocols have been proposed [1]–[9]. However, these proposals grapple with high-cost view-change protocols, often resulting in suboptimal performance and hindering practical application, albeit with limited decentralization or consistency. For example, Steward [2] ensures global ordering and durability by considering transactions of a single cluster at a time, GeoBFT [4] ensures only durability, but not global ordering, which prevents clients from interacting with information that is not maintained by their local cluster, and HiBFT [3] only orders globally, but not locally, which prevents local ahead-of-commit conflict resolution for the majority of transactions that only affect exclusively locally modified objects.

In this paper we present ORION, a hierarchical consensus protocol that, in contrast to previous works, harnesses the full potential of cluster parallelism and that proactively orders requests locally to expedite conflict resolution and global ordering, offering clients the flexibility to interact with all objects as necessary. It accomplishes this while effectively circumventing the pitfalls of expensive view-change protocols that limit its predecessors.

ORION distinguishes itself from previous protocols designed specifically for hierarchical consensus. This work leverages a novel insight in the construction of BFT protocols that was originally introduced in Steward: the construction of hierarchical protocols from non-hierarchical building blocks. More precisely, we found that cluster consensus can not only constrain Byzantine replicas to behave like crashed ones, but in fact cluster consensus can replace trusted-trustworthy components as they are found in hybrid BFT protocols, such as MinBFT [10] or Damysus [11], and used there to reduce the required number of replicas and protocol steps[2]. Cluster confirmation — i.e., the confirmation of all actions of global-group replicas (or, in our case, of replicas that originate from and represent the local cluster) through a majority of other local replicas — thereby prevents Byzantine representatives from jeopardizing the consistency of the blockchain, while a cluster-global rotation scheme ensures liveness, even if temporarily all representatives in the global group are Byzantine.

ORION exemplifies our generic construction method by leveraging as building blocks a consistent broadcast protocol, HotStuff [14] for early request-preordering and local conflict resolution, and Damysus [11] as hybrid protocol for reaching consensus in the global group. We achieve the latter by implementing Damysus' trusted-component services — checker and accumulator — through cluster confirmation. ORION achieves 20% higher throughput than GeoBFT at a slight increase in latency, measured in the time until the client receives

This work is supported through FNR/FCT grant C18/IS/12694392 (ThreatAdapt).

[1]In the Greek mythology, Orion is a blind Giant that carried his servant on his shoulders to see for him. Our protocol uses cluster-confirmation to lift Damysus to the global level and use it for global consensus.

[2]See also Gupta et al. [12] and Bessani et al. [13] for a recent discussion about hybrid BFT protocols.

confirmation that its requests are durably stored and in a manner that is robust to whole cluster crashes.

Specifically, ORION features the following key points.

- ORION leverages a local pre-ordering for conflict resolution locally at a cluster without global interaction
- It is the first hierarchical protocol based on HotStuff, which avoids complex and costly view changes
- ORION combines a consistent broadcast protocol for transaction dissemination before global ordering, and an inter-cluster SMR protocol that utilizes the Damysus hybrid BFT-SMR protocol to reduce the required number of clusters and global communication phases.
- A key achievement of ORION is its novel compositionality, effectively integrating various building blocks into a cohesive framework. This unique assembly not only achieves scalability and high throughput but also significant enhances the efficiency and robustness of Byzantine fault-tolerant consensus protocols.

After discussing related work (in Sec. II) and our fault and system models (in Sec. III), we introduce (in Sec. IV) ORION, our approach to geo-replicated Byzantine fault-tolerant state machine replication (BFT-SMR) and Blockchains. Sec. V describes the ORION steps in detail. Our construction from different base protocols is made possible by a novel compositionality result, which we prove in Sec. VI. Sec. VII evaluates our protocol and compares it to GeoBFT and non-hierarchical baselines. Sec. VIII concludes this paper.

II. RELATED WORK

Classical and streamlined BFT consensus. Classical BFT consensus algorithms assume partially synchronous networks, tolerate up to f Byzantine replicas and require at least $3f + 1$ replicas to guarantee both safety and liveness. These algorithms include the seminal PBFT [15], BFT-SMaRt [16], Zyzzyva [17], are leader-based and rely on a view-change scheme. The quadratic message complexity of these protocols typically result in low performance in geo-distributed settings. Improving over PBFT's performance, HotStuff [14] avoids all-to-all communication patterns, by integrating view-change procedure into the steady case, leading to linear message complexity and increased throughput. In return Hotstuff requires one additional communication phase. In this work, we use Hotstuff pre-order requests within a cluster and extend its rotation scheme to avoid view changes, locally and globally.

Trusted components. Another line of work aimed at leveraging trusted components inside BFT consensus algorithms to reduce the number of replicas they require, and possibly the number of communication phases they employ. Possible trusted components have included trusted logs [18], attested append-only memory (A2M) [19], and multiple trusted counters [20]. Following this line of work, MinBFT [10] builds on PBFT and uses a trusted monotonic counter to make omissions and equivocation detectable and reduce the number of replicas to $2f+1$ and the communication phases to two. Damysus [11] identifies two trusted services for streamlined BFT protocols. To ensure that replicas cannot fabricate information about the most recently prepared blocks, these services record additional block-related information. Damysus removes one communication phase (two network latencies) from HotStuff and also uses $2f+1$ replicas. Although we do not use trusted components in this work, we leverage Damysus at the global communication level. More precisely, under our system model the quorum certificate from a cluster in our protocol is assimilated as a trusted component's signature in Damysus.

Hierarchical BFT. Organizing replicas in a hierarchical manner is another way to improve the performance of consensus in wide-area networks [1]. Hierarchical protocols have mostly relied on PBFT, while we leverage recent streamlined protocols, and separate block dissemination from ordering to obtain a higher throughput. In Steward [2], replicas rely on PBFT inside clusters, and clusters use a two-phase commit protocol to reach global consensus. Steward assumes that up to one-third of a cluster's replicas can be Byzantine, while only a minority of the clusters can crash. HiBFT [3] uses PBFT at both the local and the global consensus levels. HiBFT processes one block per global consensus round, while we allow the asynchronous dissemination of blocks and the creation of superblocks by the global consensus layer. In addition, HiBFT uses PBFT's heavy view-change procedure, while we build on HotStuff and Damysus, which integrate the view-change procedure in the normal case. GeoBFT [4] relies on PBFT within each cluster, and uses a representative per cluster that shares a certified client request with other clusters. Eventually all client requests for a given view are received by a representative and deterministically ordered. GeoBFT employs a complex and expensive remote view-change protocol.

Other hierarchical protocols based on PBFT include [5]–[9], [21]. Some protocols build on different protocols in the two-layer hierarchy [22], [23], while a few adopt a variable number of layers [24], [25]. Regardless of the unique mechanisms they employ, all these protocols grapple with the challenges associated with costly view-change operations and their throughput is limited to a single cluster proposal per round.

Making BFT scale. We leverage several design paradigms that have been described in previous works. First, we separate block dissemination from their ordering, because blocks generated at the global level only contain the hashes of blocks generated by clusters. A similar mechanism was used in Narwhal/Tusk [26] and in Bullshark [27], which are DAG-based [28]–[30]. The idea of assembling superblocks was used in HoneyBadgerBFT [31] and RedBelly [32]. Other methods that could be used to improve the performance of BFT protocols include the use of multiple leaders, which Mir-BFT [33] and Alder [34] pioneered. These protocols are built around PBFT or Algorand [35] and are therefore not directly applicable to HotStuff, which is the protocol we use. We do not employ sharding techniques [36], [37] in ORION as their performance depends on the actual workloads being executed but they are fully compatible with its design.

Table I provides an overview of the BFT consensus protocols that are most closely related to this work, and compares them with ORION, our protocol. Note that ORION's threat

TABLE I
KEY METRICS OF ORION AND RELATED WORKS
- n: # replicas per cluster - f: # faulty replicas in each cluster - c: # clusters - F: # faulty clusters -

	# Clusters	Total # Replicas	Threat model cluster/global	Msg complexity of normal case	Msg complexity of view change	Communication steps (+view change)
PBFT [15]	1	$3f$+1	BFT	O(n²)	O(n²)	3(+2)
HotStuff [14]	1	$3f$+1	BFT	O(n)	-	8
Steward [2]	$2F$+1	$(3f$+1$)(2F$+1$)$	BFT/CFT	O(cn²+c²)	O(c²n²)	3(+4)
HiBFT [3]	$3F+1$	n*c	-/BFT	O(cn+c²)	O(c²n²)	3(+3)
GeoBFT [4]	c	$(3f$+1$)$c	BFT/-	O(cn²+c²)	O(c²n²)	1(+3)
ORION (this work)	$2F$+1	$(3f$+1$)(2F$+1$)$	BFT/*CFT	O(cn+c²+c)	-	6

model is close to Steward's but contains some noticeable adjustments to more effectively manage potential faulty behaviors at the global level. In both Steward and ORION, clusters may crash, however in ORION, up to F out of the $N = 2F+1$ representatives that form the global group can be Byzantine.

III. SYSTEM AND FAULT MODEL

Clusters and Replicas. We consider a static system that implements a consensus service, used for example by a permissioned blockchain, in geo-replicated settings. The system consists of N clusters $C_1, \dots C_N$. Each cluster C_i contains $n_i = 3f_i + 1$ replicas of which up to f_i replicas can be arbitrarily faulty (i.e., Byzantine). We write $r_{i,k}$ for the k^{th} replica in cluster C_i and omit indices where they are clear from the context. Clusters may crash, e.g., in case of fire in a cluster, regional blackouts or network partition. In that case, we say that a cluster has *crashed*. Our goal is to tolerate up to f_i Byzantine replicas in each cluster and up to F out of the $N = 2F + 1$ crashed clusters.

Within each cluster, one replica is a local leader (denoted r_L^i) and is responsible for pre-ordering requests locally. Leaders are rotated at each view change, which is a global event. Replicas are able to identify who leads a cluster based on the global view v (e.g., by choosing as leader r_L^i of cluster C_i the L^{th} replica where $L = v \ mod \ n_i$).

Clients. Clients interact primarily with the cluster C_i that is closest to them, but will be able to receive service from other clusters should C_i crash. We call C_i the *local cluster* of such a client. Any number of clients may be faulty.

Global Group. In addition to the cluster leaders, we distinguish a second replica in each cluster — the *representative* R^i. Representatives form the global group. Leader and representative might be the same replica, but, for load-balancing reasons, choosing different replicas suggests itself. We achieve this by rotating the representative on global view changes and by adjusting local rotation to skip over selected representatives when selecting the next local leader.

Notice that because representatives are cluster replicas, some might belong in crashed clusters. All members of the global group can be Byzantine. We tolerate these obstacles by rotating out of these configurations and eventually reaching one where at most F representatives originate from crashed clusters or are Byzantine.

Communication and Synchrony. We assume communication among replicas of a given cluster to exhibit a much lower latency and higher bandwidth on average than inter-cluster communications. We build upon the partially synchronous model [38]. That is, we assume that after an unknown global stabilization time GST there exists a known bound Δ such that communication latency remains below Δ.

Cryptographic Primitives. We assume the cryptographic functions we use to be secure, e.g., signatures cannot be forged and hashing schemes are collision-resistant. Moreover, we assume replicas to be deployed securely (e.g., using authenticated boot) and healthy ones to not leak credentials.

IV. GEO-REPLICATED BFT-SMR AND BLOCKCHAINS

We first give an overview of our geo-replicated permissioned blockchain, and of ORION, the hierarchical Byzantine-fault-tolerant consensus protocol it uses.

In the absence of faults, clients communicate with their preferred cluster to issue transactions that should be made durable by placing them in the geo-replicated blockchain before executing them. Clusters assemble client transactions into blocks, disseminate them for other clusters to store them durably, order them into the local chain, and request the global group to include them into the blockchain. The global group in turn assembles received blocks into a superblock and appends it to the blockchain. Once appended, clusters resolve global conflicts in a deterministic manner and, if required, execute the contained transactions to update their state. Fig. 1 shows the structure of our blockchain.

To minimize the amount of data that needs to be communicated across clusters, we limit the communication of

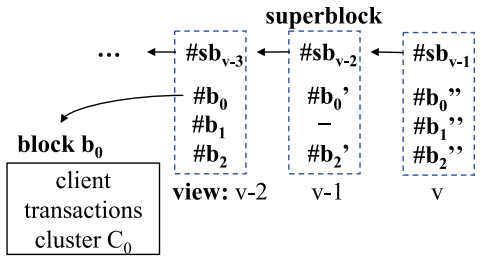

Fig. 1. Structure of our blockchain.

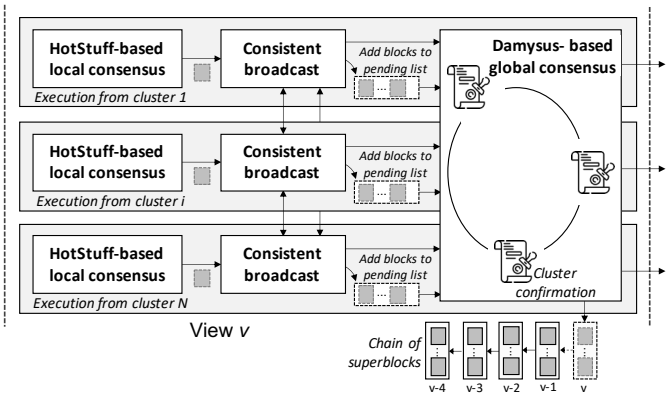

Fig. 2. Overview of ORION and the interplay of its subcomponents

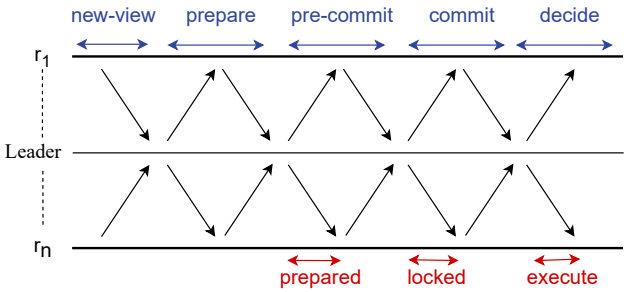

Fig. 3. Communication phases in HotStuff

transactions to a block dissemination phase and reference blocks using their secure hash during the remainder of the protocol. Consequently, superblocks store hashes to blocks plus one hash to connect to the last superblock already in the chain. A superblock contains a maximum number k of hashes of blocks proposed by the clusters and it is possible that it contains more than one block from a given cluster. The conditions for accepting a superblock are (1) that blocks from a cluster are queued in a local-order preserving manner and (2) stored durably in at least $F + 1$ clusters. Healthy global leaders will further try to balance how many blocks are stored from each cluster, a property which faulty global leaders may of course try to jeopardize.

ORION combines elements from three standard protocols, which we use as building blocks: the consensus protocols Hotstuff [14] and Damysus [11], and a consistent broadcast protocol [39], [40]. Fig. 2 depicts their interplay. Each replica instantiates three processes: a *local consensus protocol* based on HotStuff, which locally orders the transactions of each cluster to form a block; a *consistent broadcast protocol* that disseminates blocks to all clusters and tolerates the presence of f faulty replicas in each cluster, and a *global consensus protocol* based on Damysus, which forms a superblock, reaches agreement on it and includes it into the blockchain.

While Fig. 2 presents the primary steps of ORION in a sequential manner, it is crucial to note that these steps actually operate in a pipelined and asynchronous fashion. The local protocol generates blocks of transactions while other blocks are disseminated and global agreement on superblocks with hashes of already disseminated blocks is reached.

An important characteristic of our protocol is that it can handle scenarios where more than F of the replicas in the global group are faulty (Byzantine or originating from crashed clusters). In fact, the entire global group can become malicious. To address this challenge, our protocol restricts the global group's role to simply relaying decisions made by the clusters. To achieve this, we employ a mechanism called *cluster confirmation*, which serves to gather and validate the information that replicas in the global group must relay. Through this confirmation mechanism, equivocation attempts are immediately detected and will trigger a global view change. This effectively transforms clusters into trusted components that the global consensus protocol considers as entities that may only fail by crashing. Upon detecting malicious behavior or on timeout, our rotation mechanism replaces the replicas in the global group until eventually a healthy configuration with less than F faulty replicas is found. Cluster confirmation ensures that ORION is safe and its rotation scheme eventually and repeatedly returns to healthy configurations, which ensures liveness after global stabilization.

Clusters replay disseminated blocks in case they were not included in the latest superblock. In addition, clients retransmit transactions in case their local cluster crashes. In the latter case, we ensure re-transmitted transactions conflict with their original transaction to ensure only one becomes durable and to avoid executing the same transaction twice.

V. PROTOCOL DETAILS

We now describe ORION's sub-protocols in greater detail: block formation through local consensus, block dissemination through consistent broadcast and global agreement on a superblock through Damysus-based global consensus.

A. HotStuff-based local consensus

Exploiting the physical proximity of clients to their preferred cluster and of replicas within a cluster is key to the efficiency of hierarchical BFT-protocols. We leverage physical proximity in two aspects: to agree on request blocks in local clusters and thereby allow for early conflict resolution of those transactions that only affect other local clients, and by selecting a protocol — HotStuff — that has been optimized for fast intra-cluster conditions.

HotStuff is a rotating-leader-based homogeneous consensus protocol with linear communication complexity and capable of tolerating up to f Byzantine replicas out of $n = 3f + 1$. In HotStuff, replicas agree on a single block in each view, which we then pass to Damysus for inclusion into the superblock. Our choice of HotStuff over PBFT is driven by its higher throughput, in particular in data-center environments, as well as by its leader rotation scheme, which avoids the costs and complexities typically associated with view change, and its lower message complexity.

Fig. 3 shows the protocol steps of HotStuff. After entering a *new view* v, the rotated-in leader (here replica R_2) proposes a block (in *prepare*) and reaches consensus (in *pre-commit* and *commit*). Normally the leader would then execute the *decide* phase so that all replicas can execute the transaction. In ORION, the actual execution is deferred until global consensus ensures durability. Replicas can leverage this local pre-ordering to asynchronously (to local ordering) resolve local conflicts and disseminate blocks for later inclusion in the superblock. This separation of local consensus, dissemination and global consensus enables ORION to proceed through these operations independently, with a positive impact on system responsiveness and scalability.

We optimize ORION's performance by distributing the roles of local leader and cluster representative to different replicas to avoid that a single replica has to handle coordinating consensus within a cluster and managing block execution at global level.

B. Inter-cluster consistent broadcast

After executing an instance of the local consensus protocol to lock a block, each correct cluster shares its local block with the other clusters. Unlike other protocols, block dissemination is asynchronous, which separates dissemination from consensus and transaction execution. To do so, we utilize a simple broadcast protocol to consistently disseminate local blocks between clusters and their respective local replicas. For this purpose, each cluster C_i incorporates a rotating disseminator D_{C_i} that initiates the broadcast of a block of transactions previously locked through the HotSuff-based consensus protocol. The approach we employ is low-cost and effective if the disseminator is error-free. As a result, we refrain from employing view change techniques notorious for being complicated and heavy. The disseminator sends the block to $f+1$ replicas in each cluster, ensuring that at least one correct replica per cluster receives the block. Once a correct replica receives the block, it relays it its peers in the local cluster. Receiving such a block, correct replicas verify that it was properly locked in the local consensus protocol of the sending cluster before storing it in its local memory, ready to be included in the chain of superblocks.

In the event of a faulty disseminator, the rotating mechanism ensures that another healthy replica in the cluster eventually takes over the role of disseminator. Additionally, we distribute the role of disseminator within each cluster in a way that avoids potential performance bottlenecks occurring from a single replica handling the multiple responsibilities of coordinating the local consensus, broadcasting blocks, and managing the consensus protocol at the global level.

C. Damysus-based global consensus

Our system integrates a modified version of the Damysus [11] consensus protocol aligning with wide-area network characteristics for robust blockchain replication. The inter-cluster global consensus protocol adopts a three-phase communication structure where local replicas communicate with cluster representatives, which is illustrated in Fig. 4.

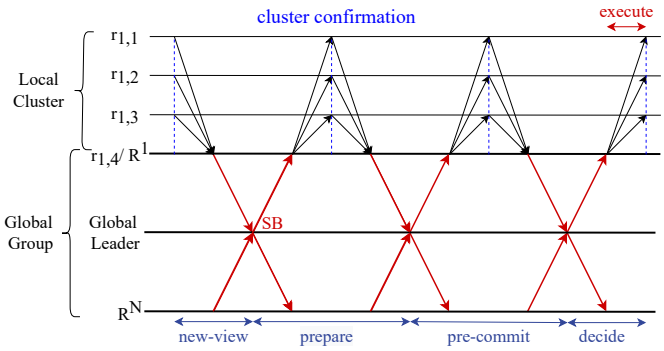

Fig. 4. Phases of the Damysus-based global consensus protocol

The global consensus layer comprises a shared overlay network involving representatives from N clusters, which participate in global consensus even when F clusters are faulty. A clear distinction is made between local leaders for local consensus and representatives for global consensus, balancing replica load and minimizing faulty replica impact.

This consensus protocol is tailored for slow connectivity and high latency, enabling representatives to reach consensus on references (hashes) rather than transmitting large amounts of transaction data or whole blocks. Coupled with asynchronous block dissemination, this design optimizes consensus by ordering fixed-size references, ensuring independence of overall and consensus throughput, while totally ordering blocks.

Unlike some hierarchical schemes [8], our global layer eliminates competition for leadership in consensus rounds through a leader rotation scheme. Global leaders generate 'superblock' proposals containing references to local blocks, resulting in a lightweight protocol. Instead of carrying full block data, PREPARE messages from the global leader contain metadata including cluster-ID, replica-ID, ViewNumber, IDs of blocks, and the hash of the preceding superblock.

Clusters confirmations. A cluster confirmation, denoted as ϕ and produced by clusters, can be expressed as $\langle h, v, h', v', \text{ph}, \vec{\sigma}_c^n \rangle$. This includes the hash values of superblocks represented by h and h', the view numbers depicted by v and v', a phase represented by ph can take on values from the set $\{\text{nv\_p}, \text{prep\_p}, \text{pcom\_p}\}$ (correspond to the new-view, prepare, and pre-commit phases), and a collection of signatures on the data $(h, v, h', v', \text{ph})$, indicated by $\vec{\sigma}_c^n = [\sigma_{c1},...,\sigma_{cn}]$.

A cluster confirmation is referred to as an n-cluster confirmation if it encompasses N signatures from separate clusters. We characterize C-combine as a function dedicated to the generation of quorum certificates from the votes of various clusters, also known as confirmations. On the other hand, C-match validates whether a sufficient number of messages have been received from different clusters during a specific phase.

Invoked functions. Cl-Prepare: This function generates a cluster confirmation as an approval or preparation vote based on the hash value of the proposed superblock from the global leader and an extension to the latter. The extension, proposed by the leader cluster, comes with a cluster signature certifying

it as the highest superblock. This function only operates when the extension is generated by the current leader cluster. Cl-Pcom: Accepts confirmations from $F+1$ clusters, checks that a quorum of clusters prepared the superblock in the current view, and produces a cluster confirmation signifying an approval or pre-commit vote. CreateClusterSign: Consists of three steps: First, the cluster leader broadcasts the message to local replicas. Second, local replicas compute a partial signature on the message. Finally, the leader combines these into a cluster certificate, proving that $2fi+1$ replicas have given their approval. ExtList: Selects the new view message for the superblock prepared at the highest view among $F+1$ clusters. It then certifies it as the highest by creating a cluster signature on it. ClusterStart: After successfully verifying the signatures of cluster confirmations from remote clusters, it sets the Ext variable. The confirmation is then converted into Ext as a qualified potential extension for the highest superblock. ClusterIterate: Checks the remaining cluster confirmations for a potential higher superblock and updates Ext accordingly. ClusterFinalize: Takes a potential extension, verifies its signatures, and then creates leader cluster signatures on it.

This lightweight global consensus protocol, encapsulated in Alg. 1, comprises Prepare, Precommit, and Decide phases, detailed in subsequent sections.

Prepare. In this phase, the global leader collects $F+1$ new-view messages from remote clusters and their representatives (Alg. 1, ln. 7). The leader selects the new-view message corresponding to the highest-viewed superblock from all $F+1$ received messages, validating its superiority via $2f+1$ local replica confirmation. Subsequently, the leader extends this highest superblock with a new one, referred to as SB (Alg. 1 ln. 9). This superblock SB is prepared using Cl-Prepare (Alg. 1 ln. 10), ensuring the transition to the next phase while generating a signature for the newly proposed superblock, also known as "cluster prepare confirmation". The global leader then disseminates the new proposal SB, the extension Ext, the cluster confirmation formed by the leader cluster to all cluster representatives. Other clusters authenticate that SB extends the superblock in Ext (Alg. 1 ln. 17) and prepare the superblock proposed by the global leader through validation from a quorum of inner-cluster replicas (Alg. 1 ln. 18).

Pre-commit. In this phase, the global leader assembles $F+1$ prepare cluster-confirmations from distant clusters and dispatches a prepare $(F+1)$-cluster confirmation to every cluster representative. The clusters invoke Cl-Pcom to store SB and secure a cluster validation that attests the new superblock SB is preserved in $2f+1$ replicas. This process yields a cluster signature, formulated via $2f+1$ local partial signatures from the same phase, ensuring that a minimum of $F+1$ clusters will keep and transmit any executed superblocks in their new-view messages. Due to the cluster confirmation functioning as Damysus's accumulator, clusters are not required to lock blocks - a mechanism employed in HotStuff during the commit phase. This locking is unnecessary in our case, as the virtual accumulator, based on reliable cluster confirmation, guarantees that the global leader only proposes superblocks that expand upon the highest prepared superblock.

Algorithm 1 ORION pseudocode

```
1:  // Protocol Initialization
2:      view ← 0 // Current view
3:      c − pubs // Cluster public keys
4:
5:  // Prepare phase
6:  As a global leader:
7:      wait for φ⃗ such that C-match ⟨φ⃗, F + 1, ⊥, view, nv_p⟩ from
    representatives
8:      Ext ← ExtList (φ)
9:      SB ← CreateLeaf (Ext.hash, list of blocks ids)
10:     φ_prep ← Cl-Prepare (H(SB), Ext)
11:     send ⟨SB, Ext, φ_prep.sign⟩ to f+1 representatives in each cluster
12:
13: As a representative:
14:     wait for ⟨SB, (view, v′, h′, F + 1, σc), σc′⟩ from the global leader
15:     Ext ← ⟨view, v′, h′, F + 1, σc⟩
16:     φ_prep_p ← ⟨H(SB), view, h′, v′, prep_p, σc′⟩
17:     Abort if ¬(VERIFY (φ_prep_p)_{c−pubs} ∧ SB > h′)
18:     send φ′ ← Cl-Prepare(H(SB), Ext) to global leader
19:
20: // Pre-commit phase
21: As a global leader:
22:     wait for φ⃗ such that C-match (φ⃗, F+1, h,view,prep_p) from represen-
    tatives
23:     send φ ← C-combine(φ⃗) to f + 1 representatives in each cluster
24:
25: As a representative:
26:     wait for ⟨h, view, h′, v′, prep_p, σc⃗^{F+1}⟩ from the global leader
27:     send φ ← Cl-Pcom (⟨h, view, h′, v′, prep_p, σc⃗^{F+1}⟩) to the global
    leader
28:
29: // Decide phase
30: As a global leader:
31:     wait for φ⃗ such that C-match (φ⃗,F+1, h,view,pcom_p) from repre-
    sentatives
32:     send φ ← C-combine(φ⃗) to f + 1 representatives in each cluster
33:
34: As a representative:
35:     wait for ⟨h, view, ⊥, ⊥, pcom_p, σc⃗^{F+1}⟩ from the global leader
36:     Multicast ⟨h, view, ⊥, ⊥, pcom_p, σc⃗^{F+1}⟩ to local replicas
37:     Abort if ¬(VERIFY ⟨h, view, ⊥, ⊥, pcom_p, σc⃗^{F+1}⟩_{c−pubs})
38:     Execute SB corresponding to h and reply to clients
39:
40: // NewView phase
41: For all replicas:
42:     When executed or timeout:
43:     (v, ph) ← (view, prep_p); view++
44:     While (v, ph) ≠ (view, nv_p) do
45:         φ ← ClusterSign(); (v, ph) ← (φ.vprep, φ.phase)
46:     End while
47:     send φ to view's leader
```

Decide. In the decide phase, the global leader collects $F+1$ pre-commit clusters-confirmations from remote representatives. Upon receiving these confirmations, the global leader issues a pre-commit $(F+1)$-cluster-confirmation to all cluster representatives. These representatives, in turn, broadcast the cluster-confirmation to their respective local replicas. This action initiates the execution of the superblock across all clusters, effectively achieving consensus on the ordering of the transactions contained within the superblock.

New-view. Each cluster utilizes timers, commencing at the inception of each view, to transition to the subsequent global view when the current one experiences delays, potentially due to a faulty representative. The new-view phase comes into

Algorithm 2 Invoked Functions

function ExtList ($\vec{\phi}$)
  $\phi_0 \leftarrow$ commitment $\phi \in \vec{\phi}$ with highest $\phi_{V_{\text{vjust}}}$
  $Ext \leftarrow$ ClusterStart (ϕ_0)
  for $\phi \in \{\phi_0\}$ **do** $Ext \leftarrow$ ClusterIterate (Ext, ϕ)
return ClusterFinalize (Ext)

function CreateClusterSign (h, h', v')
  as a leader: request cluster signature on $(h, \text{view}, h', v', \text{phase})$ from local replicas
  as a replica: compute $psig$ on $(h, \text{view}, h', v', \text{phase})$ **where** $psig \leftarrow$ SIGN $(h, \text{view}, h', v', \text{phase})$
  phase $++$
  as a leader: wait for $(n - f)$ $psig$ from different replicas, then build a σ_c on $(h, \text{view}, h', v', \text{phase})$ where $\sigma_c \leftarrow$ SIGNS $(h, \text{view}, h', v', \text{phase})_{2f_i+1}$
    return $\phi \leftarrow (h, \text{view}, h', v', \text{phase}, \sigma_c)$

function ClusterSign ()
  return $\phi \leftarrow$ CreateClusterSign ($\perp$, preph, prepv)

function Cl-Prepare (h, Ext) **where** Ext is $\langle v, v', h', F+1, \sigma_c \rangle$
  If VERIFY $(Ext)_{\text{c-pubs}} \wedge$ view $=$v $\wedge$ h $\neq \perp$ **then**
    return $\phi \leftarrow$ CreateClusterSign (h,h',v')
  end if

function Cl-Pcom (ϕ) **where** ϕ is $\langle h, v, h', v', ph, \sigma_c^{F+1} \rangle$
  If VERIFY $(\phi)_{\text{c-pubs}} \wedge$ view $=$v $\wedge$ ph $=$prep **then**
    $preph \leftarrow$ h ; $preph \leftarrow$ v ;
    return $\phi' \leftarrow$ CreateClusterSign (h,$\perp$,$\perp$)
  end if

function ClusterStart (ϕ) **where** ϕ **is** $\langle \perp, v, h', v', \text{nv\_p}, \sigma_c \rangle$
  if VERIFY $(\phi)_{\text{c-pubs}}$ **then**
  $\sigma'_c \leftarrow$ SIGN $(v, v', h', [\sigma_c.\text{clid}])_{2f+1}$
  return $Ext \leftarrow (v, v', h', [\sigma.\text{clid}], \sigma'_c)$
  end if

function ClusterIterate (Ext, ϕ) **where** Ext **is** $\langle v_1, v'_1, h_1, \vec{i}, \sigma_c \rangle$ **and** ϕ **is** $\langle \perp, v_2, h_2, \text{nv\_p}, \sigma_{c,2} \rangle$
  if $\left(\begin{array}{c} v_1 = v_2 \wedge v'_1 \geq v'_2 \wedge \sigma_{c,2}.\text{clid} \notin \vec{i} \wedge \\ VERIFY(Ext)_{\text{c-pubs}} \wedge VERIFY(\phi)_{\text{c-pubs}} \end{array} \right)$ **then**
  $\sigma'_c \leftarrow$ SIGN $\left(v_1, v'_1, h_1, \vec{i}@[\sigma_c.\text{clid}]\right)_{2fi+1}$
  return $Ext' \leftarrow (v_1, v'_1, h_1, \vec{i}@[\sigma.\text{clid}], \sigma'_c)$
  end if

function ClusterFinalize (ext) **where** ext **is** $\langle v, v', h, \vec{i}, \sigma_c \rangle$
  If VERIFY $(Ext)_{\text{c-pubs}}$ **then**
    return $Ext \leftarrow (v, v', h, |\vec{i}|, \sigma_c)$ **where** $\sigma_c \leftarrow$ SIGN $\left(v, v', h, |\vec{i}|\right)_{2f_i+1}$
  end if

play either post completion of the current view - implying successful execution of the decide phase - or upon expiration of the timer initiated at the view's start. This mechanism is activated under certain conditions, such as a representative's inability to gather sufficient votes to inaugurate a new view or local replicas' disagreement on the representative's proposal, causing a timeout and triggering the start of a new view. This phase sees clusters progress their views and provide their votes to the global leader of the new view.

VI. Compositional Safety and Liveness

This section establishes the safety and liveness of Orion by proving a more general compositionality result about hierarchical consensus with cluster confirmation and global rotation.

We assume the standard partial synchrony model and prove liveness after GST (see Section III). More precisely, we will show that if the subprotocols executed in the clusters are live and safe, then the composition using cluster confirmation and global rotation repeatedly returns the overall system into a state where these properties extend to the global case.

We identify points of progress in the subprotocols (e.g., the dissemination of a message, reaching consensus about a client request, or appending a block of transactions) and consider subprotocols that advance to their respective next progress point, provided they remain for a given amount of time t in a configuration where such progress is possible. Our proof then establishes that (1) Byzantine configuration cannot jeopardize the state of subprotocols as a whole (i.e., they can only jeopardize the Byzantine replica's state), and (2) global rotation repeatedly returns subprotocols into sufficiently healthy configurations (with at most f Byzantine replicas or up to F representatives that may itself either be Byzantine or that represent crashed clusters) and remains there for at least time t. Then, because global rotation repeatedly returns to such healthy state and configuration pairs where progress is possible, the local properties of the protocol extend to the system as a whole as far as safety and liveness are concerned.

Definition 1 (Local safety/liveness)**.** *We say clusters are* locally live/safe *if the subprotocols they execute exhibit these properties.*

Definition 2 (t-live)**.** *A configuration of the protocol is* t-live *if given a correct overall state it is capable of advancing to the next progress point and correct overall state, provided the protocol remains in this configuration for at least time t.*

Definition 3 ((t-)rotation safe)**.** *We say a global group protocol is* rotation safe *if its rotation scheme eventually and repeatedly returns to a configuration where at most F representatives are faulty. It is* t-rotation safe *if it remains in such a configuration for at least time t.*

Notice that rotation safety implies that the rotation cannot only be triggered by representatives, as they might be all faulty and not trigger rotation. State transitions of a subprotocol consist of delivery of a message, its analysis and possibly the sending of further messages to trigger further state transitions.

Definition 4 (Cluster-confirmed state transition)**.** *State transitions are executed under* cluster confirmation *if the delivered message is accepted only if $n - f$ replicas of the represented cluster approve this message (e.g., by signing it).*

To obtain cluster confirmation, representatives have to first consult their clusters' replicas, before they can send messages that trigger the execution of state transitions under this confirmation. We require protocols to advance the state they plan to change as part of obtaining such a confirmation. In particular, we require cluster confirmation to be sequenced.

Lemma 1 (Cluster confirmation retains correct states)**.** *Exclusively executing state transitions under cluster confirmation*

retains correct state in healthy replicas.

Proof. Let s be the state of a healthy replica, to confirm a state-transition triggering message, representatives must provide the state to modify s' and the message m to trigger the transition from s' to s''. *Case* $s = s'$: In this case, the replica is able to confirm immediately whether m is appropriate, by checking the conditions of the protocol. Local safety then ensures that s'' is safe as well. *Case* $s \neq s'$: In this case, the replica accepts s' only if $n - f$ replicas confirmed the transition that lead to s'. If so, it can approve m as in the previous case. Otherwise, it refuses approval. Since $n-f > f$ approvals are required to confirm a state transition, no faulty replica can jeopardize the state held by a healthy replica. □

Theorem 1 (Compositional liveness)**.** *If a protocol is t-live if at most F out of N of its group members are faulty, it remains t-live if t-rotation safety is guaranteed.*

Proof. After GST, a rotation safe protocol eventually reaches a configuration with at most F faulty replicas and a t-rotation safe protocol remains in that configuration for at least a time t. Lemma 1 implies that the state of the at least $N-F$ healthy replicas is correct and t-liveness ensures progress. Rotation safety reaches healthy configuration repeatedly and within a bounded amount of time, which ensures liveness. □

Theorem 2 (Compositional safety)**.** *If a protocol is safe provided at most F out of N replicas are faulty, then it is also safe in a t-rotation safe setting, provided all state transitions are cluster confirmed.*

Proof. Healthy replicas in the local cluster will approve state transitions only if they have learned about all previous cluster confirmations and only if the transition meets the protocol's safety condition. Since $n-f > f$, representatives only receive confirmation if at least one healthy replica agrees. This is equivalent to a trusted component that is exclusively connected to the representative and that provides a token that cannot be forged, stating that the protocol's safety checks succeeded.

Since Byzantine replicas cannot jeopardize the state of healthy replicas (Lemma 1) and since state transitions are exclusively executed under cluster confirmation, no transition can be made that would jeopardize the state of a healthy replica. Healthy replicas therefore retain the healthy state of the protocol, which ensures safety. □

Notice that Byzantine representatives may still alter their internal state and even reveal results to clients, but these revelations did not receive cluster confirmation and can therefore be detected by the client as possibly faulty.

There is one fundamental difference where cluster confirmation diverges from trusted components. While the latter may retain secrets (such as a symmetric key), this is only possible by means of secret-sharing [41], [42], since Byzantine replicas of a cluster might reveal any secrets they know.

ORION uses Damysus and Hotstuff as subprotocols. Their safety and liveness follows from the proofs in Decouchant et al. [11] and Yin et al. [14]. The extension to t-liveness is easy to see given both protocols progress through a fixed number of steps to commit a block and that transmission and computation times are bounded after GST. The same holds for the broadcast protocol we use. Our protocol rotates representatives either after a block has been added (i.e., after progress is made) or after the individual replicas in the clusters time out, triggering a global viewchange and rotation of representatives. With a timeout large enough to make progress after GST, our protocol fulfills the pre-condition for t-rotation safety.

What remains to be seen is that our rotation scheme eventually and repeatedly selects a configuration with at most F faulty representatives. In a crashed cluster all replicas are faulty, whereas in a non-crashed cluster at most f are (and at least $2f + 1$ are healthy). We can therefore pessimistically assume F clusters crashed. However, since our rotation scheme iterates through all combinations of replicas from the individual clusters it will also pass configurations where it selects healthy replicas as representatives from all $N-F$ non-crashed clusters and repeatedly so. We can therefore conclude that our protocol is safe and live.

VII. PERFORMANCE EVALUATION

We have evaluated ORION against hierarchical (GeoBFT [4]) and non-hierarchical (PBFT [15], HotStuff [14]) state-of-the-art consensus protocols. Evaluations utilized AWS EC2 replicas of different geographical regions and used a single t2.xlarge instance at each replica. We consider replicas of a region to form a cluster and associate this cluster as the preferred one to clients of that region. We distribute clients and replicas uniformly across regions. We report average performance over ten repetitions, each executed for 30 views.

In all experiments, we collect 400 transactions to form a block and use 32 Byte hashes to link blocks and to refer to a block from a superblock. That is, the size of superblocks is $32B$ times the number of blocks it refers to plus one (to link to the previous superblock). Experiments warm up for 60s.

A. Effects of Large-Scale Geographic Deployment

We start by investigating the effects of large-scale geographic deployment. For that, we scale the number F of clusters we tolerate to crash and hence the number of clusters $N = 2F+1$, while keeping their size fixed (to $n = 3f+1 = 10$ replicas, i.e., clusters can tolerate up to $f = 3$ Byzantine replicas). Fig. 5 and 6 show the throughput and latency of ORION relative to the baseline protocols. The x-axis denotes the number of clusters N, which we distribute to North Virginia, Ohio, North California, Mumbai, Singapore, Sydney, Frankfurt, Ireland, London, Central Canada, and São Paulo.

ORION outperforms GeoBFT's throughput (by 20% on average), albeit at the cost of increasing the latency for handling client requests until notification after the request is durable (by 31% on average) and while outperforming non-hierarchical protocols in both dimensions. This throughput-for-latency tradeoff is achieved by distributing and reaching global agreement independently from the local agreement. ORION

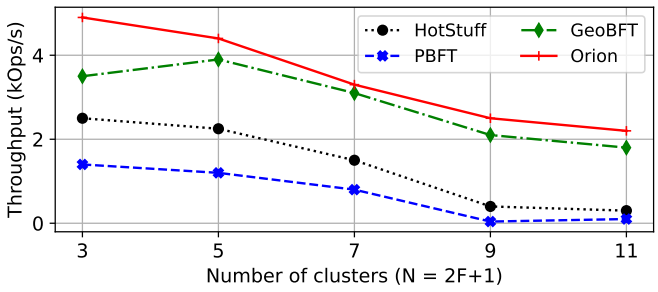

Fig. 5. Throughput depending on the number N of clusters (each cluster contains 10 replicas, and tolerates $f = 3$ Byzantine replicas).

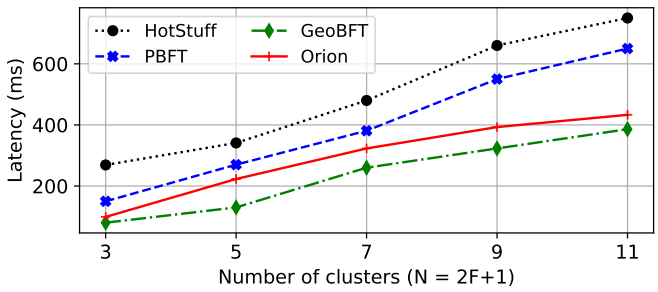

Fig. 6. Latency depending on the number N of clusters (each cluster contains 10 replicas, and tolerates $f = 3$ Byzantine replicas).

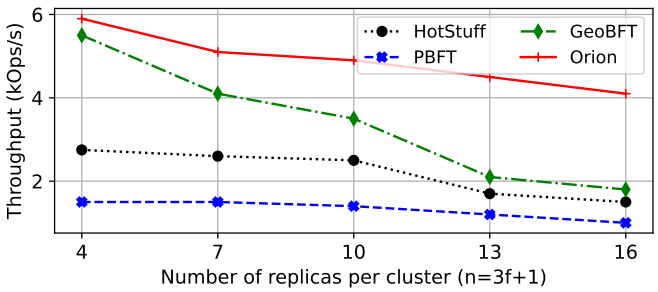

Fig. 7. Throughput as a function of the number of replicas in a cluster (with $N = 3$ clusters).

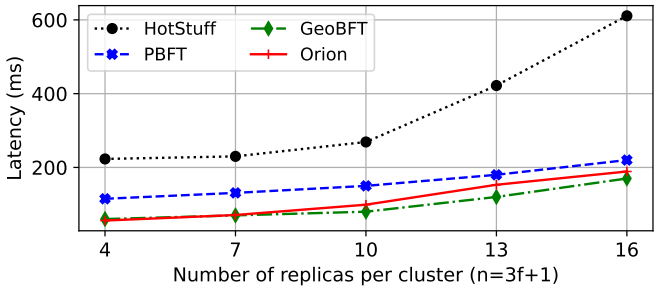

Fig. 8. Latency depending on the number of replicas in a cluster (with $N = 3$ clusters).

outperforms GeoBFT in terms of throughput under faults as it relies less on inter-cluster communications.

B. Influence of Increasing Local Clusters Size

In the second part of our analysis we focus on the performance impact of increasing the size of local clusters. For these experiments, we fix the number of clusters to $N = 3$ (i.e., we tolerate $F = 1$ cluster crash), distributed over Ohio, Sydney, and London, while increasing the number of Byzantine replicas that each cluster can tolerate from $f = 1$ to $f = 5$ and cluster sizes from $n = 4$ to $n = 16$.

Throughput and latency results are shown in Fig. 7 and 8, respectively. Again, ORION outperforms GeoBFT in terms of throughput (by 63% on average) and non-hierarchical protocols in terms of both throughput and latency. However, as local clusters grow in size, the latency advantage of GeoBFT drops to 11% on average. ORION closes the gap to GeoBFT because GeoBFT requires global messages proportional to f, while this number remains constant in our protocol (independent of the cluster sizes, as only one representative communicates globally). Additionally, every cluster has to disseminate certificates to the associated $f + 1$ remote replicas, which grow as well in size as f increases and hence n grows. Our protocol trades performance improvements in global communication for a slight increase in local communication, since each global step requires reaching $2f+1$ local replicas for cluster confirmation, which explains the increase in latency seen in Fig. 8.

Overall, ORION demonstrates that compositionally constructed protocols perform well and can sometimes even outperform specific hierarchical protocols. Compositionally also significantly reduces development costs.

VIII. CONCLUSION

We introduced ORION, a novel, compositionally-constructed, hierarchical Byzantine fault-tolerant consensus protocol that addresses the scalability and performance challenges of blockchain systems. We demonstrated that ORION achieves superior throughput compared to existing protocols, like PBFT, HotStuff and GeoBFT, albeit by marginally trading off latency. ORION is based on a novel compositionality result, enabling the use of hybrid protocols at global level without necessitating dedicated trusted components. Looking ahead, we envisage further refinement of ORION through expanded evaluations involving diverse clusters, scenarios, and speculative execution to improve latency. Our findings have broad implications for enhancing blockchain and distributed systems' performance, particularly in geo-distributed clustered networks, paving the way for more efficient and reliable distributed ledger technologies.

REFERENCES

[1] C. Berger, S. Schwarz-Rüsch, A. Vogel, K. Bleeke, L. Jehl, H. P. Reiser, and R. Kapitza, "Sok: Scalability techniques for BFT consensus," *arXiv preprint arXiv:2303.11045*, 2023.

[2] Y. Amir, C. Danilov, D. Dolev, J. Kirsch, J. Lane, C. Nita-Rotaru, J. Olsen, and D. Zage, "Steward: Scaling byzantine fault-tolerant replication to wide area networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 7, no. 1, pp. 80–93, 2008.

[3] Q. T. Thai, J.-C. Yim, T.-W. Yoo, H.-K. Yoo, J.-Y. Kwak, and S.-M. Kim, "Hierarchical byzantine fault-tolerance protocol for permissioned blockchain systems," *The Journal of Supercomputing*, vol. 75, no. 11, pp. 7337–7365, 2019.

[4] S. Gupta, S. Rahnama, J. Hellings, and M. Sadoghi, "Resilientdb: Global scale resilient blockchain fabric," *Proc. VLDB Endow.*, vol. 13, no. 6, pp. 868–883, 2020.

[5] W. Jiang, X. Wu, M. Song, J. Qin, and Z. Jia, "A scalable byzantine fault tolerance algorithm based on a tree topology network," *IEEE Access*, 2023.

[6] L. Zhang and Q. Li, "Research on consensus efficiency based on practical byzantine fault tolerance," in *2018 10th international conference on modelling, identification and control (ICMIC)*. IEEE, 2018, pp. 1–6.

[7] L. Feng, H. Zhang, Y. Chen, and L. Lou, "Scalable dynamic multi-agent practical byzantine fault-tolerant consensus in permissioned blockchain," *Applied Sciences*, vol. 8, no. 10, p. 1919, 2018.

[8] H. Qushtom, J. Mišić, V. B. Mišić, and X. Chang, "A high performance two-layer consensus architecture for blockchain-based IoT systems," *Peer-to-Peer Networking and Applications*, pp. 1–13, 2022.

[9] Y. Li, L. Qiao, and Z. Lv, "An optimized byzantine fault tolerance algorithm for consortium blockchain," *Peer-to-Peer Networking and Applications*, vol. 14, pp. 2826–2839, 2021.

[10] G. S. Veronese, M. Correia, A. N. Bessani, L. C. Lung, and P. Verissimo, "Efficient byzantine fault-tolerance," *IEEE Transactions on Computers*, vol. 62, no. 1, pp. 16–30, 2011.

[11] J. Decouchant, D. Kozhaya, V. Rahli, and J. Yu, "DAMYSUS: streamlined BFT consensus leveraging trusted components," in *Proceedings of the 17th European Conference on Computer Systems*, 2022, pp. 1–16.

[12] S. Gupta, S. Rahnama, S. Pandey, N. Crooks, and M. Sadoghi, "Dissecting bft consensus: In trusted components we trust!" in *Proceedings of the 18th European Conference on Computer Systems*, ser. EuroSys '23, New York, NY, USA, 2023, p. 521–539.

[13] A. Bessani, M. Correia, T. Distler, R. Kapitza, P. Esteves-Verissimo, and J. Yu, "Vivisecting the dissection: On the role of trusted components in bft protocols," 2023.

[14] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "HotStuff: BFT consensus with linearity and responsiveness," in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, 2019, pp. 347–356.

[15] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Transactions on Computer Systems (TOCS)*, vol. 20, no. 4, pp. 398–461, 2002.

[16] A. Bessani, J. Sousa, and E. E. Alchieri, "State machine replication for the masses with BFT-smart," in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 2014, pp. 355–362.

[17] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong, "Zyzzyva: speculative byzantine fault tolerance," in *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, 2007, pp. 45–58.

[18] A. Haeberlen, P. Kouznetsov, and P. Druschel, "Peerreview: Practical accountability for distributed systems," *ACM SIGOPS operating systems review*, vol. 41, no. 6, pp. 175–188, 2007.

[19] B. Chun, P. Maniatis, S. Shenker, and J. Kubiatowicz, "Attested append-only memory: making adversaries stick to their word," 2007, pp. 189–204.

[20] D. Levin, J. R. Douceur, J. R. Lorch, and T. Moscibroda, "TrInc: Small trusted hardware for large distributed systems," 2009, pp. 1–14.

[21] X. Xu, D. Zhu, X. Yang, S. Wang, L. Qi, and W. Dou, "Concurrent practical byzantine fault tolerance for integration of blockchain and supply

[22] L.-e. Wang, Y. Bai, Q. Jiang, V. C. Leung, W. Cai, and X. Li, "Beh-raft-chain: a behavior-based fast blockchain protocol for complex networks," *ACM Transactions on Internet Technology (TOIT)*, vol. 21, no. 1, pp. 1–17, 2021.
IEEE Transactions on Network Science and Engineering, vol. 8, no. 2, pp. 1154–1166, 2020.

[23] F. Wen, L. Yang, W. Cai, and P. Zhou, "Dp-hybrid: a two-layer consensus protocol for high scalability in permissioned blockchain," in *Blockchain and Trustworthy Systems: Second International Conference, BlockSys 2020, Dali, China, August 6–7, 2020, Revised Selected Papers 2*. Springer, 2020, pp. 57–71.

[24] W. Li, C. Feng, L. Zhang, H. Xu, B. Cao, and M. A. Imran, "A scalable multi-layer PBFT consensus for blockchain," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 5, pp. 1146–1160, 2020.

[25] M. Javad Amiri, Z. Lai, L. Patel, B. Thau Loo, E. Lo, and W. Zhou, "Saguaro: Efficient processing of transactions in wide area networks using a hierarchical permissioned blockchain," *arXiv e-prints*, pp. arXiv–2101, 2021.

[26] G. Danezis, L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman, "Narwhal and tusk: a dag-based mempool and efficient BFT consensus," in *Proceedings of the Seventeenth European Conference on Computer Systems*, 2022, pp. 34–50.

[27] A. Spiegelman, N. Giridharan, A. Sonnino, and L. Kokoris-Kogias, "Bullshark: Dag BFT protocols made practical," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 2705–2718.

[28] L. Baird, "The swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance," *Swirlds Tech Reports SWIRLDS-TR-2016-01, Tech. Rep*, vol. 34, 2016.

[29] A. Gagol, D. Lesniak, D. Straszak, and M. Swietek, "Aleph: Efficient atomic broadcast in asynchronous networks with byzantine nodes," in *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, 2019, pp. 214–228.

[30] Y. Sompolinsky and A. Zohar, "Secure high-rate transaction processing in bitcoin," in *International conference on financial cryptography and data security*. Springer, 2015, pp. 507–527.

[31] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The honey badger of BFT protocols," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 31–42.

[32] T. Crain, C. Natoli, and V. Gramoli, "Red belly: a secure, fair and scalable open blockchain," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 466–483.

[33] C. Stathakopoulou, T. David, and M. Vukolic, "Mir-BFT: High-throughput BFT for blockchains," *arXiv preprint arXiv:1906.05552*, 2019.

[34] K. Korkmaz, J. Bruneau-Queyreix, S. B. Mokhtar, and L. Réveillère, "Alder: Unlocking blockchain performance by multiplexing consensus protocols," in *2022 IEEE 21st International Symposium on Network Computing and Applications (NCA)*, vol. 21. IEEE, 2022, pp. 9–18.

[35] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Proceedings of the 26th symposium on operating systems principles*, 2017, pp. 51–68.

[36] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 17–30.

[37] M. Zamani, M. Movahedi, and M. Raykova, "Rapidchain: Scaling blockchain via full sharding," in *CCS*, 2018.

[38] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *Journal of the ACM (JACM)*, vol. 35, no. 2, pp. 288–323, 1988.

[39] G. Bracha, "Asynchronous byzantine agreement protocols," *Information and Computation*, vol. 75, no. 2, pp. 130–143, 1987.

[40] S. Bonomi, J. Decouchant, G. Farina, V. Rahli, and S. Tixeuil, "Practical byzantine reliable broadcast on partially connected networks," in *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2021, pp. 506–516.

[41] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[42] R. Vassantlal, E. Alchieri, B. Ferreira, and A. Bessani, "Cobra: Dynamic proactive secret sharing for confidential BFT services," in *2022 IEEE symposium on security and privacy (SP)*. IEEE, 2022, pp. 1335–1353.