Brief Announcement: Reconfigurable Heterogeneous Quorum Systems

Xiao Li
University of California, Riverside, USA

Mohsen Lesani
University of California, Santa Cruz, USA

Abstract

In contrast to proof-of-work replication, Byzantine quorum systems maintain consistency across replicas with higher throughput, modest energy consumption, and deterministic liveness guarantees. If complemented with heterogeneous trust and open membership, they have the potential to serve as blockchains backbone. This paper presents a general model of heterogeneous quorum systems where each participant can declare its own quorums, and captures the consistency, availability and inclusion properties of these systems. In order to support open membership, it then presents reconfiguration protocols for heterogeneous quorum systems including joining and leaving of a process, and adding and removing of a quorum, and further, proves their correctness in the face of Byzantine attacks. The design of the protocols is informed by the trade-offs that the paper proves for the properties that reconfigurations can preserve. The paper further presents a graph characterization of heterogeneous quorum systems, and its application for reconfiguration optimization.

2012 ACM Subject Classification Computer systems organization → Reliability; Computer systems organization → Availability; Computing methodologies → Distributed algorithms

Keywords and phrases Quorum Systems, Reconfiguration, Heterogeneity

Digital Object Identifier 10.4230/LIPIcs.DISC.2024.37

1 Introduction

Banks have been traditionally closed; only established institutions could hold accounts and execute transactions. With regulations in place, this centralized model can preserve the integrity of transactions. However, it makes transactions across these institutions costly and slow; further, it keeps the power in the hands of a few. In pursuit of decentralization, Bitcoin [17] provided open membership: any node can join the Bitcoin network, and validate and process transactions. It maintains a consistent replication of an append-only ledger, called the blockchain, on a dynamic set of global hosts including potentially malicious ones. However, it suffers from a few drawbacks: low throughput, high energy consumption, and only probabilistic guarantees of commitment [9, 10].

Maintaining consistent replication in the presence of malicious processes has been the topic of Byzantine replicated systems for decades. PBFT [5] and its numerous following variants [21, 16, 22, 19, 2, 20] can maintain consistent replication when the network size is at least three times the size of potentially Byzantine coalitions, have higher throughput than Bitcoin, have modest energy consumption, give participants equal power, and provide deterministic liveness guarantees. Unfortunately, however, their quorums are uniform and their membership is closed. Their trust preferences, *i.e.*, the quorums of processes are fixed and homogeneous across the network. Further, their set of participants are fixed; thus, in contrast to proof-of-work replication that provides permissionless blockchains, classical Byzantine replication only provides permissioned blockchains.

Can the best of both worlds come together? Can we keep the consistency, throughput, modest energy consumption and equity of Byzantine replicated systems, and bring heterogeneous trust [6, 4, 1] and *open membership* to it? Openness challenges classical assumptions. With global information about the processes and their quorums, classical quorum systems could be configured at the outset to satisfy consistency and availability properties. However, open quorum systems relinquish global information as processes specify their own quorums, and can further join, leave, and reconfigure their quorums. As the other processes may be unaware of these changes, consistency and availability may be violated after and even while these reconfigurations happen.

Projects such as Ripple [18] and Stellar [15] pioneered, and follow-up research [14, 13, 8, 3] moved towards this goal, and presented quorum systems where nodes can specify their own quorums, and can join and leave. In fact, the Stellar network has a high churn. In previous works, the consistency of the network is either assumed to be maintained by user preferences or a structured hierarchy of nodes, is provided only in divided clusters of processes, or can be temporarily violated and is periodically checked across the network. Reconfigurations can compromise the consistency or availability of the replicated system. The loss of consistency can be the antecedent to a fork and double-spending. An important open problem is *reconfiguration protocols for heterogeneous quorum systems with provable security guarantees.* The protocols are expected to avoid external central oracles, or downtime.

In this paper, we first present a *general model of heterogeneous quorum systems* where each process declares its individual set of quorums, and then formally capture the properties of these systems: consistency, availability and inclusion. We then consider the *reconfiguration* of heterogeneous quorum systems: joining and leaving of a process, and adding and removing of a quorum. To cater for the protocols such as broadcast and consensus that use the quorum system, the reconfiguration protocols are expected to preserve the above properties.

The safety of consensus naturally relies on the *consistency (or quorum intersection)* property: every pair of quorums intersect at a well-behaved process. Intuitively, if an operation communicates with a quorum, and a later operation communicates with another quorum, only a well-behaved process in their intersection can make the second aware of the first. A quorum system is *available* for a process if it has a well-behaved quorum for that process. Intuitively, the quorum system is responsive to that process through that quorum. The less known property is *quorum inclusion.* Roughly speaking, every quorum should include a quorum of each of its members. This property trivially holds for homogeneous quorum systems where every quorum is uniformly a quorum of all its members, but should be explicitly maintained for heterogeneous quorum systems. We show that quorum inclusion interestingly lets processes in the included quorum make local decisions while preserving properties of the including quorum. We precisely capture and illustrate these properties.

We then present *quorum graphs*, a graph characterization of heterogeneous quorum systems with the above properties. It is known that strongly connected components of a graph form a directed acyclic graph (DAG). We prove that a quorum graph has only one *sink component*, and preserving consistency reduces to preserving quorum intersections in this component. This fact has an important implication for optimization of reconfiguration protocols. Any change outside the sink component preserves consistency, and therefore, can avoid synchronization with other processes. Thus, we present a decentralized *sink discovery protocol* that can find whether a process is in the sink.

In addition to consistency, availability and inclusion, reconfiguration protocols are expected to preserve *policies.* Each process declares its own trust policy: it specifies the quorums that it trusts. In particular, it does not trust strict subsets of its individual quorums. Thus, a policy-preserving reconfiguration should not shrink any quorum. We present a *join protocol* that preserves all the above properties. We present *trade-offs* for the properties that the leave, remove and add reconfiguration protocols can preserve. We show that there is no *leave or remove protocol* that can preserve both the policies and availability. Thus, we present two protocols: a protocol that preserves policies, and another that preserves availability. Both preserve consistency and inclusion. Then, we show that there is no *add protocol* that can preserve both the policies and consistency. Therefore, since we never sacrifice consistency, we present a protocol that preserves all properties except the policies.

We observe that under reconfiguration, *quorum inclusion is critical* to preserve not only availability but also consistency. Sometimes, reconfigurations can only eventually reconstruct inclusion, but can preserve *weaker notions of inclusion* that are sufficient to preserve consistency and availability. We capture these notions, prove that they are preserved, and use them to prove that the other properties are preserved.

In summary, this project makes the following contributions.

- A graph characterization of heterogeneous quorum systems, and its application to optimize reconfiguration and a sink discovery protocol
- Trade-offs between reconfiguration guarantees
- Reconfiguration protocols for joining and leaving of a process, and adding and removing of a quorum, and their proofs of correctness

In this short paper, we present an overview of the leave protocol. The full paper [12] presents all the above contributions more coherently.

2 Quorum Systems

Processes. A quorum system is hosted on a set of processes $\mathcal{P}$. In each execution, $\mathcal{P}$ is partitioned into *Byzantine* $\mathcal{B}$ and *well-behaved* $\mathcal{W} = \mathcal{P} \setminus \mathcal{B}$ processes. Well-behaved processes follow the given protocols; however, Byzantine processes can deviate from the protocols arbitrarily. Furthermore, a well-behaved process does not know the set of well-behaved processes $\mathcal{W}$ or Byzantine processes $\mathcal{B}$. The active processes $\mathcal{A} \subseteq \mathcal{P}$ are the current members of the system. As we will see, quorum systems can be reconfigured, and the active set can change: processes can join and the active set grows, and conversely, processes can leave, and the active set shrinks. We consider partially synchronous networks [7], *i.e.*, if both the sender and receiver are well-behaved, the message will be eventually delivered within a bounded delay after an unknown GST (Global stabilization Time). Processes can exchange messages on authenticated point-to-point links.

Individual Quorums. Processes can have different trust assumptions: trust is a subjective matter, and therefore, heterogeneous. We capture a heterogeneous model of quorum systems where each process can specify its individual set of quorums.

An *individual quorum* q of a process p is a non-empty subset of processes in $\mathcal{P}$ that p trusts to collectively perform an operation. Every quorum of a process p naturally contains p itself. (However, this is not necessary for any theorem in this paper.) By the above definition, any superset of a quorum of p is also a quorum of p. Thus, the set of quorums of p is superset-closed and has minimal members. (Consider a set of sets $S = \{\overline{s}\}$. We say that S is superset-closed, if any superset s' of any member s of S is a member of S as well.) A process p doesn't need to keep any quorum other than its minimal quorums: any of its other quorums include extra processes that p can perform operations without. Thus, we consider only the *(individual) minimal quorums* of p. Any superset of such a quorum is a *quorum* for p. We denote a set of quorums as Q. We denote the union of a set of quorums Q as $\cup Q$.

▶ **Definition 1** (Quorum System). *A heterogeneous quorum system (HQS)* $\mathcal{Q}$ *maps each* *active process to a non-empty set of individual minimal quorums.*

The mapping models the fact that each process has only a local view of its own individual minimal quorums. Further, since the behavior of Byzantine processes can be arbitrary, we leave their individual quorums unspecified.

The *consistency, availability and inclusion* properties are expected to be provided by a quorum system, and maintained by a reconfiguration protocol. We adapt consistency and availability for HQS [11], and define the new notion of inclusion.

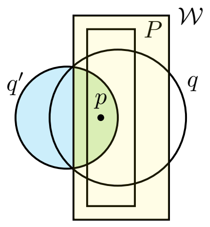

Figure 1 Quorum inclusion of q for P. Process p is a member of q that falls inside P, and q' is a quorum of p. Well-behaved processes of q' (shown as green) should be a subset of q.

▶ **Definition 2** (Consistency, Quorum Intersection). *A quorum system $\mathcal{Q}$ is consistent (i.e., has quorum intersection) at a set of well-behaved processes P iff the quorums of well-behaved processes have quorum intersection at P, i.e., $\forall p, p' \in \mathcal{W}.\ \forall q \in \mathcal{Q}(p),\ q' \in \mathcal{Q}(p').\ q \cap q' \cap P \neq \emptyset$.*

▶ **Definition 3** (Availability). *A quorum system is available for processes P at a set of well-behaved processes P' iff every process in P has at least a quorum that is a subset of P'. We say that a quorum system is available* inside *P iff it is available for P at P.*

▶ **Definition 4** (Blocking Set). *A set of processes P is a blocking set for a process p (or is p-blocking) iff P intersects every quorum of p.*

▶ **Lemma 5.** *In every quorum system that is available inside a set of processes P, every blocking set of every process in P intersects P.*

▶ **Definition 6** (Quorum inclusion). *Consider a quorum system $\mathcal{Q}$, and a subset P of its well-behaved processes. A quorum q is quorum including for P iff for every process p in the intersection of q and P, there is a quorum q' of p such that well-behaved processes of q' are a subset of q, i.e., $including(q, P) \coloneqq \forall p \in q \cap P.\ \exists q' \in \mathcal{Q}(p).\ q' \cap \mathcal{W} \subseteq q$. A quorum system $\mathcal{Q}$ is quorum including for P iff every quorum of well-behaved processes of $\mathcal{Q}$ is quorum including for P, i.e., $\forall p \in \mathcal{W}.\ \forall q \in \mathcal{Q}(p).\ including(q, P)$.*

The set P is often implicitly the set of all well-behaved processes $\mathcal{W}$. Quorum inclusion was inspired by and weakens quorum sharing [14]. Quorum sharing requires conditions on the Byzantine processes in q and q', and is too strong to maintain. We presented quorum inclusion that is weaker than quorum sharing. It requires a quorum q' only for well-behaved processes of q, and requires only the well-behaved subset of q' to be a subset of q. We will see in Section 3 that quorum inclusion is sufficient to support quorum intersection.

▶ **Definition 7** (Outlived). *A quorum system $\mathcal{Q}$ is outlived for a set of well-behaved processes $\mathcal{O}$ iff (1) $\mathcal{Q}$ is consistent at $\mathcal{O}$, (2) available inside $\mathcal{O}$, and (3) quorum including for $\mathcal{O}$.*

The safety and liveness properties of outlived processes outlive Byzantine attacks, hence the name. The protocols reconfigure an outlived quorum system into another.

3 Leave Protocol

Based on the trade-offs presented in the full paper, we present the availability-preserving and consistency-preserving protocols (AC protocols) in Algorithm 1. We then intuitively explain how it preserves the properties of the quorum system.

Algorithm 1 Leave Protocol

```
1  Implements: Leave and Remove
2    request : $Leave$
3    response : $LeaveComplete$ | $LeaveFail$
4  Variables:
5    $Q$  ▷ Individual minimal quorums of self
6    $tomb : 2^{\mathcal{P}} \leftarrow \emptyset$
7    $(in\text{-}sink : \text{Boolean}, F : 2^{\mathcal{P}}) \leftarrow Discovery(Q)$
8  Uses:
9    $tob$ : TotalOrderBroadcast
10   $apl : (\cup Q) \cup F \mapsto$ AuthPPoint2PointLink
11 upon request $Leave$
12     if $in\text{-}sink$ then
13         if $\forall q_1, q_2 \in Q, (q_1 \cap q_2) \setminus \{\mathbf{self}\}$ is
             self-blocking then
14             $tob$ request $Check(\mathbf{self}, Q)$
15         else
16             response $LeaveFail$
17     else
18         response $LeaveComplete$
19         $apl(p)$ request $Left(\mathbf{self})$
             for each $p \in F$
20 upon response $tob$, $Check(p', Q')$
21     if $\exists q_1, q_2 \in Q'. (q_1 \cap q_2) \setminus (\{p'\} \cup tomb)$
         is not $p'$-blocking then
22         if $p' = \mathbf{self}$ then
23             response $LeaveFail$
24     else
25         $tomb \leftarrow tomb \cup \{p'\}$
26         if $p' = \mathbf{self}$ then
27             response $LeaveComplete$
28             $apl(p)$ request $Left(\mathbf{self})$
                 for each $p \in F$
29 upon response $apl(p)$, $Left(p)$
30     $Q \leftarrow \{q \setminus \{p\} \mid q \in Q\}$
```

Variables and sub-protocols. Each process keeps its own set of individual minimal quorums Q. It also keeps the set *tomb* that records the processes that might have left. The full paper presents an optimization opportunity for the coordination needed to preserve consistency: when the quorum system has quorum sharing, only processes in the sink component need coordination. Therefore, each process stores whether it is in the sink component as the *in-sink* boolean, and its follower processes (*i.e.*, processes that have this process in their quorums) as the set F. (The sink information is just used for an optimization, and the protocol can execute without it.)

The protocol uses a total-order broadcast *tob*, and authenticated point-to-point links *apl* (to processes in the quorums Q and followers F). Total-order broadcast provides a broadcast interface on top of consensus [15, 14, 8, 11]. The consensus and total-order broadcast abstractions [11] require quorum intersection for safety, and quorum availability and inclusion for liveness. As we will show, the reconfiguration protocols preserve both of these properties for outlived quorum systems. We note that if a protocol naively uses *tob* to globally order and process reconfigurations, then since each process only knows its own quorums, it cannot independently check if the properties of the quorum system are preserved.

Protocol. When a process requests to leave (at L. 11), it first checks whether it is in the sink component (at L. 12). If it is not in the sink, then it can apply the optimizations that are shown with the blue color. The process can simply leave without synchronization (at L. 18); it only needs to inform its follower set so that they can preserve their quorum availability. It sends a *Left* message to its followers (at L. 19). Every well-behaved process that receives the message (at L. 29) removes the sender from its quorums (at L. 30). If the quorum system does not have quorum sharing or the sink information is not available, the protocol can be conservative (remove the blue lines) and always perform the coordination that we will consider next.

On the other hand, when the requesting process is in the sink component, its absence can put quorum intersection in danger. Therefore, it first locally checks a condition (at L. 13). The check is just an optimization not to attempt leave requests that are locally known to fail. We will consider this condition in the next subsection. If the check fails, the leave request fails (at L. 16). If the local check passes, the process broadcasts a *Check* request together with its quorums (at L. 14). If processes receive and check concurrent leave requests in different orders, they may concurrently approve leave requests for all processes in a quorum intersection. Therefore, a total-order broadcast *tob* is used to enforce a total order for processing of *Check* messages. When a process receives a *Check* request with a set of quorums Q, it locally checks a condition for Q (at L. 21). This check is similar to the check above but is repeated in the total order of deliveries by the *tob*. If the condition fails, the leave request fails (at L. 23). If it passes, the leaving process is added to the *tomb* set (at L. 25), and the leaving process informs its followers, and leaves (at lines 27 and 28). Let's now consider the condition and see how it preserves quorum intersection and inclusion.

Quorum Intersection. Let us first see an intuitive explanation of the condition, and why it preserves quorum intersection. We assume that the quorum system is outlived: there is a set of processes $\mathcal{O}$ such that the quorum system has quorum intersection at $\mathcal{O}$, quorum inclusion for $\mathcal{O}$, and quorum availability inside $\mathcal{O}$. As shown in Figure 2, consider well-behaved processes p_1 and p_2 with quorums q_1 and q_2 respectively, and let p^* be a process at the intersection of q_1 and q_2 in $\mathcal{O}$. The goal is to allow p^* to leave only if the intersection of q_1 and q_2 contains another process in $\mathcal{O}$. By the quorum inclusion property, p^* should have quorums q_1^* and q_2^* such that their well-behaved processes are included inside q_1 and q_2 respectively. Each process adds to its *tomb* set every process whose *Check* request passes. The total-order-broadcast *tob* delivers the *Check* requests in the same order across processes. Therefore, the result of the check and the updated *tomb* set is the same across processes after processing each request. Consider a *Check* request of a process p' which is ordered before that of p^*. If the check for p' is passed and it leaves, then the *tomb* set of p^* contains p'. Consider when the *Check* request of p^* is processed. The check ensures that p^* is approved to leave only if the intersection of q_1^* and q_2^* modulo the *tomb* set and p^* is p^*-blocking. By Lemma 5, since the quorum system is available inside $\mathcal{O}$, this means that the intersection of q_1^* and q_2^* after both p' and p^* leave still intersects $\mathcal{O}$. A process p in $\mathcal{O}$ remains in the intersection of q_1^* and q_2^*. Therefore, by quorum inclusion, p remains in the intersection of q_1 and q_2. Thus, outlived quorum intersection is preserved for q_1 and q_2.

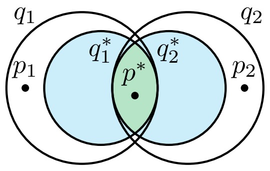

Figure 2 The Leave Protocol, Preserving Quorum Intersection.

Once the *tob* delivers the *Check* message of the leaving process p^* to p^* itself, it can locally decide whether it is safe to leave. We note that the local check ensures a global property: quorum intersection for the whole quorum system. We also note that both quorum inclusion and quorum availability are needed to preserve quorum intersection. Further, we note that outlived quorum intersection is not affected if a Byzantine process leaves: the outlived processes where quorums intersect are by definition a subset of well-behaved processes.

Quorum inclusion. Now let us elaborate on the quorum inclusion property that we just used. When a process p' leaves, it sends *Left* messages to its followers (at either L. 19 or L. 28). The followers later remove p' from their quorums (at L. 29-L. 30). These updates are not atomic and happen over time. Therefore, there might be a window when a process p' is removed from the quorum q_1 (that we saw above), but not yet removed from q_1^*. Therefore, quorum inclusion only eventually holds. However, we observe that in the meanwhile, a weaker notion of quorum inclusion, that we call *active quorum inclusion*, is preserved. It considers inclusion only for the active set of processes $\mathcal{A} = \mathcal{P} \setminus \mathcal{L}$, *i.e.*, it excludes the subset $\mathcal{L}$ of processes that have already left. It requires the quorum q_1^* to be a subset of q_1 modulo $\mathcal{L}$. More precisely, it requires $q_1^* \cap \mathcal{W} \setminus \mathcal{L} \subseteq q_1$. This weaker notion is enough to preserve quorum intersection. In the above discussion for quorum intersection, the process p that remains in the intersection is not in the *tomb* set; therefore, it is an active process. Since it is in q_1^* and q_2^*, by active quorum inclusion, it will be in q_1 and q_2 as well.

References

1 Orestis Alpos, Christian Cachin, and Luca Zanolini. How to trust strangers: Composition of byzantine quorum systems. In *2021 40th International Symposium on Reliable Distributed Systems (SRDS)*, pages 120–131. IEEE, 2021.

2 Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the thirteenth EuroSys conference*, pages 1–15, 2018.

3 Andrea Bracciali, Davide Grossi, and Ronald de Haan. Decentralization in open quorum systems: Limitative results for ripple and stellar. In *2nd International Conference on Blockchain Economics, Security and Protocols (Tokenomics 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

4 Christian Cachin and Luca Zanolini. From symmetric to asymmetric asynchronous byzantine consensus. *arXiv preprint arXiv:2005.08795*, 2020.

5 Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

6 Ivan Damgård, Yvo Desmedt, Matthias Fitzi, and Jesper Buus Nielsen. Secure protocols with asymmetric trust. In *Advances in Cryptology–ASIACRYPT 2007: 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007. Proceedings 13*, pages 357–375. Springer, 2007.

7 Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, 35(2):288–323, 1988.

8 Álvaro García-Pérez and Maria A Schett. Deconstructing stellar consensus (extended version). *arXiv preprint arXiv:1911.05145*, 2019.

9 Andrew Lewis-Pye and Tim Roughgarden. Byzantine generals in the permissionless setting. *arXiv preprint arXiv:2101.07095*, 2021.

10 Andrew Lewis-Pye and Tim Roughgarden. Permissionless consensus. *arXiv preprint arXiv:2304.14701*, 2023.

11 Xiao Li, Eric Chan, and Mohsen Lesani. Quorum subsumption for heterogeneous quorum systems. In *37th International Symposium on Distributed Computing (DISC 2023)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2023.

12 Xiao Li and Mohsen Lesani. Reconfigurable heterogeneous quorum systems. `https://mohsenlesani.github.io/companion/disc24/FullPaper.pdf`.

13 Marta Lokhava, Giuliano Losa, David Mazières, Graydon Hoare, Nicolas Barry, Eli Gafni, Jonathan Jove, Rafał Malinowsky, and Jed McCaleb. Fast and secure global payments with stellar. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 80–96, 2019.

14 Giuliano Losa, Eli Gafni, and David Mazières. Stellar consensus by instantiation. In *33rd International Symposium on Distributed Computing (DISC 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

15 David Mazieres. The stellar consensus protocol: A federated model for internet-level consensus. *Stellar Development Foundation*, 32:1–45, 2015.

16 Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of bft protocols. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 31–42, 2016.

17 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *White paper*, 2008.

18 David Schwartz, Noah Youngs, and Arthur Britto. The ripple protocol consensus algorithm. *Ripple Labs Inc White Paper*, 5(8):151, 2014.

19 Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. Bullshark: Dag bft protocols made practical. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2705–2718, 2022.

20 Chrysoula Stathakopoulou, Tudor David, and Marko Vukolic. Mir-bft: High-throughput bft for blockchains. *arXiv preprint arXiv:1906.05552*, page 92, 2019.
21 Giuliana Santos Veronese, Miguel Correia, Alysson Neves Bessani, Lau Cheuk Lung, and Paulo Verissimo. Efficient byzantine fault-tolerance. *IEEE Transactions on Computers*, 62(1):16–30, 2011.
22 Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 347–356, 2019.