Leadership Uniformity in Raft Consensus Algorithm

Elias Iosif(✉), Klitos Christodoulou, Marios Touloupou, and Antonios Inglezakis

Institute For the Future, University of Nicosia, Nicosia, Cyprus
{iosif.e,christodoulou.kl,touloupos.m,inglezakis.a}@unic.ac.cy
https://www.unic.ac.cy/iff/

Abstract. The Raft consensus algorithm constitutes a widely-used algorithm not only in the broader area of distributed systems, ut also in private/permissioned blockchains such as Hyperledger Fabric. A Raft-based distributed system (RDS) strongly relies on leader election, which involves a number of time-related parameters. In the Raft-related literature, the process according to which those parameters are set is an under-researched area. Specifically, the use of the uniform distribution is the dominant approach. Motivated by this realization, in this work, we focus on these time parameters proposing the notion of "leadership uniformity" in combination with a series of performance metrics. Leadership uniformity is based on the desirable characteristic of having equality among the nodes who serve as leaders. The proposed performance metrics are straightforward adaptations of widely-used measurements from broad disciplines such as estimation theory. The experimental results of this work justify the appropriateness of the proposed notion of leadership uniformity. Specifically, the best performance was yielded by the utilization of normal distribution from which the time parameters under investigation were drawn.

Keywords: Consensus algorithms · Blockchain · Raft

1 Introduction

In the field of distributed systems, reliability of services is considered a fundamental challenge, especially in the presence of faulty or Byzantine processes. To achieve reliability of services, consensus algorithms are proposed for enhancing distributed systems with fault tolerance capabilities. A consensus algorithm aims to provide an agreement with regards to the ordering of events, along with presenting to all processes, that participate to the system, a valid common state. Paxos [1] is acknowledged in the literature as the main paradigm of such algorithms. However, the inherent complexity of Paxos has been regarded by many as the main reason for its limited adoption [2]. This complexity has motivated the implementation of a number of variants [3–5]. In this paper we focus on Raft [6], which was proposed to the literature as an alternative to Paxos [1] cluster of consensus algorithms. In addition to the traditional field of distributed systems, the Raft consensus algorithm is proposed as an alternative to the proof-of-X family of consensus algorithms used in the area of public blockchains [7]. Specifically, the design principles of Raft make it suitable for private or consortium (aka *permissioned*) blockchain systems. For example, the Hyperledger Fabric [8], is considered by many as one of the major proposals for deploying a private/permissioned blockchains fueled by a plug-able consensus algorithm including an implementation of Raft.

A Raft-based distributed system (RDS) strongly relies on the process of leader election. This constitutes a special phase of the overall algorithm and it is controlled by a number of time parameters including the follower timeout. The follower timeout determines the timing of the transition between the default node state (follower) to the state of leader candidate. As such, the follower timeout requires special investigation. Motivated by the significance of the follower timeout, as well as, by the realization that this parameter is under-investigated, in this work we follow an experimental approach, which, to the best of our knowledge, has the following contributions to the Raft-related literature:

1. Proposal of the “leadership uniformity”, which is well-aligned with Raft’s leadership-centric design and it aims to quantify the equality among nodes that served as leaders.
2. Adaptation of a series of performance measures with respect to leadership uniformity.
3. Experimental investigation of the effect of different probability distributions from which the follower timeout is drawn.

Overall, the main findings from this work aim to further enhance the understanding of Raft’s parameter space. Furthermore, the performance metrics discussed in subsequent sections aim to serve as a tool for designers during consensus algorithms development.

The rest of the paper is organized as follows. A brief literature review along with an overview of the Raft algorithm is presented in Sect. 2. The proposed approach is presented in Sect. 3 including a number of performance metrics. The experimental settings and the respective performance scores are reported in Sect. 4 and Sect. 5, respectively. Section 6 concludes this work.

2 Literature Review and Background

This section is organized into two parts. Firstly, we briefly discuss the consensus algorithms landscape through a literature review. Lastly, we present the internal mechanics of Raft.

Specifically, Sect. 2.1 explores the Raft-related literature in an attempt to highlight the motivation behind Raft, and further relate it with the area of blockchain technologies. In the broader area of distributed systems (including blockchains) the agreement on data/information shared between nodes is known as “consensus”. Nodes need to collaborate with each other in order to make decisions on the basis of shared data/information. For the specific case of applications based on shared ledgers (including blockchains), this agreement takes the form of globally recognized ledger of transactions that are validated and, thus, trusted (e.g., see [9–11]). The set (cluster) of nodes should function as a coherent whole being able to survive (i.e., preserve a proper operational mode including the integrity of shared data/information) even in the presence of individual failures (i.e., when a number[1] of nodes crashed).

Following from the above, Sect. 2.2 details the Raft-specific background information that relates to the present work. Raft [12] was proposed as an easy to built, easy to understand fault-tolerant consensus algorithm[2]. A fundamental design characteristic of Raft, is the leader-centric design where the consensus relies heavily on a leader election process which carries the responsibility of reaching agreements. The leadership lasts for a (short) period of time which is parameterizable. In a nutshell, all client requests arriving in the network, are handled by the current leader. A client can be regarded as any machine being connected to the network which, however, is not eligible for serving as a leader (i.e., it consumes information without having the opportunity to play a role in the consensus that guarantees the integrity of the information). The above topics are further elaborated below.

2.1 Literature Review

Paxos is a fundamental paradigm of a consensus algorithm in the area of fault-tolerant distributed system [1]. The problem that Paxos successfully solved is summarized as follows: let a set of processes that propose values; the consensus algorithm should choose and verify only one value among the proposed ones. However, Paxos is considered by several researchers complex to understand and, therefore, hard to implement. Indicative descriptions of those difficulties are presented in [13–15].

After Paxos, several related research studies have been performed such as the work demonstrated in [16]. The authors have presented a series of design decisions for the creation of a fault-tolerant database using the Paxos algorithm. Furthermore, Paxos is considered by many as the “father” of the Raft consensus algorithm [6]. The complexity of the Paxos algorithm is used to serve educational purposes on state machine replication. The need for having an easy–to–understand consensus algorithm, which can be used not only for systems development but also for education purposes, created the conditions for the proposal of Raft.

Recent technological advancements with the introduction of blockchain technologies (including consensus algorithms) pose a series of educational challenges in training courses, especially in mixed student groups (e.g., student from engineering and business backgrounds). These challenges are detailed in [17] along

[1] In many cases, this number should be less than the 50% of total number of nodes.

[2] Raft can only tolerate crash failures and not malicious nodes.

with a series of recommendations. Specifically, Raft's authors introduced new techniques compared to Paxos, including feature decomposition and state space reduction. Numerous investigations have been conducted, analysing the performance of Raft, e.g., [18]. Other research attempts have demonstrated Raft variants such as the realization of a dynamic Raft cluster [19], introducing the dynamic utilization of resources and the concept of disqualified nodes. Furthermore, Raft has been tested in several use cases such as in distributed Software Defined Networks (SDNs) [20]. In [20], a thorough overview of Raft is provided along with a discussion regarding the implementation of SDN controller platforms in OpenDaylight [21] and ONOS [22].

Since Raft was introduced for the development of distributed fault-tolerant systems, with the rise of the blockchain and Distributed Ledger Technologies (DLTs), researchers have also introduced Raft in different blockchain systems such as Quorum [23] and Hyperledger Fabric [8]. Among the numerous approaches that appear in the literature, the work presented in [18] highlights consensus as a challenging problem in blockchain systems. This work is concerned with a study on the efficiency of Raft in non-negligible packet-loss rate networks. The authors proposed a detailed computational model that is used to evaluate the possibility of crashing the distributed network. Moreover, the work reported in [24], demonstrates a non Byzantine fault–tolerant algorithm called KRaft. KRaft is similar to Raft retaining most of its structure and it can be used in private blockchain systems. KRaft utilizes the K-Bucket node relationships in the Kademlia protocol [25].

In [26], the performance of the Quorum blockchain was evaluated followed by an investigation regarding the support of different consensus algorithms. The main findings propose that in a permissioned setting, where participants are known, it is unnecessary to use a consensus algorithm such as Proof–Of–Work (PoW); which is used in public blockchains like Bitcoin [9]. It seems that algorithms similar to Raft are better suited in private settings. However, this goes against the open-participation, and decentralized philosophy adopted by public blockchains. A recent study presented in [27] suggests the investigation of the interplay between the decentralization degree, that characterize a blockchain system, and the underlying consensus algorithm. For the case of non-public permissioned blockchains [27] constitutes one of the first experimental approaches towards this direction. Specifically, it is reported that the decentralization degree of the Ripple protocol can be determined as a function of the adversarial attacks or network faults. Under certain conditions the centralization degree of private/permissioned networks can be relaxed.

2.2 Raft: Background

As stated above, the nodes in a distributed system need to agree on the shared data/information. Enabling consensus in a fault-tolerant environment is also essential for the development of decentralized applications (abbreviated as dApps) that are deployed over a blockchain [28]. Being inline with this spirit, the main characteristic of the Raft algorithm is the provision of consensus in fault-tolerant distributed systems and dApps [12]. The notion of fault, refers to the case of node crashes excluding any Byzantine faults. In addition, Raft was proposed as a clearer alternative to Paxos, as well as, to provide a deeper understanding on how consensus can be accomplished in distributed systems. This was motivated by the widely-accepted view according to which its predecessor, Paxos [29], was considered difficult to understand and adopt (mainly due to the respective design complexity).Raft was intended to be more understandable by separating the logic of Paxos in a more comprehensible way. Furthermore, Raft was formally proven to be safe while offering some additional features.I n particular, Raft provides an inclusive way for handling a state machine in a set of nodes. Thus, the agreement on the same series of state transitions is achieved. The background information regarding Raft is presented in subsequent sections focusing on: (i) the election of leaders; (ii) the replication of the (shared) log; and (iii) a series of safety features. For completeness we note that this work relies on [6,30] where the algorithm is explained in detail.

Leader Election. According to [31], in non-faulty conditions, a node in a cluster can remain in one of the following states {follower, candidate, leader} during a certain time unit. The default state is "follower". Any node can become leader through the transition by the "candidate" state. The leader node transmits messages (also referred to as heartbeats or pulses) to other nodes declaring its leadership. That is, it lets the rest nodes know that an active leadership exists. When a follower signals a time-out while waiting for a pulse from the leader, a leader election phase may take place. Specifically, the timed-out node switches to the "candidate" state, votes for itself, and issues a number of Remote Procedure Calls (RPC). The aforementioned RPC calls are labeled as "RequestVotes", meaning that votes from the rest nodes are needed in order the candidate to become the leader. This phase can result into one of the following three outcomes:

1. By gaining the majority of votes, the candidate becomes the leader.
2. If vote majority is not reached, the phase may end without assigning leadership and the candidate returns to the state of "follower".
3. In case of multiple candidates, the leader is determined by the respective term number which is an index denoting whether the node is updated with respect to the shared data (also referred to as the replicated log).

Log Replication. Raft breaks down the "consensus problem" by introducing the leader election as a separate process in combination with the log replication and safety. The replicated log cannot be manipulated while it supports append–only entry storage. The log exists in every node enabling tolerance to faults, as well as, high availability.

The leader is responsible for having replicas of the logs on all nodes. Requests can be sent to the leader by the clients, which can be regarded as read/write commands to be executed with effect on the shared log. After a successful write operation, a new entry is appended to the leader's log, and this change is forwarded to the followers via an "AppendEntries" RPC message. If a follower (or more) is unavailable, the leader will retry sending the "AppendEntries" RPC until the change is adopted by all followers. A log entry typically contains three fields as follows:

1. **Command**: Denotes the command requested by the client.
2. **Index**: Specifies the position of the entry in the log.
3. **Term number**: Encodes the relative[3] time.

As stated in [12], the logs can become unreliable in the event of leader crash. In this case, the latest version of the (available) log is identified and taken into account.

Safety. The following safety features are supported by Raft:

1. **Election safety:** Only one leader can exist in the context of a term.
2. **Leader append-only:** The leader can perform only append operations in the log (i.e., no substitutions and/or deletions).
3. **Log matching:** Two logs are considered to be identical up to an index, if they have an entry that corresponds to the same index and term.
4. **Leader completeness:** The leaders' logs contain the log entries that were committed in previous terms.
5. **State machine safety:** When a node attaches a log entry to its state machine, then other nodes can not apply any changes to it.

In the sections that follow, we focus on *timeouts* that play a key role in leader election (briefly described in Sect. 2.2). Firstly, we introduce a series of definitions followed by the description of the experimental setup and the respective performance results. These results are reported for various performance metrics and different timeout configurations.

3 Model

In this section, we provide a series of definitions regarding the characterization of an RDS (see Sect. 3.1). Those definitions are used for introducing a series of performance metrics in order to quantify the leadership uniformity of the respective RDS (see Sect. 3.2).

3.1 Definitions

At each any point in time, an RDS can be characterized by the following model:

$$RDS = (N,\ D_f,\ D_c,\ g(t_f, t_c)), \tag{1}$$

[3] In a message–passing distributed system a global clock does not apply [32], so, the timings are encoded as (numerical) indices.

where, N is the number of nodes in RDS, D_f is the distribution used for setting the follower timeout t_f, and D_c is the distribution from which the candidate timeout t_c is drawn. In addition, $g(t_f, t_c)$ denotes a function reflecting the relation between t_f and t_c. Note that the model defined by (1) is focused on the critical timeouts that determine the election phase of the Raft algorithm. That is, this model is not meant to encode an exhaustive description of all Raft's parameters (i.e., parameters that apply in addition to the election phase).

For a cluster of N nodes, the following sequence is defined

$$(n_{1,L}, n_{2,L}, ..., n_{i,L}, ..., n_{N,L}) \text{ for } i = 1, ..., N, \tag{2}$$

where, $n_{i,L}$ denotes the absolute frequency (i.e., number of times) with which the i–th node, n_i, was elected as leader ($n_{i,L} \in \mathbb{Z}_0^+$). The sequence of frequencies can be treated as a set of unordered values as, by design, the ordering of nodes does not play any role in the election phase.. This "bag-of-leadership-frequencies" can be directly used for computing basic yet informative measurements as explained next.

3.2 Performance Metrics

Here, four metrics are defined meant to quantify the leadership uniformity and used as performance metrics. Those metrics can be organized in two categories: (i) variation of leadership frequencies, and (ii) divergence from ideal leadership.

Variation of Leadership Frequencies. Two intuitive measurements of the variation of the leadership frequencies in (2) are the standard deviation (denoted as *Std*) and variance (denoted as *Var*). Semantically, a desirable leadership uniformity implies low variation of leadership frequencies. In an ideal scenario, all participating nodes should be characterized by equal $n_{i,L}$ values. As the RDS under investigation approaches the ideal leadership uniformity, the respective *Std* and *Var* of (2) tend to zero.

Distance from Ideal Leadership. The second category of performance metrics relies on the idea of measuring the (average) distance from the ideal leadership uniformity. The latter is denoted by $\bar{n}_L$ and is defined as:

$$\bar{n}_L = \frac{T}{N}, \tag{3}$$

where, T stands for the number of successful election trials (rounds), i.e., trials in context of which exactly one leader has been elected. Based on this definition, we can adopt the notion of mean absolute error (a widely-used measurement applied in various disciplines such as estimation theory [33]) as follows:

$$MAE = \frac{1}{N} \sum_{i=1}^{N} |\, n_{i,L} - \bar{n}_L \,| \,. \tag{4}$$

The meaning of "error" is equivalent to the meaning of the distance of $n_{i,L}$ from the ideal value $\bar{n}_L$. A variation of (4) can be defined as follows:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(n_{i,L} - \bar{n}_L)^2. \tag{5}$$

Similarly to the case of *MAE*, *MSE* is motivated by the commonly used measurement of mean squared error. In general, *MSE* is a fundamental criterion that is widely-used in the broad context of optimal estimators [33]. As the RDS under investigation tends to the ideal leadership uniformity, the respective *MAE* and *MSE* scores tend to zero.

The quadratic operations used in *Var* and *MSE*, are meant to "amplify" the respective differences in an attempt to facilitate the comparison of the considered distributions from which the timeouts are drawn.

4 Experiments

The experimental setup used throughout the experiments reported by this work is described as follows.

1. Number of nodes (N): In total, five nodes were utilized which constitutes a typical cluster size according to the Raft reference paper [6]. For the sake of clarity, we note that the $N = 5$ configuration deals with the number of nodes that can be deemed as candidate leaders (and therefore being responsible for system's consensus) and it does not limit the number of nodes that can participate in the overall network as clients. For example, we may have five nodes deemed as candidate leaders along with thousands (or more) of other nodes acting as clients. The latter type of nodes are not eligible for leadership and they can only exchange information with the leader.
2. Distributions from which the timeouts are drawn (D_f, D_c): The uniform distribution is widely used in numerous Raft-related works, while it seems that other distributions are less-researched. Motivated by this gap, the focus of this work is the investigation of the role of two additional distributions, namely, *normal* and *lognormal*, for the case of follower timeout. The normal distribution was chosen due its vast applicability for modeling various phenomena. The employment of the lognormal distribution was motivated by the observation that the latency of nodes in distributed systems can be modeled by distributions that belong to the lognornal family. For example, an indicative early study can be found in [34], while a more recent lognormal-based modeling is presented in [35].

For the case of candidate timeout, t_c, a uniform distribution was used taking values in the [0,500] ms interval. Regarding the follower timeout, t_f, and the normal/lognormal distributions, the following three cases were investigated: (i) $\bar{t}_f = 50$ ms, (ii) $\bar{t}_f = 500$ ms, and (iii) $\bar{t}_f = 5000$ ms, where $\bar{t}_f$ stands for the mean value of the respective distribution. These timeout values imply a linear relationship (as a possible form of $g(t_f, t_c)$) between follower and candidate timeouts. The standard deviation, σ, of those distributions was set to $\frac{1}{4}\bar{t}_f$. For the case of t_f's uniform distribution, the following range was used $[\bar{t}_f - 2\sigma, \bar{t}_f + 2\sigma]$. In order to compare the performance yielded for the aforementioned experimental settings, the metrics defined in Sect. 3.2 were used for 100 successful election trials. All experiments reported were performed by extending *pyraft*, a Python-based implementation of Raft[4].

5 Analysis of Performance

In this section, the performance of the Raft consensus algorithm is presented with respect to *leadership uniformity* using four evaluation measurements, namely, *Std*, *Var*, *MAE*, and *MSE*. Regarding the normal and lognormal distributions, the respective scores are reported for the following cases (also referred to as $\bar{t}_f$ scenarios): (i) $\bar{t}_f = 50\,\text{ms}$, (ii) $\bar{t}_f = 500\,\text{ms}$, and (iii) $\bar{t}_f = 5000\,\text{ms}$. Also, the performance yielded by the uniform distribution is also presented.

Table 1. Follower timeout: leadership uniformity ($\bar{t}_f = 50\,\text{ms}$).

Distribution	*Std*	*Var*	*MAE*	*MSE*
Uniform	8.53	72.80	11.20	172.80
Normal	**6.90**	**47.60**	**10.80**	**147.60**
Lognormal	10.00	100.00	12.40	200.00

Table 1 demonstrates the performance scores for the case of $\bar{t}_f = 50\,\text{ms}$. The best performance is achieved by the normal distribution, while the utilization of the lognormal distribution exhibits the lowest scores. As expected –and by definition– all four metrics are correlated. However, *Var* and *MSE* are characterized by larger ranges compared to *Std* and *MAE* amplifying the performance differences among the distributions under comparison. Regarding $\bar{t}_f = 500\,\text{ms}$, the performance analysis is presented in Table 2. Similarly to the previous case, the best results are yielded by the normal distribution. Slightly different observations hold when increasing the follower timeout. These are presented for the

Table 2. Follower timeout: leadership uniformity ($\bar{t}_f = 500\,\text{ms}$).

Distribution	*Std*	*Var*	*MAE*	*MSE*
Uniform	8.72	76.00	12.00	176.00
Normal	**6.07**	**36.80**	**10.40**	**136.80**
Lognormal	6.08	37.00	12.00	209.60

[4] https://pypi.org/project/pyraft/.

Table 3. Follower timeout: leadership uniformity ($\bar{t}_f = 5000\,\text{ms}$).

Distribution	*Std*	*Var*	*MAE*	*MSE*
Uniform	**8.07**	**65.20**	**11.60**	**165.20**
Normal	8.94	80.00	12.80	180.00
Lognormal	11.15	124.40	13.60	224.40

$\bar{t}_f = 5000\,\text{ms}$ case as shown in Table 3. In this case, the uniform distribution constitutes the top performing distribution followed by the uniform and lognormal distribution.

Overall, the top performance was achieved for the middle $\bar{t}_f$ scenario (i.e., $\bar{t}_f = 500\,\text{ms}$) that achieved $MSE = 136.80$. An example where the effectiveness of *MSE* is clearly shown, is the comparison between the normal and lognormal distribution, as shown in Table 2. According to *Std* and *Var*, the two distributions exhibit slight differences (e.g., 6.07 vs. 6.08 for *Std*). However, this difference is emphasized when using *MSE*, i.e., 136.80 vs. 209.60.

6 Conclusions

In this work, we explored an under-investigated aspect of the Raft algorithm. This aspect deals with the setting of follower timeout that constitutes a critical parameter for the leader election phase. In addition, the notion of leadership uniformity was proposed based on the idea that all nodes should be granted equal leadership roles. This is well-aligned with the core spirit of Raft according to which the consensus is managed by the (current) leader. With regards to leadership uniformity we have shown the straightforward adaptation of a number of intuitive performance metrics. Those metrics provided a comprehensive quantification of performance with respect to leadership uniformity. The experimental results suggest that the *MSE* performance metric is more descriptive compared to *MAE* and *Std*. The variance of leadership frequencies, *Var*, exhibits similar behavior with *MSE*. A key finding of the present work is the observation that the normal distribution yields the best performance when the average follower timeout, $\bar{t}_f$, is kept less than 1s. This was verified for $\bar{t}_f = 50\,\text{ms}$ and $\bar{t}_f = 500\,\text{ms}$. For those timeouts, the superiority of the normal distribution over the uniform distribution can be attributed to the fact that all timeouts according to the latter distribution are equiprobable. This finding contributes to the Raft-related literature where the utilization of uniform distribution seems to be the usual (i.e., default) choice.

Future work deals with the automatic estimation of the distribution parameters. In this context, we also aim to investigate the dynamic selection of the optimal timeout. This will require the employment of a machine learning based component which should be trained on the network characteristics (e.g., based on the relative differences of nodes' timeouts). Last but not least, we aim to explore the effect of other distributions exhibiting asymmetric characteristics (skewness).

Acknowledgement. This research was funded by the Ripple's Impact Fund, an advised fund of Silicon Valley Community Foundation (Grant id: 2018-188546).

References

1. Lamport, L.: The part-time parliament. In: Concurrency: The Works of Leslie Lamport, pp. 277–317 (2019)
2. Howard, H., Schwarzkopf, M., Madhavapeddy, A., Crowcroft, J.: Raft refloated: do we have consensus? ACM SIGOPS Oper. Syst. Rev. **49**(1), 12–21 (2015)
3. Lamport, L., Massa, M.: Cheap Paxos. In: International Conference on Dependable Systems and Networks, 2004, pp. 307–314. IEEE (2004)
4. Lamport, L.: Fast paxos. Distrib. Comput. **19**(2), 79–103 (2006)
5. Mazieres, D.: Paxos made practical (2007)
6. Ongaro, D., Ousterhout, J.: In search of an understandable consensus algorithm. In: 2014 USENIX Annual Technical Conference, pp. 305–319 (2014)
7. Bano, S., et al.: SoK: consensus in the age of blockchains. In: Proceedings of the 1st ACM Conference on Advances in Financial Technologies, pp. 183–198 (2019)
8. IBM: Build and run a smart contract on a Hyperledger Fabric network with the Raft ordering service (2020)
9. Nakamoto, S., Bitcoin, A.: A Peer-to-peer Electronic Cash System. Bitcoin (2008). https://bitcoin.org/bitcoin.pdf
10. Wood, G., et al.: Ethereum: a secure decentralised generalised transaction ledger. Ethereum Proj. Yellow Pap. **151**(2014), 1–32 (2014)
11. Schwartz, D., Youngs, N., Britto, A., et al.: The Ripple protocol consensus algorithm. Ripple Labs Inc. White Pap. **5**(8) (2014)
12. Howard, H.: ARC: analysis of raft consensus. Technical report UCAM-CL-TR-857, University of Cambridge, Computer Laboratory (2014)
13. Meling, H., Jehl, L.: Tutorial summary: Paxos explained from scratch. In: Baldoni, R., Nisse, N., van Steen, M. (eds.) Principles of Distributed Systems. OPODIS 2013. Lecture Notes in Computer Science, vol. 8304, pp. 1–10. Springer, Cham (2013). https://doi.org/10.1007/978-3-319-03850-6_1
14. Chand, S., Liu, Y.A.: What's live? understanding distributed consensus. arXiv preprint arXiv:2001.04787 (2020)
15. Liu, Y.A., Chand, S., Stoller, S.D.: Moderately complex Paxos made simple: high-level specification of distributed algorithm. Computing Research Repository (2017)
16. Chandra, T.D., Griesemer, R., Redstone, J.: Paxos made live: an engineering perspective. In: Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing, pp. 398–407 (2007)
17. Themistocleous, M., Christodoulou, K., Iosif, E., Louca, S., Tseas, D.: Blockchain in academia: where do we stand and where do we go? In: Proceedings of the 53rd Hawaii International Conference on System Sciences (2020)

18. Huang, D., Ma, X., Zhang, S.: Performance analysis of the raft consensus algorithm for private blockchains. IEEE Trans. Syst. Man Cybern. Syst. **50**(1), 172–181 (2020)
19. Nakagawa, T., Hayashibara, N.: Resource management for raft consensus protocol. Int. J. Space-Based Situated Comput. **8**(2), 80–87 (2018)
20. Sakic, E., Kellerer, W.: Response time and availability study of RAFT consensus in distributed SDN control plane. IEEE Trans. Netw. Serv. Manage. **15**(1), 304–318 (2017)
21. : The Linux Foundations Projects. OpenDaylight (2019)
22. Open Networking Foundation (2019)
23. Quorum: Home. Quorum (2020)
24. Wang, R., Zhang, L., Xu, Q., Zhou, H.: K-Bucket based Raft-like consensus algorithm for permissioned blockchain. In: 2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS), pp. 996–999 (2019)
25. Kademlia: a design specification. XLattice (2019)
26. Baliga, A., Subhod, I., Kamat, P., Chatterjee, S.: Performance evaluation of the Quorum blockchain platform. arXiv preprint arXiv:1809.03421 (2018)
27. Christodoulou, K., Iosif, E., Inglezakis, A., Themistocleous, M.: Consensus crash testing: exploring Ripple's decentralization degree in adversarial environments. Future Internet **12**(3), 53 (2020)
28. Ren, L., Ward, P.A.S.: Distributed consensus and fault tolerance mechanisms. In: Li, K.-Ch., Bertino, E., Chen, X., Jiang, H. (eds.) Essentials of Blockchain Technology. 1st edn. (2019)
29. Lamport, L., et al.: Paxos made simple. ACM Sigact News **32**(4), 18–25 (2001)
30. Ongaro, D.: Consensus: bridging theory and practice. Ph.D. thesis, Stanford University (2014)
31. Arora, V., Mittal, T., Agrawal, D., El Abbadi, A., Xue, X., et al.: Leader or majority: why have one when you can have both? improving read scalability in Raft-like consensus protocols. In: 9th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 17) (2017)
32. Coulouris, G.F., Dollimore, J., Kindberg, T.: Distributed Systems: Concepts and Design. Pearson Education (2005)
33. Kay, S.M.: Fundamentals of Statistical Signal Processing. Prentice Hall PTR (1993)
34. Almeida, V., Bestavros, A., Crovella, M., De Oliveira, A.: Characterizing reference locality in the www. In: Fourth International Conference on Parallel and Distributed Information Systems, pp. 92–103. IEEE (1996)
35. Underwood, R., Anderson, J., Apon, A.: Measuring network latency variation impacts to high performance computing application performance. In: Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering, pp. 68–79 (2018)