Orchestra-Paxos: A Revision of the Paxos with Dynamic Number of Quorum Participants

Mikhail Diagilev and Vladimir Korkhov(✉)

Saint Petersburg State University, 7-9 Universitetskaya Embankment, St Petersburg, Russia 199034
st072182@student.spbu.ru, v.korkhov@spbu.ru

Abstract. The present research focuses on applying consensus methods to ensure data consistency and fault tolerance in distributed systems, with particular attention to the Paxos consensus algorithm, widely used to coordinate decisions among distributed nodes. Despite its reliability, Paxos has practical limitations that affect system performance and availability in unpredictable network conditions. These include challenges in forming a quorum when nodes fail or become temporarily unavailable, as well as performance degradation caused by requiring participation from all acceptors in the consensus process. Message loss and network instability further reduce the efficiency of consensus mechanisms. To address these issues, the paper proposes a modified version of Paxos, called Orchestra-Paxos. This approach introduces partial consistency and improves fault tolerance by allowing consensus to be reached through an active subset of nodes, while others can be temporarily marked as unavailable. Unlike traditional Paxos, it does not require the involvement of a majority of accepting nodes in every replication cycle. The study uses comparative analysis between Orchestra-Paxos and Multi-Paxos, combining system modeling with experimental performance evaluation. Key metrics include average replication round time, total completion time, and the number of rounds required to accept all proposals. Results show that Orchestra-Paxos significantly improves performance under high message loss conditions (60%), reducing both the number of rounds and overall completion time by an average factor of 12 compared to Multi-Paxos.

Keywords: Computer science · Consensus algorithm · Distributed systems · Concurrent computing · Computing cluster architecture · Improved algorithm · Paxos · Dynamic quorum

1 Introduction

Modern distributed systems, data storage systems, and financial networks frequently require high availability, dependability, and security. The consensus

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2027
O. Gervasi et al. (Eds.): ICCSA 2026 Workshops, LNCS 16759, pp. 39–52, 2027.
https://doi.org/10.1007/978-3-032-30518-3_3

mechanism is a major component of such systems. This enables the system nodes to agree on conclusions about the sequence of transaction execution and the state of data, despite probable faults and delays. This is precisely why consensus algorithms are essential in achieving these goals.

However, as distributed systems become more complicated and scalable, classic consensus approaches like the Paxos algorithm [1] and the Raft algorithm [2] encounter a variety of issues. The primary issues are decreased performance, longer reaction times, and challenges maintaining consistency under high loads. Therefore, it is essential to modernize and optimize current consensus techniques, especially when dealing with extensive distributed systems consisting of dozens or even millions of nodes.

In this paper, the Paxos algorithm is selected for modernisation. Paxos is used in large distributed systems such as Megastore [3], Chubby [4], Cassandra [5] and MySQL [6]. It has a number of drawbacks. A major issue is the necessity of majority quorum in the voting phase. The system may shortly forfeit its capacity to achieve consensus for an unspecified duration, either due to an escalation in the response time of quorum members or as a consequence of failures. Such situations can lead to a loss of profit for the company and call its reputation into question, as was the case with Google in 2014 [7]. In view of this, this work is dedicated to developing an algorithm that will essentially build upon the concept of the Paxos algorithm [1]. On the other hand, at each stage of the voting process, gathering all quorum members will be optional. Any nodes that do not respond will be marked as temporarily unavailable. However, they will be able to resume decision-making to reach consensus once their operation has been restored. It is therefore anticipated that this approach can guarantee partial-incomplete system consistency and significantly increase the overall system performance.

In order to compare the developed algorithm with Paxos variants, namely Multi-Paxos [8], the following criteria will be used: the average time of a single replication round of the algorithm, the waiting time for the completion of all replication rounds, and the number of rounds required to fulfil all client requests [9].

The contribution to the field is that this is the first consensus algorithm that uses dynamical number of quorum participants based on the Paxos algorithm.

The topic of modernising consensus algorithms in large distributed systems and analysing failures within them is extremely relevant, as it demands the development of new solutions to improve the resilience and performance of modern systems while reducing the risks associated with various failures. This paper examines modern approaches to the modernisation of consensus algorithms.

2 Related Work

Consensus algorithms in the Paxos family have been extensively studied and optimized to address performance and scalability challenges in distributed systems. One of the most widely adopted variants is Multi-Paxos [8], which amortizes the cost of leader election by allowing a stable leader to propose multiple

values without repeating the full consensus protocol. This significantly reduces latency in steady-state operation and is commonly used in production systems.

Fast Paxos [10] improves upon classical Paxos by reducing the number of communication rounds required to reach consensus under certain conditions. By allowing proposers to bypass the leader in some cases, it achieves lower latency, though at the cost of increased complexity and a higher likelihood of conflicts.

Another algorithm is Flexible Paxos [11], in which communication does not involve all nodes or a fixed majority. Messages are sent only to a quorum of nodes, which can be chosen more flexibly. Flexible quorum configurations can improve availability and performance in distributed systems.

Building on these ideas, Fast Flexible Paxos [12] combines the reduced latency of Fast Paxos with the quorum flexibility of Flexible Paxos. This hybrid approach aims to achieve both lower latency and improved fault tolerance, although it introduces additional design and implementation complexity.

These variants highlight the ongoing effort to refine Paxos-based consensus mechanisms, balancing trade-offs between latency, fault tolerance, and system complexity. The algorithm proposed in this work builds upon these foundations, aiming to further modernize and optimize consensus in distributed environments.

3 Replicated State Machine

Large-scale distributed cloud systems often handle client data. Multiple computer nodes are frequently used to provide dependable data storage, modification, and deletion inside the system. Multiple nodes allow the system to tolerate failures and outages that impact only part of them. This enhances the issue of duplicating data over several nodes while preserving consistency. To ensure consistency among nodes, a technique for determining the order in which incoming messages are processed is required. Consensus acts as such a process. Various algorithms are used to reach consensus among nodes. Paxos [1] is one of the most well-known and commonly used.

It is important to bear in mind the FLP theorem of Fischer, Lynch and Paterson [13]. It states that it is impossible to achieve even a non-bounded consensus for $N > 2$ processes, even on a single bit, under the following conditions in a distributed system:

1. The system is asynchronous;
2. Any node in the system may experience a failure for an indefinite period of time;
3. The algorithm must terminate within a finite time;
4. The algorithm must be deterministic.

Thus, the theorem highlights the challenges of achieving reliable coordination in distributed systems in the presence of failures. Furthermore, in large cloud-based distributed systems, the order in which data arriving from the client is recorded may be important (for example, the order of financial transactions in banking systems), or it may be irrelevant (for example, non-sensitive data such as system process metrics).

4 Paxos

Paxos is a family of protocols designed to solve the consensus problem in distributed systems with unreliable computations. The creator of the family's leading algorithm, namely Paxos, is Leslie Lamport [1].

For simplicity, let us consider a record of a specific value in the system as the object for which the consensus problem is being solved. Each node in the distributed system can play one of three roles:

1. Proposer;
2. Acceptor;
3. Learner.

The proposer wants the value proposed by the client for the record to be selected in the system. The acceptors agree on the proposal and notify all learners of it. The learner wishes to know the accepted value for the record.

In the algorithm, the proposer attempts to put a proposal forward for approval by the acceptors. If another value has already been accepted, the proposer becomes aware of this and informs the learners. The proposal process is divided into two phases:

1. Preparation and promise:
   (a) The proposer receives a value v from the client and selects a unique round identifier p, then sends a message containing p to all acceptors to collect commitments;
   (b) Acceptors store p locally as the highest accepted proposal and respond to the proposer with a promise if they have not previously committed to any other proposer to participate in a round with a unique identifier $p' > p$. Otherwise, they refuse to participate in the vote with number p. If any of the acceptors has managed to accept value v' from a proposer with round number p', then the acceptor informs the proposer with round number p of a refusal with value v';
   (c) If a quorum (at least half of the acceptors) of promises is collected from the acceptors, the proposer proceeds to the second phase. Otherwise, they may attempt to select a unique round identifier greater than p and repeat the first phase.
2. Proposal and confirmation:
   (a) If the proposer has already received an accepted promise of participation in the voting from the acceptors, they must select the value with the highest unique identifier as v. If this has not occurred, the proposer is entitled to use their own proposal as v. The proposer sends a proposal with number p and v to the acceptors;
   (b) If an acceptor receives a proposal with number p, which is no less than the unique identifier of the round in which the acceptor promised to vote, the acceptor accepts the value v and notifies all learners. Otherwise, the acceptor rejects the proposal and responds to the proposer with the unique identifier $p' > p$ and the value v', which corresponds to p';

(c) Upon receiving a simple majority quorum (at least half of the acceptors) accepting the value from the acceptors, the proposer learns that the value has been accepted. Otherwise, the proposer may repeat the first phase with a unique identifier greater than p.

Paxos assures that once a value v is selected, no other value v' may be chosen. The procedure achieves consensus only if $[n/2]+1$ of the system's n nodes do not fail. The Paxos consensus algorithm addresses the issue of ending a consensus mechanism in the presence of malfunctioning nodes in an asynchronous system. Paxos, on the other hand, is a 'one-shot algorithm', which means that it does not expect to process more than one proposal from a client. This is precisely why, in real-world systems, Paxos is used to establish a consensus on the order of value acceptance inside cycles. Within these cycles, Paxos is employed to overcome the consensus problem. Generally, Paxos undergoes modifications in real-world systems and is not used in its classical form, as it is considered a theoretical algorithm [4].

Figure 1 illustrates an ideal implementation of Paxos for solving the consensus problem among system nodes. The following notation is used in Fig. 1: P—proposer, A—acceptor, L—learner, p—unique round identifier, v—value proposed by the proposer. The notes on Fig. 1 reflect the moments when each of the two phases began and ended. For better visual perception, only one learner is shown.

5 An Enhanced Paxos Algorithm – Orchestra-Paxos

Standard Paxos has a number of drawbacks. Two main issues stand out.

1. The inability to establish a quorum of acceptors in the first phase if more than half of the system's confirming nodes fail, as illustrated in Fig. 2;
2. Endless competition among several proposers in the first phase to assemble a quorum of acceptors, as shown in Fig. 3.

Whilst the second problem can be solved by configuring a random timeout following each failed attempt in the first phase, the first problem cannot be solved in a straightforward manner.

The Paxos algorithm has a number of variants, such as EPaxos [14], Flexible Paxos [11], Fast Flexible Paxos [12] and others. Each addresses a specific problem in Paxos, whilst also improving fault tolerance or performance.

One of the main modernised algorithms is Multi-Paxos. The difference from the classical version is that the proposer becomes the master after successfully passing the first phase. All subsequent proposals from the system's client go exclusively through this master. However, the drawback of this approach lies in the selection of a new master in the event of a failure of the current one. Furthermore, Paxos originally positions itself as an algorithm that does not depend on a master. Furthermore, performance studies of systems using Multi-Paxos demonstrate that the system's throughput is determined by the speed of

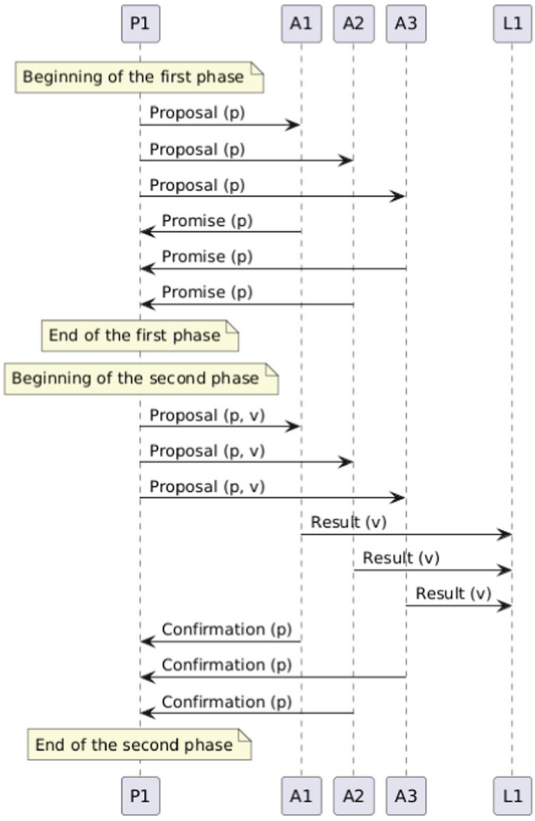


Fig. 1. Paxos's perfect execution

the slowest node. This, in turn, is a drawback of the classic Paxos algorithm as well.

Thus, two main problems of the standard Paxos algorithm are identified. It is for this reason that this paper proposes a modernised version of the Paxos algorithm—Orchestra-Paxos—which resolves the issues described above and is applied to the problem of replicated state machine in large distributed systems for values where the order of writing is not strictly mandatory. It also consists of two phases:

1. Preparation and promise:
   (a) The proposer selects a unique round identifier p and sends a message containing p to all acceptors to collect commitments. A timer is started to wait for commitments from the acceptors;
   (b) Acceptors store p locally as the highest accepted proposal and respond to the proposer with a promise if they have not previously committed to any other proposer to participate in a round with a unique identifier $p' > p$.

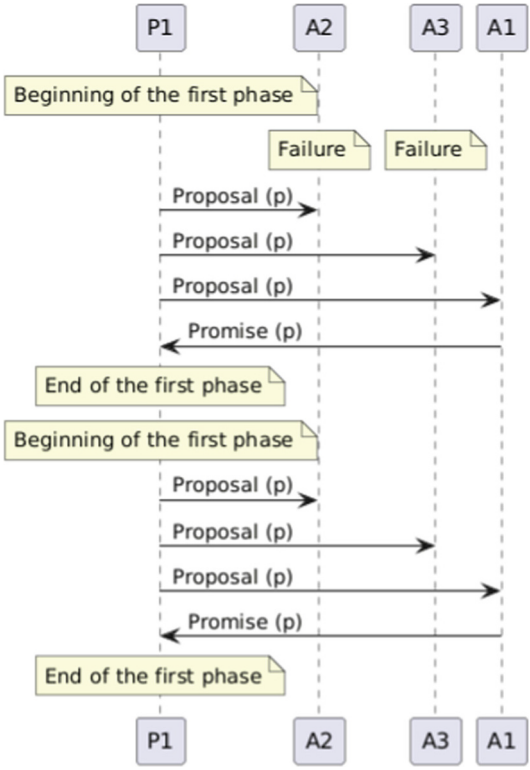


Fig. 2. Failure to gather a quorum of acceptors

Otherwise, they refuse to participate in the vote with number p and reply to the proposer with p' and the value v', if it was proposed earlier;

(c) The proposer, in the absence of replies refusing to participate in their vote, collects a set of acceptors (active acceptors) upon the expiry of the timer and proceeds to the second phase. Otherwise, the proposer may attempt to select a unique round identifier greater than p and repeat the first phase.

2. Proposal and confirmation:
   (a) The proposer sends an offer with number p, v and set of active acceptors to all members of the set of active acceptors. A timer is started to wait for the acceptors to accept the values;
   (b) If an acceptor receives a proposal with number p, which is no less than the unique identifier of the round in which the acceptor promised to vote, the acceptor takes the value v and notifies all learners about the acceptance of the value, and also transmits the set of active acceptors. The acceptor then notifies the proposer of the confirmation of the value. Otherwise, the acceptor rejects the proposal and replies to the proposer with a unique identifier $p' > p$ and a value v' corresponding to p';

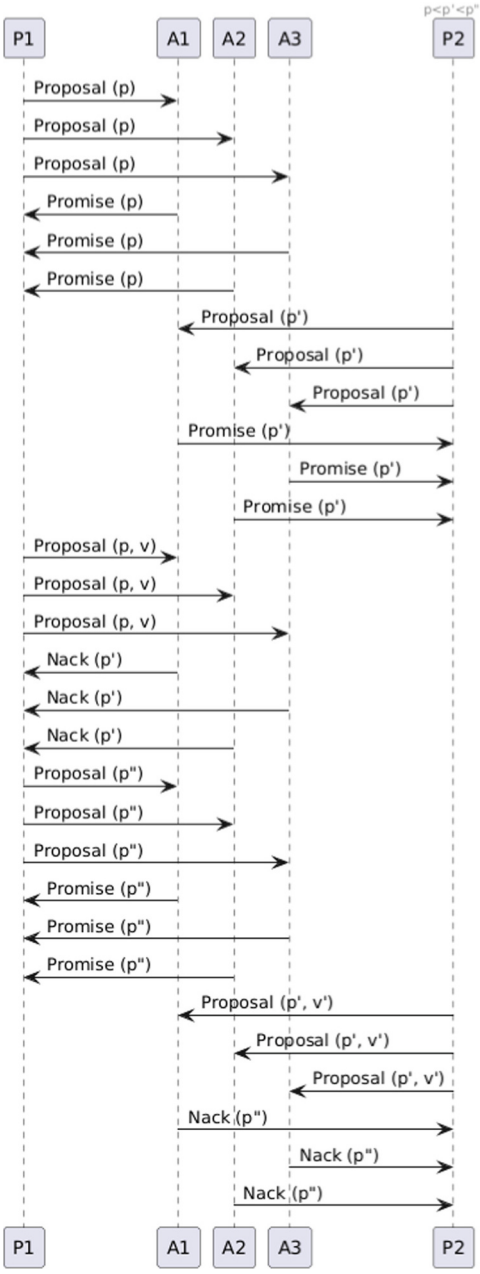


Fig. 3. Endless competition between several proposers

(c) Upon receiving a simple majority quorum (at least half of the acceptors) from the set of acceptors obtained in the first phase regarding the acceptance of the value, the proposer learns that the value has been accepted. If a proposer does not obtain a quorum of confirmations or the timer expires, they may repeat the first phase with a unique identifier after the expiry of a random timer that is greater than p. The set of active acceptors also becomes empty.

Thus, this algorithm resolves the problem of the impossibility of gathering a quorum in the first phase. Should such a situation arise, it is optionally possible to additionally notify the system participants responsible for collecting node survivability metrics. Furthermore, timers in both phases help to avoid waiting for lagging nodes, which has a direct impact on system performance. Thus, Orchestra-Paxos achieves consensus provided that $[n'/2] + 1$ nodes have not failed, where $n' \in N$ and N is the set of all nodes in the system. Thus, even if the set of active acceptors contains only a single node, it is possible to solve the consensus problem in the system. Furthermore, the second phase uses the classic Paxos algorithm, which also guaranties that consensus is reached.

Orchestra-Paxos is not leader-oriented due to the fact that each proposer's value has the exact same probability of being chosen in the round of replication.

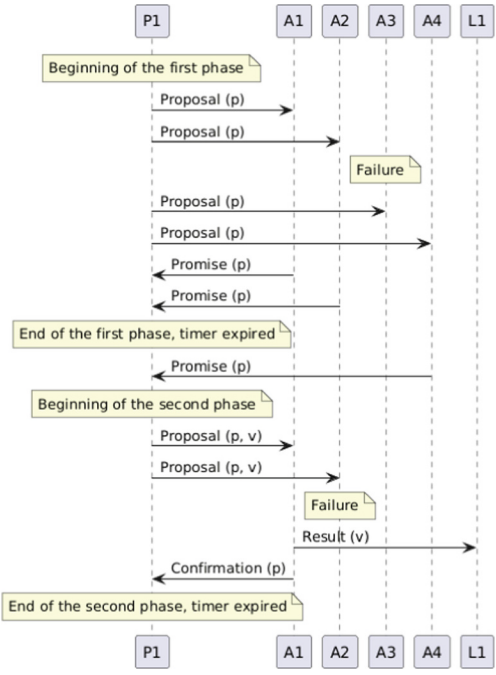


Fig. 4. Successful execution of Orchestra-Paxos

Figure 4 illustrates the execution of Orchestra-Paxos under the condition of failure and slowdown of some of the system's confirming nodes. In the first phase, it is shown that proposer $P1$ adds only $A1$ and $A2$ to the set of active acceptors. Acceptor $A3$ does not make it into the set of active acceptors because it has failed. Confirming node $A4$ is not included in the set of active acceptors, as it did not manage to respond with a confirmation to proposer $P1$ within the timer duration of the first phase. In the second phase, proposer $P1$ sends the proposal only to members of the set of active acceptors. Acceptor $A1$ successfully accepts the value v and reports this to the learner $L1$, then confirms its acceptance by sending a message to proposer $P1$. Acceptor $A2$ experiences a failure and therefore does not accept proposal v. Upon expiry of the second-phase timer, proposer $P1$ gathers a simple majority quorum (at least half of the acceptors), and thus successfully completes the round.

6 Preliminary Assessment

The Orchestra-Paxos algorithm was implemented in the *Go* programming language. To test the algorithm, the TCP/IP network protocol was used as the transport layer between the system's nodes. Furthermore, the following transport parameters were emulated:

1. The channel bandwidth was 1.25 megabits per second;
2. The round-trip time for each message was 20 ms;
3. The probability of message loss between nodes was configured using a parameter.

Each experiment used $3N$ nodes, where N is the number of proposers, acceptors and learners. Each proposer sent a new proposal once per second for 120 s. The timer for each phase of the algorithm was 100 ms. Furthermore, if the first phase needed to be repeated, the proposer waited between 15 and 20 s after an unsuccessful current phase. In the second phase the same method was used. This timer was chosen because the standard Go programming language library uses 10 connection recovery attempts at 1-s intervals. After 120 s had elapsed, the algorithm checked every 5 s whether the number of messages sent matched the number of messages received. If the proposals matched, the experiment was considered successfully completed, and the elapsed time was recorded. Messages from clients and proposers were always delivered. The experiments were conducted on a machine running *macOS*, with 48 GB of RAM and 14 cores.

Similar experiments were also conducted using Multi-Paxos. The master proposed N (N – number of acceptors and learners) proposals within 120 s.

Orchestra-Paxos was compared to Multi-Paxos on the following metrics:

1. Average time per round of the algorithm;
2. Waiting time for the completion of all replication rounds;
3. Number of rounds required to fulfill all client requests.

Table 1. Results of the experiments with three participants in each role

Algorithm	Probability of message loss %	Average duration of a single replication round sec	The time taken to complete all replication rounds sec	The number of rounds required to process all client requests
Orchestra-Paxos	0	0.35	213.277	603
Multi-Paxos	0	**0.3**	**121.276**	**399**
Orchestra-Paxos	0.01	0.39	245.27	629
Multi-Paxos	0.01	**0.33**	**130.730**	**401**
Orchestra-Paxos	5	0.36	255.313	702
Multi-Paxos	5	**0.29**	**135.323**	**468**
Orchestra-Paxos	30	0,58	826.239	**1432**
Multi-Paxos	30	**0,23**	**480.367**	1580
Orchestra-Paxos	60	0.53	**3003.12**	**5690**
Multi-Paxos	60	**0.21**	9452.631	45293

Table 2. Results of the experiments with five participants in each role

Algorithm	Probability of message loss %	Average duration of a single replication round sec	The time taken to complete all replication rounds sec	The number of rounds required to process all client requests
Orchestra-Paxos	0	0,19	420.332	2210
Multi-Paxos	0	**0.16**	**104.388**	**666**
Orchestra-Paxos	0.01	0.19	420.273	2197
Multi-Paxos	0.01	**0.16**	**120.282**	**779**
Orchestra-Paxos	5	0.2	445.321	2178
Multi-Paxos	5	**0.14**	**120.317**	**838**
Orchestra-Paxos	30	0,29	1030.845	3497
Multi-Paxos	30	**0.10**	**215.11**	**2078**
Orchestra-Paxos	60	0.51	**4072.325**	**7962**
Multi-Paxos	60	**0.09**	14997.527	178129

The results of experiments using Orchestra-Paxos and Multi-Paxos are shown in Table 1, with the number of proposers (except for the Multi-Paxos algorithm, where there is always a single proposer—the master node), acceptors and learners set to 3. Similarly, for 5 participants in each role, the results of the experiments are shown in Table 2. The results of the testing are averaged over three runs of each experiment. Moreover, the scalability of the algorithm is not the primary

focus of this work, so experiments were conducted on three and five participants in each of the roles in the system, which is generally a common replication factor.

The results of the experiment show that Multi-Paxos outperforms Orchestra-Paxos by a factor of three on average across the specified measures, assuming minimal message loss probability. Under high message loss probability, Orchestra-Paxos surpasses Multi-Paxos by a factor of twelve in terms of the number of rounds required and overall completion time. The improved technique is originally developed for a more flexible role allocation and, with more modifications, may perform better in scalable and fault-tolerant complicated network topologies.

An implementation of the Orchestra-Paxos algorithm in the Go programming language, along with test scenarios for verifying it and a comparison with the Multi-Paxos algorithm, are available on the GitHub version control system in the Orchestra-paxos repository [15].

7 The Disadvantages and the Future of the Algorithm

There are some disadvantages of the Orchestra-Paxos:

1. In the current implementation of the modernized algorithm, the system is in risk of segmentation;
2. The performance results of Orchestra-Paxos leave much to be desired, when the probability of message loss is minimal.

Future work will focus on extending the current algorithm by adding a group of replicated leaders that will check the availability of the system's nodes through a heartbeating mechanism, orchestrate the number of dynamic quorum participants and replicate data on revived nodes. The development direction aims to address the issue of system segmentation and ensure full consistency across the entire system. Moreover, adjusting the timer durations for each phase may positively impact the experimental results, potentially achieving performance comparable to Multi-Paxos or even surpassing it when the probability of message loss is minimal.

Lastly, after upgrading the algorithm, information about strict consistency and formal guaranties of the algorithm's correctness will be added to future research. In the present article, strict consistency is out of scope.

8 Conclusion

The research presented in this paper introduced Orchestra-Paxos, an enhanced consensus algorithm that replaces the fixed majority quorum requirements of traditional Paxos with a dynamic set of active participants. By utilizing timers to identify and exclude unresponsive nodes during the replication cycle, the algorithm ensures that consensus can be reached even when a majority of acceptors are temporarily unavailable. This approach effectively addresses the problem of

stalled progress in unstable networks, providing a robust foundation for high-availability distributed systems where the strict requirement for a fixed majority often leads to performance degradation or total system failure.

The experimental evaluation demonstrates that Orchestra-Paxos significantly enhances system resilience under adverse network conditions. Specifically, in environments with 60% message loss, the algorithm reduced the total completion time and the number of required replication rounds by an average factor of 12 compared to Multi-Paxos. Although Multi-Paxos maintains a performance advantage in stable network conditions due to its leader-oriented optimizations, Orchestra-Paxos proves to be a superior solution for large-scale, unpredictable environments. These results confirm that the flexibility of dynamic quorums allows for continued operations in scenarios where classic consensus models would otherwise reach a deadlock.

The Orchestra-Paxos algorithm has substantial stability features and optimisation potential, making it a compelling option for systems that demand high fault tolerance and predictability. A new generation of scalable, fault-tolerant, and high-performance consensus algorithms may now be developed thanks to dynamic quorum size, carefully chosen algorithm parameters, and pragmatic system architecture.

References

1. Lamport, L.: The part-time parliament. ACM Trans. Comput. Syst. **16**(2), 133–169 (1998)
2. Ongaro, D., Ousterhout, J.: In search of an understandable consensus algorithm. In: 2014 USENIX Conference on USENIX Annual Technical Conference (USENIX ATC'14), pp. 305–320. USENIX Association, Philadelphia, PA, USA (2014)
3. Baker, J., et al.: Megastore: providing scalable, highly available storage for interactive services. In: 5th Biennial Conference on Innovative Data Systems Research (CIDR 2011), Proceedings Conference Innovative Data Syst Res (CIDR), Asilomar, CA, USA, pp. 224–234 (2011)
4. Chandra, T., Griesemer, R., Redstone, J.: Paxos made live: an engineering perspective. In: 26th ACM Symposium on Principles of Distributed Computing (PODC 2007), pp. 398–407. Association for Computing Machinery, New York, NY, USA (2007)
5. Lakshman, A., Malik, P.: Cassandra: a decentralized structured storage system. ACM SIGOPS Operating Syst. Rev. **44**(2), 35–40 (2010)
6. MySQL Blog Archive. https://dev.mysql.com/blog-archive/the-king-is-dead-long-live-the-king-our-homegrown-paxos-based-consensus/. Accessed 10 Mar 2026
7. Gunawi, H.-S., et al.: Why does the cloud stop computing? Lessons from hundreds of service outages. In: 7th ACM Symposium on Cloud Computing (SoCC 2016), pp. 1–16. Association for Computing Machinery, Santa Clara, CA, USA (2016)
8. Camargos, L.-J., Schmidt, R.-M., Pedone, F.: Multicoordinated Paxos. In: 26th ACM Symposium on Principles of Distributed Computing (PODC 2007), pp. 316–317. Association for Computing Machinery, New York, NY, USA (2007)
pg

9. Moraru, I., Andersen, D.-G., Kaminsky, M.: There is more consensus in egalitarian parliaments. In: 24th ACM Symposium on Operating Systems Principles (SOSP 2013), pp. 358–372. Association for Computing Machinery, New York, NY, USA (2013)
10. Lamport, L.: Fast Paxos. Distrib. Comput. **19**(2), 79–103 (2006)
11. Howard, H., Malkhi, D., Spiegelman, A.: Flexible Paxos: quorum intersection revisited. In: 20th International Conference on Principles of Distributed Systems (OPODIS 2016), pp. 25:1–25:14. Dagstuhl Publishing, Madrid, Spain (2016)
12. Howard, H., Charapko, A., Mortier, R.: Fast flexible Paxos: relaxing quorum intersection for fast Paxos. In: 22nd International Conference on Distributed Computing and Networking (ICDCN 2021), pp. 186–190. ACM Digital Library, Nara, Japan (2021)
13. Fischer, M.-J., Lynch, N.-A., Paterson, M.-S.: Impossibility of distributed consensus with one faulty process. J. ACM **32**(2), 374–382 (1985)
14. Ryabinin, F., Gotsman, A., Sutra, P.: Making democracy work: fixing and simplifying egalitarian Paxos. In: 29th International Conference on Principles of Distributed Systems (OPODIS 2025), pp. 22:1–22:19. Dagstuhl Publishing, Iaşi, Romania (2025)
15. Orchestra-Paxos code repository. https://github.com/ACE-777/orchestra-paxos. Accessed 11 Feb 2026