Benedikt Beckermann, Mtr. Nr. 1599203

Instructor's Name: Prof. Sturm

Couse: SoSe 2024 Distributed Systems

17 September 2024

The Paxos Algorithm, Übungsblatt 3, SoSe 2024

1. Introduction

The Paxos algorithm is a consensus algorithm for distributed systems. It is still used in modern distributed systems, such as the key-value store FoundationDB [5], or the NoSQL multi model database management system OrientDB [4]. This paper describes the basic Paxos algorithm as in [2].

The paper is structured as follows: Section 2 explains the basic Paxos algorithm, as described by L. Lamport. Section 3 describes an implementation of the basic Paxos algorithm using the sim4da framework. Section 4 concludes the paper.

2. The Basic Paxos Algorithm

The goal of the basic Paxos algorithm [2] is to achieve a consensus between several sites for the value of a variable. Because it only needs a majority of the sites to agree upon a value, it can handle failures of up to $\lceil \frac{\#sites}{2} \rceil - 1$ sites.

The algorithm uses three different roles: The Proposers, the Acceptors, and the Learners. The Proposers handle requests to set the value for the consensus, the Acceptors decide individually if they accept an incoming value, and the Learners learn the consensus value once a majority of acceptors agree upon one. While these roles can and often are combined, the following description of the algorithms will treat them as separate roles.

The algorithm is divided into the Prepare phase and the Accept phase. In the Prepare phase, a Proposer request a majority of Acceptors to promise that they will not accept any proposal number lower than the proposal number of the new request. Once the proposer receives answers of a majority of Acceptors, the Accept phase starts in which the Proposer request a majority of Acceptors to accept a suggested value. If this second majority is reached, the suggested value is the consensus value.

The Prepare phase starts with a Proposer selecting a unique proposal number *n* that is higher than any of its used proposal numbers before, and sending a *prepare* message containing *n* to a majority of Acceptor nodes. The uniqueness of the proposal number can be achieved by assigning an unique id between 0 and the number of Proposers to each Proposer, and constructing the next unique proposal number at each Proposer by starting with the individual id and adding the number of proposers to it for the next proposal number. If you have three Proposers, this would lead to Proposer 0 using the numbers 0, 3, 6, 9, …, Proposer 1 the numbers 1, 4, 7, 10, …, and Proposer 2 using the numbers 2, 5, 8, 11, … . If an Acceptor receives a *prepare* message containing the proposal number *n*, it tests if *n* is higher than any proposal number that it received before. If it is higher, it sends a promise to not accept any proposal number lower than *n*, and if the Acceptor already accepted a value (see Accept phase), it also sends the proposal number of the highest-numbered accepted proposal together with its value. If *n* is not higher than any previously received proposal number, the incoming request is ignored.

The Accept phase starts once a Proposer received Promise messages as answers to its prepare messages from a majority of Acceptors. If this happens, the Proposer sends an accept message to a majority of Acceptors containing the proposal number *n* and a proposed value. The proposed value is the value of the highest-num-

bered accepted proposal received via the promise messages from the Acceptors, or, if no accepted proposal was send, any value chosen by the Proposer. If an Acceptor receives an *accept* message, it confirms the proposed value, unless it already responded to an *prepare* message with a higher-numbered proposal. If it already responded to a *prepare* message with a higher-numbered proposal, the incoming *accept* message is ignored. Once a Proposer received confirmations from a majority of Acceptors, the value send is the consensus value. The Proposer can now inform all Learners about the decision.

The Paxos algorithm relies upon the use of proposal numbers to differentiate and decide upon values. While two (or more) Proposers can get a majority of Acceptors to promise not to accept any number lower than the send proposal numbers, if both proposers then send their *accept* messages, only the Proposer with the higher proposal number can achieve a majority of Acceptors to confirm the accept messages.

3. Implementation

To demonstrate the Paxos algorithm in action, it is implemented using the sim4-da-v2[1] framework. The code of the Paxos implementation can be found in a public git repository[2].

The implementation consists out of five classes: Proposer, Acceptor, Listener, Client, Messages, and BasicPaxos. The Proposer, Acceptor and Listener classes represent the three different roles in the basic Paxos Algorithm, as discussed in Section 2. They react to incoming messages according to their roles and log incoming messages and and the chosen responses into the Terminal output. An instance of the Proposer class starts a proposal when it receives a PROPOSAL message. The Client class repre-

[1] https://github.com/oxoo2a/sim4da-v2

[2] https://github.com/greenBene/24S-DS-Paxos

sents a client that wants to propose its name as value for the consensus. This simulates situations where a leader needs to be agreed upon. When the simulation starts, the client waits a variable amount of time (between 0 and 500 ms in the current implementation), before it sends a PROPOSE message to a Proposer containing its name as suggested value. The class Messages contains static cloneable instances of the messages sends between the other classes. The BasicPaxos class contains the main method of the implementation. When started, it creates a number of Proposer, Acceptor, Listener, and Client objects, as defined by constants, and starts the simulation.

An example output of a run of the described basic Paxos implementation can be found in the repository in the file *output.txt*. Its run is setup to use four client nodes, four proposal nodes, seven acceptor nodes, and five learner nodes.

4. Conclusion

This paper describes the basic Paxos algorithm and an implementation of the algorithm using sim4da-v2 to demonstrate the algorithm.

The implementation is designed to closely represent the original algorithm as described by Lamport in [2], and thus does not contain common optimizations. To make use of the implementation in more complex situations, there are several possible optimizations. E.g., the described implementation only supports the consensus on a single value is supported. Many use cases require the consensus of multiple values. This optimization, called Multi Paxos, is described in the original paper by Lamport [3] as “The Multidegree Parliament”.

Sources

[1] L. Lamport, ‘Fast Paxos’, *Distributed Comput.*, vol. 19, no. 2, pp. 79–103, 2006, doi: 10.1007/S00446-006-0005-X.

[2] L. Lamport, ‘Paxos made simple’, ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001), pp. 51–58, Dec. 2001.

[3] L. Lamport, ‘The Part-Time Parliament’, *ACM Trans. Comput. Syst.*, vol. 16, no. 2, pp. 133–169, 1998, doi: 10.1145/279227.279229.

[4] D. Ritter, L. Dell’Aquila, A. Lomakin, and E. Tagliaferri, ‘OrientDB: A NoSQL, Open Source MMDMS’, in *Proceedings of the The British International Conference on Databases 2021, London, United Kingdom, March 28, 2022*, H. Pirk and T. Heinis, Eds., in CEUR Workshop Proceedings, vol. 3163. CEUR-WS.org, 2021, pp. 10–19. Accessed: Jun. 04, 2024. [Online]. Available: https://ceur-ws.org/Vol-3163/BICOD21_paper_3.pdf

[5] J. Zhou *et al.*, ‘FoundationDB: A Distributed Unbundled Transactional Key Value Store’, in *Proceedings of the 2021 International Conference on Management of Data*, Virtual Event China: ACM, Jun. 2021, pp. 2653–2666. doi: 10.1145/3448016.3457559.