Software Engineering for Consensus

Harald Ng

Aug 2023

1 Introduction

Over the last two decades, the daily lives of humans have become increasingly dependent on replicated services in the cloud. Digital documents, photos, and wallets are stored in distributed databases that appear consistent and available from any device. One of the key challenges in building such highly available distributed systems is to coordinate the servers of the system so that they remain consistent even under failures such as network disconnections and server crashes. The standard approach to achieve this is via state machine replication, where the application state gets replicated on multiple servers such that even if some servers are down or disconnected, the state is still consistent and accessible via the other servers. In this way, the replicated state machine as a whole provides the view of a single consistent system state for the users.

State machine replication is typically implemented using a consensus algorithm such as Raft [8] and Paxos [5]. These algorithms replicate a log on the servers, which acts as a single source of truth; all changes to the appliaction state are placed in the log, and all servers execute operations according to the log order to maintain consistency. Consensus algorithms have formally proven correctness and have also gained wide adoption in practice in the form of distributed databases [1, 9, 2] and general-purpose coordination services such as etcd and ZooKeeper [3].

2 Robert's lectures

2.1 Static and dynamic techniques for verification and validation

Consensus protocols are notoriously known for being complicated, and specific protocols have been designed with the main objective of being understandable [8]. Thus, a major challenge when implementing the protocols in practice is to make a source code and API that are intuitive and understandable. Various static techniques can be applied for this purpose. Automatic tools such as lints can aid with consistent naming and API designs. Code reviews can also help with that and can additionally be useful for ensuring that the implementation follows the formal specification of the protocols.

However, real implementations typically vary from the formal specification or pseudo-code of the protocol. The programming language and platform can influence the design and structure of the implementation, while performance optimizations such as multi-threading may lead to deviations from the high-level pseudo code. Therefore, dynamic techniques such as testing are important. Unit tests can be applied to individual components such as leader election and log replication, while more advanced tests that simulate failures can be used for integration tests.

2.2 Behavioral Software Engineering

Behavioral software engineering (BSE) aims to make software engineering effective by focusing on humans. In the context of engineering for consensus protocols, an important concept in BSE is team diversity. Consensus protocols are important in various metrics: safety, performance, and liveness. A team may have value diversity where different people consider some of the metrics more important than the others, which can make the team disagree over what to focus on and hinder progress. Different sources or power may affect what direction the team will be focused on. This is also dependent on the organization of the team. For instance, in a team with a more flat hierarchy, information power, i.e., the ability to convince others, might be the deciding factor. In other organizations, it may require a formal team leader with legitimate power to decide the team's direction.

3 DevBots for Consensus

DevBots are typically used for the automation of tasks using human-like traits. In the context of consensus, one possible use case of DevBots is to make the source code intuitive and easy to read. As previously mentioned, consensus protocols are typically considered hard to understand, and real implementations further deviate from the specification, making the source code difficult to read and error-prone. DevBots can help improve the naming, structure, and comments of the source code to make it more intuitive for developers. Another use case for DevBots in the context of consensus is for documentation and learning. The DevBot can answer developer's questions, such as which parts of the protocol correspond to where in the actual implementation. This can help developers understand the theoretical concepts better and ease the introduction for developers who are not consensus protocol experts.

4 Software Engineering for Consensus

4.1 Architecture and Design

The architecture and design are two critical stages in software engineering. The software architecture lays down a blueprint for the system that defines how different components and parts will interact. The design typically delves deeper into the implementation details and focuses on how the different functionalities of a system will be realized. These could include what kind of algorithms, interfaces, and data structures that are used.

The architecture of a replicated state machine or consensus system is a highly debated one. A modular architecture was first introduced in the 1990s and early 2000s [5] where liveness and safety are decoupled such that different protocols could be used for them. In 2014, the Raft [8] protocol was introduced with a monolithic architecture that combined both liveness and safety for the sake of understandability and has become widely adopted since. However, the popularity of Raft has not translated into many popular libraries. Instead, companies and open source systems typically implement their own versions based on the formal specification of it. This can be attributed to its monolithic design which makes it not possible to modify parts of the protocol, such as leader election. The monolithic design also proved to have sub-optimal performance and resilience. To tackle this problem, we published Omni-Paxos [6], a consensus system that goes back to the modular architecture. Omni-Paxos consist of three distinct protocols specifically designed for leader election, replication, and reconfiguration. The modular architecture allows each of the protocols to be optimized for one specific purpose which allows it to improve the resilience and reconfiguration performance. Furthermore, the modular design enabled us to build a user-friendly consensus library based on Omni-Paxos[1]. Since the architecture is modular in theory, we can provide interfaces that make changing out individual components (such as the network and storage) easy for the user. From my experience, this modularity has also shown to be advantageous for the devel-

[1]omnipaxos.com

opment progress. Developers can work on different components in parallel without conflicts and cross-dependent complexities.

4.2 Sustainability

Sustainability in software engineering refers to the practice of developing and maintaining software that minimizes negative impacts on the environment and society. As consensus protocols provide replicated data that is highly available, they are typically deployed as long-running distributed systems. Thus, an important aspect that should be considered is the energy consumption of running the system. For instance, similar protocols that also tolerate byzantine failures (BFT) can be found in cryptocurrencies that use proof-of-work. These sorts of systems have been shown to consume more energy than whole countries [4]. Although non-BFT protocols typically don't need to consume the same amounts of energy, it is still highly important to design protocols to become efficient since they become software that runs for a long time on multiple servers across the globe. One open problem is that the consensus protocols are theoretical and do not consider the characteristics of the deployment of the actual software. This implies that many opportunities to make the protocol more efficient are not exploited. To this end, we have published a paper on how to make consensus replication more efficient by reducing the data amounts sent over the network through learning data trends [7]. Future work can also exploit other characteristics, such as heterogeneous server configurations where backup servers use fewer resources. This idea can be further developed to possibly become a commercial product, where the system continuously monitors itself and automatically scales in or scales down the system when the workload is low. In this way, we can have a self-driving replication service that both saves money and reduces its environmental impact by adapting to the workload.

5 Future Trends for SE with AI/ML

Existing consensus libraries are typically considered hard to use for people unfamiliar with the protocols in theory. Ideally, new libraries with more familiar and intuitive APIs that hide the complexities of consensus should address this problem. However, AI/ML can help improving the usability of the existing libraries. Large language models (LLMs) can be trained on specific libraries to provide interactive chatbots that help users use the library properly. In general, LLMs will continue to help generate code; thus, developers will increasingly focus more on prompt-engineering and declaring what kind of program they want rather than writing the code that describes how the program should run. A consequence of this is that the manual work of developers will more likely shift to debugging and correcting generated code rather than writing code from scratch.

From a systems researcher's perspective, these automation tools for software engineering will also accelerate the progress of the research itself. Prototype systems can be built faster, and benchmarks can easier be automatically implemented to compare with other systems easily.

References

[1] James C Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, Jeffrey John Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, et al. Spanner: Google's globally distributed database. *ACM Transactions on Computer Systems (TOCS)*, 31(3):1–22, 2013.

[2] Dongxu Huang, Qi Liu, Qiu Cui, Zhuhe Fang, Xiaoyu Ma, Fei Xu, Li Shen, Liu Tang, Yuxing Zhou, Menglong Huang, et al. Tidb: a raft-based htap database. *Proceedings of the VLDB Endowment*, 13(12):3072–3084, 2020.

[3] Patrick Hunt, Mahadev Konar, Flavio Paiva Junqueira, and Benjamin Reed. Zookeeper: Wait-free coordination for internet-scale systems. In *USENIX annual technical conference*, volume 8, 2010.

[4] Anh Ngoc Quang Huynh, Duy Duong, Tobias Burggraf, Hien Thi Thu Luong, and Nam Huu Bui. Energy consumption and bitcoin market. *Asia-Pacific Financial Markets*, 29(1):79–93, 2022.

[5] Leslie Lamport et al. Paxos made simple. *ACM Sigact News*, 32(4):18–25, 2001.

[6] Harald Ng, Seif Haridi, and Paris Carbone. Omni-paxos: Breaking the barriers of partial connectivity. In *Proceedings of the Eighteenth European Conference on Computer Systems*, pages 314–330, 2023.

[7] Harald Ng, Kun Wu, and Paris Carbone. Unicache: Efficient log replication through learning workload patterns. 2023.

[8] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 305–319, Philadelphia, PA, June 2014. USENIX Association.

[9] Rebecca Taft, Irfan Sharif, Andrei Matei, Nathan VanBenschoten, Jordan Lewis, Tobias Grieger, Kai Niemi, Andy Woods, Anne Birzin, Raphael Poss, et al. Cockroachdb: The resilient geo-distributed sql database. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1493–1509, 2020.