Challenges in Verifying Strict Serializability in Distributed Datastores

by

Anuja Chandrashekhar Wani

May 13, 2024

A thesis submitted to the

Faculty of the Graduate School of

the University at Buffalo, The State University of New York

in partial fulfilment of the requirements for the

degree of

Master of Science

Department of Computer Science and Engineering

Table of Contents

List of Figures

Abstract

Strict serializability or external consistency is considered the gold standard for modern day distributed datastores ensuring data consistency, reliability, and robustness. While many systems claim to have achieved this using diverse mechanisms, relying solely on such claims can leave the clients vulnerable to false assumptions. Consequently, verifying these datastores becomes very crucial.

This work argues that clock-based serialization techniques are insufficient for verifying strict serializability consistency. It shows how prevailing system verifiers' assumptions about timestamps often overlook scenarios that do not conform to strict serializability. We present a proof demonstrating the infeasibility of constructing a clock-based serialization verifier for a strictly serializable system. Grounded on the assumption of such a verifier's existence, we construct our proof through logical deduction.

Additionally, the work explores various approaches to constructing a checker for verifying a strictly serializable system. Several existing checkers rely on graph-based techniques for verification of lower-level consistency which are unsuitable for verifying strictly serializable systems. This paper proposes mechanisms for identifying vulnerabilities of existing systems claiming strict serializability.

Acknowledgments

I am deeply grateful to Dr. Haonan Lu for his invaluable guidance, mentorship, and encouragement throughout the course of this thesis. His expertise, insightful feedback, and unwavering support have been instrumental in shaping the direction of this research and fostering my academic growth.

I would also like to extend my sincere appreciation to my colleagues and collaborators who have contributed to this project. Their dedication, collaboration, and shared enthusiasm have enriched the research process and contributed to its success.

Furthermore, I am thankful to my friends and family for their unconditional support, understanding, and encouragement. Their belief in me has been a constant source of motivation, and I am truly grateful for their presence in my life.

Lastly, I would like to acknowledge the institute, University at Buffalo for providing the resources and facilities to support this research endeavor. Their support has been essential in enabling the completion of this thesis.

Chapter1

Introduction

Modern systems extensively rely on distributed datastores to effortlessly scale and manage the escalating data volumes. This infrastructure allows users to seamlessly expand their storage capacity to adapt to evolving business demands while maintaining optimal system performance and reliability. Distributed datastores function as interconnected networks of computers, storing data across multiple nodes. To ensure data availability, even in the event of node failures, these systems employ replication mechanisms.

Strict serializability, often equated with External Consistency, stands as the most robust form of transactional consistency.[1] It imposes a stringent real-time order constraint on serializability, dictating that the total ordering of any two transactions must adhere to real-time sequencing. For instance, if transactions *tx1* and *tx2* are to be strictly serializable, *tx1* must commit before *tx2* commences, thus establishing a real-time order denoted as $tx1 \rightarrow tx2$. In essence, a system adhering to strict serializability guarantees that any transaction executed will invariably perceive the effects of all preceding transactions in the global serial order.[2]

Enforcing strict serializability not only ensures data consistency and integrity but also plays a crucial role in managing concurrency in distributed systems. By enforcing strict serializability, the system maintains a consistent order of transactions, preventing conflicts and ensuring that concurrent transactions do not interfere with each other. This approach helps to mitigate anomalies such as lost updates, dirty reads, and non-repeatable reads, thereby promoting reliable and coherent data interactions across distributed components.[3]

Verifying the correctness of a system is very important from the implementation point of view. Clients typically operate under the assumption that the system is functioning correctly during the development phase i.e. they assume the system to provide claimed consistency and concurrency. Consistency in a distributed system is ensuring all the users viewing the data should be able to see the same data irrespective of their machines or geographic locations. To put it in simple terms, consistency means all future reads should reflect the value of the writes completed before it. However, the absence of stated consistency within the system can introduce inaccuracies or errors that compromise its integrity and effectiveness. Therefore, meticulous attention to verifying the correctness of the system's behavior is essential. This involves rigorous testing, validation, and verification procedures to identify and rectify any discrepancies or inconsistencies before deployment. By prioritizing thoroughness in the verification process, developers can enhance the robustness and trustworthiness of the system, thereby mitigating potential risks and ensuring optimal performance in real-world scenarios.

Due to the complexity of distributed systems, verifying them is a tedious process. It can always lead to a failure for complete verification, making one assume its correctness. From our references to NCC[4], we see that notable systems such as DrTM and TAPIR are some of the systems stated to be strictly serializable and which fail to provide that. This can cause a huge issue for applications which use these systems assuming their correct operations. A prevalent pitfall contributing to this discrepancy is Timestamp Inversion, a phenomenon prevalent in many systems claiming strict serializability. Identifying such systems is pivotal in addressing the potential ramifications for applications that rely on them.

There are two ways to verify the correctness of a system, using a formal verification technique such as the TLA+, or using verification in the wild by generating unit tests and integration tests. While formal verification is expensive, it is exhaustive however it is useful only to verify the protocols or algorithms. TLA+ doesn't help verify the implementation hence cannot identify faults in the implementation of systems which deviate from their

algorithms.

Testing in the wild also includes fault injection, where a particular scenario is generated by some tools to verify the results. This is particularly useful when one specific case is to be tested rigorously. Jepsen[2] is one of such well known tools useful for testing. Jepsen tests distributed systems by running hundreds of tests for data consistency and fault tolerance.

Serialization graphs serve as a valuable tool for verifying serializability within systems. By constructing these graphs from transactional dependencies, they can effectively identify conflicts within transactions. If cycles exist within the graph, it indicates conflicts among transactions, rendering the schedule non-serializable. Conversely, the absence of cycles confirms the serializability of the schedule.[5]

While serialization techniques are invaluable for verifying strict serializability, they fall short in accurately validating this property due to its imposition of real-time ordering constraints. Current verifiers rely on timestamps to order transactions in real-time, but our research reveals this approach's inadequacy, as it can yield incorrect assessments of system consistency.

In this thesis, our objective is to demonstrate the infeasibility of achieving a strictly serializable system verifier using clock based serialization techniques. Additionally, we endeavor to provide a formal proof illustrating this impossibility. We also suggest some designs to construct a strict serializability verifier.

Chapter2

Background

Lamport[6] states atomicity as a fundamental assumption in a system, where an atomic operation is said to be an operation whose execution is performed as an indivisible action. If there are two operations A and B, if A precedes B then it can influence the execution of operation B. An operation is either a read or a write and using a global clock we assign start time and end time to all the operations. [7]

In distributed systems, the essence lies in making strategic trade-offs. Similarly, when considering consistency models, we encounter a balance between the concurrency and ordering of operations. Put differently, it's a trade-off between operational efficiency and accuracy.[8]

When testing distributed systems, we are essentially testing the validity of some properties. All these properties are essentially the intersection of two important properties:

1. Safety property - Something bad will never happen e.g. replicas are strongly consistent

2. Liveness property - Good things will eventually happen e.g. eventual consistency

Based on these properties, various checkers are devised to verify different levels of consistency guarantees.[7]

Verifying distributed systems is a complicated process. It is integral to ensuring the reliability of applications built upon them. Numerous testing methods are employed to assess different levels of consistency. Serializability, which dictates that transactions should appear

to occur in a single sequential order, doesn't impose real-time constraints[9]. Tools like COBRA[10] utilize directed acyclic graphs (DAGs) to detect cycles, which signal violations of serializability.

While these techniques are valuable for verifying serializability, they fall short in capturing the real-time ordering between transactions. Strict serializability imposes a constraint on the real-time execution order of transactions. Real-time ordering dictates that if transaction *tx2* begins after transaction *tx1* concludes, *tx2* must be sequenced after *tx1* in the total ordering, meaning transactions occur in the order they are received by the system. However, the system lacks the means to precisely determine when a transaction begins in real time. Therefore, alternative mechanisms are required to identify real-time transaction ordering and construct graphs that may aid in detecting violations of strict serializability.

Chapter3

Solution

Physical clocks alone cannot be useful when ordering events in a system. Two clocks can never run at the same rate and drift further apart. This forms the basis of our study to show why clock based solutions to order events can not be relied upon. To explain this we consider an example from[11], suppose a person issues some request A on computer A, calls a friend to issue request B on computer B. It is very much possible for the system to give request B a lower timestamp and order request B before request A. Since the system has no way of knowing which request was preceding the other since that information is external to the system.

The solution suggested to avoid such cases is 1) Introduce the necessary information to the system e.g. users can specify the timestamp of their requests (B specifies timestamp of Tb as something later than Ta, this introduces users the responsibility for avoiding anomalous behavior ; 2) Construct a new system of clocks such that for any events $a \rightarrow b$, clock value for $a <$clock value for b.

From NCC[4], we found the concept of Timestamp Inversion Pitfall that has impacted and motivated our research. It states that timestamp based techniques sometimes fail to guard against a total order that violates the real time ordering. An example is if we have 3 transactions *tx1*, *tx2* and *tx3*; *tx2* starts after *tx1* finished so using real time ordering we should have *tx1* $\rightarrow$*tx2*. *tx3* on the other hand is a multishard transaction interleaving with *tx1* and *tx2*. Time stamps of the 3 transactions as 10,5,7 giving us an order *tx2*$\rightarrow$*tx3*$\rightarrow$*tx1*

which violated the real time ordering between *tx1* and *tx2*.

This pitfall has impacted many existing systems such as TAPIR and DrTM which are claimed to be strictly serializable. There are many other instances where incorrect verification of systems have been identified. Hence, a thorough verification of the system is essential to have reliance on any system concurrency claims.

To prove our solution we take the approach of proof by contradiction.

3.1 Conjecture

3.1.0.0.1 Conjecture: It is not possible to verify strict serializability consistency using clock based serialization graphs

3.2 Proof

3.2.0.0.1 Proof: To demonstrate the impossibility of having a strictly serializable system verifier which uses clock based serialization graphs, we employ proof by contradiction.

Firstly, when the verifier constructs serialization graphs, there can be two instances:

1. There are no cycles in the graph and the verifier concludes the system is strictly serializable

2. There are cycles in the graph and the verifier concludes the system is not strictly serializable

We have 2 transactions *t1* and *t2* and two servers *S1* and *S2*.

Consider the first scenario: ref fig.3.1

1. Let *t1* and *t2* be two transactions with real-time ordering relationship, denoted as *t1*$\rightarrow$*t2*.

2. Transaction *t1* executes on server *S1* while *t2* executes on *S2*.

3. However, there is a clock skew between the servers and hence they assign the two transactions timestamps based on these skews.

4. *t1* is assigned timestamps 3 for the start and 10 for the end, and *t2* is assigned timestamps 5 for start and 13 for end.

5. The verifier constructs a serialization graph based on these timestamps.

6. However, because of the overlapping timestamps, the verifier mistakenly assumes the transactions are concurrent.

7. Consequently the verifier might order them as *t1*→*t2* or *t2*→*t1*.

8. This erroneous assumption leads to a scenario where a cycle that should have existed in the graph is missed.

9. As a result the verifier incorrectly concludes the system is strictly serializable showing an example of a failure of the verifier to accurately verify the system.

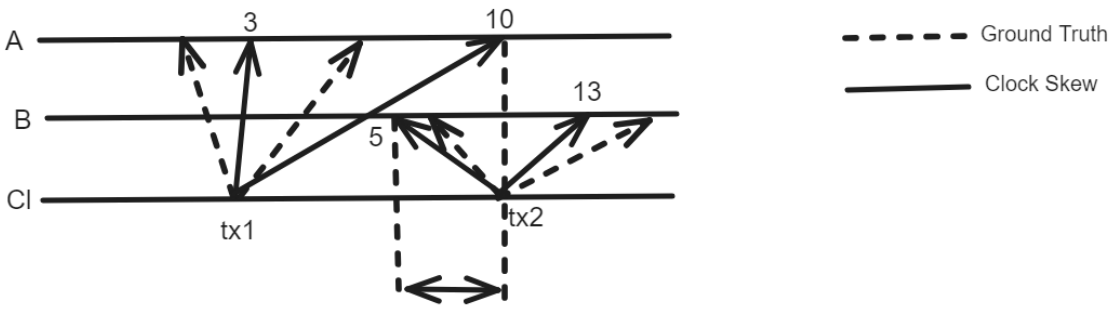

Figure 3.1: Case1

Now, consider the second scenario:ref fig.3.2

1. Let t1 and t2 be two concurrent transactions with no real-time ordering relationship.

2. Transaction t1 executes on server S1 while t2 executes on S2.

3. However, there is a clock skew between the servers and hence they assign the two transactions timestamps based on these skews.

4. t1 is assigned timestamps 2 for the start and 6 for the end, and t2 is assigned timestamps 7 for start and 12 for end.

5. The verifier constructs a serialization graph based on these timestamps.

6. However, because of the non overlapping timestamps, the verifier mistakenly assumes the transactions have a real-time ordering such that t1-¿t2.

7. Consequently the verifier orders them as t1-¿t2.

8. This erroneous assumption leads to a scenario where a cycle is created among concurrent transactions.

9. As a result the verifier incorrectly concludes the system is not strictly serializable showing another example of a failure of the verifier to accurately verify the system.

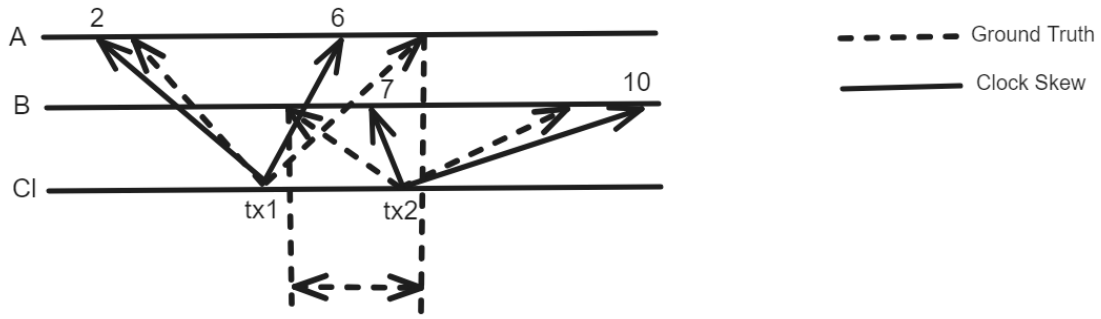

Figure 3.2: Case2

Based on these two scenarios, we can see how incorrect timestamps caused due to clock skews might result in incorrect verification of the system. Hence our assumption is proved wrong. Therefore, through this contradiction, we conclude that constructing a strictly serializable system verifier using clock based serialization graphs, is not feasible.

Chapter4

Approaches

Our approach began with an exploration of existing verifiers for strictly serializable systems, revealing a scarcity in the field. Most verifiers focus solely on verifying serializability, neglecting the stricter criteria of strict serializability. Among those that do attempt to verify strict serializability, reliance on timestamping in serialization techniques is prevalent. However, these techniques fail to scrutinize the correctness of the timestamps assigned by the system.

Our conjecture and subsequent proof illuminated the shortcomings of timestamp-based verification methods. This prompted us to explore alternative approaches that circumvent the pitfalls associated with timestamping.

Initially, we considered implementing a centralized tracker to oversee transaction ordering by assigning unique transaction IDs. This approach aimed to facilitate comparison of transactions across servers to identify violations of strict serializability. However, concerns arose regarding the potential bottleneck at the master server responsible for generating these IDs.

Further analysis led us to scrutinize scenarios where strict serializability fails. For instance, we examined a scenario where transaction A commits a write operation before notifying transaction B, which subsequently reads the updated data. In such cases, adherence to real-time ordering mandates that transaction A precedes transaction B. We conceptualized these communication pathways between transactions as external channels that dictate real-time ordering.

Our goal was to internalize these external channels within the system architecture. However, achieving this proved challenging, as these channels operate beyond the system's control. We explored the possibility of generating comprehensive test cases to identify instances of strict serializability failure, particularly focusing on timestamp inversion pitfalls. While this approach offers a degree of flexibility, its efficacy may be limited by the inability to exhaustively cover all edge cases.

Additionally, we entertained the idea of constructing a secondary network that operates faster than the underlying network, facilitating rapid communication between nodes to preserve real-time ordering. While promising, this approach necessitates further research to determine the feasibility of constructing such a network and ensuring its consistent performance compared to the primary network.

Chapter5

Related Work

Research in distributed systems has identified challenges in achieving strict serializability, particularly in verifying systems for this property. A significant study known as the Timestamp-Inversion Pitfall explores the limitations of timestamp-based techniques in ensuring strict serializability. This work emphasizes the need to minimize communication overhead for naturally consistent transactions and introduces the Natural Concurrency Control (NCC) [4] approach to address these challenges. By highlighting the pitfalls associated with timestamp-based verification, this research contributes valuable insights into the complexities of ensuring strict serializability in distributed systems.

Furthermore, investigations into consistency models, such as those conducted at Facebook [12], provide additional context on the impact of weaker consistency guarantees on system behavior. Understanding how consistency models affect programming complexity and user experience is crucial for designing effective verification mechanisms for strict serializability. The examination of Facebook's TAO storage system and its consistency models sheds light on the trade-offs involved in ensuring consistency while maintaining system performance.

Additionally, insights from industry practices, such as those observed in Google's Spanner database [3], offer practical perspectives on enforcing strict serializability in real-world scenarios. By exploring how Spanner utilizes clock skews and the TrueTime API, researchers can gain practical insights into the challenges of implementing clock-based solutions for strict serializability verification. Synthesizing insights from research endeavors and industry

implementations can inform the development of robust verification mechanisms for strict serializability, addressing the inherent challenges posed by real-time ordering constraints in distributed systems.

Chapter6

Conclusion

Strict serializability, also known as external consistency, represents the gold standard of data consistency in modern day datastores. It offers the invaluable advantage of executing transactions seamlessly, akin to operating on a single machine. This standard not only streamlines the development process but also avoids issues such as stale reads, thereby enhancing both user experience and developer productivity. However, the verification of systems purporting to adhere to strict serializability is of great importance. Any misstep in ensuring system consistency can lead to significant errors and inaccuracies.

While verifying consistency up to serializability presents relatively straightforward approaches, the verification of strict serializability remains an area yet to be fully explored. Our research has revealed significant shortcomings in clock-based serialization graph verifiers, highlighting their inadequacy in accurately assessing the strict serializability of systems.

In our attempt to address this gap, we have proposed various approaches aimed at designing a robust verifier for strict serializability. It is important to note that these approaches are not foolproof solutions but rather preliminary suggestions for further exploration. The development of an effective verifier for strict serializability remains an open area for future research, requiring deeper investigative and innovation.

Bibliography

[1] *Serializability vs "Strict" Serializability: The Dirty Secret of Database Isolation Levels*. https://fauna.com/blog/serializability-vs-strict-serializability-the-dirty-secret-of-database-isolation-levels.

[2] *Jepsen*. https://jepsen.io/consistency/models/strict-serializable.

[3] *Spanner under the hood: Understanding strict serializability and external consistency*. https://cloud.google.com/blog/products/databases/strict-serializability-and-external-consistency-in-spanner.

[4] Haonan Lu et al. "NCC: Natural Concurrency Control for Strictly Serializable Datastores by Avoiding the Timestamp-Inversion Pitfall". In: *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. Boston, MA: USENIX Association, July 2023, pp. 305–323. ISBN: 978-1-939133-34-2. URL: https://www.usenix.org/conference/osdi23/presentation/lu.

[5] *Serializability*. https://www.mydistributed.systems/2020/08/distributed-transactions-serializability.html.

[6] Leslie Lamport. "Time, clocks, and the ordering of events in a distributed system". In: *Commun. ACM* 21.7 (1978), 558–565. ISSN: 0001-0782. DOI: 10.1145/359545.359563. URL: https://doi.org/10.1145/359545.359563.

[7] Eric Anderson et al. "What consistency does your key-value store actually provide?" In: *Proceedings of the Sixth International Conference on Hot Topics in System Dependability*. HotDep'10. Vancouver, BC, Canada: USENIX Association, 2010, 1–16.

[8] *Consistency Guarantees in Distributed Systems Explained Simply*. https://kousiknath.medium.com/consistency-guarantees-in-distributed-systems-explained-simply-720caa034116.

[9] *Linearizability vs Serializability*. https://ajaygupta-spark.medium.com/linearizability-and-vs-serializability-in-distributed-databases-9da2462589d.

[10] Cheng Tan et al. "Cobra: Making Transactional Key-Value Stores Verifiably Serializable". In: *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, Nov. 2020, pp. 63–80. ISBN: 978-1-939133-19-9. URL: https://www.usenix.org/conference/osdi20/presentation/tan.

[11] *On Interprocess Communications*. https://lamport.azurewebsites.net/pubs/interprocess.pdf.

[12] Haonan Lu et al. “Existential consistency: Measuring and understanding consistency at Facebook”. In: *Proceedings of ACM Symposium on Operating Systems Principles (SOSP)*. Oct. 2015.