CHAIR OF DECENTRALIZED INFORMATION SYSTEMS & DATA MANAGEMENT

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Computer Science

Concurrent Range Locking

Thua-Duc Nguyen

CHAIR OF DECENTRALIZED INFORMATION SYSTEMS & DATA MANAGEMENT

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Computer Science

Concurrent Range Locking

Author:	Thua-Duc Nguyen
Supervisor:	Prof. Dr. Viktor Leis
Advisor:	Lam-Duy Nguyen
SubmissionDate:	19.08.2024

Acknowledgments

I would like to express my sincere gratitude to my dedicated supervisor Prof. Dr. Viktor Leis and advisor Lam-Duy Nguyen, for their unwavering support and guidance. They have consistently provided support and motivation, even amid their busy schedule.

My heartfelt appreciation to my girlfriend, Thuy-Trang Nguyen, for standing by me through the challenges of my thesis and undergraduate journey. She has consistently provided support and motivation.

Eventually, I would like to send a special acknowledgment to my family and friends whose encouragement has been a source of strength and an integral part during the course of my studies.

I would not have been able to accomplish this work without the support of each of these people.

Abstract

Effective resource access management is pivotal in modern computing systems such as filesystems, operating systems, and databases. Traditional single-locking methods often lead to bottlenecks and hinder parallelism among writers. Range locks have emerged as a promising solution to this issue by dividing shared resources into smaller segments, allowing concurrent access and enhancing system efficiency.

This thesis investigates scalable range-locking techniques motivated by the increasing interest in replacing traditional locking mechanisms, such as the mmap_lock in the Linux kernel, with more refined approaches. By focusing on critical ranges rather than coarse-grained locking, range locks facilitate concurrent transactions, thereby reducing memory overhead and improving throughput.

The research builds upon existing work in scalable range locks, exploring implementations based on skip and linked lists as alternatives to interval trees. A new concurrent range lock design is proposed, leveraging a probabilistic concurrent skip list, which offers efficient insertion and lookup operations while addressing contention points inherent in previous approaches.

The proposed range lock mechanism will be evaluated based on performance, correctness, and comparison with existing state-of-the-art solutions. Performance testing under varying loads and concurrent accesses will provide insights into the scalability and efficiency of the proposed mechanism. Ensuring data consistency and correctness, especially in scenarios involving overlapping data ranges and concurrent operations, will be a crucial aspect of the evaluation process. Additionally, a comparative analysis will highlight the advantages and potential trade-offs of the proposed solution against existing approaches.

The expected outcome of this research is the development of a scalable range lock that outperforms existing solutions in terms of performance and resource utilization. The findings will enhance resource access management in critical computing systems, paving the way for improved parallelism and efficiency in various applications.

Contents

1 Introduction

Range locks acquire exclusive locks on consecutive values within a specified range. They play a vital role in filesystems[1], operating systems[2], and databases[3]. Unlike traditional single-locking methods, range locks offer a more refined approach to resource access management. By dividing a shared resource into smaller segments, range locks allow multiple writers to modify different segments simultaneously. This approach overcomes the limitations and bottlenecks of single-lock approaches and fosters parallelism among writers. Consequently, range locks enhance overall efficiency within these critical computing systems.

Recently, there has been an increase in interest in range-locking techniques. One notable example is that the Linux kernel community is considering using range-locking techniques to replace the `mmap_lock` [2, 4, 5]. The `mmap_lock` uses a per-process semaphore to control access to the whole `mm_struct` [6] and serialize changes to address spaces. Despite previous efforts to overcome the scalability issues of `mmap_lock`, a resolution is yet to be found [5].

In the context of database management systems, range locks also offer a solution to the issue of coarse-grained locking in large databases and indexes. With the storage sizes and the number of pages and indexes increasing exponentially, there are better options than locking the entire index, considering that such a coarse-grained locking mechanism inherently blocks other transactions from progressing, leading to poor throughput and high latency. As a solution to this problem, range locks focus on key ranges, thus allowing multiple transactions to operate concurrently on separate key ranges, reducing memory overhead and lock acquisition bottlenecks [3].

2 Related work

Previous research has explored various approaches to scalable range locks [7, 8, 9]. The current implementation of range lock in the Linux kernel uses a range tree that keeps track of acquired ranges and an internal spin lock to protect it[7]. Since every range request relies on this single spinlock, it becomes a point of contention.

Song et al. try to improve the Linux kernel's implementation by combining a skip list with a spinlock to manage locked ranges[8]. This technique leverages the skip list, which is more lightweight and efficient than the interval tree and can still conduct intensive searches for overlapping ranges. Despite these advancements, the problem of contention points still requires resolution.

In another research, Kogan et al. designed a range lock based on a linked list, where each node represents an acquired range [9]. By using a linked list, this approach can maintain a lock-free range lock, thus solving the bottleneck problem. However, insertion and lookup operations on the linked list are less efficient than tree-like structures.

3 Approach

In this research's scope, we propose a new concurrent range lock design that leverages a probabilistic concurrent skip list[10, 11]. It consists of two main functions:

- **try_lock**: The `try_lock` function searches for the required range `([start, start+len])` in the skip list. If an overlapping range exists, indicating another thread is modifying that range, the requesting thread must wait and retry. If not, the range is added to the list, signaling that the range is reserved.
- **release_lock**: The `release_lock` function releases the lock by finding the address range in the skip list and removing it accordingly.

Our range lock design also utilizes the per-node lock instead of an interval lock, thus addressing the bottleneck problem of the spinlock-based range lock and maintaining the lock's high level of performance.

4 Evaluation

The proposed approach will be evaluated under these evaluation criteria:

- **Performance:** We will test the range lock mechanism under increasing load and concurrent accesses to measure its performance.
- **Correctness:** We will ensure the consistency and correctness of data accesses, especially when there are overlapping data ranges and concurrent operations.
- **Comparison:** We will compare the performance of the proposed solution with existing state-of-the-art approaches.

5 Result

We aim to develop a scalable range lock that performs better than the existing range locks. The evaluation results will provide insights into the performance characteristics and potential trade-offs of the proposed mechanism.

6 Conclusion

Conclusion

List of Figures

List of Tables

Bibliography

[1] C.-G. Lee, S. Noh, H. Kang, S. Hwang, and Y. Kim. “Concurrent file metadata structure using readers-writer lock”. In: *Proceedings of the 36th Annual ACM Symposium on Applied Computing*. 2021, pp. 1172–1181.

[2] J. Corbet. “Range reader/writer locks for the kernel”. In: *LWN.net* (2022). Accessed: 2024-04-21. URL: `https://lwn.net/Articles/724502/`.

[3] G. Graefe. “Hierarchical locking in B-tree indexes”. In: *On Transactional Concurrency Control*. Springer, 2007, pp. 45–73.

[4] M. Rybczynska. “Introducing maple trees”. In: *LWN.net* (2022). Accessed: 2024-04-21. URL: `https://lwn.net/Articles/845507/`.

[5] J. Corbet. “The ongoing search for mmap_lock scalability”. In: *LWN.net* (2022). Accessed: 2024-04-21. URL: `https://lwn.net/Articles/893906/`.

[6] S. Boutnaru. “Linux Kernel — mm_struct”. In: *medium.com* (2023). Accessed: 2024-04-21. URL: `https://medium.com/@boutnaru/linux-kernel-mm-struct-fafe50b57837`.

[7] J. Kara. “Implement range locks”. In: *lkml.org* (2013). Accessed: 2024-04-21. URL: `https://lkml.org/lkml/2013`.

[8] X. Song, J. Shi, R. Liu, J. Yang, and H. Chen. “Parallelizing live migration of virtual machines”. In: *Proceedings of the 9th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*. 2013, pp. 85–96.

[9] A. Kogan, D. Dice, and S. Issa. “Scalable range locks for scalable address spaces and beyond”. In: *Proceedings of the Fifteenth European Conference on Computer Systems*. 2020, pp. 1–15.

[10] H. Maurice, L. Yossi, L. Victor, and S. Nir. “A provably correct scalable concurrent skip list”. In: *Conference On Principles of Distributed Systems (OPODIS). Citeseer*. Vol. 103. 2006.

[11] H. Maurice, S. Nir, L. Victor, and S. Michael. *The art of multiprocessor programming*. Newnes, 2020.