Tiyiso Hlungwani u21598127

1. Introduction (But, like, not the best one ever)

So, here's the deal. We've got threads—lots of them—and we need to stop them from fighting over the same data like toddlers over a toy. To do this, we use these things called locks. And just like in real life, not all locks are created equal. Some make you wait forever; some make you think you're getting in, but nope (looking at you, TAS and TTAS), and others are a bit smarter about letting you in without creating chaos (hello, Backoff).

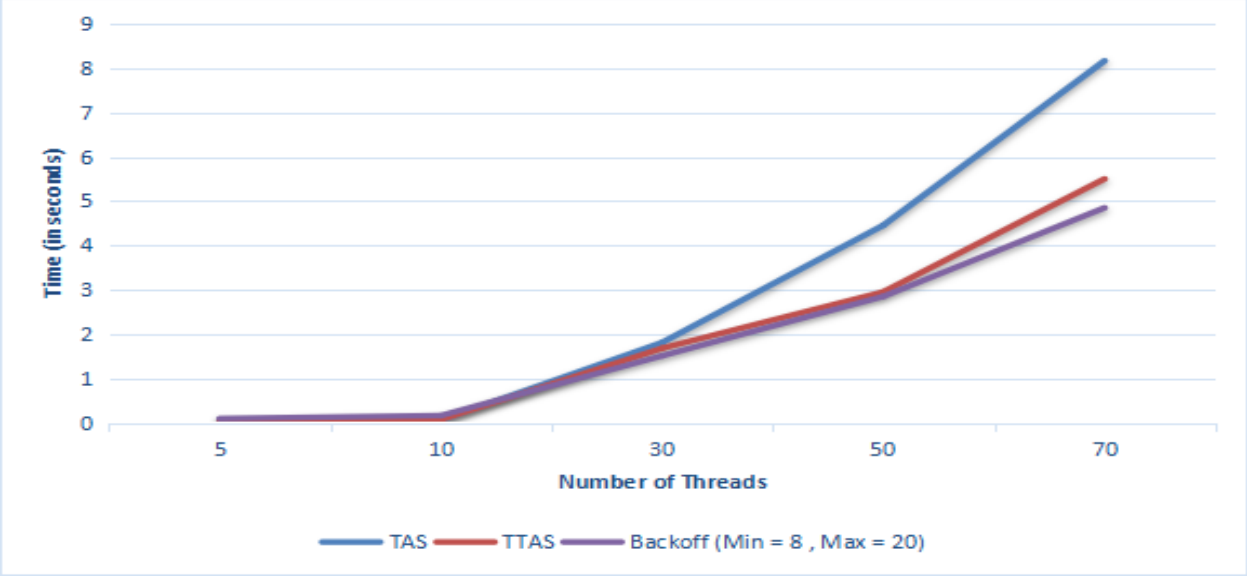
In this report, I'm going to show you how these different locks perform when a bunch of threads try to access the same resource. It's not going to be pretty for TAS, let's just say that. But in the end, we'll see which lock handles the stress best, and, spoiler alert, it's not TAS. Anyway, on to the graphs!

2. Graphical Performance Analysis

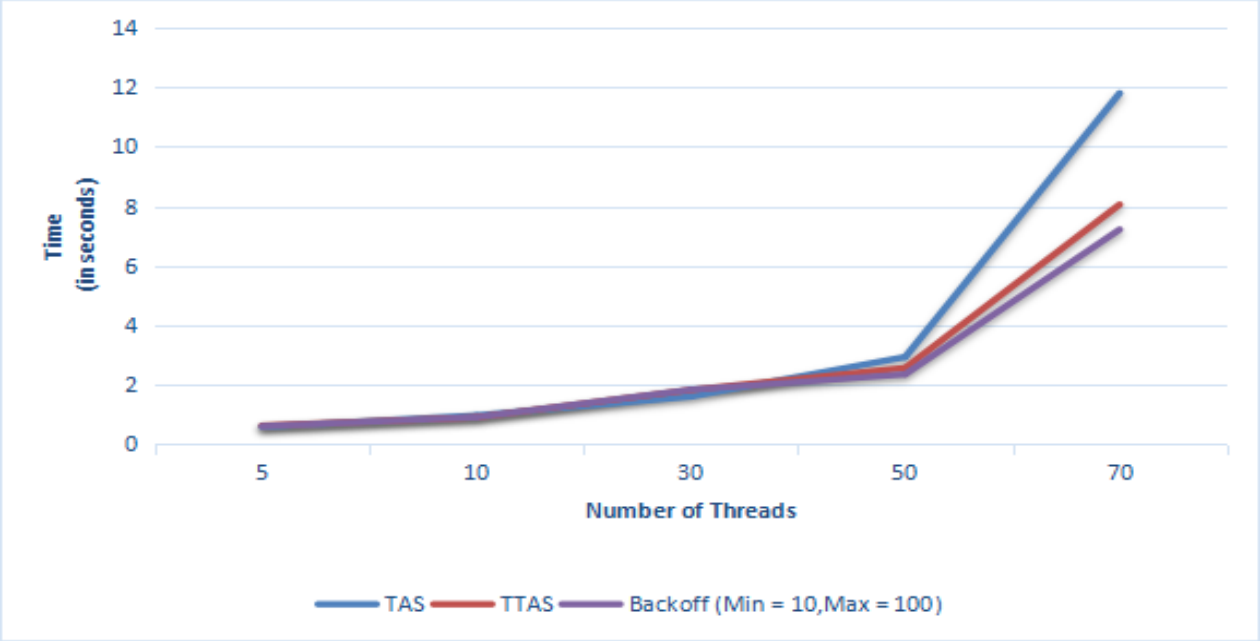
The performance of the locks was evaluated under two scenarios:

- **Scenario 1:** 15 iterations
- **Scenario 2:** 50 iterations
- I have reported MinDelay and MaxDelay values taken for experiments.
- I have compared the performance of the Lock Algorithms using the best MinDelay and MaxDelay value observed.

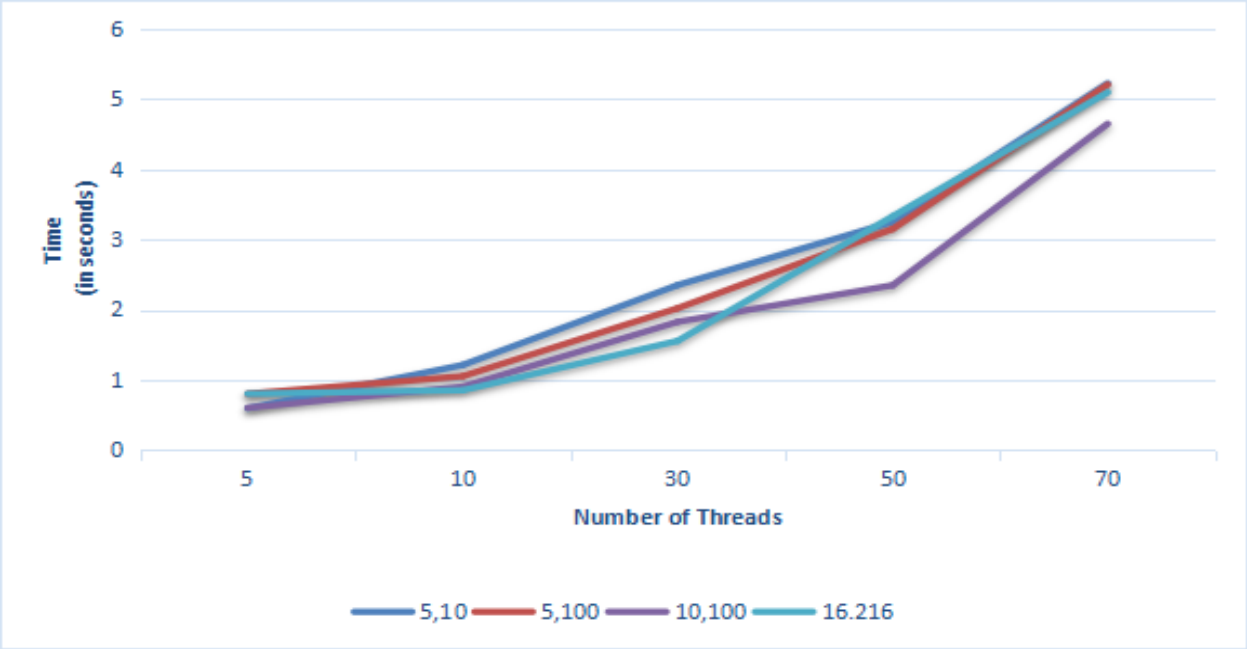
Number of Iterations: 15



Number of Iterations: 50



Backoff Performance with varying Min/Max Delays



3. Explanation of Trends

- **TAS Lock:** The performance of TAS locks worsens significantly as thread contention increases. This is due to its inability to handle contention gracefully—each thread repeatedly tries to acquire the lock, which leads to a heavy load on the memory system and higher overall time.
- **TTAS Lock:** TTAS mitigates some of the performance issues of TAS by testing the lock before attempting to acquire it. This reduces the frequency of expensive operations in high-contention scenarios. The improvement is especially noticeable when the number of threads is moderate.
- **Exponential Backoff:** Backoff strategies dynamically adapt the wait time between attempts to acquire the lock, reducing the contention overhead. As more threads are introduced, backoff locks perform better by minimizing the number of failed lock acquisition attempts. However, if the backoff delay is not tuned appropriately, performance may degrade, as observed in the third graph.

4. Scenarios

The locks were tested under two main scenarios:

1. **15 Iterations:** For a small number of iterations, the performance differences between the locks were less pronounced. However, TAS still exhibited the highest contention overhead.
2. **50 Iterations:** With more iterations, the differences between the locks became clearer. TAS scaled poorly, while TTAS and Backoff locks demonstrated better handling of high-contention scenarios.

In the Backoff scenarios, various Min/Max delays were tested, revealing that an appropriately chosen delay can significantly improve performance. For example, a Min/Max delay of 10/100 performed better than other configurations under the 50-iteration scenario.

5. References

I referenced a GitHub repository by Ankit7590, which provided implementations of the TAS, TTAS, and Backoff lock algorithms. The repository also included various optimizations and explanations that helped guide my implementation and understanding of these locks.