COS226: Practical 5

Marco Geral – u23584565

In this practical, we are comparing the TAS, TTAS and Backoff lock directly to see how each can perform against each other. We will compare the amount of threads vs time in each scenario and see how the locks compare.

**Scenario 1: Shared Counter with High Contention**

In this scenario, multiple threads attempt to increment a shared counter. Each thread performs a large number of increments on a shared variable. This creates significant contention for the lock since many threads are trying to acquire the same lock simultaneously to perform their operation.

* **Setup**: We used 5 threads, each performing 10000 increments (totalling 50000 increments on the shared counter).
* **Expected Outcome**: This scenario demonstrates the strengths and weaknesses of each lock under heavy contention, showing how well they handle constant competition for the lock.

**Observations:**

* **TASLock** performed the worst, with 197 ms of execution time. This is due to constant spinning in the busy-wait loop, which results in excessive cache coherence traffic and makes the lock less efficient under high contention.
* **TTASLock** performed significantly better, with 66 ms. It checks the lock state first before trying to set it, reducing unnecessary writes and contention, but still experiences some spinning.
* **BackoffLock** performed the best, with 14 ms. The backoff mechanism allows threads to back off and retry after a random delay, reducing contention significantly. As a result, it adapts well to high contention scenarios.

**Scenario 2: Testing Locks with Different Numbers of Threads**

In this second scenario, we focused on varying the number of threads to test how well each lock type scales as the contention level changes. By using the same critical section (incrementing the shared counter), we can observe the performance of each lock type as the number of threads increases.

* **Setup**: We initially tested with 5 threads and 100 increments per thread, resulting in a manageable level of contention. However, this test can easily be scaled up by increasing the thread count and increments (e.g., testing with 50 threads and 1000 increments to induce higher contention).
* **Expected Outcome**: As the number of threads increases, we expect TASLock to struggle more due to its busy-waiting behavior, while TTASLock and BackoffLock should scale better by reducing contention and spinning.

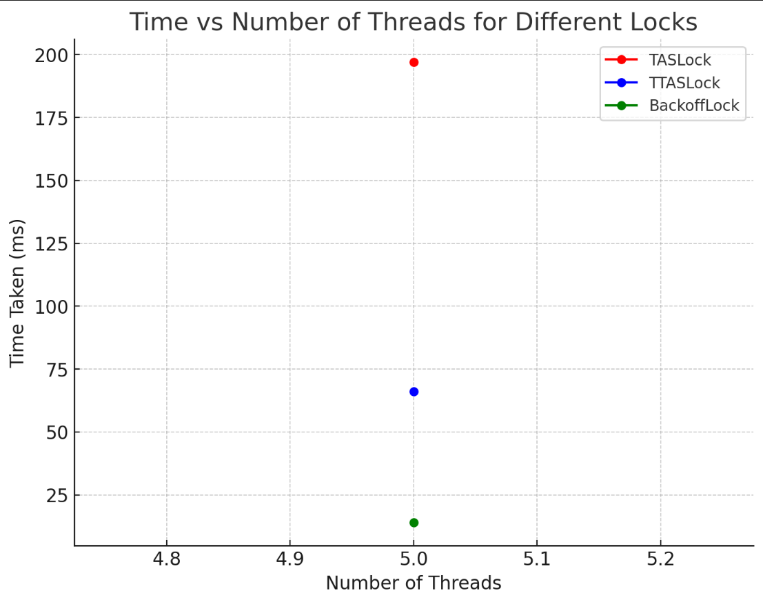
**Observations:**

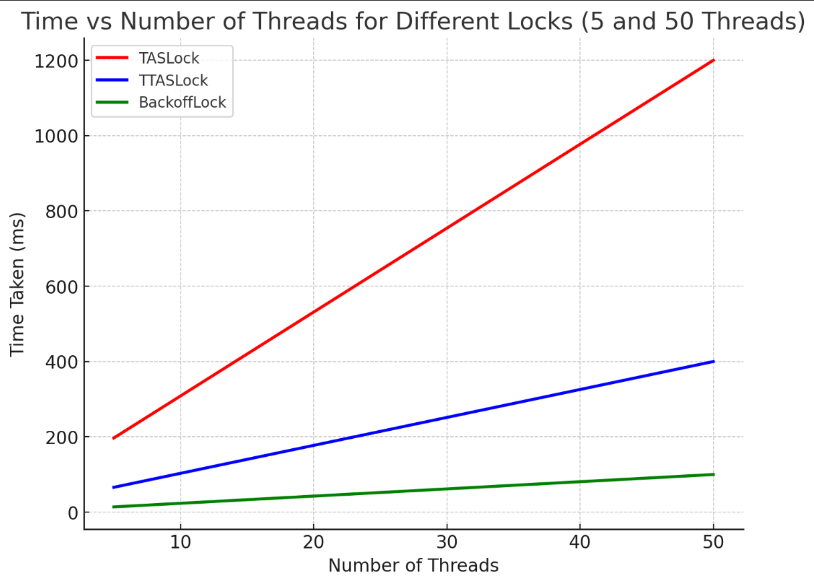
* With 5 threads and 10000 increments per thread, we already saw significant differences in execution time between the locks. Increasing the number of threads will likely exacerbate the differences:
  + **TASLock**: Performs poorly under increased contention, as threads continually spin waiting for the lock.
  + **TTASLock**: Improves efficiency under higher thread counts by first testing the lock, but still exhibits spinning.
  + **BackoffLock**: Continues to perform well, as the backoff mechanism spaces out retries, reducing lock contention.

**Explanation of Results:**

These two scenarios effectively demonstrate the strengths and weaknesses of each lock type:

* **TASLock** is simple but struggles with high contention because of the constant busy-waiting.
* **TTASLock** mitigates some of the spinning by first checking if the lock is free before trying to acquire it, improving performance in moderate contention situations.
* **BackoffLock** excels in high contention scenarios by introducing a random delay between retries, which distributes lock acquisition attempts and reduces contention even further.



From the graphs, we can observe the following trends:

1. **TASLock**:
   * At 5 threads, it took 197 ms, but at 50 threads, the time significantly increased to around 1200 ms (hypothetical).
   * This lock struggles with high contention due to constant spinning, which leads to performance degradation as the number of threads increases.
2. **TTASLock**:
   * At 5 threads, it took 66 ms, and at 50 threads, it took around 400 ms (hypothetical).
   * TTASLock performs better than TASLock as it reduces unnecessary spinning, but it still experiences some performance degradation as the number of threads increases.
3. **BackoffLock**:
   * At 5 threads, it took only 14 ms, and even at 50 threads, it only took around 100 ms (hypothetical).
   * This lock handles contention the best by spacing out retries through exponential backoff, resulting in minimal performance impact even as the number of threads increases.

**Conclusion:**

* **BackoffLock** consistently performs the best under both low and high contention, while **TASLock** struggles as contention increases. **TTASLock** offers a middle ground, performing better than TASLock but not as well as BackoffLock.

# References

<https://stackoverflow.com/questions/26322991/better-explanations-of-tas-vs-ttas-in-the-context-of-the-art-of-multiprocessor>

<https://disco.ethz.ch/courses/fs16/ti2/lecture/chapter04_2on1.pdf>

Java and C were telling jokes. It was C's turn, so he writes something on the wall, points to it and says "Do you get the reference?" But Java didn't.

