Comparison of TAS, TTAS and BackOff Locks

Design

- **TAS**
  - The TAS algorithm was implemented as given in the book
  - exchange is the equivalent method of getAndset() of Java, and this was used
  - store is the equivalent method of set() of Java
  - load() is the equivalent method of get() of Java

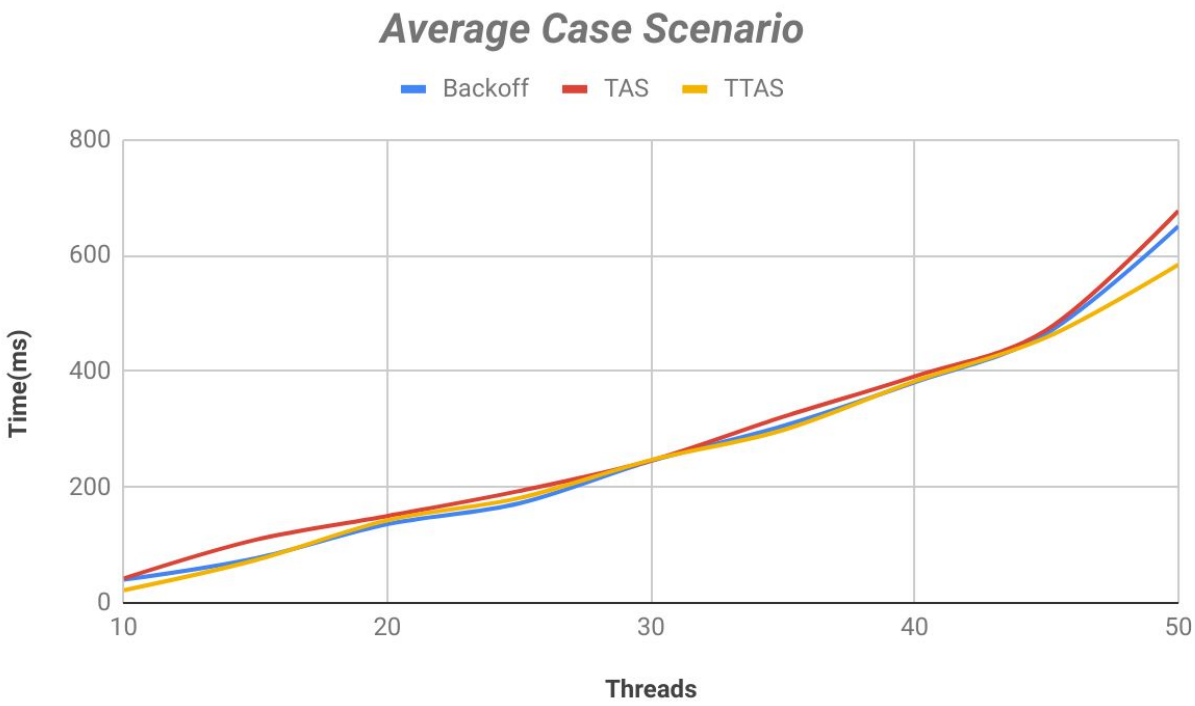
- **TTAS**
  - TTAS algorithm was implemented as given in the book

- **BackOff**
  - BackOff algorithm was implemented as given in the book
  - MAX_DELAY of 63 ms was imposed and a MIN_DELAY of 1ms was set

Graphs and Analysis

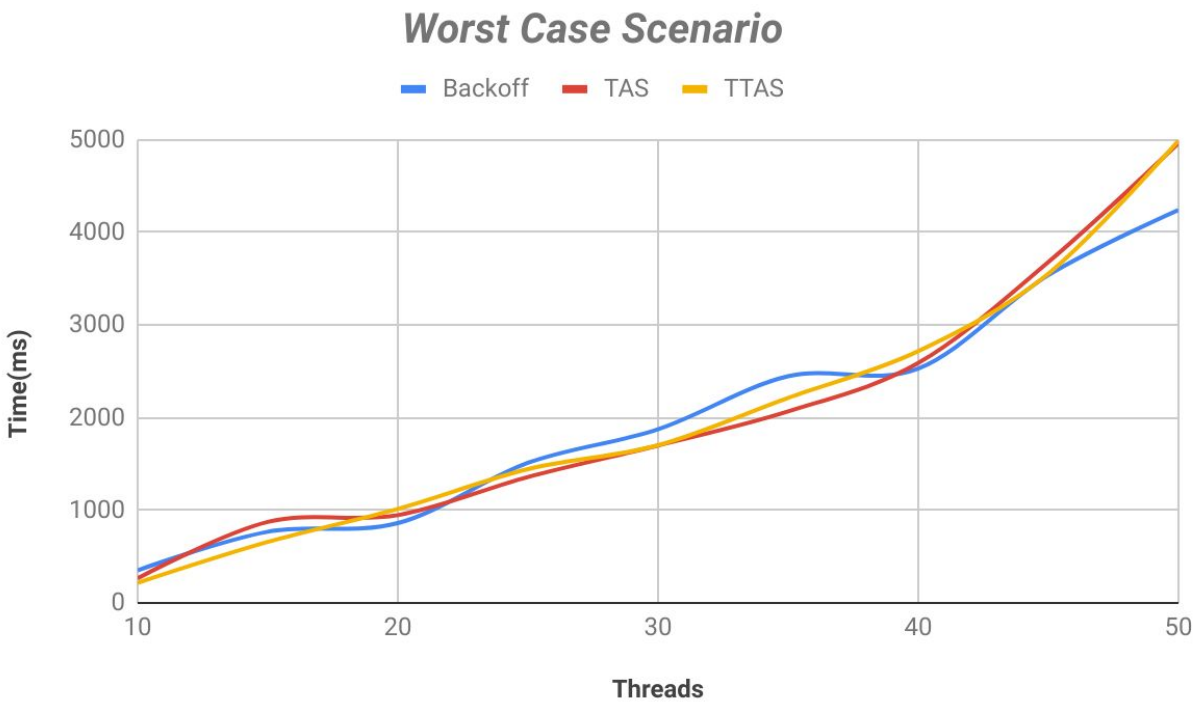
n = [10, 15, 20, 25, 30, 35, 40, 45, 50] (Number of threads)
k = 10 (Number of times to enter the CS)
lamda1 = 1 (average sleep time within the CS, which is exponentially distributed)
lamda2 = 2 (average sleep time outside the CS, which is exponentially distributed)

★ Each test was run **5 times** and the **average value** was calculated to plot the graphs



- TTAS seems to perform better most of the times followed by BackOff which performed better than TTAS for some cases. TAS performs the worst among the three algorithms
- TTAS, BackOff performing better than TAS is intuitive as the cache invalidation does not happen as frequently in TTAS, BackOff as in TAS due to the optimization made of reading the lock via get() before performing getAndSet()
- BackOff performance is variable and it depends highly on the value of MIN_DELAY, MAX_DELAY and the contention faced during acquiring the lock
- We can on average say that BackOff should perform better than TTAS when the contention is high as the memory bus traffic can be reduced by backing off than trying the lock again when the lock is freed
- TTAS performing better than BackOff can be explained by giving the reason :
    - The contention was not high enough for BackOff advantage to be observed
    - In these specific runs, BackOff just got unlucky and two threads again chose the same value of backoff(this is very unlikely, but can happen)
    - The values of MIN_DELAY and MAX_DELAY was not optimal (I have tied several combinations but did not know exact heuristics to choose the parameters optimally, I chose parameters which are generally chosen in wireless networks)

★ Note: BackOff improves upon TTAS algorithm by choosing random backoff values when the thread fails to acquire the lock



- The worst-case time taken by a thread to enter the CS gives an indication of starvation in threads.
- Here we see TTAS having consistent statistics compared to BackOff and TAS algorithms
- The statistics of TAS and BackOff is very variable, for all three algorithms it is difficult to comment upon which is fairer by seeing the graph since the overall graph is very similar for all 3
- TTAS, TAS is supposed to be fairer than BackOff going by the algorithm as everyone gets a chance to acquire the lock without backing off
- In absolute terms, TTAS should be better than TAS because of the performance improvement already discussed

Data

Average case	Backoff	TAS	TTAS
10	40	42.2	21.8
15	77	109	73.8
20	135.8	150	142.6
25	171.8	193.2	181
30	245.2	245.8	246.6
35	305.4	321.2	298.2
40	380.8	391.4	382.8
45	465.2	471.4	458.6
50	650.6	677	584.2

Worst case	Backoff	TAS	TTAS
10	351.6	266.2	216.8
15	767.8	873.6	658.2
20	861	946.8	1013.6
25	1511.4	1359.2	1443.4
30	1874.6	1698.4	1702
35	2446	2066.8	2213.2
40	2529.2	2589	2716.6
45	3537.6	3678.6	3553.4
50	4241.4	4961.2	4987.6

Note1: The above tables contains time in milliseconds

Correctness of the locking algorithms:

TAS:

```
0th CS Entry at 2:35:42 by thread 46
0th CS Exit at 2:35:42 by thread 46
1th CS Request at 2:35:42 by thread 46
0th CS Entry at 2:35:42 by thread 43
0th CS Exit at 2:35:42 by thread 43
1th CS Request at 2:35:42 by thread 43
0th CS Entry at 2:35:42 by thread 38
0th CS Exit at 2:35:42 by thread 38
1th CS Request at 2:35:42 by thread 38
1th CS Entry at 2:35:42 by thread 43
1th CS Exit at 2:35:42 by thread 43
```

TTAS:

```
2th CS Entry at 2:35:50 by thread 1
2th CS Exit at 2:35:50 by thread 1
0th CS Entry at 2:35:50 by thread 21
3th CS Request at 2:35:50 by thread 1
0th CS Exit at 2:35:50 by thread 21
1th CS Entry at 2:35:50 by thread 0
1th CS Exit at 2:35:50 by thread 0
```

BackOff:

```
0th CS Entry at 2:35:57 by thread 29
0th CS Request at 2:35:57 by thread 11
0th CS Exit at 2:35:57 by thread 29
0th CS Request at 2:35:57 by thread 48
1th CS Request at 2:35:57 by thread 29
1th CS Entry at 2:35:57 by thread 1
0th CS Request at 2:35:57 by thread 2
1th CS Exit at 2:35:57 by thread 1
0th CS Entry at 2:35:57 by thread 19
2th CS Request at 2:35:57 by thread 1
0th CS Exit at 2:35:57 by thread 19
```

- The above snippets show that the locking algorithms implemented are correct

Note:

- The above snippets were taken from the logs obtained from the execution of the programs
- Disable compiler optimizations before running the algorithms