Sharan Babu Paramasivam Murugesan (sxp141731)

Vasudev Karthick (vxr143630)

Report – Project 2

Experimental setup:

Programming Language : Java
Operating system : Linux (CentOS distribution)
Hardware:
Processor - Dual 8-core Intel Xeon E5-2680 2.7Ghz
Cache – 32K L1 data, 32K L1 instruction, 256K L2 and 20MB L3
Range of Values
Number of threads – 1, 2, 4, 8, 16, 32
Average Inter-request delay – 0, 20, 40, 60, 80, 100 Time Units (1 Time Unit or 1 TU = 0.1ms)
Each data point in the graphs are averaged over 10 runs

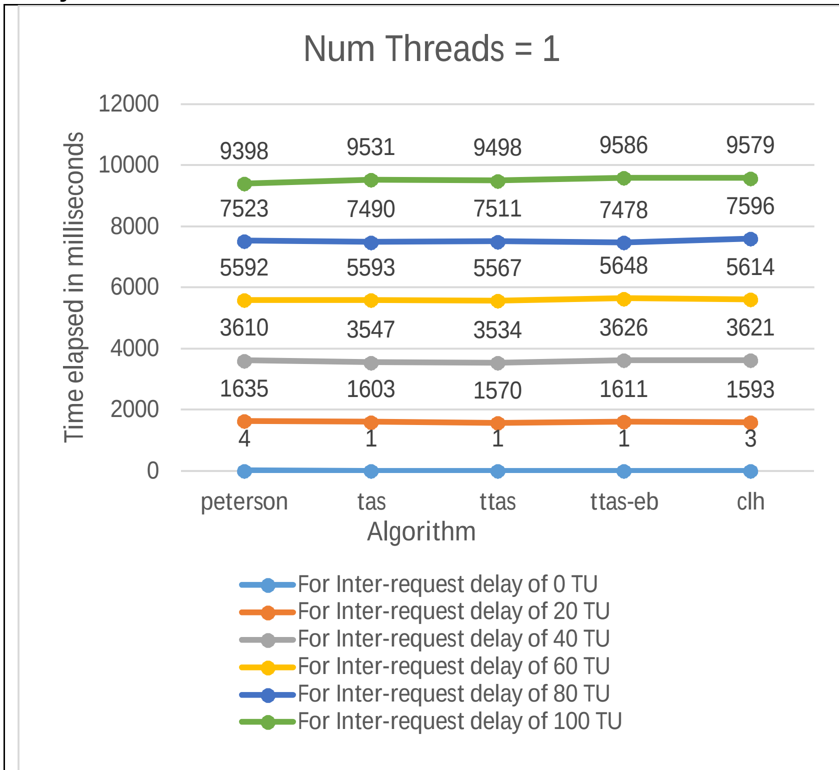
Other libraries - External library for exponential random number generation which is used to generate interlock request delay
Crtitical section used - Integer counter – each thread executes 1000 times (so final value expected is num_threads * 1000).
Through put measured – time elapsed for all threads to complete

Algorithms compared :

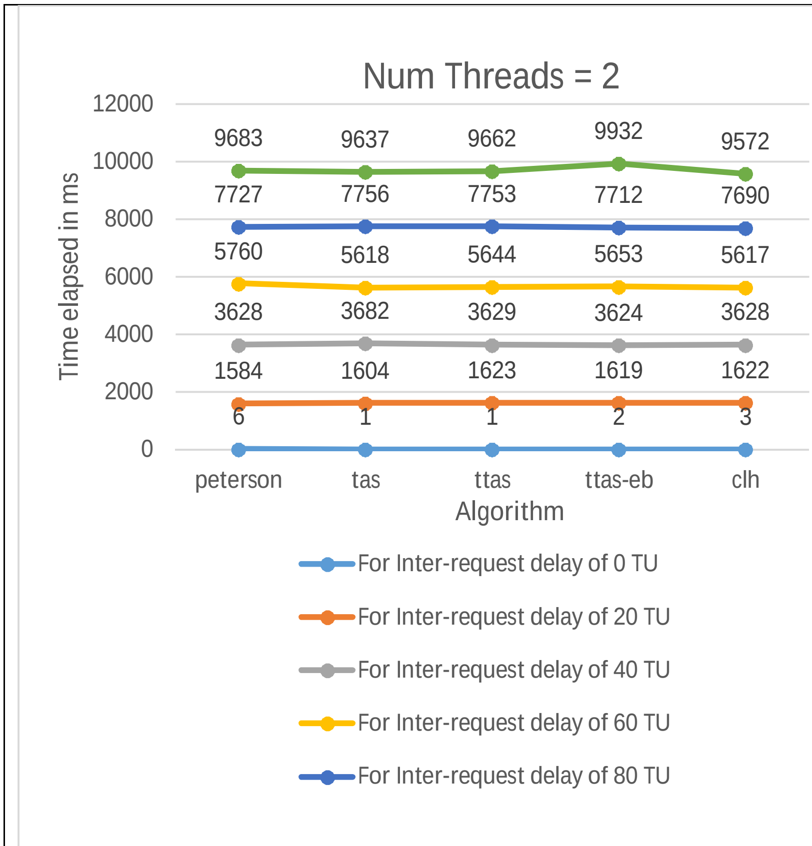
1. Extension of Peterson's lock for n threads using Binary tree
2. TAS
3. TTAS
4. TTAS – with Exponential backoff
   Minimum waiting time used – 1 ms
   Maximum waiting time used – 16 ms
   Experimented with different combination of values and compared the runtime with TTAS. The values that produced better improvement over TTAS were 1 and 16 for min and max waiting time respectively. Since these values are sensitive to the processor speed which directly influences the average time for each thread to execute the CS – and since in this experiment the CS is very small, the same values produced similar results in a different system with different processor but with similar clock speed.
5. CLHLock

Results:

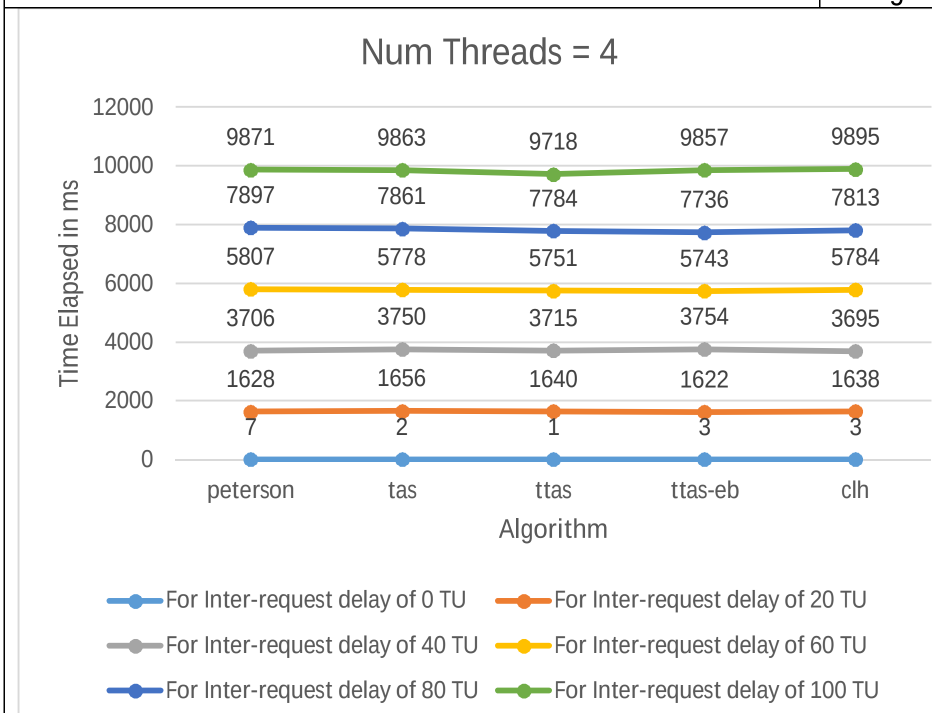
Below shown results are averaged over 10 runs.
Analysis of Runtime Performance:



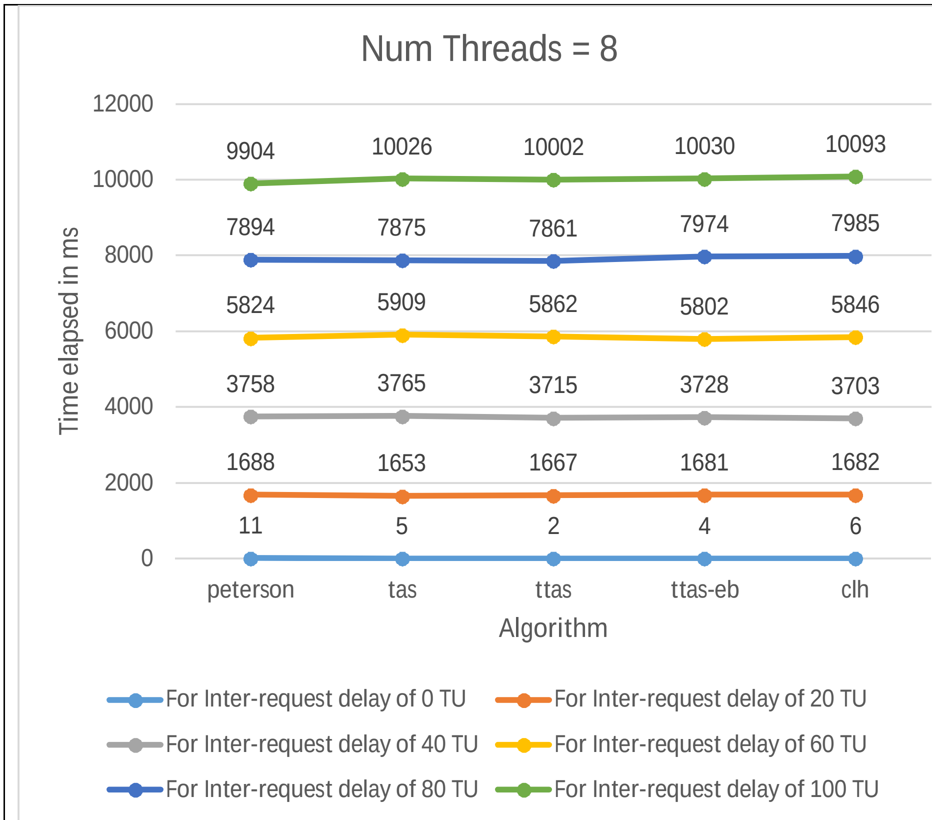
Observation:
When the number of threads are 1, TTAS provides the best performance in terms of runtime. Though TTAS and TAS are comparable, TTAS outperforms TAS by a slight margin. Clearly, the exponential back-off doesn't provide any improvement in runtime when there is only one thread as a single thread backing off for a certain amount of time will definitely be an overhead for the performance.



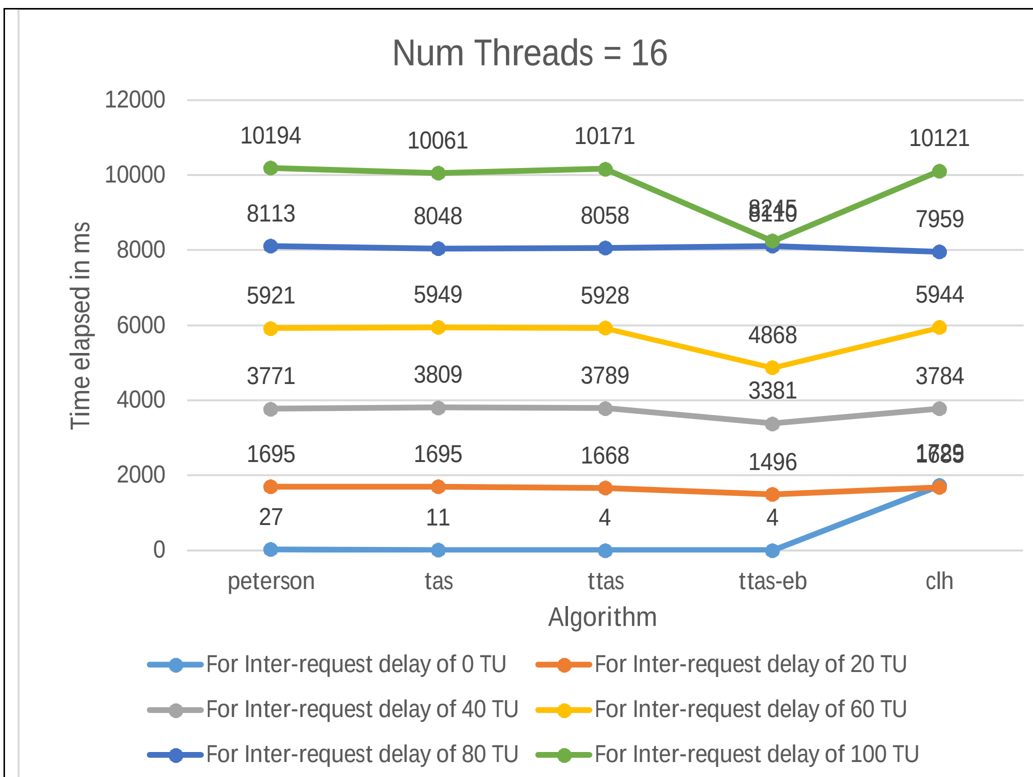
Observation:
When there are 2 threads, TTAS with EB provides the best performance except when the inter-request delay is very large (100 TU). Though TTAS-EB and CLH provides comparable results, TTAS-EB outperforms CLH. This behavior of TTAS with EB confirms with the explanation from Herlihy and Shavit. The exponential back off reduces the time taken for threads to finish by reducing the amount of cache traffic. But when the inter-request delay is very high, the advantage provided by the EB is overcome by the overhead associated with the inter-request delay. But during such high inter-request delay, CLHLock outperforms all others. This behavior also confirms with the explanation provided in the book that using queue based locks reduces the cache traffic to a large extent.



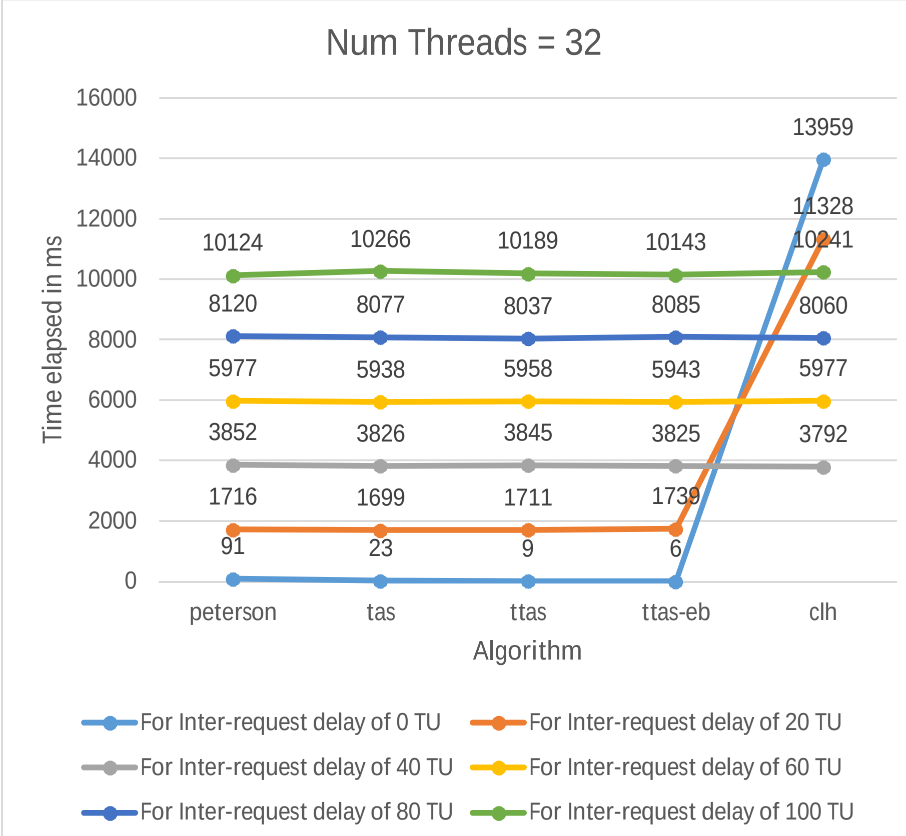
Observation:
Similar to the scenario of 2 threads, here also when there are 4 threads, TTAS-EB and TTAS provides comparable results but TTAS-EB outperforms the others. But when the inter-request delay is high, TTAS provides the best results. In CLH, spinning on other threads' memory node must have increased the cache as the number of threads have increased from previous experiment.



Observation:
Here also, the behavior confirms with the explanation given in the above scenarios. Both TTAS and TTAS-EB seems to be performing better than other algorithms. Though there is no significant improvement observed in TTAS-EB, the EB doesn't become an overhead here, confirming the correctness of values chosen. There are other scenarios explained below, where EB provides significant improvement.

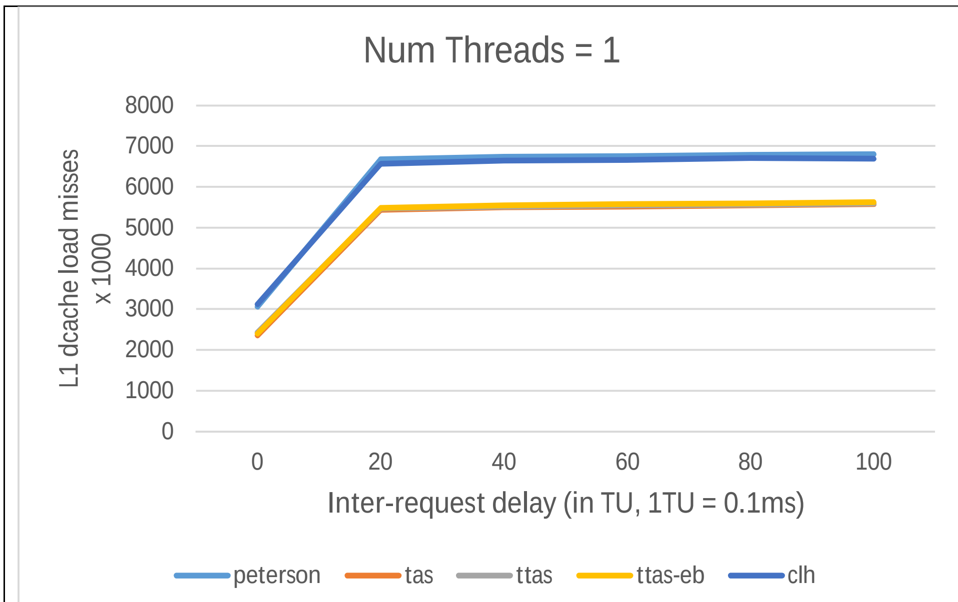


Observation:
Here is a scenario where EB pays off. TTAS-EB outperforms all others significantly. Though CLH doesn't provides the best results, it seems to be improving when the inter-request delay increases.

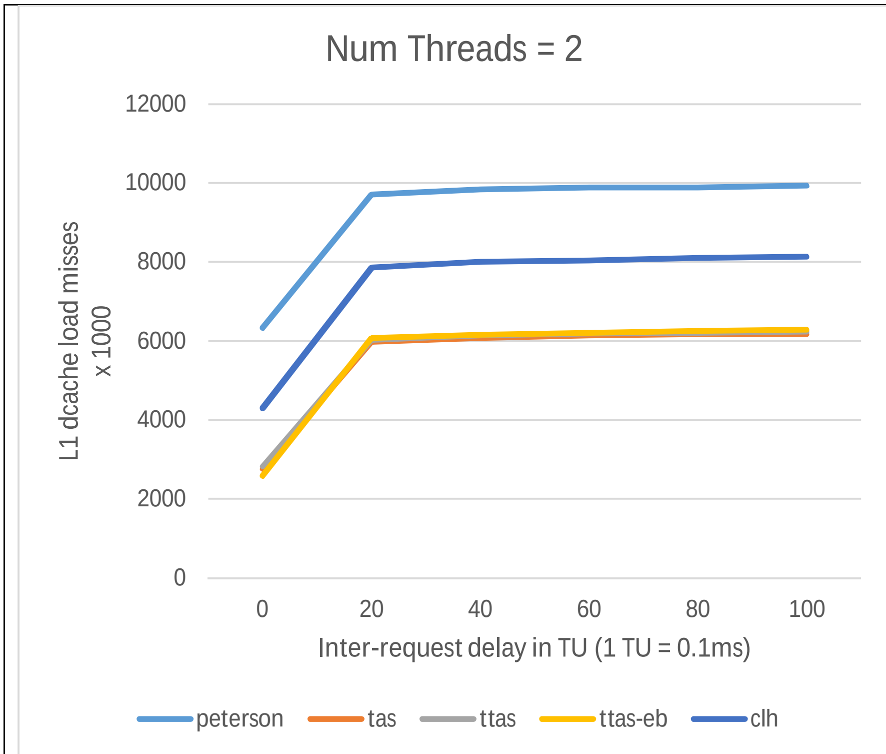


Observation:
This particular experiment when the number of threads are twice the available cores, explains the behavior of CLH that the increased cache traffic associated with a thread spinning on other thread's memory node must be causing this huge performance degradation in CLH. TTAS-EB seems to be performing better than most others.

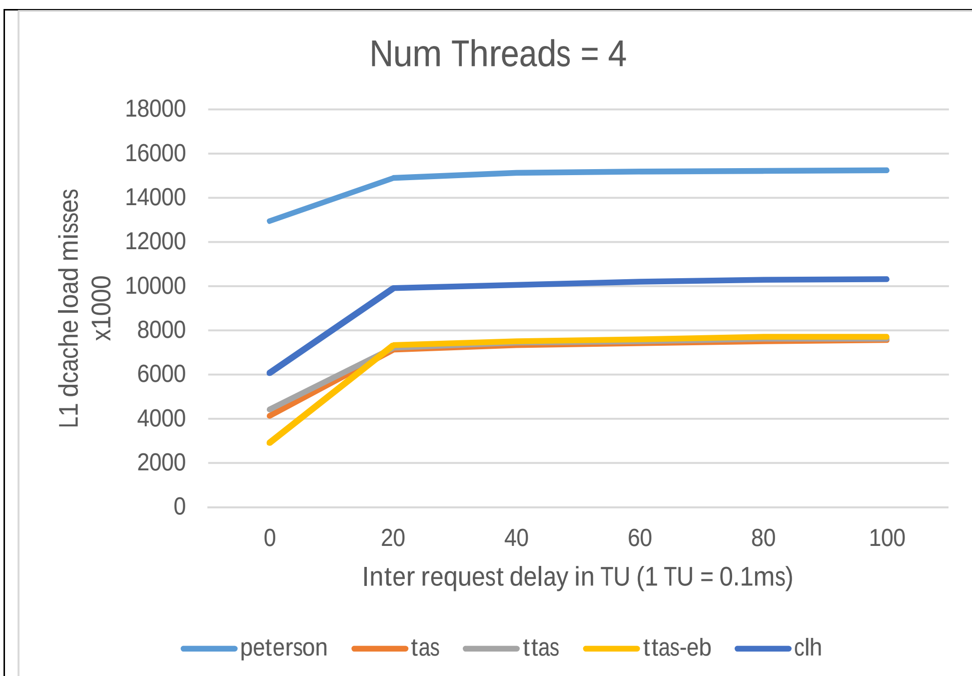
Analysis of Cache Performance:



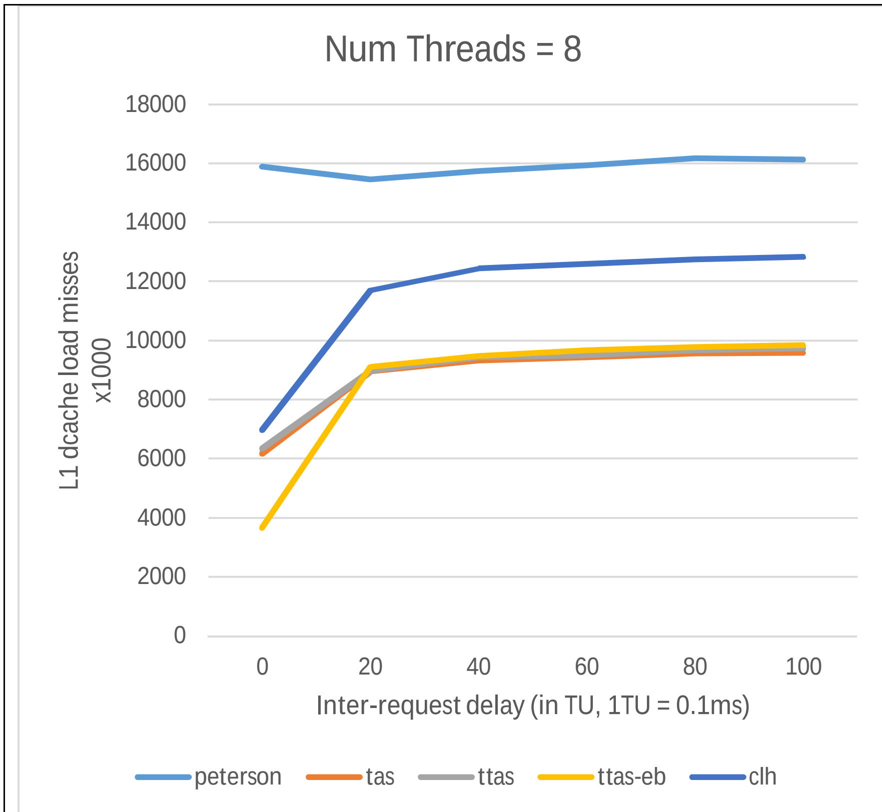
Observation:
Since there is only thread, there is no cache invalidation of other thread's cache memory and there will be no reread of cache values, no storm of bus traffic and there will be no local spinning at all. Hence, all TAS, TTAS and TTAS-EB should produce identical results and the experiment confirms this prediction.



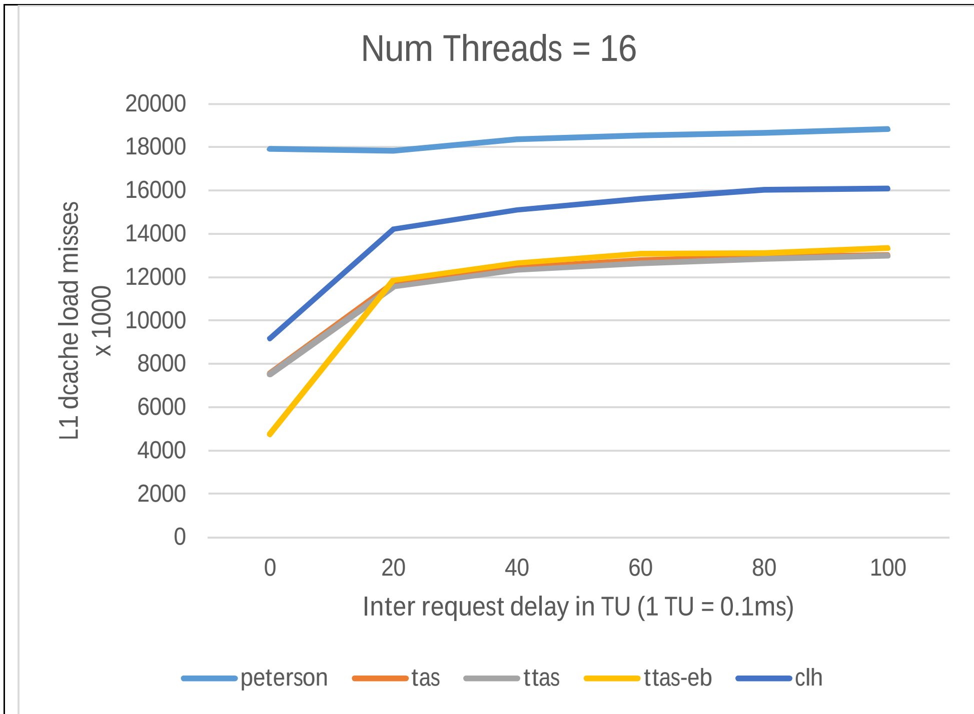
Observation:
When there are only two threads, there seems to be less bus traffic and less impact of cache misses on the performance of TAS, TTAS and TTAS-EB. A close look into the numbers suggests, TTAS producing better results among all. Between CLH and Peterson's, Peterson's lock produces higher cache misses as this algorithm uses larger memory (binary tree for n thread implementation + memory required for original Peterson's Algorithm) compared to others.



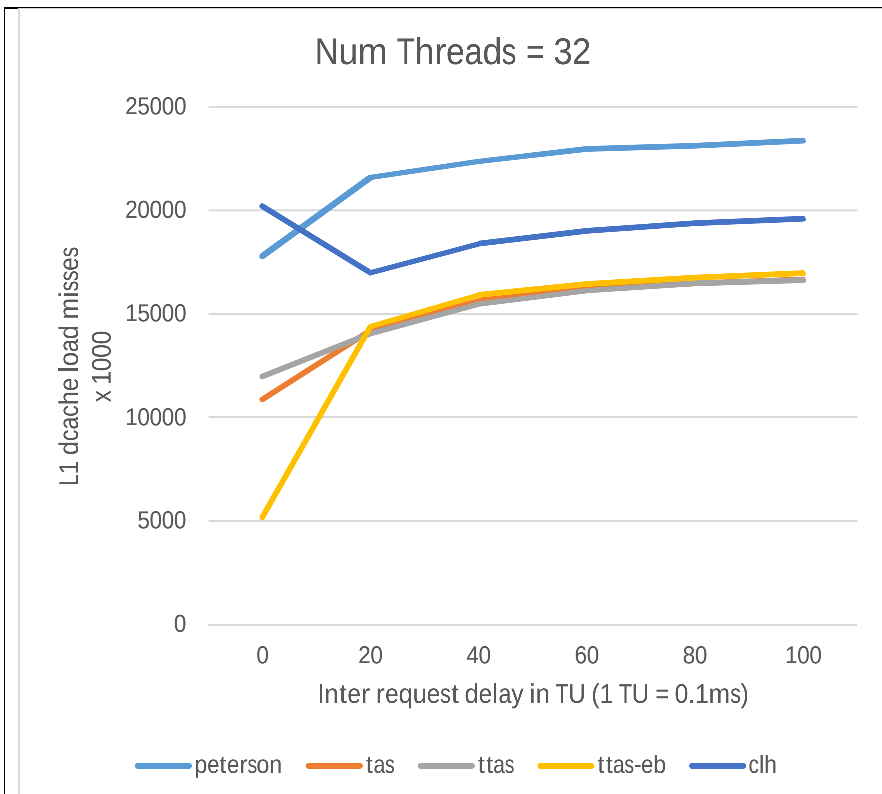
Observation:
Here when there are 4 threads, both TTAS and TTAS-EB seems to produce comparable cache activity when the inter-request delay is larger. During short inter-request delays, when there is will be more system load, TTAS-EB produces better results. Though there is only very little difference between the figures, TTAS-EB works better during high system load.



Observation:
When there are 8 threads, during high system load (short inter-request delay), TTAS-EB produces better results by a significantly higher margin. But as the inter-request delay increases, when there is less system load, there seems to be less cache traffic and hence less cache misses observed between TAS, TTAS and TTAS-EB. This confirms with the understanding of the behavior of these algorithms. Although, when the system load is very high, TAS seems to be producing better results as opposed to be expectation of observing better results from TTAS which is very odd.



Observation:
When there are 16 threads, it is observed that the results are similar to that of 8 thread experiment. The only difference is that during high system load, between TAS and TTAS, TTAS seems to be producing better results as expected rather than the odd behavior like in the previous experiment.



Observation:

When there are 32 threads, TTAS-EB produces almost 50% less traffic than TTAS and TAS. This confirms the understanding of the behavior of the algorithm. When the system is gradually reduced, the results are comparable between TAS, TTAS and TTAS-EB. Whereas between CLH and Peterson’s, though initially during high load Peterson’s performs better, CLH seems to be performing better between these two algorithms.
.