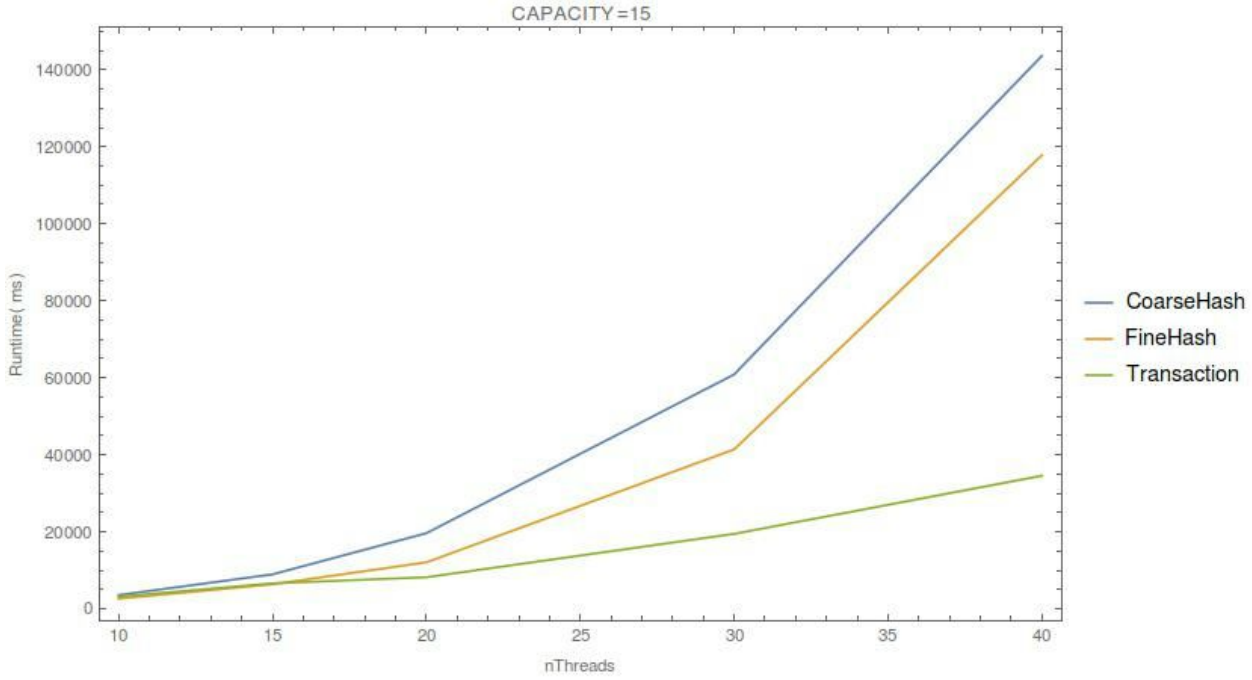
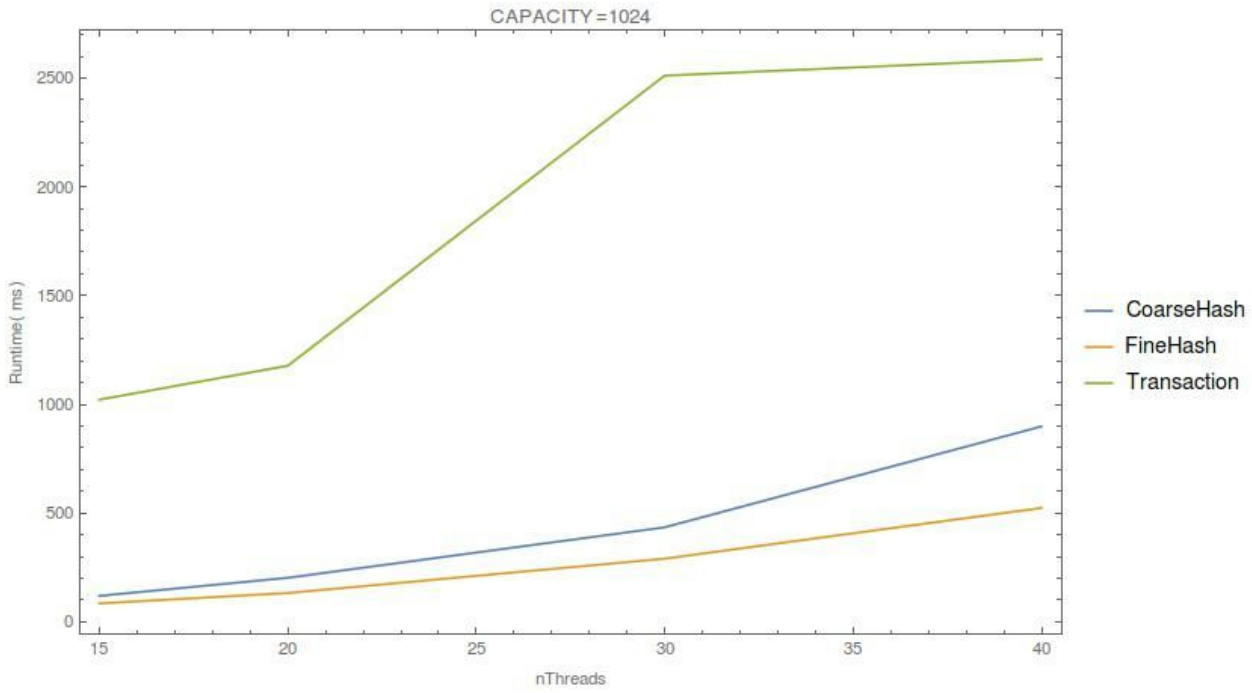
1) The comparisons of performances between Coarse-grained lock, Fine-Grained lock and transaction are done on my laptop under the same environment. For each thread, 6000 insertions are performed and then 6000 queries performed. First with capacity of the HashSet 15, we can see that transaction version almost always performs better than the locked versions except for number of threads less than the capacity, in which case the probability that more than one thread accessing one bucket is low. The transaction method's advantage gets more obvious as multiple threads accessing the same bucket

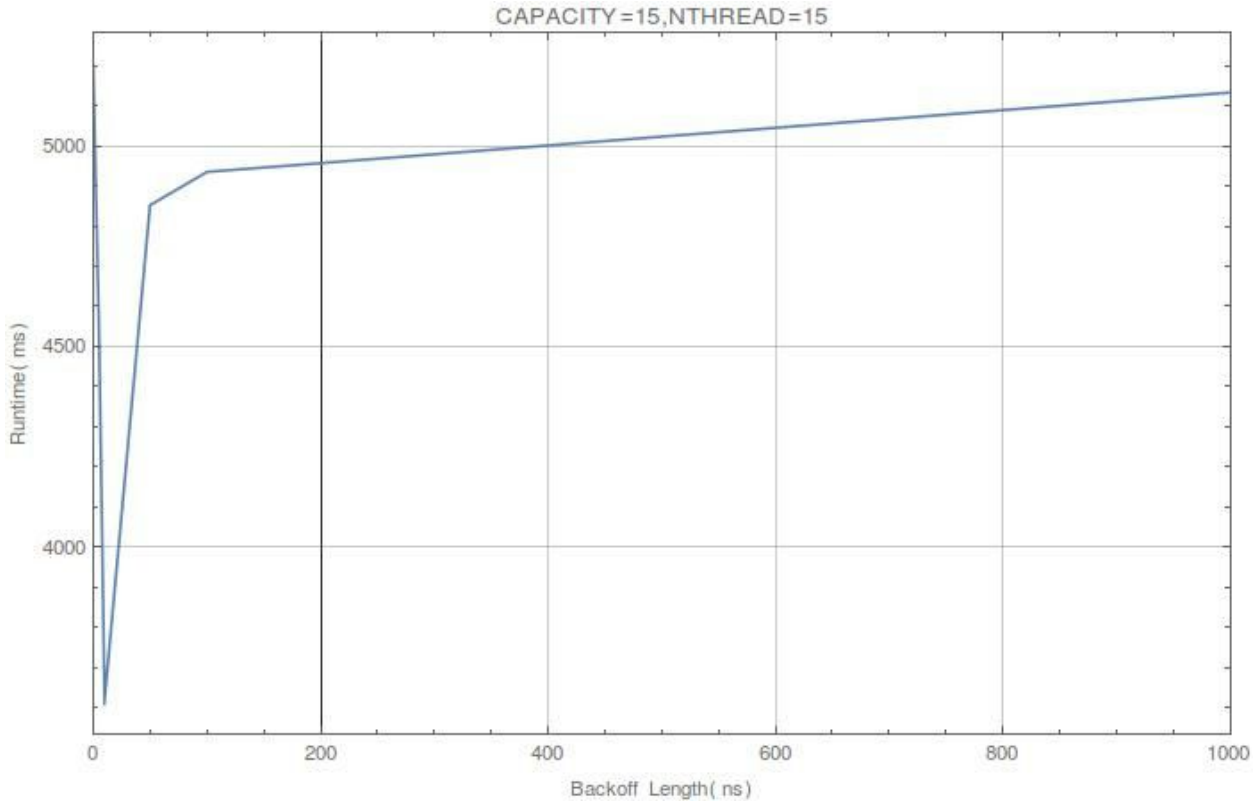




becomes more frequent. By contrast, when the capacity of the HashSet is high (1024)

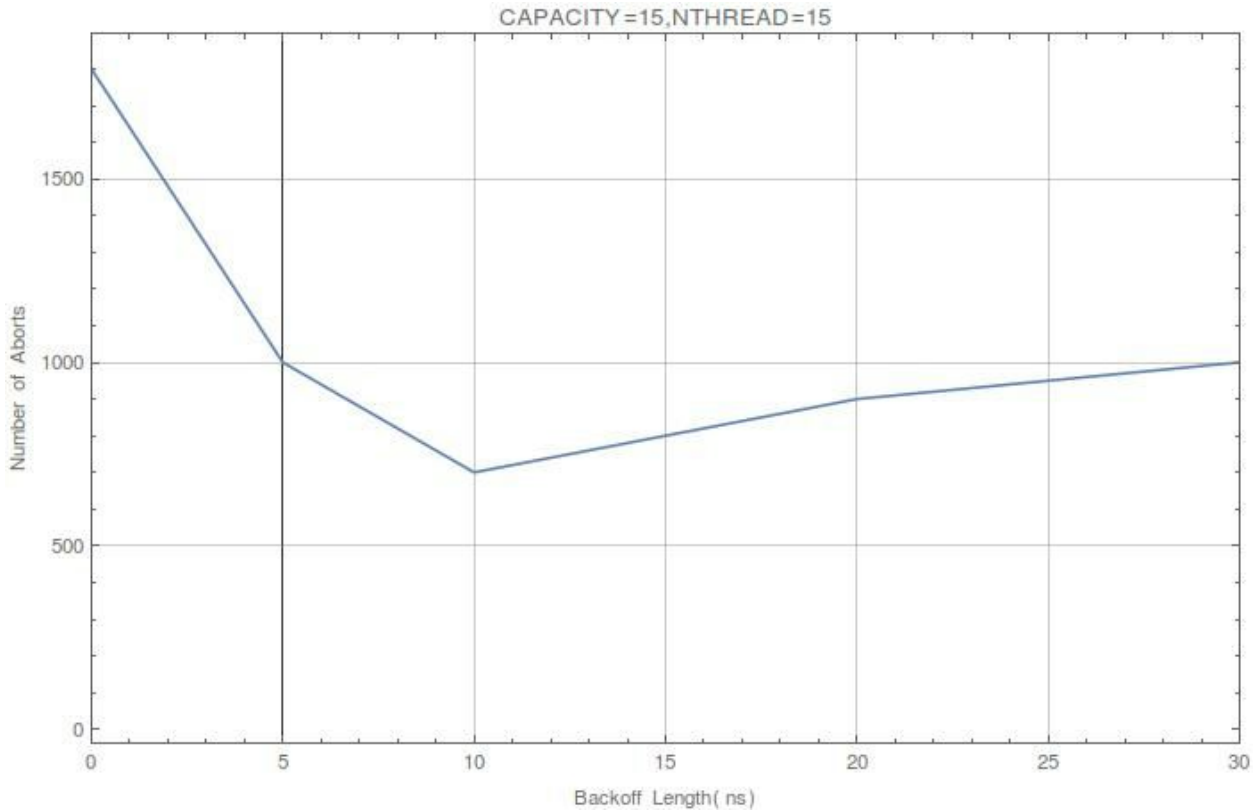
and much larger than the number of threads, it is highly unlikely for multiple threads to access the same bucket. Henceforth, the overheads of transactions dominate its computational costs while its optimization over interleaving operations are not realized. As a result, the transaction-implemented HashSet constantly does worse than the lock-implemented HashSet and the difference gets larger as the number of threads goes up. In both cases, the fine-grained lock scheme is faster than the coarse-grained one.

2) In this case, the same amount of work as in 1) is done in each thread with 15 threads and a capacity of 15 for the HashSet to allow for frequent enough simultaneous access to buckets.



At backoff length 0 (no exponential backoff), the performance of the HashTable is at its highest. Then the performance is optimized at around 10ns backoff length. It is interesting that after 10ns the runtime increases sharply and then really slowly, presumably because the program never doubles the initial waiting time.

3) Finally, we repeat the same experiment but have the program prints the number of commits and aborts. The number of commits are always the same regardless of the backoff length since all work needs eventually to be done. But the number of aborts vary with the backoff length. Note that the abort count shown in the following graph are averages taken heuristically across the threads.



Note that in this graph, the range of backoff length is much smaller than that of the previous one since the number of aborts vary very little after it reaches its optimum. It is clear that when no exponential backoff is implemented we have a lot more aborts and at the optimal backoff length we have the fewest aborts and then the number of aborts stays approximately the same.