29.03.2019

CS342 OPERATING SYSTEMS

PROJECT 2

Elif Gülşah Kaşdoğan – 21601183

Sıla İnci – 21602415

SECTION 2

We conducted tests with a function named “test_func”, we noted the elapsed time for execution of hash functions we have implemented. test_func takes a number of key values W, inserts W distinct keys to the hash table, updates, calls hash_get function on it then deletes. Each thread executes test_func for respective value. We used a various number of thread values(T) and calculated total elapsed time. The elapsed time shows how many milliseconds each case took while completing the jobs that could be done with all the functions enabled by the hash library.

K (number of locks)	Time (milliseconds)
10	2.195
20	1.952
30	1.794
40	1.824
50	1.965
60	1.973
70	1.752
80	1.138
90	0.742
100	0.756

Table 1: Time vs Number of Locks

N = 100 , T = 10, W = 100

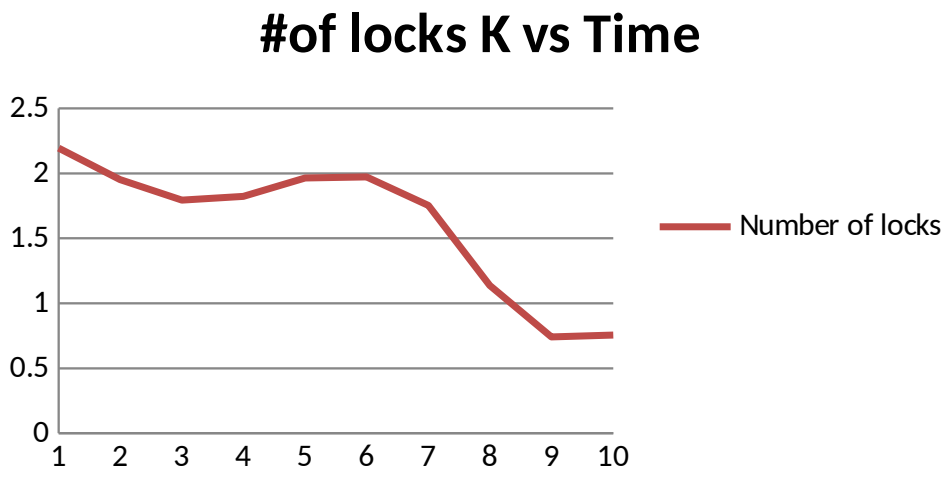

Fig 1: Graph of Table 1

In Table 1, the experiment is conducted with 10 different values of K (number of locks) with the default values of N = 100, T = 10 and W = 100. As it is seen in the table, with the increase of the number of locks, the time the entirety of hash operations take decreases.

This is because each value of the number of locks decide how many regions our hash table is going to be divided into. When K = 10, our hash table which has 100 buckets divide into 10 regions which allow only 10 concurrent threads to do operations on the table. For example, if we have set the number of locks to 1, we would only allow one lock at a time operate on the table. This would be time inefficient since the threads other than the one which is active would have to wait for the other to be done

K (number of locks)	T (number of threads)	t (milliseconds)
50	1	1.492
50	2	2.470
50	5	6.732
50	10	12.184
50	100	92.088
100	1	1.571
100	2	2.369
100	5	6.043
100	10	12.018
100	100	120.291

When the number of locks is constant, execution time increase as we increase the number of threads. Each thread calls test_func of its own, performs respective operations and joins. Threads might be CPU bound, memory bound or I/O bound, in our case our threads are mostly bounded by the memory and CPU. Each newly created thread competes with other threads for access of shared hash table. Each thread waits for other threads operating on the region it wants to access, because of the time spent on waiting for the region to be available and competition, an increase of the number of threads results in an increase of time execution.

N (table size)	W (number of keys/operations)	t (milliseconds)
100	10	0.258
100	100	0.967
100	1000	6.243
100	5000	24.958
100	10000	109.986
500	10	0.333
500	100	0.848
500	1000	5.024
500	5000	12.898
500	10000	35.763
1000	10	0.351
1000	100	0.983
1000	1000	1.373
1000	5000	21.163
1000	10000	24.584

Another important factor for execution time is the number of collision of buckets. For two small W values, we cannot see a distinct execution time difference, but we can say that small data set for largely sized hashtable might not be efficient. W/N value is directly correlated to the number of collisions. For example for W = 10000 minimum number of collisions will occur for N = 1000 then for N = 500 and N = 100 respectively.

We resolve collisions by chaining, everytime we encounter a collision we have to manipulate linked list of the corresponding bucket. In other words collisions might become an overhead if it occurs too much. execution time increases as the number of collisions increase.

As a conclusion, there is a relation between the number of locks ,the number of threads and the execution time. Increase in the number of threads while the other values stay the same, results in the decrease in time elapsed when they work on the same operation.