CS 342 Operating Systems Project II Report

Berke Egeli - 21601673 - Section 1, Mert Alp Taytak - 21602061 - Section 2

T/K	1	25	50	100
1	208.3µs	209.3 µs	192 µs	197 µs

Table 1. Execution time of 1000 operations with distinct keys where T = 1, N = 1000, W = 1000 (Average of 3 experiments)

When the thread number is set to 1, the number of locks do not have any effect on the execution time of the experiments. This makes sense, since at any point in time only 1 region of hash table will be accessed by the CPU.

T/K	25
1	295.3 µs
2	248.3 µs
5	965.6 µs
10	2162.6 µs

Table 2. Execution time of 1000 (across all of the threads) operations with distinct keys where K = 25, N = 1000, W = 1000 (Average of 3 experiments)

The experiments were conducted on a VM that utilized a total of 4 cores. There is a speed up of the total execution time when the number of threads increase from 1 to 2*, however, when we look at the total execution times, it becomes clear that once the number of threads pass the number of available cores, the execution time increases considerably. This is due to the number context switches needed by the CPU as well as the contention that is created when multiple threads try to access the locked regions of the shared hash table.

*(This is mostly an experimental anomaly, when more than 3 experiments are conducted, 2 total execution time starts to increase.)

T/N	100	1000
1	275.3 µs	251 µs
2	258.3 µs	242.3 µs
5	875 µs	714 µs
10	2020.6 µs	2186.6 µs

Table 3. Execution time of 1000 (across all of the threads) operations with distinct keys where K= 100, W= 1000 (Average of 3 experiments)

When we look at the Table 3 we see that the effect of number of threads is similar when N = 100 and N = 1000. There isn't a considerable speed up that comes with the increase in the size of the hash table. However, there is a slight increase in the overall execution time of the experiments when N is set to 1000 as opposed to N is set to 100. This might come from the fact that keys are less likely to collide when N is equal to 1000 because of the hashing algorithm that we use.

T/W	1000	5000
1	251 µs	570.6 µs
2	242.3 µs	831.6 µs
5	714 µs	931.3 µs
10	2186.6 µs	1737.3 µs

Table 4. Execution time of experiments with different amount of distinct keys where N = 1000, K = 100 (Average of 3 experiments)

In the experiment where W is set to 1000 there are 500 distinct keys and, in the experiment, where W is set to 5000 there are 2500 distinct keys. From the table we can clearly see that when the number of operations and the keys increase the total execution time increase as well.

The interesting thing to note here is that, once the number of distinct keys increase, the contention in the hash table increases as well, and therefore unlike W = 1000 case when W = 5000 the total execution time of T = 2 is more than T = 1. (The fact that execution time of T = 10 for W = 1000 being more than the execution time of W = 5000 is probably an experimental error.)

T/K	1	25	50	100
1	208.3 µs	209.3 µs	192 µs	197 µs
2	404.3 µs	317.3 µs	228.3 µs	240.3 µs
5	1050 µs	430 µs	391.3 µs	434.3 µs
10	2532.6 µs	1778 µs	1708.6 µs	1919.3 µs

Table 5. Execution time of the experiments where N = 1000 and W = 1000 (Average of 3 experiments)

From table 5 it is easy to see that the overall performance of the hash table increases when the lock amount increases and the performance decreases when the number of threads increase. This makes sense, since smaller regions will be locked when the lock amount increases and when the number of threads increase the contention increases as different threads try to access the same region simultaneously.