**CS 342 Operating Systems Project II Report**

**Berke Egeli - 21601673 - Section 1**

**Mert Alp Taytak - 21602061 - Section 2**

**Experiment 1: Effect of K on Execution Time**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 one region of the hash table will be accessed by the CPU.

**Experiment 2: Effect of T on Execution Time**

|  |  |
| --- | --- |
| 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 four 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 three experiments are conducted, T = 2 total execution time starts to increase. The reason for this anomaly is that our key generation is the same for all threads and depending on the scheduling some runs gets more contention.

**Experiment 3: Effect of T versus N on Execution Time**

|  |  |  |
| --- | --- | --- |
| 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 is not a considerable speed up that comes with the increase in the size of the hash table. However, there is a slight improvement 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 with the hashing algorithm that we use.

**Experiment 4: Effect of T versus W on Execution Time**

|  |  |  |
| --- | --- | --- |
| 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 increases the total execution time increases 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. 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 caused by external factors.)

**Experiment 5: Effect of T versus K on Execution Time**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 increases. This makes sense since smaller regions will be locked when the lock amount increases, and when the number of threads increases the contention increases as different threads try to access the same region simultaneously.

**Conclusions**

Following is the summary of our conclusions listed,

* Increasing the number of threads up to the number of cores improves performance.
* Having more threads than cores decreases performance through increased number of context switches.
* Increasing the number of locks decreases the region sizes and improves performance through reduced contention.
* Increasing the table size improves performance through reduced contention.