Fundamentals of Computing Systems Homework Assignment #4 - EVAL

1. EVAL Problem 1

   (a) **Maximum Number of CPUs that I can use for this EVAL problem**
   In order to check CPU numbers in my current interactive session of SCC, I use "lscpu" command in my terminal. The result is as following:

```
Architecture:        x86_64
CPU op-mode(s):      32-bit, 64-bit
Byte Order:          Little Endian
CPU(s):              32
On-line CPU(s) list: 0-31
Thread(s) per core:  1
Core(s) per socket:  16
Socket(s):           2
NUMA node(s):        4
Vendor ID:           GenuineIntel
CPU family:          6
Model:               85
Model name:          Intel(R) Xeon(R) Gold 6242 CPU @ 2.80GHz
Stepping:            7
CPU MHz:             2800.000
BogoMIPS:            5600.00
Virtualization:      VT-x
L1d cache:           32K
L1i cache:           32K
```

```
L2 cache:              1024K
L3 cache:              22528K
NUMA node0 CPU(s):     0,4,8,12,16,20,24,28
NUMA node1 CPU(s):     1,5,9,13,17,21,25,29
NUMA node2 CPU(s):     2,6,10,14,18,22,26,30
NUMA node3 CPU(s):     3,7,11,15,19,23,27,31
```

The forth row "CPU(s)" says I have 32 CPUs, which is larger than $w + 2 = 4$.

(b) **Compare the Utilization of two threads**

Just like the previous Eval Problems, I measured each threads' Utilizations.

	Thread 0	Thread 1
Number of Requests	725	775
Busy Resource Time	37.25493199995253 msec	36.92602499912027 msec
Total Available time	40.83810000005178 msec	40.742694000015035 msec
Utilization	91.33651416896219 %	90.63226157579723 %

Table 1: Each Thread Utilizations

The Utilizations of each threads are 91.336 % and 90.632 %. The request row presented for Thread 0 and Thread 1 indicates that the load is quite balanced between the two threads. Thread 0 handled 725 requests, while Thread 1 handled 775, showing a small difference of 50 requests out of a total of 1500. Despite this slight variation, the busy resource times are nearly identical, with Thread 0 at 37.255 msec and Thread 1 at 36.926 msec, reflecting only a minor difference of about 0.33 msec. Similarly, the total available times for both threads are almost the same, with Thread 0 having 40.838 msec and Thread 1 having 40.743 msec, differing by just 0.095 msec. The utilization rates are also very close, with Thread 0 at 91.34% and Thread 1 at 90.63%, showing a minor difference of 0.7%. Overall, this data suggests that the workload has been evenly distributed between the two threads, with both exhibiting similar performance metrics, leading to the conclusion that the system is well balanced in terms of thread utilization.

(c) **Average Response Time Comparison**

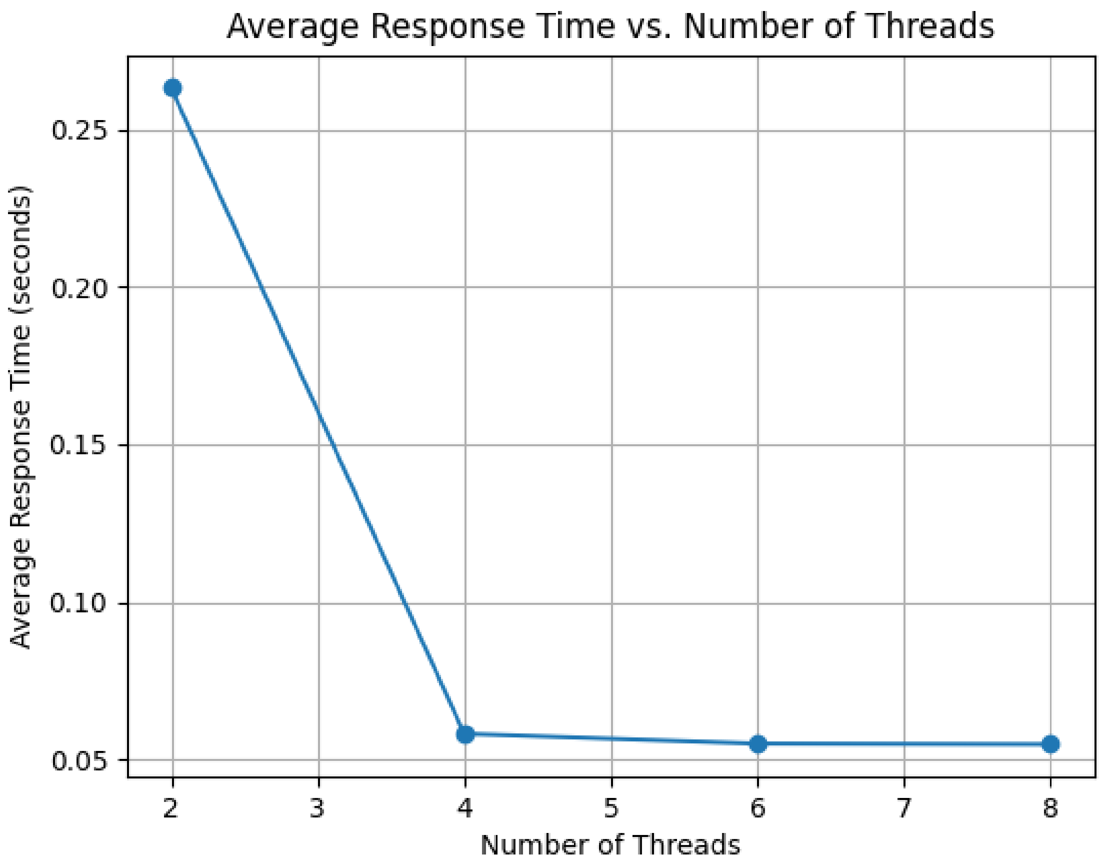

Figure 1: Average Response Time of Each Threads Comparison

As the number of threads increases, the average response time decreases significantly, particularly between 2 and 4 threads, where we observe a super-linear improvement. The average response time drops from 0.263 milliseconds with 2 threads to 0.058 milliseconds with 4 threads, indicating a more-than-proportional performance boost due to added parallelism. However, as the thread count increases further to 6 and 8 threads, the improvement becomes less pronounced, with the average response time decreasing only marginally. Due to resource contention or overhead associated with managing additional threads, adding more threads produces only slight improvements in the system, suggesting diminishing returns. As a result, thread count increases initially provide substantial gains, but after a certain point, the benefits plateau.

(d) **Rejection Rate as more workers available**

The statement that "if X is the rejection rate with 1 worker, then X/W is the rejection rate with W workers" suggests that the rejection rate should decrease proportionally as the number of workers increases. The results of my experiment suggest that this is not fully accurate.

With 1 worker, the rejection rate was 53 out of 1500 requests (around 3.53%). However, with 2 workers, there were no rejections, which is a substantial improvement beyond what would be predicted by the "X/W" formula. It would be expected that 2 workers would have half the rejection rate (X/2) but instead the system handled all requests without any rejections.

The reason this happens is because increasing the number of workers allows the server to process more requests concurrently, reducing the likelihood of requests being delayed long enough to be rejected. With 1 worker, the single thread may not process requests quickly enough to prevent some from being rejected due to timeouts or queue overflows. A server can handle multiple requests in parallel when you add more workers, improving response times and reducing the load on each individual thread, thereby eliminating rejections.

Thus, the assumption that the rejection rate scales linearly with "X/W" does not hold in this case. The reduction in rejection rate is non-linear and likely depends on the server's ability to process requests efficiently as more threads are added, along with factors such as queue size and request distribution.