Effects of Engine Average Execution Time on CPU Usage Steadiness

Hypothesis

If the engine's average execution time is greater than the scan period then the computer's total CPU usage will not be stable.

Materials

- KC Small Training Room Laptop 8. Laptop Details – Model: XPS L702X, OS: Windows 7 SP1 Enterprise, Processor: Intel Core i7
- Wonderware System Platform 2012 R2
- Performance Monitor
- Fog Image: KCTrainingLaptopOptimizing

Methods

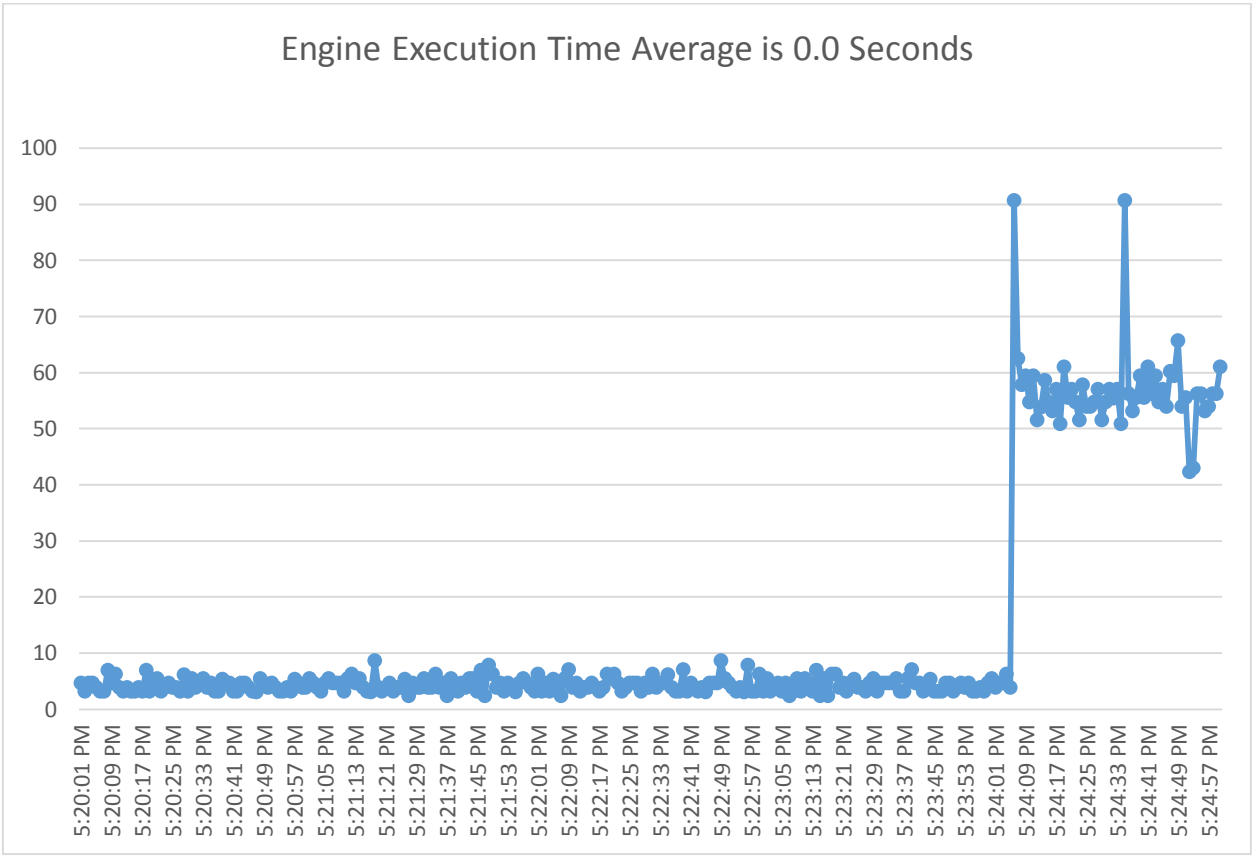
- Set engine scan time to 10,000 MS to help with measure performance. Performance monitors lowest resolution is once a second. Using an engine scan time of 1 second could have made for confusing results.
- Imaged KC Small Training Room Laptop 8 with fog image KCTrainingLaptopOptimizing. The image already had a galaxy with object to generate load. Only needed to adjust the engine execution time to 10,000 MS.
- Installed 2012 R2 to the physical machine's operating system instead of using a virtual machine to avoid invalidating the CPU usage data with vmware's automatically adjusting CPU capacity algorithms.
- Of the 8 available cores in the core i7 cpu 6 were disabled. Only using two cores makes it simpler to understand how object CPU usage is divided among cores. Disable cores by going to start->run->msconfig->boot->advanced options->number of processors.
- When running experiments closed all applications then waited 1 minute before recording any results. This would reduce noise in the experiment from an application taking cpu or deployment using cpu.
- Used an object in the galaxy named "Loader". Loader is an object in the galaxy already deployed on the fog image "KCTrainingLaptopOptimizing". Loader is a simple object configured with a script to create cpu load. The script is an execution type script with a while true condition checking an area object discrete attribute called "DoLoadTest". The float UDAs are defined "squareRoot","sine","count". There is also "cosine" as a Boolean, althought it should probably be float(just a mistake from creators of optimizing application server class) :

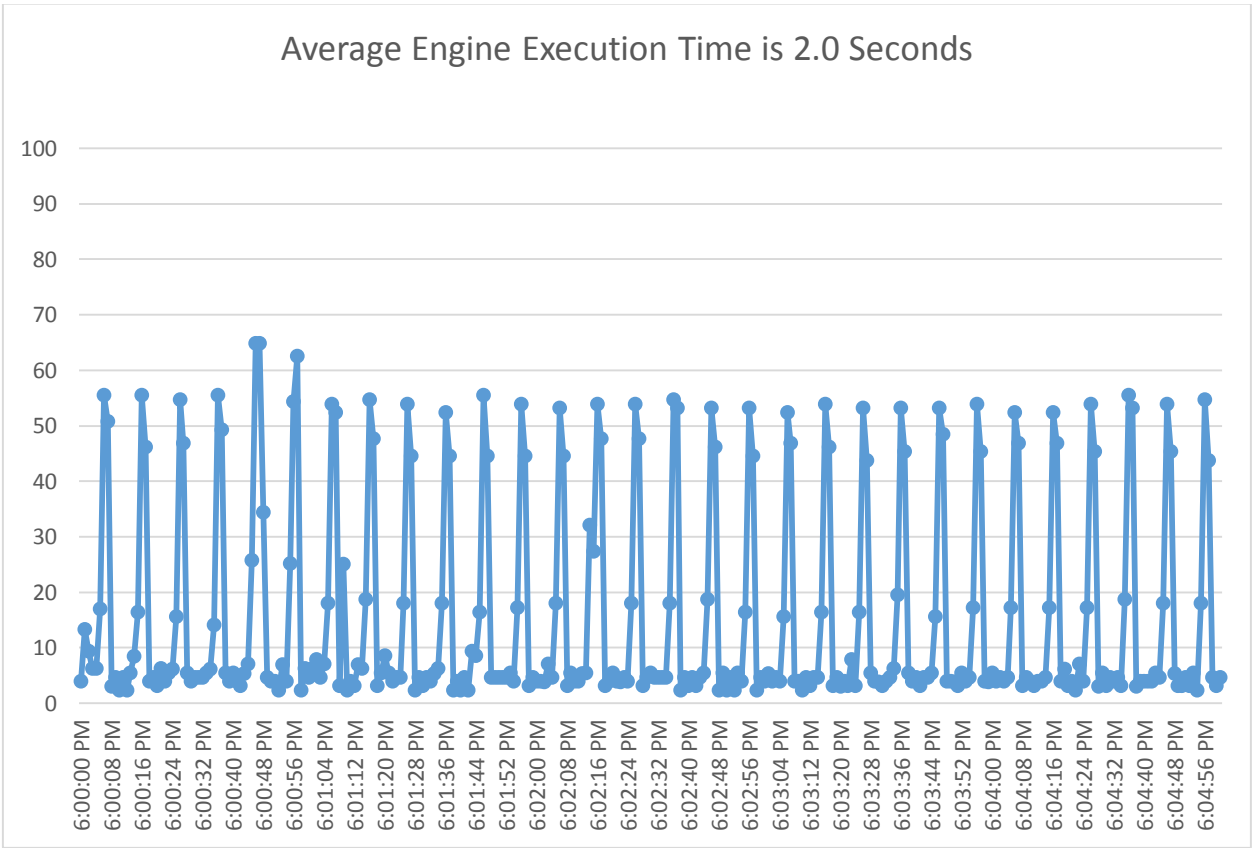
```
dim i as integer;

me.count = me.count+1;

for i = 1 to 1000
      me.sine = Sin( me.count );
      me.squareRoot = Sqrt( me.count );
      me.cosine = Cos (me.count );
      'me.sine = i;
      'me.squareRoot = i;
      'me.cosine = i;
next;
```

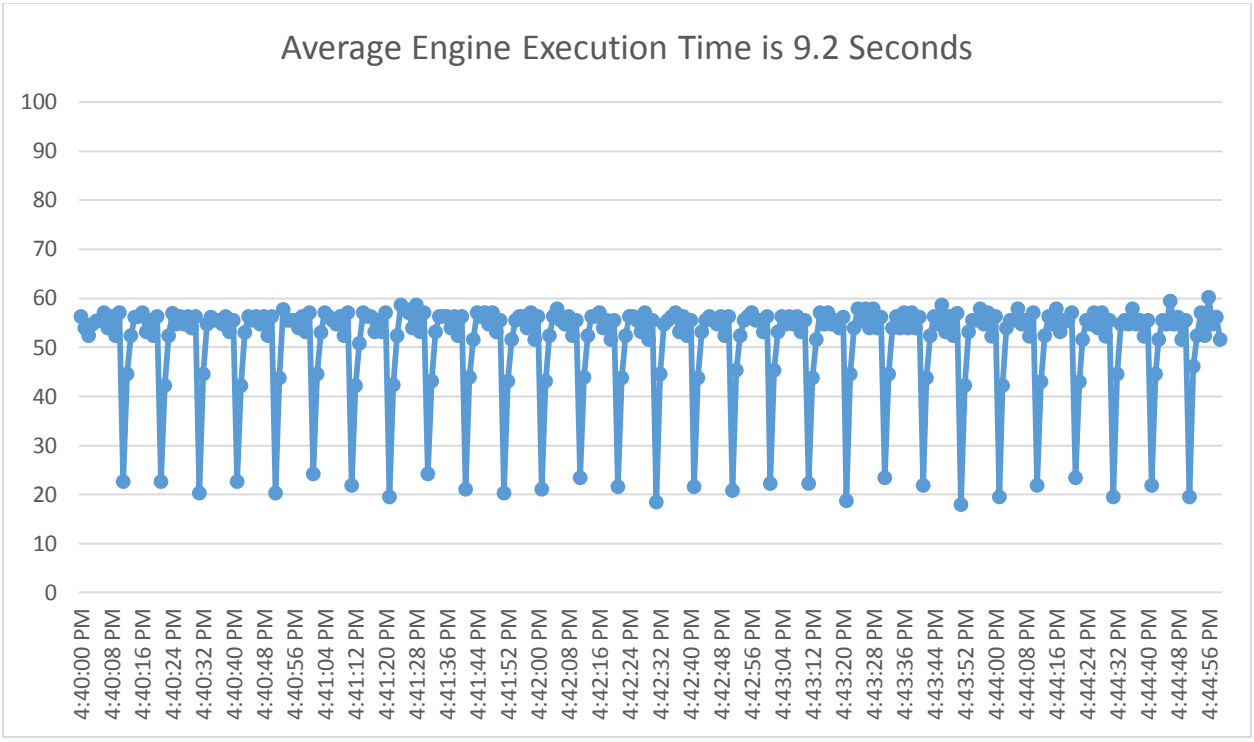
- Made 10,000 Loader objects using dbdump/load. The naming convention was Loader_XXXX. Starting at Loader_0000 and ending at Loader_9999. For a total for 10,000 loader objects.

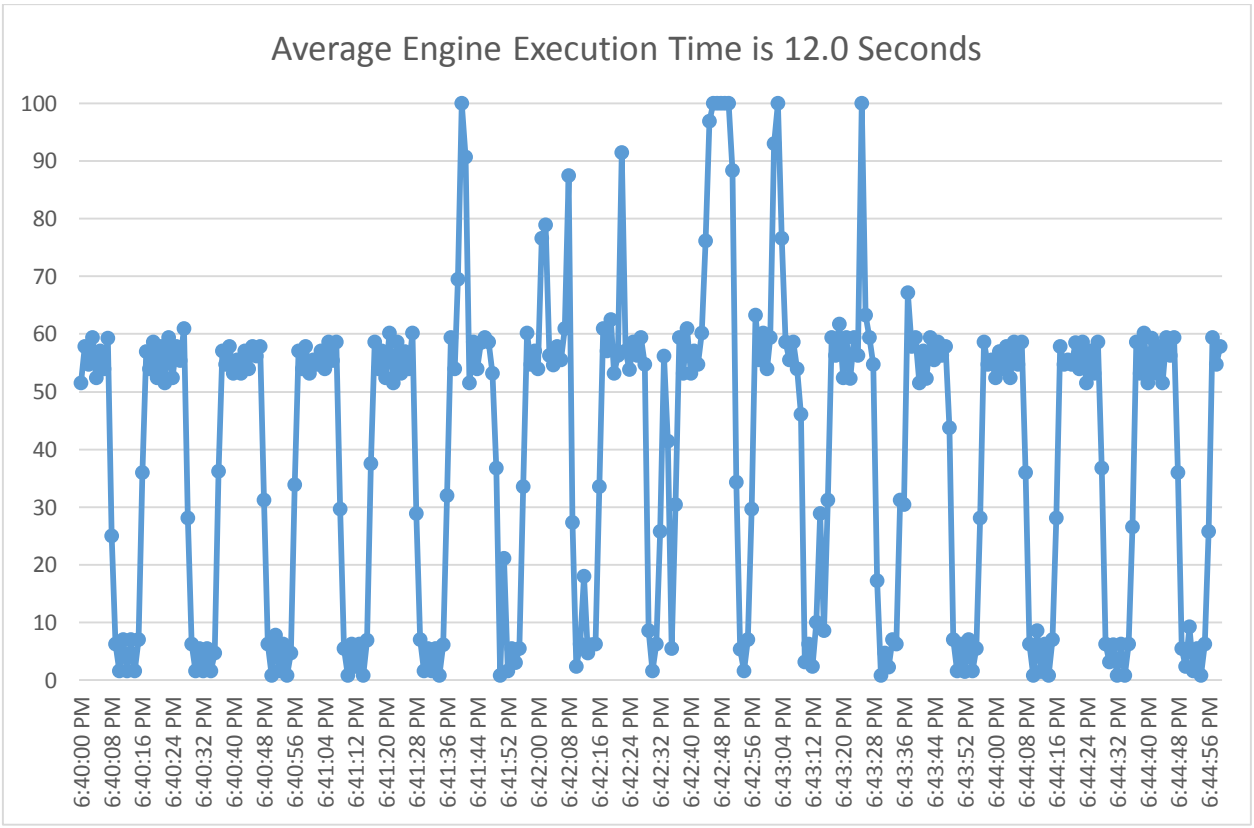
Data

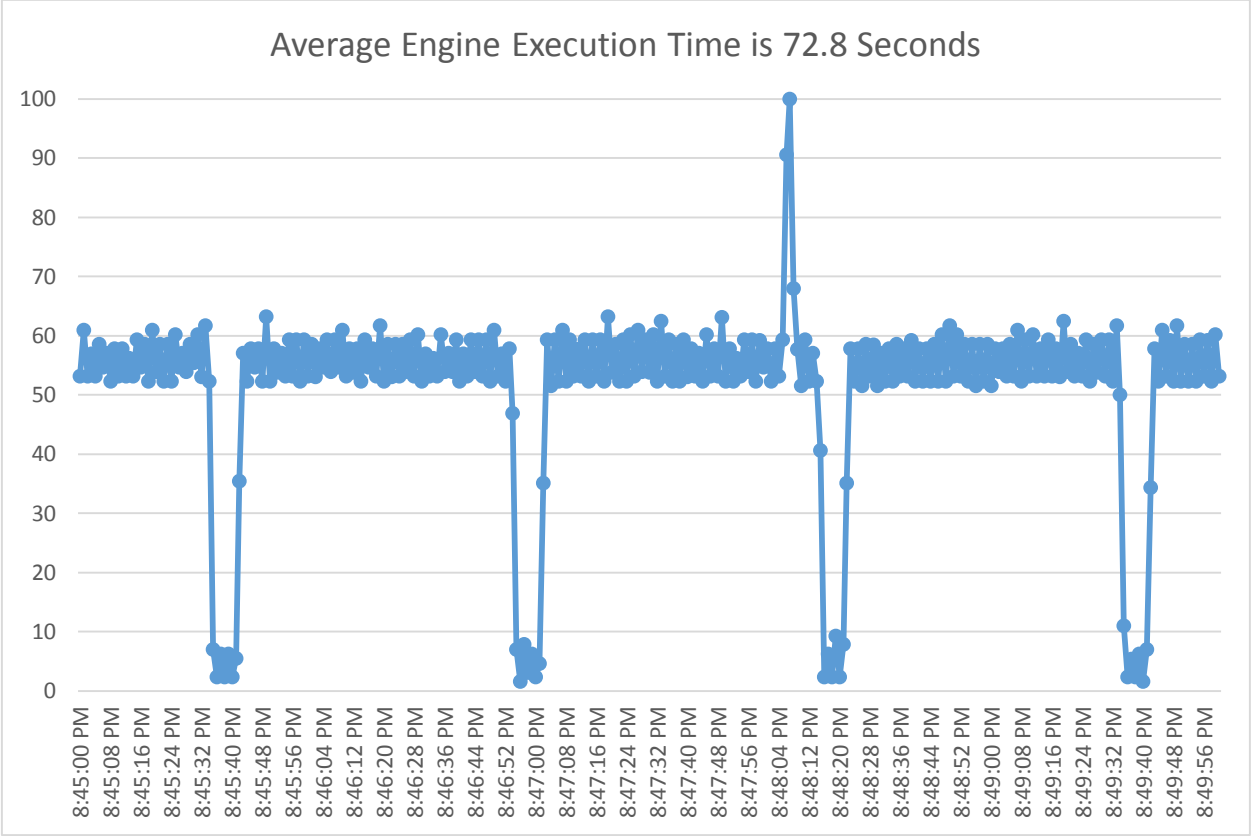
Results






Average Engine Execution Time is 9.2 Seconds
100
90
80
70
60
50
40
30
20
10
0
4:40:00 PM
4:40:08 PM
4:40:16 PM
4:40:24 PM
4:40:32 PM
4:40:40 PM
4:40:48 PM
4:40:56 PM
4:41:04 PM
4:41:12 PM
4:41:20 PM
4:41:28 PM
4:41:36 PM
4:41:44 PM
4:41:52 PM
4:42:00 PM
4:42:08 PM
4:42:16 PM
4:42:24 PM
4:42:32 PM
4:42:40 PM
4:42:48 PM
4:42:56 PM
4:43:04 PM
4:43:12 PM
4:43:20 PM
4:43:28 PM
4:43:36 PM
4:43:44 PM
4:43:52 PM
4:44:00 PM
4:44:08 PM
4:44:16 PM
4:44:24 PM
4:44:32 PM
4:44:40 PM
4:44:48 PM
4:44:56 PM


Average Engine Execution Time is 12.0 Seconds
100
90
80
70
60
50
40
30
20
10
0
6:40:00 PM
6:40:08 PM
6:40:16 PM
6:40:24 PM
6:40:32 PM
6:40:40 PM
6:40:48 PM
6:40:56 PM
6:41:04 PM
6:41:12 PM
6:41:20 PM
6:41:28 PM
6:41:36 PM
6:41:44 PM
6:41:52 PM
6:42:00 PM
6:42:08 PM
6:42:16 PM
6:42:24 PM
6:42:32 PM
6:42:40 PM
6:42:48 PM
6:42:56 PM
6:43:04 PM
6:43:12 PM
6:43:20 PM
6:43:28 PM
6:43:36 PM
6:43:44 PM
6:43:52 PM
6:44:00 PM
6:44:08 PM
6:44:16 PM
6:44:24 PM
6:44:32 PM
6:44:40 PM
6:44:48 PM
6:44:56 PM



Analysis

The hypothesis was shown to be true. All of the trend graphics demonstrate CPU usage that is not stable. The explanation for sudden reductions is CPU usage is the engine finishes executing objects then is left idle until the end of the scan period. Each trend chart would be well served for it's own analysis

Engine Execution time is 0.0

The point of this experiment was to establish a baseline. The trend chart shows about 5% CPU usage in a steady state. There was also a time of about 50% CPU usage. It is unknown what program was using this CPU, but the program(s) did add some noise to the other experiments. Experiments with execution times of 72.8 and 12.0 have noise in them. While this noise does cast some doubt on the experiments the trends are still clear to see and it seem unlikely the "noise" is really significant data or distorting the significant data. If these experiments were to be conducted again it would be advisable to identify sources of noise then eliminate them. The sources of noise were not eliminated in this experiment because of time constraints to conduct the experiment.

Engine Execution time is 2.0 Seconds

This experiment used the least amount of CPU. The CPU usage is at about 55% for 2 seconds then about 5% for 8 seconds. Up and down CPU usage is repeated of the course of the 5 minute experiment.

Engine Execution time is 9.2 Seconds

The minimum CPU usage was the least for this of all experiments. Because the script execution took 9.2 seconds this left only .8 of IDLE time. The operating system likely had a backlog of non-wonderware related process that needed to execute. Leaving the minimum CPU usage at about 20% instead of about 5%.

Engine Execution time is 12.0 Seconds

The trend is very similar to the 2.0 Second experiment. In the 2.0 Second experiment there was 8 seconds of about 5% CPU usage, the same is true of this experiment. The only significant difference is there was 12 seconds of about 55% CPU usage instead of 2 seconds of about 55% CPU usage. What this experiments mean is when the execution time exceeds the scan time another scan isn't immediately started, there is still an idle period. There were several spikes of 100% CPU usage and near 100% CPU usage, likely just noise from other processes.

Engine Execution time is 72.8 Seconds

The idea of this experiment was to do an extreme over run of the scantime to see if strange results happened. The results were consistent with other experiments. After running for 72.8 seconds there was a 7.2 second idle period.

Conclusions

Many people assume as object count increases the CPU usage will linearly increase meaning CPU loading is as simple as adding more CPUs for more Objects. These experiments break the assumption.

What this means for systems in the field is changing scan times, moving engines, creating more engines, etc. are adjustments that can be made to make a system more responsive or performance. Although these kinds of adjustments are likely not worth the effort on small or medium sized system. On a small system any issue with responsiveness or performance will not occur because of how small they are are. On a medium system any issue can be fixed by buying adding just a little more physical resources, add a little more physical resources will be much cheaper than spending money on engineering to load balance CPUs. On medium systems it is still worth understanding an engine only uses one core's worth of CPU capacity and adding more engines will use more cpus, but optimizing scan times does not seems worth the engineering to do so. On large systems however it will be worth the engineering to get as much responsiveness and performance as possible. Making savings on performance could save a dozen blade servers. Making responsiveness improvements will improve dozens of operator's effectiveness with the HMI every day.

References

- Used this article as a guide to write this Lab Report: http://chemistry.about.com/od/chemistrylabexperiments/a/labreports.htm