SQL Elapsed Time Collection Method Comparision Analysis

Author: Craig Shallahamer (craig@orapub.com), Version 1a 22-July-2011

Background and Purpose

The purpose of this notepad is to investigate three methods of gathering SQL elapsed times; SQL trace, source code instrumentation, and OraPub's Elapsed Time Sampler (beta v4).

Experimental Data

Below is all the experimental data. The experiment was run on a Dell single four-core CPU, Oracle 11.2G. According to "cat /proc/version": Linux version 2.6.18-164.el5PAE (mockbuild@ca-build10.us.oracle.com) (gcc version 4.1.2 20080704 (Red Hat 4.1.2-46)) #1 SMP Thu Sep 3 02:28:20 EDT 2009. There was a tremendous CBC latch contention load, the OS was CPU bottlenecked he sample set interval was 5 minutes.

The order of sample data is elapsed time (seconds).

The three sets are:

ssToolV3 data was gathered using the OraPub SQL Elapsed Sampler version 3a.
ssIntrumentedV1 data was gathered using time gathered just before and after the sql statement was executed each time.
ssTraceV4 data was gathered using a shell script to parse through a trace file, which only had data for the specific sqlid under observation.

In[1]:=
```
ssToolV3 = {2.704671, 1.796647, 1.632453, 1.837511, 1.884671, 1.905275, 2.18978, 2.025389,
   1.611901, 1.836863, 2.505555, 3.153465, 2.84024, 2.476605, 2.094691, 2.173646, 2.519876,
   2.298478, 2.374658, 1.93859, 2.888561, 2.364635, 2.087664, 2.666636, 1.667784, 2.177537,
   1.899662, 1.731623, 1.72645, 1.75771, 3.55947, 1.919624, 1.844708, 2.75665, 3.034486,
   1.814696, 1.837491, 1.898573, 1.627535, 1.910661, 1.756651, 1.98259, 1.834711, 2.560435,
   2.606642, 2.395607, 1.671747, 1.673795, 3.231439, 3.096578, 2.059662, 5.881336, 2.974423,
   2.341667, 3.629601, 5.089597, 3.400618, 3.744663, 4.535504, 3.479637, 2.544644, 3.372578,
   2.440392, 3.622236, 3.436203, 3.70184, 5.330186, 2.818602, 2.459942, 3.818559, 2.416704,
   2.770282, 3.768572, 3.536641, 2.861564, 2.161639, 6.221447, 2.427597, 2.355631};
ssInstrumentedV1 = {2.711167, 1.801678, 1.642338, 1.843705, 1.85499, 1.883658, 2.181674,
   2.073582, 1.629159, 1.825775, 2.674796, 3.171358, 2.865554, 2.544855, 2.099589,
   2.316396, 2.561801, 2.306012, 2.379955, 1.915868, 2.896979, 2.346932, 2.117401,
   2.658505, 1.662047, 2.169463, 1.909734, 1.745047, 1.726111, 1.767757, 3.581717,
   1.925027, 1.855934, 2.852958, 3.159873, 1.81302, 1.877463, 1.86855, 1.637116, 1.917331,
   1.860883, 1.998871, 1.846868, 2.576076, 2.627712, 2.458729, 1.679608, 1.686958,
   3.22731, 3.118615, 2.076141, 2.916338, 2.366096, 2.996268, 2.31827, 3.675232, 5.263303,
   3.428657, 3.759504, 4.56331, 3.481395, 2.554238, 3.373345, 2.424349, 3.665441, 3.418634,
   3.709849, 5.330086, 2.85883, 2.45115, 3.825466, 2.398828, 2.859167, 3.778904, 3.539237,
   2.973861, 2.168657, 6.318705, 2.418443, 2.367844, 2.141678, 1.605836, 1.688933};
ssTraceV4 = {1.797346, 1.640636, 1.842014, 1.853523, 1.882048, 2.180093, 2.055014, 1.626876,
   1.823635, 2.573840, 3.169362, 2.863241, 2.543169, 2.097604, 2.214430, 2.524729, 2.304538,
   2.377945, 1.914216, 2.894981, 2.345491, 2.115687, 2.656515, 1.659811, 2.167793, 1.907696,
   1.742786, 1.723897, 1.765280, 3.579856, 1.922513, 1.853502, 2.746276, 3.042696, 1.810769,
   1.875932, 1.866923, 1.635630, 1.890313, 1.775378, 1.996670, 1.844642, 2.573766, 2.613248,
   2.412404, 1.667942, 1.684852, 3.225041, 3.117141, 2.074641, 2.905923, 2.347282, 2.994653,
   2.316767, 3.673653, 5.061426, 3.427076, 3.747277, 4.561126, 3.479720, 2.552670, 3.371810,
   2.422721, 3.663710, 3.416866, 3.708151, 5.328181, 2.855186, 2.449320, 3.823706, 2.396365,
   2.856855, 3.776408, 3.537316, 2.872711, 2.167043, 6.234509, 2.416293, 2.365536};
```

Basic Statistics

In this section I calculate the basic statistics, such as the mean and median. My objective is to ensure the data has been collected and entered correctly and also to compare the two datasets to see if they appear to be different.

In[4]:=
```
ssTool = ssToolV3;
ssInstrumented = ssInstrumentedV1;
ssTrace = ssTraceV4;
myData =
 {
  {"Trace", Mean[ssTrace], Median[ssTrace], StandardDeviation[ssTrace], Length[ssTrace]
  },
  {"Instr", Mean[ssInstrumented], Median[ssInstrumented],
   StandardDeviation[ssInstrumented], Length[ssInstrumented]
  },
  {"Tool", Mean[ssTool], Median[ssTool], StandardDeviation[ssTool], Length[ssTool]
  }
 }
toGrid = Prepend[myData, {"Method", "Mean", "Median", "Std Dev", "Samples"}];
Grid[toGrid, Frame → All]
```

Out[7]=
```
{{Trace, 2.61018, 2.39637, 0.900277, 79},
 {Instr, 2.59723, 2.37996, 0.902959, 83}, {Tool, 2.64538, 2.4167, 0.969826, 79}}
```

Out[9]=

Method	Mean	Median	Std Dev	Samples
Trace	2.61018	2.39637	0.900277	79
Instr	2.59723	2.37996	0.902959	83
Tool	2.64538	2.4167	0.969826	79

Sample Comparison Tests (when normality does NOT exist)

If our sample sets are **not normally distributed,** we can not perform a simple t-test. We can perform what are called location tests. I did some research on significance testing when non-normal distributions exists. I found a very nice reference:

http://www.statsoft.com/textbook/nonparametric-statistics

The paragraph below (which is from the reference above) is a key reference to what we're doing here:

...the need is evident for statistical procedures that enable us to process data of "low quality," from small samples, on variables about which nothing is known (concerning their distribution). Specifically, nonparametric methods were developed to be used in cases when the researcher knows nothing about the parameters of the variable of interest in the population (hence the name nonparametric). In more technical terms, nonparametric methods do not rely on the estimation of parameters (such as the mean or the standard deviation) describing the distribution of the variable of interest in the population. Therefore, these methods are also sometimes (and more appropriately) called parameter-free methods or distribution-free methods.

Being that I'm not a statistician but still need to determine if these sample sets are significant different, I let *Mathematica* determine the appropriate test. Notice that one of the above mentioned tests will probably be the test *Mathematica* chooses.

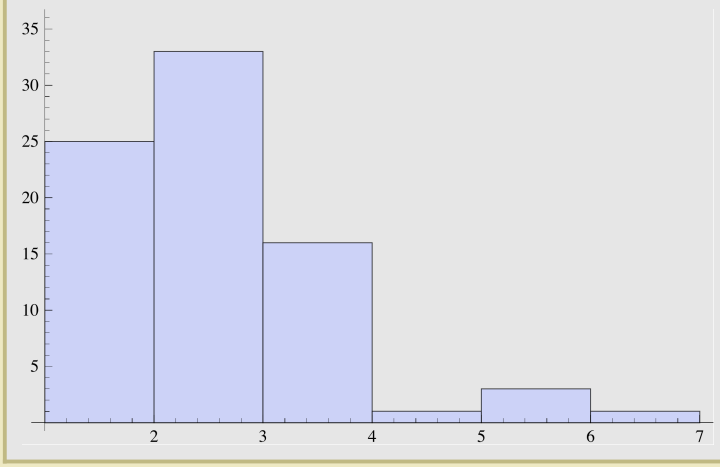
Note: If we run our normally distributed data through this analysis (speically, the "LocationEquivalenceTest"), *Mathematica* should detect this and use a more appropriate significant test, like a t-test.

Here we go with the hypothesis testing (assuming our sample sets are not normally distributed):

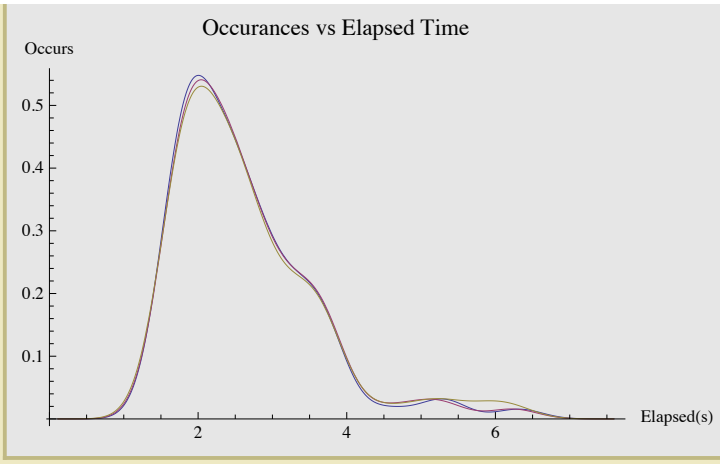
1. Our P value threshold is 0.05, which is our alpha.
2. The null hypotheses is the two populations have the same mean. (Remember we have to sample sets, which is not the population.)
3. Do the statistical test to compute the P value.
4. Compare the result P value to our threshold alpha value. If the P value is less then our threshold, we will reject the null hypothesis and say the difference between our samples is significant. (Which is what I'm hoping to see.) However, if the P value is greater than the threshold, we cannot reject the null hypothesis and any difference between our samples are not statistically significant; randomness, picked the "wrong" samples, etc.

In[10]:=
```
Histogram[ssTool]
SmoothHistogram[{ssInstrumented, ssTrace, ssTool},
 PlotLabel → "Occurances vs Elapsed Time", AxesLabel → {"Elapsed(s)", "Occurs"}]
SmoothHistogram[{ssInstrumented, ssTrace}, PlotLabel → "Occurances vs Elapsed Time",
 AxesLabel → {"Elapsed(s)", "Occurs"}]
t1 = LocationEquivalenceTest[{ssInstrumented, ssTool}, {"TestDataTable", "AutomaticTest"}]
t2 = LocationEquivalenceTest[{ssInstrumented, ssTrace}, {"TestDataTable", "AutomaticTest"}]
t3 = LocationEquivalenceTest[{ssTrace, ssTool}, {"TestDataTable", "AutomaticTest"}]
```

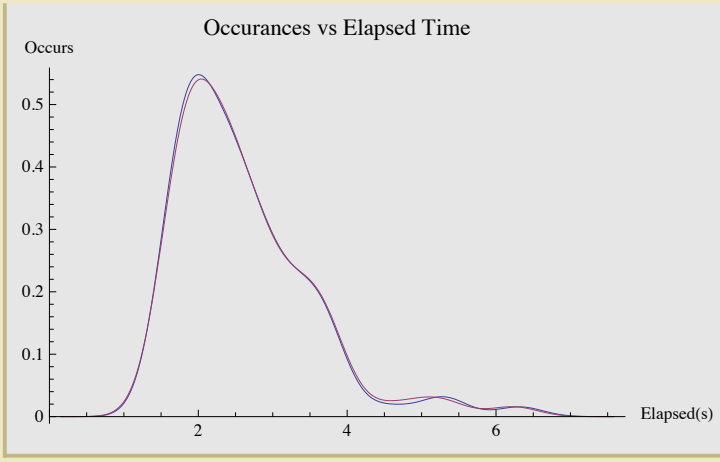
Out[10]=



Out[11]=



Out[12]=



Out[13]=

	Statistic	P-Value
Kruskal-Wallis	0.0264102	0.871495

, KruskalWallis

Out[14]=

	Statistic	P-Value
Kruskal-Wallis	0.000631555	0.980044

, KruskalWallis

Out[15]=

	Statistic	P-Value
Kruskal-Wallis	0.00220393	0.962735

, KruskalWallis

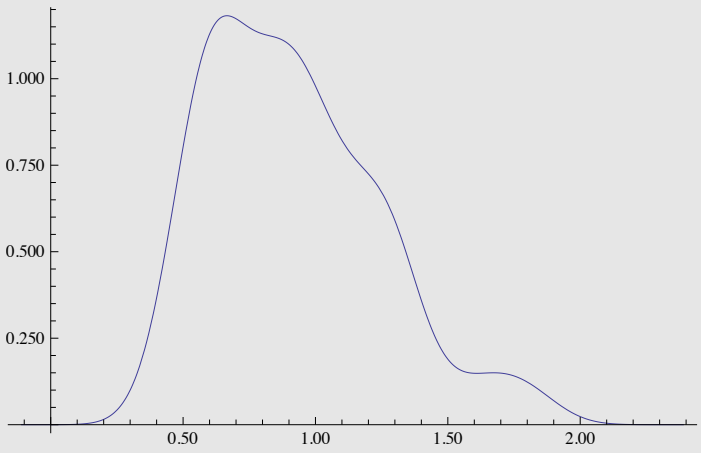
Log Normal Distribution Fit Test

In[16]:=
```
transformedSS = {};
Table[
  AppendTo[transformedSS, Log[ssTool[[i]]]],
  {i, 1, Length[ssTool]}
 ];
transMean = Mean[transformedSS]
transSD = StandardDeviation[transformedSS]
SmoothHistogram[transformedSS]
```

Out[18]=
```
0.918189
```

Out[19]=
```
0.319816
```

Out[20]=

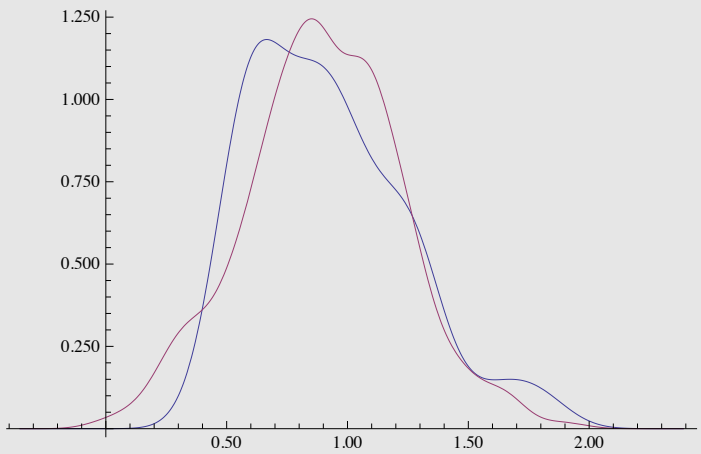


In[21]:=
```
ndSS = RandomVariate[NormalDistribution[transMean, transSD], 500];
Take[ndSS, 5]
```

Out[22]=
```
{1.09804, 1.05787, 0.496877, 0.549886, 0.789067}
```

In[23]:=
```
SmoothHistogram[{transformedSS, ndSS}]
```

Out[23]=



In[24]:=
```
h = DistributionFitTest[transformedSS]
```

Out[24]=
```
0.0140837
```