SQL Statement Elapsed Time Distribution Analysis

Author: Craig Shallahamer, Version 3e on 31-Jan-2011

The Point and Purpose

In my performance work I often am presented with a Statspack or AWR report. After performing my 3-Circle Analysis the high resource SQL statement(s) are easily identified. This presents me the total elapsed time and the number of executions that occured during the report interval. This allow me to easily compute the average statement elapsed time. However, I find that in communicating using only the average SQL elapsed time implies the situation seem worse than it really is. And sometimes people are surprised that the SQL statement takes "that long." I want to better understand, describe, and communicate SQL statement elapsed time. So in essence, my desire to be able to responsibly provide more detail with only limited inputs.

Now if SQL statement elapsed time confirms to a specific statistical distribution, then with minimal inputs I will be able to say much more, communicate better, and serve my clients and students better. But even if it does not, if a general pattern develops I may be able to say something like, "The typical elapsed time will most likely be much less than the average." Just saying that with responsible confidence will be valuable to me.

So my quest simply is:

1. Does the sample data conform to an existing and well understood statistical distribution. (test both statistically an visually)
2. If so, what else can I say about the data.
3. Validate the "what else can I say about the data" by comparing several real and actual samples statistics (actual values) with the mathematical predicted values.
4. Provide a few example using Statspack and AWR data.

This notebook will focus on number 1. This notebook is not related to a specific data set. As the *Data Loading and Verification* section makes obvious, it is more an experimental template that can be used on any appropriately formatted dataset.

A note about hypothesis testing. Hypothesis tests give quantitative answers to common questions, such as how good the fit is between my data and a particular distribution, whether these distributions have the same mean or median, and whether these datasets have the same variability. The hypothesis testing done in this notebook is comparing a data set with a known statistical distribution. It is not about comparing two sample sets. In my performance work, I many times want to know if a change made something faster. In these types of situations, I am comparing two sample sets to see if they are significantly different. In contrast, what I'm doing in this experiment and this associated notebook is taking a sample set of SQL elapsed times and seeing if their elapsed time pattern fits one of the common statistical distributions. So when you see the hypothesis testing section below, it may seem a little strange to many performance-minded DBAs.

The data source is from a tool kit developed by OraPub's Craig Shallahamer. The tool kit gathers performance statistics (e.g., CPU consumption) regarding a specific SQL statement over a given period of time. The data collected is inserted into an Oracle table therein queried and used in this analysis. The data collection tool kit can be found at OraPub's web-site, http://www.orapub.-com and then doing a search for "sql distribution".

Data Loading and Verification

Set the full path location of the raw data file. There is to be no headers in the data file, just data. You can download my data gathering tool from my web-site. Just do an OraPub search for "sql elapsed".

In[1]:=
```
filename = "/Users/cshallah/Desktop/Garret8qtkxy0g5d1p3_1.txt";
```

Import the experimental results. The "Table" option allows all the data to be imported easily and perfectly. I also asked for the number of data rows to ensure all my samples have been imported.

In[2]:=
```
allData = Import[filename, "Table"];
Length[allData]
```

Out[3]=
```
506
```

Let's check the first and last data row...just to be sure.

In[4]:=
```
firstRow = allData[[1, All]]
lastRow = allData[[Length[allData], All]]
```

Out[4]=
```
{8qtkxy0g5d1p3,2282376281, 1, 1, 0., 3., 0.1, 0.1}
```

Out[5]=
```
{8qtkxy0g5d1p3,2282376281, 506, 1, 8., 127., 10., 44.259}
```

All the grunt statistical work will be performed on the samples from a particular column, so we need to segment the data by column. I labeled each column so it will be easier to work with.

In[6]:=
```
executions = allData[[All, 3]];
piorExe = allData[[All, 4]];
lioExe = allData[[All, 5]];
cpuMsExe = allData[[All, 6]];
wallMsExe = allData[[All, 7]];
```

I want to show the complete first complete data sample by its columns, just to ensure the data has been parsed correctly. Compare the below with the first row listing above.

In[11]:=
```
Print[executions[[1]], ", ", piorExe[[1]], ", ",
 lioExe[[1]], ", ", cpuMsExe[[1]], ", ", wallMsExe[[1]]]
```

```
1, 0., 3., 0.1, 0.1
```

If all looks good, then let's pick which data element we want to analyze. Since the main purpose of this notebook is elapsed time analysis, regardless of the *sampleDataSet* variable setting, all the graphical headings will reference elapsed time (which is the same as "wall time").

In[12]:=
```
sampleDataSet = wallMsExe;
```

Statistical Analysis

Core Numeric Statistics (from actual sample set)

In[13]:=

```
sampleCount = Length[sampleDataSet];
sampleMean = Mean[sampleDataSet];
sampleMedian = Median[sampleDataSet];
sampleStdDev = StandardDeviation[sampleDataSet];
sampleMinValue = Min[sampleDataSet];
sampleMaxValue = Max[sampleDataSet];
quantile10pct = Quantile[sampleDataSet, 0.10];
quantile25pct = Quantile[sampleDataSet, 0.25];
quantile50pct = Quantile[sampleDataSet, 0.50];
quantile75pct = Quantile[sampleDataSet, 0.75];
quantile90pct = Quantile[sampleDataSet, 0.90];
quantile95pct = Quantile[sampleDataSet, 0.95];
Print["Quantile Values: 10%=", N[quantile10pct], " 25%=",
 N[quantile25pct], " 50%=", N[quantile50pct], " 75%=", N[quantile75pct],
 " 90%=", N[quantile90pct], " 95%=", N[quantile95pct]]
Print["Count=", sampleCount, " Mean=", N[sampleMean, {99, 3}], " Median=",
 N[sampleMedian, {99, 3}], " StdDev=", N[sampleStdDev, {99, 3}],
 " Minimum=", N[sampleMinValue], " Maximum=", N[sampleMaxValue]]
transformedSampleSet = {};
Table[
  AppendTo[transformedSampleSet, Log[sampleDataSet[[i]]]],
  {i, 1, Length[sampleDataSet]}
 ];
sampleMeanLog = Mean[transformedSampleSet];
sampleMedianLog = Median[transformedSampleSet];
sampleStdDevLog = StandardDeviation[transformedSampleSet];
Print["Log Transformed: Mean=", sampleMeanLog,
 " Median=", sampleMedianLog, " StdDev=", sampleStdDevLog]
Print["e=", N[e]]
```

```
Quantile Values: 10%=0.1 25%=0.19 50%=25.643 75%=65.615 90%=117.949 95%=183.295

Count=506 Mean=47.8747 Median=25.8148 StdDev=67.1709 Minimum=0.082 Maximum=475.927

Log Transformed: Mean=2.09084 Median=3.25092 StdDev=2.7534

e=2.71828
```

Hypothesis Testing: Introduction, Data Set, and Distribution Setup

For each selected distribution type (e.g., normal, uniform), I am comparing the actual sample data to two similarly created distributions. This allows me to create two tests related to each distribution. **The first test** (e.g., expoDist) is based on what I can personally determine from a Statspack/AWR report, which is only the average elapsed time. (I actually push this a little to far because from Statspack/AWR I can't determine the standard deviation...but I can from the experimental data.) **The second test** (e.g., expoDistEst) allows *Mathematica* to determine the best input parameters for a given distribution from the actual data set. This is actually very cool, because if *Mathematica* derives the parameters and its created distribution does not satisfactorily conform to our experimental data, then surely our experimental data is not "like" the given distribution. If the hypothesis test

claims the data (first test) and the similarly created distributions (second test) are significantly different, that's a very strong case that the sample data is truly different than the compared statistical distribution.

Note: I don't actually use the m and s variables, but they are required when using the EstimatedDistribution function. This is why they are repeated. The Estimated Distribution function amazingly trolls through the dataset and determines the best parameters (e.g., m, s) for the specified distribution. If the data set is truly, let's say exponential, then the derived mean should match the actual mean.

You'll notice I allowed the log normal distribution variables to print. I have personally tested that the *lm* (log mean) and *ls* (log standard deviation) variables can be easily derived by applying each sample to the log function and then simply calculating the mean and standard deviation. Apparently *Mathematica* is doing something similar!

In[34]:=
```
expoDist = ExponentialDistribution[1 / sampleMean];
expoDataSet = RandomVariate[expoDist, sampleCount];

expoDistEst =
 EstimatedDistribution[sampleDataSet, ExponentialDistribution[1 / m]]
expoDataSetEst = RandomVariate[expoDistEst, sampleCount];

normalDist = NormalDistribution[sampleMean, sampleStdDev];
normalDataSet = RandomVariate[normalDist, sampleCount];

normalDistEst = EstimatedDistribution[sampleDataSet, NormalDistribution[m, s]]
normalDataSetEst = RandomVariate[normalDistEst, sampleCount];

lognormalDist = LogNormalDistribution[sampleMeanLog, sampleStdDevLog];
lognormalDataSet = RandomVariate[lognormalDist, sampleCount];

lognormalDistEst =
 EstimatedDistribution[sampleDataSet, LogNormalDistribution[lm, ls]]
lognormalDataSetEst = RandomVariate[lognormalDistEst, sampleCount];

poissonDist = PoissonDistribution[sampleMean];
poissonDataSet = RandomVariate[poissonDist, sampleCount];

poissonDistEst =
 EstimatedDistribution[IntegerPart[sampleDataSet], PoissonDistribution[m]]
poissonDataSetEst = RandomVariate[poissonDistEst, sampleCount];

poissonConsulDistEst = EstimatedDistribution[
   IntegerPart[sampleDataSet], PoissonConsulDistribution[x, y]];
```

Out[36]=
```
ExponentialDistribution[0.0208879]
```

Out[40]=
```
NormalDistribution[47.8747, 67.1045]
```

Out[44]=
```
LogNormalDistribution[2.09084, 2.75068]
```

Out[48]=

```
PoissonDistribution[47.4783]
```

Hypothesis Testing: The Actual Tests

Hypothesis tests give quantitative answers to common questions, such as how good the fit is between my data and a particular distribution, whether these distributions have the same mean or median, and whether these datasets have the same variability. In this notepad the hypothesis testing is comparing an experimental data set with an established statistical distribution (e.g., normal). We are **not** testing two sample data sets checking to see if they are from the sample population or not.

The key here is to set the *dataDist* variable to the distribution you are testing against (i.e., comparing with) the experimental data. I am using a 5% alpha, which is very common. If the data set and the data distribution statistically match, then the p-value will be greater than 5%. If they do not statistically match, the p-value will be less then 5%. All of my tests based on real experimental data showed the p-value much less then 5%. It wasn't even close!

Note: A good way to test to see if the notepad is set up and working correctly to set the *dataDist* and the *dataSet* to the same distribution and another test would be to set them differently. For example, if you set the *dataDist* to *normalDist* and set the *dataSet* to *normalDataSet* the hypothesis test should show a very strong match with the resulting p-value being much greater than 5%. Check it out! In contrast, if I then change the *dataSet* to *poissonDataSet* and run the hypothesis test, the p-value should be much less than 5%. This is an important test to do yourself so you believe the experimental results! *Visually* some of the distributions match very closely to the experimental data set, yet the hypothesis tests clearly fail (p-value < 5%).

In[51]:=
```
dataDist = lognormalDistEst;
dataSet = sampleDataSet;

generatedDataSet = RandomVariate[dataDist, 1 000 000];
α = 0.05;

h = DistributionFitTest[dataSet, dataDist, "HypothesisTestData"];
h["TestDataTable", All]
If[AndersonDarlingTest[dataSet, dataDist] > α,
 Print["Null hypo accepted: Data is statistically similar"],
 Print["Null hypo rejected: Data is statistically
    different. The results cannot be explained by randomness,
    so there must be something else causing the differences."]
]
predMedian1 =
  ℮^Mean[Table[Log[generatedDataSet[[i]]], {i, 1, Length[generatedDataSet]}]];
predMedian2 = ℮^Mean[Table[Log[dataSet[[i]]], {i, 1, Length[dataSet]}]];
predMean = ℮^(sampleMeanLog + (sampleStdDevLog^2) / 2);
Print["Log normal prediction: actual sample median=",
 N[Median[dataSet]], " pred median (actual transformed)=",
 predMedian2, " pred median (generated transformed)=", predMedian1]
Print["Log normal prediction: actual sample mean=",
 N[Mean[dataSet]], " pred mean=", predMean]
```

Out[56]=

	Statistic	P-Value
Anderson-Darling	36.4917	0.
Cramér-von Mises	5.87377	4.80727×10^{-14}
Pearson χ^2	885.7	1.5536×10^{-171}

```
Null hypo rejected: Data is statistically different. The results cannot be explained
  by randomness, so there must be something else causing the differences.
Log normal prediction: actual sample median=25.8148
  pred median (actual transformed)=8.09172 pred median (generated transformed)=8.10417
Log normal prediction: actual sample mean=47.8747 pred mean=358.325
```

Visually Analysis

The visual analysis is very interesting. In part because some of the experimental data sets seem to match a distribution rather nicely (check using the log normal distribution) yet they clearly fail the hypothesis test. The visual analysis is also a good double-check to ensure you are using the correct data distribution...because you can see it!

I show both the theoretically pure data distribution histogram and then below it the histogram based on the actual experimental data. The final and third histogram overlays them. The overlay sometimes provides a nice way to observe how visually close (or not) the experimental data is with the pure distribution. But be aware, that what may look like a very nice match visually, is not a good match statistically (p-value < alpha).

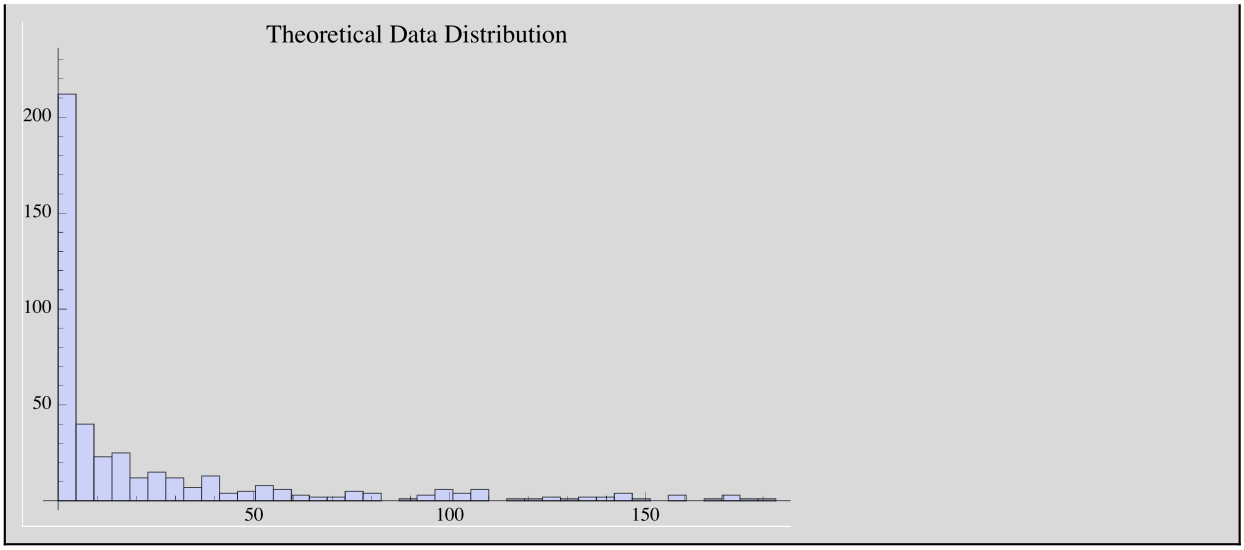
The combination of a massive relative data value range and massive clustering near the origin makes this data very difficult to view all together. After some analysis, showing the 0 to 90 to even to 97 percentile usually presents us with a respectable histogram. But we can't forget that the remaining few percentage of data is *extremely* skewed to the right. Picturing where the

mean and median reside will help complete a more realistic picture of the data set. By setting the variables below, you can probably create a very pleasing and informative histogram.

In[63]:=
```
percentOfData = 95;
histogramBins = 40;
histogramBinSize = Quantile[dataSet, percentOfData / 100] / histogramBins;
histgramMaxPoint = Quantile[dataSet, percentOfData / 100];
theoryDataSet = RandomVariate[dataDist, sampleCount];
```

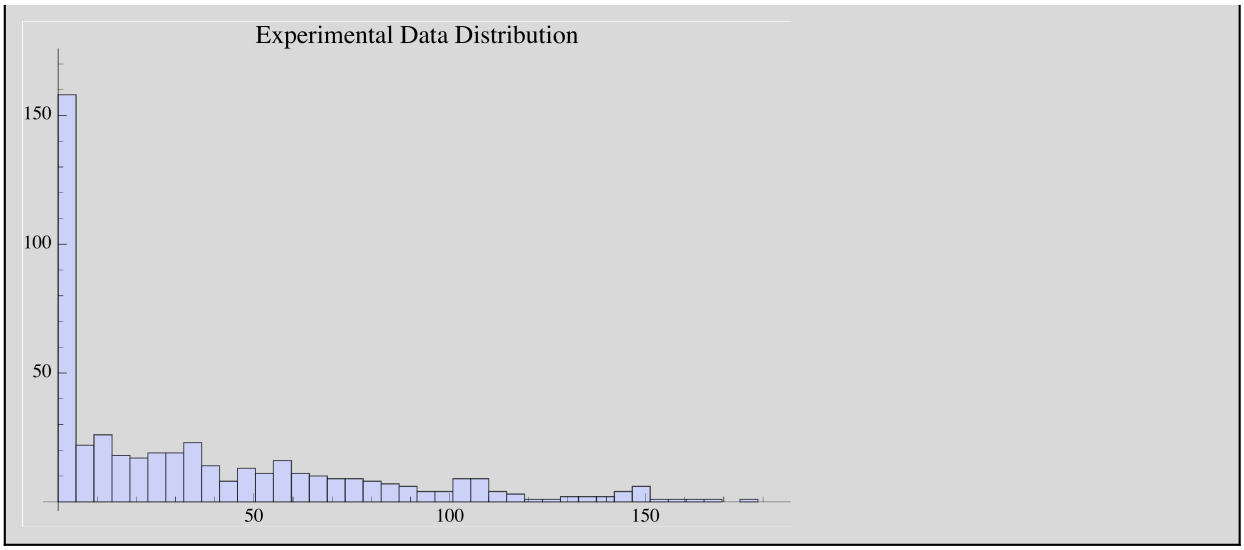
In[68]:=
```
Histogram[theoryDataSet, {0, histgramMaxPoint, histogramBinSize},
 PlotLabel → "Theoretical Data Distribution"]
```

Out[68]=



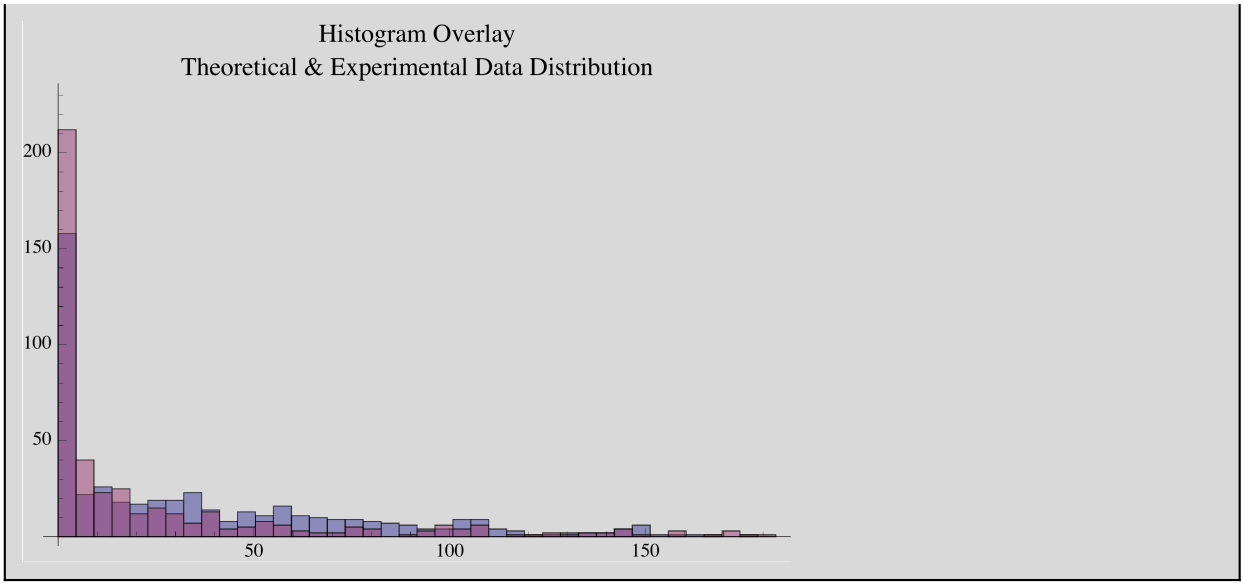
In[69]:=
```
Histogram[dataSet, {0, histgramMaxPoint, histogramBinSize},
 PlotLabel → "Experimental Data Distribution"]
```

Out[69]=



In[70]:=
```
Histogram[{dataSet, theoryDataSet},
 {0, histgramMaxPoint, histogramBinSize}, PlotLabel →
  "Histogram Overlay\nTheoretical & Experimental Data Distribution"]
```

Out[70]=



If the sample data set is log normal distributed, the below histogram will be very normal-like. The below histogram was created by taking each data sample value, appying it to the log function, and placing the result into a new data set (transformedSampleSet), and then plotting the histogram.

In[71]:=
```
Histogram[transformedSampleSet, 40,
 PlotLabel → "Histogram of Sample Data, Log Transformed"]
```

Out[71]=