SQL Statement Inter-Arrival Time Distribution Analysis

Author: Craig Shallahamer, Version 2b on 1-Feb-2011

The Point and Purpose

The purpose of this notepad is to aid in understanding the distribution of SQL statement inter-arrival times. Most DBAs expect SQL statement arrival times to be pretty much the same and when pressed they're likely to say they are probably normally distributed. But to my knowledge, no one has ever actually verified this experimentally...and that is my objective.

More specifically, I want to:

1. Gather SQL statement inter-arrival times
2. Perform a statistical fitness test to determine if the inter-arrival times conform to standard statistical distributions
3. Make some useful conclusions relating to Oracle performance analysis

In computing system capacity planning, transaction inter-arrival times are assumed to be exponentially distributed. Said another way, given bunch of samples with the average inter-arrival time of 10ms, more of the samples will have an inter-arrival times less then 10ms than greater than 10ms. With a normal distribution you're likely to see just as many samples greater than and also less than the mean.

The value of statistically matching the collected data to a known statistical distribution is, even with a limited sample set or simply the average inter-arrival time, additional information can be gleaned. This will aid in describing SQL statement arrival rates and in developing predictive models.

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 arrivals for 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 along with examples and sample data can be found here.

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.

In[68]:=
```
filename = "/Users/cshallah/Desktop/craig_summary_list_clean_oltp1.txt";
```

Import the experimental results. The "List" option allows all the data to be imported easily and perfectly. The data is in seconds between arrivals, but we typically talk in milliseconds and the graphs and math is more visibly pleasing, so I multiply by 1000 using *Mathematica*'s Table command. I also asked for the number of data rows to ensure all my samples have been imported. There is also an option to only work with a subset of the data. This is nice when there are litteraly thousands of samples or we want to focus on a subset of the data.

In[69]:=
```
rawSet = Import[filename, "List"];
sampleStartElement = 1;
sampleEndElement = 100 000;
sampleDataSet = Table[1000 rawSet[[i]],
   {i, sampleStartElement, Min[sampleEndElement, Length[rawSet]]}];
Length[rawSet]
Length[sampleDataSet]
```

Out[73]=
```
321
```

Out[74]=
```
321
```

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

In[75]:=
```
firstRow = sampleDataSet[[1]]
lastRow = sampleDataSet[[Length[sampleDataSet]]]
```

Out[75]=
```
83.492
```

Out[76]=
```
63.145
```

If all looks good, then let's move forward and do some analysis.

Statistical Analysis

Core Numeric Statistics (from actual sample set)

In[77]:=

```
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]]
Print["Log Mean=", N[Log[sampleMean]], " Log StdDev=", N[Log[sampleStdDev]]]
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%=8.758 25%=26.06 50%=59.112 75%=84.485 90%=91.821 95%=120.778

Count=321 Mean=186.792 Median=59.112 StdDev=1384.8 Minimum=0.135 Maximum=19 182.7

Log Mean=5.23 Log StdDev=7.23331

Log Transformed: Mean=3.72194 Median=4.07943 StdDev=1.28799

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[99]:=
```
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[101]=
```
ExponentialDistribution[0.00535359]
```

Out[105]=
```
NormalDistribution[186.792, 1382.64]
```

Out[109]=
```
LogNormalDistribution[3.72194, 1.28598]
```

Out[113]=
```
PoissonDistribution[186.315]
```

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[116]:=
```
dataDist = lognormalDistEst;
dataSet = sampleDataSet;
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."]
]
predMedian = ℮^sampleMeanLog;
predMean = ℮^(sampleMeanLog + (sampleStdDevLog^2) / 2);
Print["Log normal prediction: actual sample median=",
 N[Median[dataSet]], " pred median=", predMedian]
Print["Log normal prediction: actual sample mean=",
 N[Mean[dataSet]], " pred mean=", predMean]
```

Out[120]=

	Statistic	P-Value
Anderson-Darling	17.5966	1.87443×10^{-6}
Cramér-von Mises	3.13741	3.75916×10^{-8}
Pearson χ^2	369.514	4.19043×10^{-66}

```
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=59.112 pred median=41.3446
Log normal prediction: actual sample mean=186.792 pred mean=94.7646
```

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!

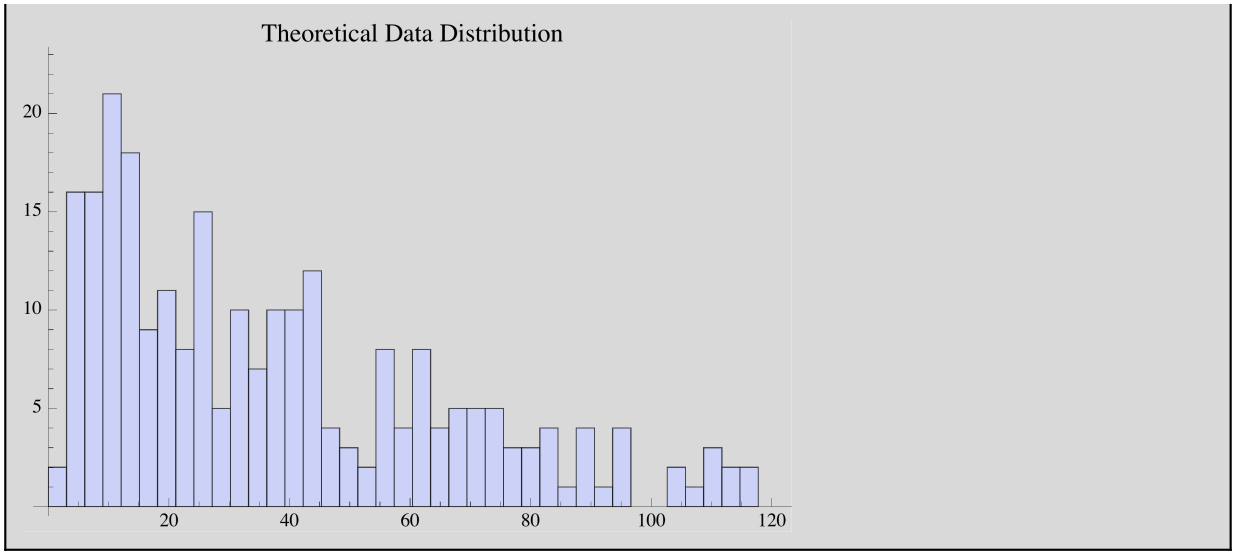
I show both the theoretically pure data distribution histogram and then below it the histogram based on the actual experimental data. The final and third histogram overlays them. The overlay sometimes provides a nice way to observe how visually close (or not) the experimental data is with the pure distribution. 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[126]:=
```
percentOfData = 95;
histogramBins = 40;
histogramBinSize = Quantile[dataSet, percentOfData / 100] / histogramBins;
histgramMaxPoint = Quantile[dataSet, percentOfData / 100];
theoryDataSet = RandomVariate[dataDist, sampleCount];
```

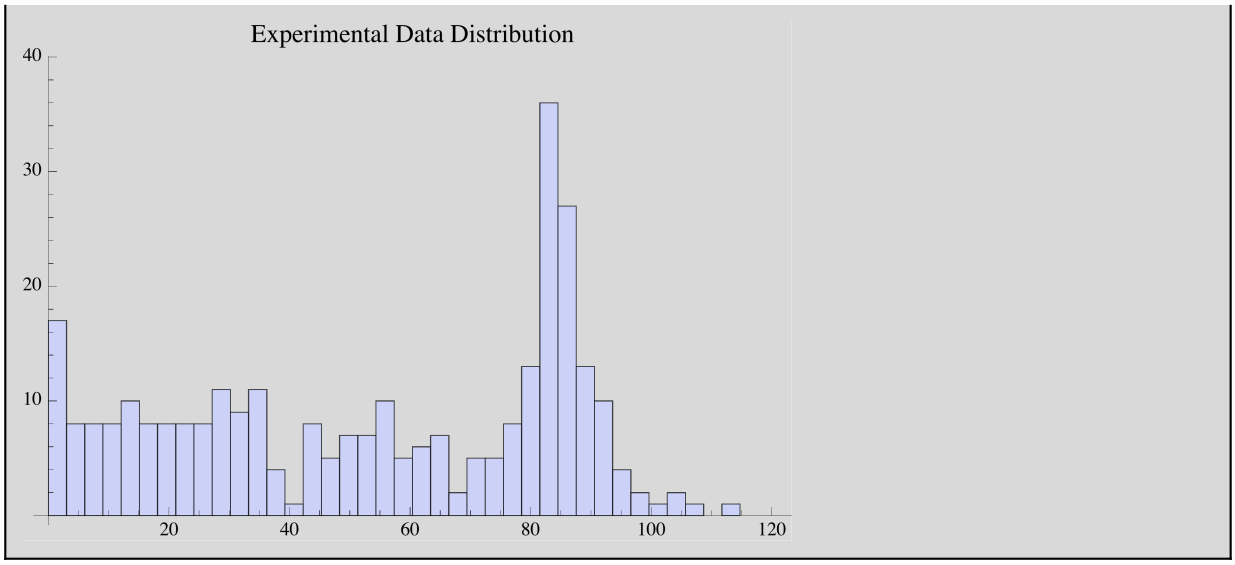
In[131]:=
```
Histogram[theoryDataSet, {0, histgramMaxPoint, histogramBinSize},
 PlotLabel → "Theoretical Data Distribution"]
```

Out[131]=



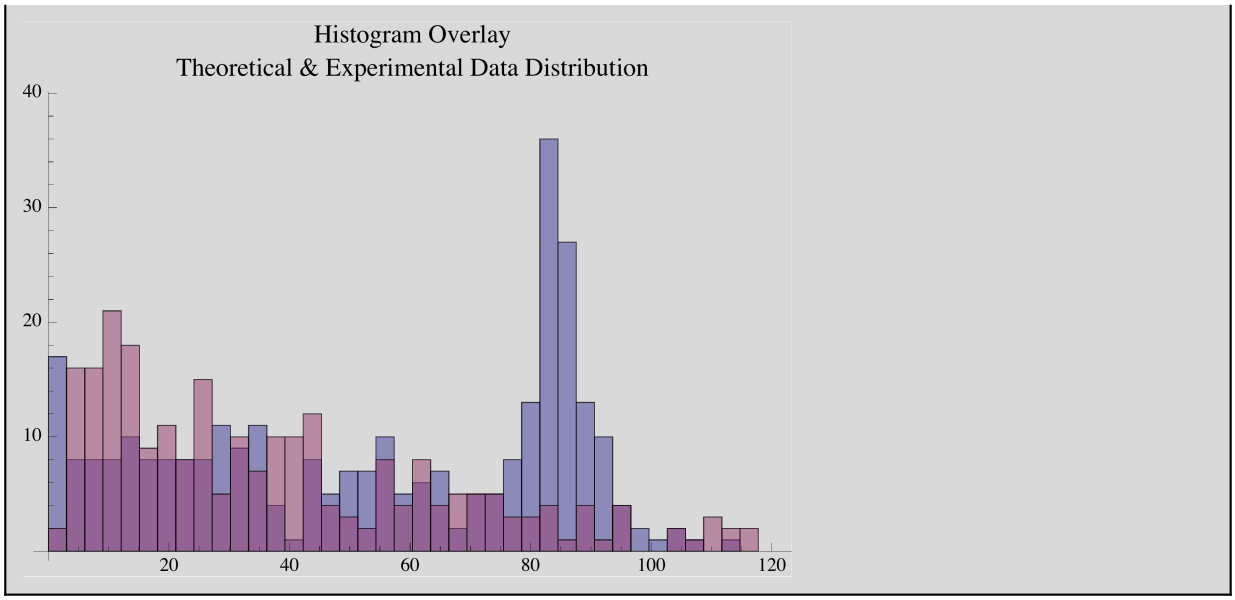
In[132]:=
```
Histogram[dataSet, {0, histgramMaxPoint, histogramBinSize},
 PlotLabel → "Experimental Data Distribution"]
```

Out[132]=



In[133]:=
```
Histogram[{dataSet, theoryDataSet},
 {0, histgramMaxPoint, histogramBinSize}, PlotLabel →
  "Histogram Overlay\nTheoretical & Experimental Data Distribution"]
```

Out[133]=



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[134]:=

```
Histogram[transformedSampleSet, 40,
 PlotLabel → "Histogram of Sample Data, Log Transformed"]
```

Out[134]=