
Universal Scalability Law (USL) in an Oracle OLTP database

Dave Abercrombie, and@rahul.net
August 14 2011

Introduction

This notebook is the second in a series on applying the Universal Scalability Law (USL) to Oracle OLTP. It follows up on my July post that gave an overview of the USL, and showed how Oracle “Average Active Sessions” (AAS) metric could be used to model concurrency “N”.

The goal here is to estimate seriality and coherency parameters, α and β respectively, that can be used to predict OLTP throughput as a function of concurrency “N”. This is achieved by fitting the USL to observations of Oracle performance metrics that are easily queried from the Automatic Workload Repository (AWR). This model can be used for capacity planning and to set realistic monitoring targets. It also provides insight into the relative importance of serialization and coherency limits to system throughput.

The USL was developed by Dr. Neil Gunther, and is well described in his book *Guerrilla Capacity Planning* (Springer 2007, chapters 4-6). Craig Shallahamer places the USL into an Oracle context in his book *Forecasting Oracle Performance* (Apress 2007, chapter 10). The USL can be thought of as an extension of Amdahl’s Law, which concerns speeding up a data processing task through parallelization. Amdahl’s Law shows that the serialized parts of the data processing task will limit its speedup, so performance will eventually level off. The USL extends Amdahl’s law by accounting for coherency in addition to concurrency. The USL’s inclusion of coherency means that performance will actually decrease with excessive parallelization, while it merely levels off under Amdahl’s Law.

In the previous paragraph, I was intentionally vague with my use of the word parallelization. Does this term mean adding more CPU processors, or does it mean adding more users? In other words, will the USL predict how many CPUs are needed to support a certain workload, or will it predict how many concurrent users can be supported by an existing software system? The answer: both! Dr. Gunther showed that the **USL works for both hardware and software** (Gunther 2007, chapter 6).

This notebook is divided into these sections:

- USL model detailed description
- Observations from a production database
- Estimate throughput at $N = 1$
- Normalize throughput observations to convert to **capacityRatio**
- Fit USL model to observations to estimate parameters α and β

USL model detailed description

Define N as concurrency, the number of concurrent “users.” Here it is modelled as Oracle “Average Active Sessions” (AAS), as explained in my July post.

Define X as “throughput.” Here it is modelled as Oracle “buffer gets” per millisecond as a function of N , and given the variable name `throughPut` (Gunther 2007 p. 52 eq. 4.20 as modified by p. 101 Section 6.3). Note that you have the choice to model your database whichever throughput metrics are most appropriate. For example, many of Shallahamer’s example used the Oracle metric “user calls.” No single correct throughput metric exists, you might want to experiment with several to see which has the best predictive ability.

$$\text{throughPut} = X(N)$$

Then $X(1)$ is the throughput where $N=1$ (Ibid.). It is obviously not possible to set $N=1$ in a production Oracle database when using this $N=AAS$ model, since AAS is estimated through observation rather than set to arbitrary values. The variable name `throughPutOne` is used here for $X(1)$ and its value is estimated by linear regression as shown below.

$$\text{throughPutOne} = X(1)$$

Define $C(N)$ as the “capacity ratio” or “relative capacity” as the ratio of $X(N)$ to $X(1)$ at a given value of N (Gunther 2007 p. 77). The variable `capacityRatio` is used here for $C(N)$.

$$\text{capacityRatio} = C(N) = X(N)/X(1)$$

Gunther’s Universal Scalability Law (USL) as modified for software scalability is shown below. (Gunther 2007 p. 101 eq. 6.7). The variable n is used here for N . A goal of this study is to estimate parameters α and β , which characterize seriality and coherency respectively.

$$\text{capacityRatio} = n / (1 + \alpha (n - 1) + \beta n (n - 1))$$

In[1]:=

```
capacityRatio = n / (1 + \alpha (n - 1) + \beta n (n - 1))
```

Out[1]:=

$$\frac{n}{1 + (-1 + n) \alpha + (-1 + n) n \beta}$$

As an aside, shown below is how *Mathematica* simplifies this formula.

In[2]:=

```
capacityRatio // FullSimplify
```

Out[2]:=

$$\frac{n}{1 + (-1 + n) (\alpha + n \beta)}$$

Observations from a production database

Import a CSV file containing observations from an AWR query on a production system. This query calculates deltas of hourly snapshots of several Oracle resource usage counters (see this article for more about AWR).

Start with a sanity check on current working directory, to make sure `Import ["file"]` works. Of course, modify this as necessary on your system.

```
In[3]:= SetDirectory["C:\\Users\\abe\\Documents\\math-svn\\usl"]
Out[3]= C:\\Users\\abe\\Documents\\math-svn\\usl
```

For this USL model, we need only these two columns:

`DB_TIME_CSPH` = centiseconds of "DB Time" used in one hour. As explained here, "DB time" is the sum of time spent on the CPU and in "active" waits.

`BUFFER_GETS_PH` = buffer gets done in one hour, a measure of throughput.

Since *Mathematica* does not support column names in its nested lists, I create a list of column names by parsing the CSV file. This helps to verify that we are using the correct column of the nested list.

```
In[4]:= awrObservationsCSV = Import["db9-awr.csv", "CSV"];
awrObservationsHeader = awrObservationsCSV[[1]]
Out[5]= {SNAP_ID, BEGIN_HOUR, DB_CPU_CSPH, DB_TIME_CSPH,
PCNT_CPU_UTILIZE, WAIT_CSPH, USER_CALLS_PH, BUFFER_GETS_PH}
```

The data are in all but the first row (element) of the CSV. Use a negative second parameter to `Take [list, n]` so that the extraction will count backwards from the last CSV element. Double check the extracted subset's dimensions, and echo the first few elements out as a visual sanity check.

```
In[6]:= awrObservations = Take[awrObservationsCSV, 1 - Length[awrObservationsCSV]];
Dimensions[awrObservations]
Take[awrObservations, 3]
Out[7]= {175, 8}
Out[8]= {{27434, 2010-08-09 08:00, 545835, 715558, 38, 169723, 28348925, 594757228},
{27435, 2010-08-09 09:00, 517703, 701535, 36, 183832, 30689450, 549076714},
{27436, 2010-08-09 10:00, 551088, 753833, 38, 202745, 32247830, 573609119}}
```

Define names for the two columns we need, and double check with the header.

```
In[9]:= dbTimeCsphColumn = 4;
bufferGestPhColumn = 8;
{awrObservationsHeader[[dbTimeCsphColumn]],
awrObservationsHeader[[bufferGestPhColumn]]}
Out[11]= {DB_TIME_CSPH, BUFFER_GETS_PH}
```

Create a table with these two columns, and in the model's units of measure:

n in the first column (i.e., AAS or N, concurrency), and
throughPut in the second column (mean buffer gets per millisecond).

Divide *centiseconds per hour of DB Time* by the number of centiseconds in one hour to calculate AAS.

Divide *buffer gets per hour* by the number of milliseconds in one hour to calculate *mean throughput in units of buffer gets per millisecond*.

Echo back the first ten rows for a sanity check.

In[12]:=

```
nThroughPut = Table[
  { N[awrObservations[[i, dbTimeCsphColumn]] / (3600 * 100)],
    N[awrObservations[[i, bufferGestPhColumn]] / (3600 * 1000) ]
  }, {i, Length[awrObservations]}
];
Take[ nThroughPut, 10]
```

Out[13]=

```
{{1.98766, 165.21}, {1.94871, 152.521}, {2.09398, 159.336},
 {2.29625, 167.687}, {2.37329, 165.111}, {2.19681, 163.358},
 {2.22491, 157.062}, {1.79068, 142.206}, {1.38164, 121.516}, {1.21838, 112.132}}
```

Estimate throughput at N = 1

As mentioned above, it is obviously not possible to set N=1 in a production Oracle database when using this N=AAS model, since AAS is estimated through observation rather than set to arbitrary values. We will use linear regression on this subset to estimate the throughput at AAS=1.

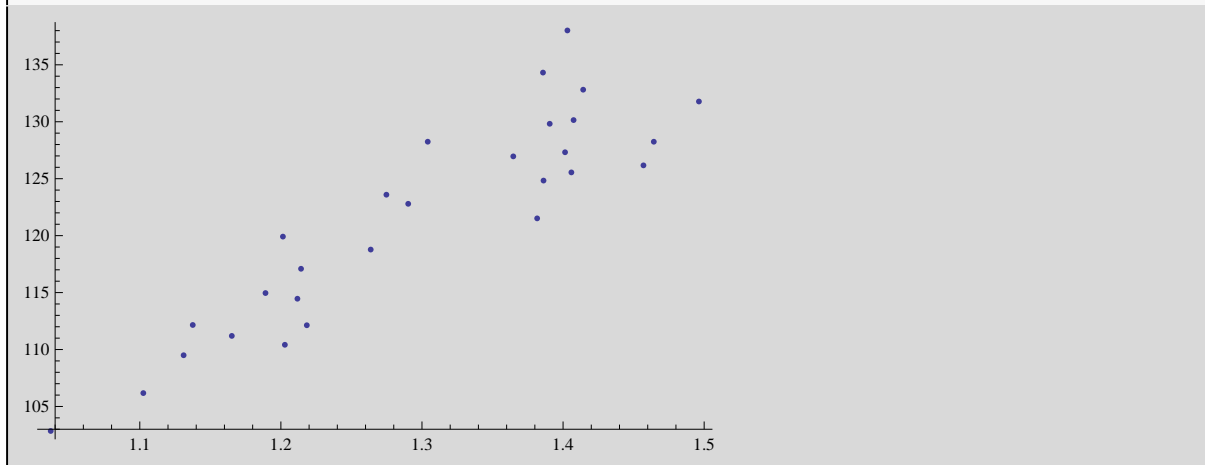
Select the subset of observations where AAS was less than 1.5. Do a quick plot as a sanity check. This estimated throughput at AAS=1 is needed to normalize throughput observations to unitless **capacityRatio**.

Note the unusual "pure function" syntax used for specifying the first part of this nested list (i.e. table).

In[14]:=

```
nThroughPutSubset = Select[nThroughPut, #[[1]] < 1.5 &];
p1 = ListPlot[nThroughPutSubset]
```

Out[15]=



Estimate **throughPutOne** = X(1) by applying linear regression to the subset of throughput observations.

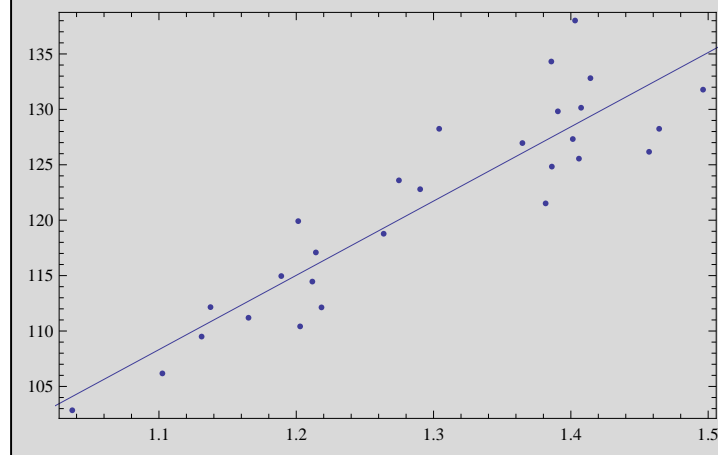
In this case, the intercept was not zero. Perhaps a low rate of buffer gets, 34 per millisecond, can happen without active sessions getting noticed by ASH? Hmm...

```
In[16]:=
nThroughPutSubsetLm = LinearModelFit[nThroughPutSubset, x, x]
throughPutOne = nThroughPutSubsetLm[1]
p2 = Plot[nThroughPutSubsetLm[x], {x, 0, 2}];
Show[ p1, p2, Frame -> True]
```

```
Out[16]= FittedModel[ 34.6025 + 67.0177 x ]
```

```
Out[17]= 101.62
```

```
Out[19]=
```



Normalize throughput observations to convert to capacityRatio

Normalize throughput observations to convert to **capacityRatio** = $C(N) = X(N)/X(1)$.

Sanity check and plot.

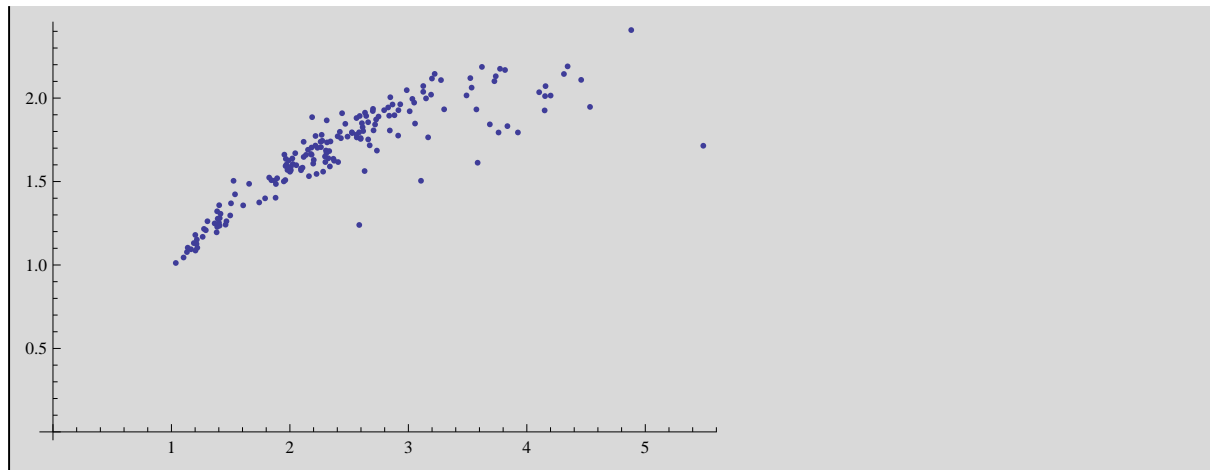
In[20]=

```
nCapacityRatio = Table[
  { nThroughPut[[i, 1]],
    nThroughPut[[i, 2]] / throughPutOne
  }, {i, Length[nThroughPut]}
];
Take[nCapacityRatio, 10]
p1 = ListPlot[nCapacityRatio, AxesOrigin -> {0, 0}]
```

Out[21]=

```
{{1.98766, 1.62576}, {1.94871, 1.5009}, {2.09398, 1.56795},
 {2.29625, 1.65013}, {2.37329, 1.62479}, {2.19681, 1.60754},
 {2.22491, 1.54558}, {1.79068, 1.39939}, {1.38164, 1.19579}, {1.21838, 1.10344}}
```

Out[22]=



Fit USL model to observations to estimate parameters α and β

Use `FindFit[data, expr, pars, vars]` to estimate parameters α and β , which characterize seriality and coherency respectively. The capacity ratio form of the USL is repeated below. (Gunther 2007 p. 101 eq. 6.7). The variable n is used here for N and is the first column in the `nCapacityRatio` (nested list) table and can be thought of as the independent variable. The second column of data, the normalized capacity ratio from immediately above, is considered to be the observed value of the capacityRatio formula.

$$\text{capacityRatio} = n / (1 + \alpha (n - 1) + \beta n (n - 1))$$

In[23]=

```
capacityRatio
bestFitUSL = FindFit[nCapacityRatio, capacityRatio, {\alpha, \beta}, n]
```

Out[23]=

$$\frac{n}{1 + (-1 + n) \alpha + (-1 + n) n \beta}$$

Out[24]=

```
{\alpha -> 0.189005, \beta -> 0.0335166}
```

Prepare USL model by substituting best fit parameters and simplifying.

In[25]:=

```
uslModel = capacityRatio /. bestFitUSL // FullSimplify
```

Out[25]=

$$\frac{n}{0.810995 + (0.155489 + 0.0335166 n) n}$$

Plot USL model along with observations.

In[26]:=

```
p2 = Plot[uslModel, {n, 0, 8}];  
Show[p1, p2]
```

Out[27]=

