Package 'usl'

March 2, 2020

Type Package

Title Analyze System Scalability with the Universal Scalability Law

Version 3.0.0

Date 2020-02-29

BugReports https://github.com/smoeding/usl/issues

Depends R (>= 3.0), methods

Imports graphics, stats, nlsr

Suggests knitr

VignetteBuilder knitr

Description The Universal Scalability Law (Gunther 2007) <doi:10.1007/978-3-540-31010-5> is a model to predict hardware and software scalability. It uses system capacity as a function of load to forecast the scalability for the system.

License BSD_2_clause + file LICENSE

Encoding UTF-8

RoxygenNote 7.0.2

NeedsCompilation no

Author Neil J. Gunther [aut],
Stefan Moeding [aut, cre]

Maintainer Stefan Moeding <stm@moeding.net>

Repository CRAN

Date/Publication 2020-03-02 00:30:02 UTC

R topics documented:

`usl-package` *Analyze system scalability with the Universal Scalability Law*

Description

The Universal Scalability Law is a model to predict hardware and software scalability. It uses system capacity as a function of load to forecast the scalability for the system.

Details

Use the function `usl` to create a model from a formula and a data frame.

The USL model produces two coefficients as result: `alpha` models the contention and `beta` the coherency delay of the system.

The Universal Scalability Law has been created by Dr. Neil J. Gunther.

References

Neil J. Gunther. Guerrilla Capacity Planning: A Tactical Approach to Planning for Highly Scalable Applications and Services. Springer, Heidelberg, Germany, 1st edition, 2007.

See Also

`usl`

confint,USL-method *Confidence Intervals for USL model parameters*

Description

Estimate confidence intervals for one or more parameters in a USL model. The intervals are calculated from the parameter standard error using the Student t distribution at the given level.

Usage

```
## S4 method for signature 'USL'
confint(object, parm, level = 0.95)
```

Arguments

object	A USL object.
parm	A specification of which parameters are to be given confidence intervals, either a vector of numbers or a vector of names. If missing, all parameters are considered.
level	The confidence level required.

Details

Bootstrapping is no longer used to estimate confidence intervals.

Value

A matrix (or vector) with columns giving lower and upper confidence limits for each parameter. These will be labelled as (1-level)/2 and 1 - (1-level)/2 in % (by default 2.5% and 97.5%).

See Also

usl

Examples

```
require(usl)

data(specsdm91)

## Create USL model
usl.model <- usl(throughput ~ load, specsdm91)

## Print confidence intervals
confint(usl.model)
```

`efficiency,USL-method` *Efficiency of the system*

Description

The efficiency of a system expressed in terms of the deviation from linear scalability.

Usage

```
## S4 method for signature 'USL'
efficiency(object)
```

Arguments

`object`	A USL object.

Details

The function returns a vector which contains the deviation from linearity for every measurement of the model input. A value of 1 indicates linear scalability while values less than 1 correspond to the fraction of the measurement compared to linear scalability.

Value

A vector of numeric values.

References

Neil J. Gunther. Guerrilla Capacity Planning: A Tactical Approach to Planning for Highly Scalable Applications and Services. Springer, Heidelberg, Germany, 1st edition, 2007.

See Also

`usl`

Examples

```
require(usl)

data(raytracer)

## Show the efficiency
efficiency(usl(throughput ~ processors, raytracer))
```

`limit.scalability,USL-method` *Scalability limit of a USL model*

Description

Calculate the scalability limit for a specific model.

Usage

```
## S4 method for signature 'USL'
limit.scalability(object, alpha, beta, gamma)
```

Arguments

`object`	A USL object.
`alpha`	Optional parameter to be used for evaluation instead of the parameter computed for the model.
`beta`	Optional parameter to be used for evaluation instead of the parameter computed for the model.
`gamma`	Optional parameter to be used for evaluation instead of the parameter computed for the model.

Details

The scalability limit is defined as:

$$Xroof = \frac{\gamma}{\alpha}$$

This is the upper bound (Amdahl asymptote) of system capacity.

The parameters `alpha`, `beta` and `gamma` are useful to do a what-if analysis. Setting these parameters override the model parameters and show how the system would behave with a different contention or coherency delay parameter.

The scalability limit is undefined if `alpha` is zero.

This function accepts an argument for `beta` although the value is not required to perform the calculation. This is on purpose to provide a coherent interface.

Value

A numeric value for the system capacity limit (e.g. throughput).

See Also

`usl`, `peak.scalability,USL-method` `optimal.scalability,USL-method`

Examples

```
require(usl)

data(specsdm91)

limit.scalability(usl(throughput ~ load, specsdm91))
## The throughput limit is about 3245
```

`optimal.scalability,USL-method`

Point of optimal scalability of a USL model

Description

Calculate the point of optimal scalability for a specific model.

Usage

```
## S4 method for signature 'USL'
optimal.scalability(object, alpha, beta, gamma)
```

Arguments

`object`	A USL object.
`alpha`	Optional parameter to be used for evaluation instead of the parameter computed for the model.
`beta`	Optional parameter to be used for evaluation instead of the parameter computed for the model.
`gamma`	Optional parameter to be used for evaluation instead of the parameter computed for the model.

Details

The point of optimal scalability is defined as:

$$Nopt = \frac{1}{\alpha}$$

Below this point the existing capacity is underutilized. Beyond that point the effects of diminishing returns become visible more and more.

The value can be constructed graphically by projecting the intersection of the linear scalability bound and the Amdahl asymptote onto the x-axis.

The parameters `alpha`, `beta` and `gamma` are useful to do a what-if analysis. Setting these parameters override the model parameters and show how the system would behave with a different contention or coherency delay parameter.

The point of optimal scalability is undefined if `alpha` is zero.

This function accepts a arguments for `beta` and `gamma` although the values are not required to perform the calculation. This is on purpose to provide a coherent interface.

Value

A numeric value for the load where optimal scalability will be reached.

See Also

`usl`, `peak.scalability,USL-method` `limit.scalability,USL-method`

Examples

```
require(usl)

data(specsdm91)

optimal.scalability(usl(throughput ~ load, specsdm91))
## Optimal scalability will be reached at about 36 virtual users
```

---

oracledb *Performance of an Oracle database used for online transaction processing*

Description

A dataset containing performance data for an Oracle OLTP database measured between 8:00am and 8:00pm on January, 19th 2012. The measurements were recorded for two minute intervals during this time and a timestamp indicates the end of the measurement interval. The performance metrics were taken from the `v$sysmetric` family of system performance views.

Format

A data frame with 360 rows on 8 variables

Details

The Oracle database was running on a 4-way server.

The data frame contains different types of measurements:

- Variables of the "time" type are expressed in seconds per second.
- Variables of the "rate" type are expressed in events per second.
- Variables of the "util" type are expressed as a percentage.

The data frame contains the following variables:

- `timestamp` The end of the two minute interval for which the remaining variables contain the measurements.
- `db_time` The time spent inside the database either working on a CPU or waiting (I/O, locks, buffer waits ...). This time is expressed as seconds per second, so two sessions working for exactly one second each will contribute a total of two seconds per second of `db_time`. In Oracle this value is also known as *Average Active Sessions* (AAS).
- `cpu_time` The CPU time used during the interval. This is also expressed as seconds per second. A 4-way machine has a theoretical capacity of four CPU seconds per second.
- `call_rate` The number of user calls (logins, parses, or execute calls) per second.
- `exec_rate` The number of statement executions per second.
- `lio_rate` The number of logical I/Os per second. A logical I/O is the Oracle term for a cache hit in the database buffer cache. This metric does not indicate if an additional physical I/O was necessary to load the buffer from disk.
- `txn_rate` The number of database transactions per second.
- `cpu_util` The CPU utilization of the database server in percent. This was also measured from within the database.

`overhead,USL-method` *Overhead method for Universal Scalability Law models*

Description

`overhead` calculates the overhead in processing time for a system modeled with the Universal Scalability Law. It evaluates the regression function in the frame `newdata` (which defaults to `model.frame(object)`). The result contains the ideal processing time and the additional overhead caused by contention and coherency delays.

Usage

```
## S4 method for signature 'USL'
overhead(object, newdata)
```

Arguments

`object`	A USL model object for which the overhead will be calculated.
`newdata`	An optional data frame in which to look for variables with which to calculate the overhead. If omitted, the fitted values are used.

Details

The calculated processing times are given as percentages of a non-parallelized workload. So for a non-parallelized workload the ideal processing time will always be given as *100%* while the overhead for contention and coherency will always be zero.

Doubling the capacity will cut the ideal processing time in half but increase the overhead percentages. The increase of the overhead depends on the values of the parameters `alpha` and `beta` estimated by `usl`.

The calculation is based on *A General Theory of Computational Scalability Based on Rational Functions*, equation 26.

Value

`overhead` produces a matrix of overhead percentages based on a non-parallelized workload. The column `ideal` contains the ideal percentage of execution time. The columns `contention` and `coherency` give the additional overhead percentage caused by the respective effects.

References

Neil J. Gunther. Guerrilla Capacity Planning: A Tactical Approach to Planning for Highly Scalable Applications and Services. Springer, Heidelberg, Germany, 1st edition, 2007.

Neil J. Gunther. A General Theory of Computational Scalability Based on Rational Functions. Computing Research Repository, 2008. `http://arxiv.org/abs/0808.1431`

See Also

`usl`, `USL-class`

Examples

```
require(usl)

data(specsdm91)

## Print overhead in processing time for demo dataset
overhead(usl(throughput ~ load, specsdm91))
```

---

peak.scalability,USL-method

Point of peak scalability of a USL model

Description

Calculate the point of peak scalability for a specific model.

Usage

```
## S4 method for signature 'USL'
peak.scalability(object, alpha, beta, gamma)
```

Arguments

object	A USL object.
alpha	Optional parameter to be used for evaluation instead of the parameter computed for the model.
beta	Optional parameter to be used for evaluation instead of the parameter computed for the model.
gamma	Optional parameter to be used for evaluation instead of the parameter computed for the model.

Details

The peak scalability is the point where the throughput of the system starts to go retrograde, i.e., starts to decrease with increasing load.

The parameters alpha, beta and gamma are useful to do a what-if analysis. Setting these parameters override the model parameters and show how the system would behave with a different contention or coherency delay parameter.

See formula (4.33) in *Guerilla Capacity Planning*.

This function accepts an argument for gamma although the value is not required to perform the calculation. This is on purpose to provide a coherent interface.

Value

A numeric value for the point where peak scalability will be reached.

References

Neil J. Gunther. Guerrilla Capacity Planning: A Tactical Approach to Planning for Highly Scalable Applications and Services. Springer, Heidelberg, Germany, 1st edition, 2007.

See Also

usl, optimal.scalability,USL-method limit.scalability,USL-method

Examples

```
require(usl)

data(specsdm91)

peak.scalability(usl(throughput ~ load, specsdm91))
## Peak scalability will be reached at about 96 virtual users
```

plot,USL-method *Plot the scalability function from a USL model*

Description

Create a line plot for the scalability functionh of a Universal Scalability Law model.

Usage

```
## S4 method for signature 'USL'
plot(
  x,
  from = NULL,
  to = NULL,
  xlab = NULL,
  ylab = NULL,
  bounds = FALSE,
  alpha,
  beta,
  ...
)
```

Arguments

x	The USL object to plot.
from	The start of the range over which the scalability function will be plotted.
to	The end of the range over which the scalability function will be plotted.
xlab	A title for the x axis: see title.
ylab	A title for the y axis: see title.
bounds	Add the bounds of scalability to the plot. This always includes the linear scalability bound for low loads. If the contention coefficient alpha is a positive number, then the Amdahl asymptote for high loads will also be plotted. If the coherency coefficient beta is also a positive number, then the point of peak scalability will also be indicated. All bounds are show using dotted lines. Some bounds might not be shown using the default plot area. In this case the parameter ylim can be used to increase the visible plot area and include all bounds in the output.
alpha	Optional parameter to be used for evaluation instead of the parameter computed for the model.
beta	Optional parameter to be used for evaluation instead of the parameter computed for the model.
...	Other graphical parameters passed to plot (see par, plot.function).

Details

plot creates a plot of the scalability function for the model represented by the argument x.

If from is not specified then the range starts at the minimum value given to define the model. An unspecified value for to will lead to plot ending at the maximum value from the model. For add = TRUE the defaults are taken from the limits of the previous plot.

xlab and ylab can be used to set the axis titles. The defaults are the names of the regressor and response variables used in the model.

If the parameter bounds is set to TRUE then the plot also shows dotted lines for the theoretical bounds of scalability. These are the linear scalability for small loads and the Amdahl asymptote for the limit of scalability as load approaches infinity.

The parameters alpha or beta are useful to do a what-if analysis. Setting these parameters override the model parameters and show how the system would behave with a different contention or coherency delay parameter.

See Also

usl, plot.function

Examples

```
require(usl)

data(specsdm91)

## Plot result from USL model for demo dataset
plot(usl(throughput ~ load, specsdm91), bounds = TRUE, ylim = c(0, 3500))
```

---

predict,USL-method *Predict method for Universal Scalability Law models*

Description

predict is a function for predictions of the scalability of a system modeled with the Universal Scalability Law. It evaluates the regression function in the frame newdata (which defaults to model.frame(object)). Setting interval to "confidence" requests the computation of confidence intervals at the specified level.

Usage

```
## S4 method for signature 'USL'
predict(
  object,
  newdata,
  alpha,
  beta,
```

```
  interval = c("none", "confidence"),
  level = 0.95
)
```

Arguments

object	A USL model object for which prediction is desired.
newdata	An optional data frame in which to look for variables with which to predict. If omitted, the fitted values are used.
alpha	Optional parameter to be used for evaluation instead of the parameter computed for the model.
beta	Optional parameter to be used for evaluation instead of the parameter computed for the model.
interval	Type of interval calculation. Default is to calculate no confidence interval.
level	Confidence level. Default is 0.95.

Details

The parameters alpha or beta are useful to do a what-if analysis. Setting these parameters override the model parameters and show how the system would behave with a different contention or coherency delay parameter.

predict internally uses the function returned by scalability,USL-method to calculate the result.

Value

predict produces a vector of predictions or a matrix of predictions and bounds with column names fit, lwr, and upr if interval is set to "confidence".

References

Neil J. Gunther. Guerrilla Capacity Planning: A Tactical Approach to Planning for Highly Scalable Applications and Services. Springer, Heidelberg, Germany, 1st edition, 2007.

See Also

usl, scalability,USL-method, USL-class

Examples

```
require(usl)

data(raytracer)

## Print predicted result from USL model for demo dataset
predict(usl(throughput ~ processors, raytracer))

## The same prediction with confidence intervals at the 99% level
predict(usl(throughput ~ processors, raytracer),
        interval = "confidence", level = 0.99)
```

print,USL-method *Print objects of class "USL"*

Description

print prints its argument and returns it invisibly (via invisible(x)).

Usage

```
## S4 method for signature 'USL'
print(x, digits = max(3L, getOption("digits") - 3L), ...)
```

Arguments

x	An object from class USL.
digits	Minimal number of *significant* digits, see print.default.
...	Other arguments passed to other methods.

Value

print returns the object x invisibly.

See Also

usl, USL-class

Examples

```
require(usl)

data(raytracer)

## Print result from USL model for demo dataset
print(usl(throughput ~ processors, raytracer))
```

raytracer *Performance of a ray-tracing software on different hardware configurations*

Description

A dataset containing performance data for a ray-tracing benchmark.

Format

A data frame with 11 rows on 2 variables

Details

The benchmark measured the number of ray-geometry intersections per second. The data was gathered on an SGI Origin 2000 with 64 R12000 processors running at 300 MHz.

The data frame contains the following variables:

- `processors` The number of CPUs used for the benchmark (1–64).
- `throughput` The number of operations per second.

Source

Neil J. Gunther. Guerrilla Capacity Planning: A Tactical Approach to Planning for Highly Scalable Applications and Services. Springer, Heidelberg, Germany, 1st edition, 2007. Original dataset from http://sourceforge.net/projects/brlcad/

`scalability,USL-method`

Scalability function of a USL model

Description

`scalability` is a higher order function and returns a function to calculate the scalability for the specific USL model.

Usage

```
## S4 method for signature 'USL'
scalability(object, alpha, beta, gamma)
```

Arguments

`object`	A USL object.
`alpha`	Optional parameter to be used for evaluation instead of the parameter computed for the model.
`beta`	Optional parameter to be used for evaluation instead of the parameter computed for the model.
`gamma`	Optional parameter to be used for evaluation instead of the parameter computed for the model.

Details

The returned function can be used to calculate specific values once the model for a system has been created.

The parameters `alpha` and `beta` are useful to do a what-if analysis. Setting these parameters override the model parameters and show how the system would behave with a different contention or coherency delay parameter.

Value

A function with parameter x that calculates the scalability value of the specific model.

References

Neil J. Gunther. Guerrilla Capacity Planning: A Tactical Approach to Planning for Highly Scalable Applications and Services. Springer, Heidelberg, Germany, 1st edition, 2007.

See Also

usl, peak.scalability,USL-method optimal.scalability,USL-method limit.scalability,USL-method

Examples

```
require(usl)

data(raytracer)

## Compute the scalability function
scf <- scalability(usl(throughput ~ processors, raytracer))

## Print scalability for 32 CPUs for the demo dataset
print(scf(32))

## Plot scalability for the range from 1 to 64 CPUs
plot(scf, from=1, to=64)
```

---

show,USL-method *Show objects of class "USL"*

Description

Display the object by printing it.

Usage

```
## S4 method for signature 'USL'
show(object)
```

Arguments

object	The object to be printed.

Value

show returns an invisible NULL.

See Also

usl, USL-class

Examples

```
require(usl)

data(raytracer)

## Show USL model
show(usl(throughput ~ processors, raytracer))
```

---

sigma,USL-method *Extract Residual Standard Deviation 'Sigma'*

Description

sigma Extract Residual Standard Deviation 'Sigma'

Usage

```
## S4 method for signature 'USL'
sigma(object, ...)
```

Arguments

object	An object from class USL.
...	Other arguments passed to other methods.

Value

A single number.

See Also

usl, USL-class

Examples

```
require(usl)

data(raytracer)

## Print result from USL model for demo dataset
print(sigma(usl(throughput ~ processors, raytracer)))
```

specsdm91 *Performanced of a Sun SPARCcenter 2000 in the SPEC SDM91 benchmark*

Description

A dataset containing performance data for a Sun SPARCcenter 2000 (16 CPUs)

Format

A data frame with 7 rows on 2 variables

Details

A Sun SPARCcenter 2000 with 16 CPUs was used for the SPEC SDM91 benchmark in October 1994. The benchmark simulates a number of users working on the UNIX server and measures the number of script executions per hour.

The data frame contains the following variables:

- `load` The number of simulated users (1–216).
- `throughput` The achieved throughput in scripts per hour.

Source

Neil J. Gunther. Guerrilla Capacity Planning: A Tactical Approach to Planning for Highly Scalable Applications and Services. Springer, Heidelberg, Germany, 1st edition, 2007. Original dataset from http://www.spec.org/osg/sdm91/results/results.html

summary,USL-method *USL Object Summary*

Description

`summary` method for class "`USL`".

Usage

```
## S4 method for signature 'USL'
summary(object, ...)
```

Arguments

`object`	A USL object.
`...`	Other arguments passed to other methods.

See Also

usl, USL-class

Examples

```
require(usl)

data(raytracer)

## Show summary for demo dataset
summary(usl(throughput ~ processors, raytracer))

## Extract model coefficients
summary(usl(throughput ~ processors, raytracer))$coefficients
```

---

usl *Create a model for the Universal Scalability Law*

Description

usl is used to create a model for the Universal Scalability Law.

Usage

```
usl(formula, data, method = "default")
```

Arguments

formula	An object of class "formula" (or one that can be coerced to that class): a symbolic description of the model to be analyzed. The details of model specification are given under 'Details'.
data	A data frame, list or environment (or object coercible by as.data.frame to a data frame) containing the variables in the model. If not found in data, the variables are taken from environment(formula), typically the environment from which usl is called.
method	Character value specifying the method to use. The possible values are described under 'Details'.

Details

The Universal Scalability Law is used to forcast the scalability of either a hardware or a software system.

The USL model works with one independent variable (e.g. virtual users, processes, threads, ...) and one dependent variable (e.g. throughput, ...). Therefore the model formula must be in the simple "response ~ predictor" format.

The model produces two main coefficients as result: alpha models the contention and beta the coherency delay of the system. The third coefficient gamma estimates the value of the dependent variable (e.g. throughput) for the single user/process/thread case. It therefore corresponds to the scale factor calculated in previous versions of the usl package.

The function coef extracts the coefficients from the model object.

The argument method selects which solver is used to solve the model:

- "nls" for a nonlinear regression model. This method estimates all coefficients alpha, beta and gamma. The R base function nls with the "port" algorithm is used internally to solve the model. So all restrictions of the "port" algorithm apply.
- "nlxb" for a nonliner regression model using the function nlxb from the nlsr package. This method also estimates all three coefficients. It is expected to be more robust than the nls method.
- "default" for the default method using a transformation into a 2nd degree polynom has been removed with the implementation of the model using three coefficients in the **usl** package 2.0.0. Calling the "default" method will internally dispatch to the "nlxb" solver instead.

The Universal Scalability Law can be expressed with following formula. C(N) predicts the relative capacity of the system for a given load N:

$$C(N) = \frac{\gamma N}{1 + \alpha(N-1) + \beta N(N-1)}$$

Value

An object of class USL.

References

Neil J. Gunther. Guerrilla Capacity Planning: A Tactical Approach to Planning for Highly Scalable Applications and Services. Springer, Heidelberg, Germany, 1st edition, 2007.

John C. Nash. nlsr: Functions for nonlinear least squares solutions, 2017. R package version 2017.6.18.

See Also

efficiency,USL-method, scalability,USL-method, peak.scalability,USL-method, optimal.scalability,USL-met
limit.scalability,USL-method, summary,USL-method, sigma,USL-method predict,USL-method,
overhead,USL-method, confint,USL-method, coef, fitted, residuals, df.residual

Examples

```
require(usl)

data(raytracer)

## Create USL model for "throughput" by "processors"
usl.model <- usl(throughput ~ processors, raytracer)
```

```
## Show summary of model parameters
summary(usl.model)

## Show complete list of efficiency parameters
efficiency(usl.model)

## Extract coefficients for model
coef(usl.model)

## Calculate point of peak scalability
peak.scalability(usl.model)

## Plot original data and scalability function
plot(raytracer)
plot(usl.model, add=TRUE)
```

USL-class *Class "USL" for Universal Scalability Law models*

Description

This class encapsulates the Universal Scalability Law. Use the function usl to create new objects from this class.

Slots

`frame` The model frame.

`call` The call used to create the model.

`regr` The name of the regressor variable.

`resp` The name of the response variable.

`coefficients` The coefficients alpha, beta and gamma of the model.

`coef.std.err` The standard errors for the coefficients alpha and beta.

`coef.names` A vector with the names of the coefficients.

`fitted` The fitted values of the model. This is a vector.

`residuals` The residuals of the model. This is a vector.

`df.residual` The degrees of freedom of the model.

`sigma` The residual standard deviation of the model.

`limit` The scalability limit as per Amdahl.

`peak` A vector with the predictor and response values of the peak.

`optimal` A vector with the optimal predictor and response values.

`efficiency` The efficiency, e.g. speedup per processor.

`na.action` The `na.action` used by the model.

See Also

usl

Index