Evalys Documentation

Release 4.0.4

Olivier Richard

Feb 01, 2019

Contents

CHAPTER 1

Evalys - Overview

"Infrastructure Performance Evaluation Toolkit"

It is a data analytics library made to load, compute, and plot data from job scheduling and resource management traces. It allows scientists and engineers to extract useful data and visualize it interactively or in an exported file.

- Free software: BSD license
- Documentation: https://evalys.readthedocs.org.

1.1 Features

- Load and all Batsim outputs files
  - Compute and plot free slots
  - Simple Gantt visualisation
  - Compute utilisation / queue
  - Compute fragmentation
  - Plot energy and machine state
- Load SWF workload files from Parallel Workloads Archive
  - Compute standard scheduling metrics
  - Show job details
  - Extract periods with a given mean utilisation

1.2 Examples

You can get a simple example directly by running ipython and discover the evalys interface. For example:

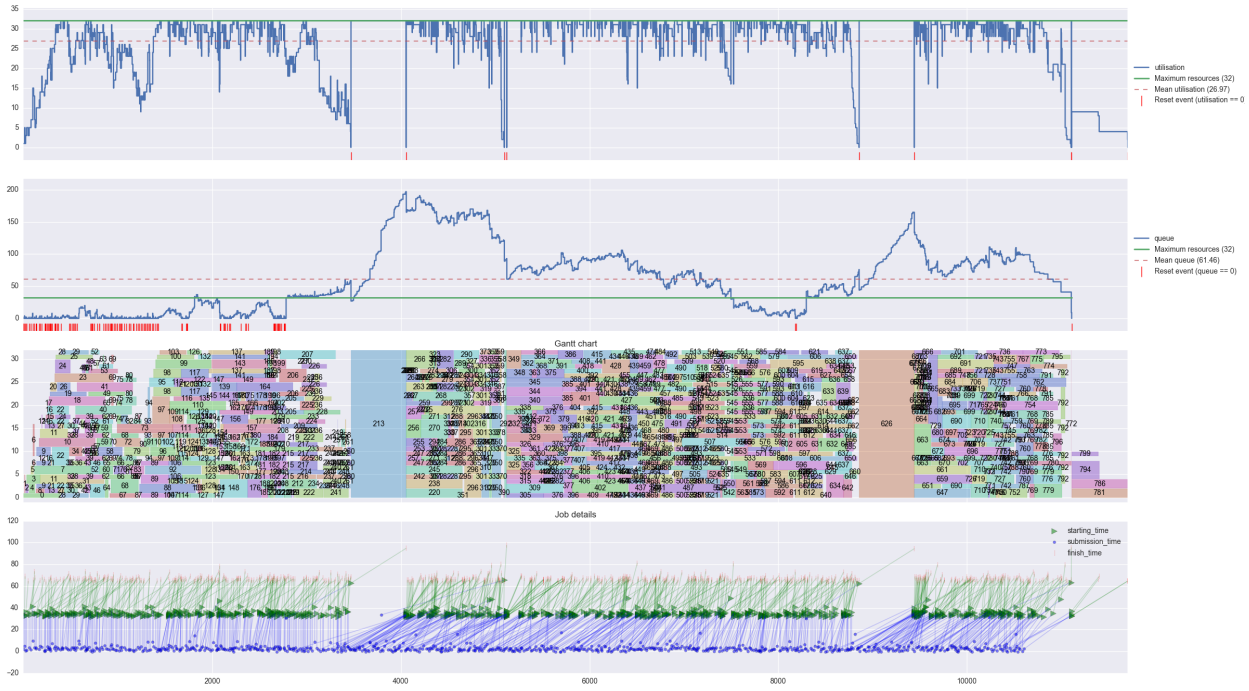
```
from evalys.jobset import JobSet
import matplotlib.pyplot as plt

js = JobSet.from_csv("evalys/examples/jobs.csv")
js.plot(with_details=True)
plt.show()
```

This also works for SWF files but the Gantt chart is not provided because job placement information is not provided in this format.

You can find a lot of examples in the *./examples* directory.

1.3 Gallery



Contents:

1.3.1 Installation

You can install, upgrade, uninstall evalys with these commands:

```
pip install [--user] evalys
pip install [--user] --upgrade evalys
pip uninstall evalys
```

Or from git (last development version):

```
pip install git+https://github.com/oar-team/evalys.git
```

Or if you already pulled the sources:

```
pip install path/to/sources
```

Or if you don't have pip:

```
easy_install evalys
```

1.3.2 Evalys module documentation

Workload: Handle Feitelson's SWF

This module contains the data and metadata given in the headers of the SWF format.

SWF is the default format for parallel workload defined here: see: http://www.cs.huji.ac.il/labs/parallel/workload/swf.html

class evalys.workload.**Workload**(*dataframe*, *wkd_format='swf'*, ***kwargs*)

This class is a derived from the SWF format. SWF is the default format for parallel workload defined here: http://www.cs.huji.ac.il/labs/parallel/workload/swf.html

the data format is one line per job, with 18 fields:

0. Job Number, a counter field, starting from 1
1. Submit Time, seconds. submittal time
2. Wait Time, seconds. diff between submit and begin to run
3. Run Time, seconds. end-time minus start-time
4. Number of Processors, number of allocated processors
5. Average CPU Time Used, seconds. user+system. avg over procs
6. Used Memory, KB. avg over procs.
7. Requested Number of Processors, requested number of processors
8. Requested Time, seconds. user runtime estimation
9. Requested Memory, KB. avg over procs.
10. status (1=completed, 0=killed), 0=fail; 1=completed; 5=canceled
11. User ID, user id
12. Group ID, group id
13. Executable (Application) Number, [1,2..n] n = app# appearing in log
14. Queue Number, [1,2..n] n = queue# in the system
15. Partition Number, [1,2..n] n = partition# in the systems
16. Preceding Job Number, cur job will start only after ...
17. Think Time from Preceding Job, seconds should elapse between termination of this preceding job. Together with the next field, this allows the workload to include feedback as described below.

classmethod from_csv(*filename*)
Import SWF or OWF CSV file. :param filename: SWF or OWF file path

to_csv(*filename*)
Export the workload as SWF format CSV file :param filename: exported SWF file path

queue
Calculate cluster queue size over time in number of procs.

Returns a time indexed serie that contain the number of used processors It is based on real time if UnixStartTime is defined

utilisation

Calculate cluster utilisation over time: nb procs used / nb procs available

Returns a time indexed serie that contain the number of used processors It is based on real time

TODO: Add ID list of jobs running and ID list of jobs in queue to the returned dataframe

plot(*normalize=False*, *with_details=False*, *time_scale=False*)
Plot workload general informations.

Args with_details if True show the job submission, start and finish time (Warning: don't use this on large traces.

extract_periods_with_given_utilisation(*period_in_hours*, *utilisation*, *variation=0.01*, *nb_max=None*, *merge_basic=False*, *merge_change_submit_times=False*, *randomize_starting_times=False*, *random_seed=0*, *max_nb_jobs=None*)
This extract from the workload a period (in hours) with a given mean utilisation (between 0 and 1).

Returns a list of workload of the given periods, with the given utilisation, extracted from the this workload.

extract(*periods*, *notes=''*, *merge_basic=False*, *merge_change_submit_times=False*, *max_nb_jobs=None*)
Extract workload periods from the given workload dataframe. Returns a list of extracted Workloads. Some notes can be added to the extracted workload. It will be stored in Notes attributs and it will appear in the SWF header if you extract it to a file with to_csv().

For example:

```
>>> from evalys.workload import Workload
>>> w = Workload.from_csv("./examples/UniLu-Gaia-2014-2.swf")
>>> periods = pd.DataFrame([{"begin": 200000, "end": 400000},
...                         {"begin": 400000, "end": 600000}])
>>> extracted = w.extract(periods)
```

Handle Batsim output files

class evalys.jobset.**JobSet**(*df*, *resource_bounds=None*, *float_precision=6*)
A JobSet is a set of jobs with it state, its time properties and the resources it is associated with.

It takes a dataframe in input that are intended to have the columns defined in :py::*JobSet.columns*.

The *allocated_resources* one should contain the string representation of an interval set of the allocated resources for the given job, i.e. for this interval:

```
# interval_set representation
[(1, 2), (5, 5), (10, 50)]
# strinf representation
1-2 5 10-50
```

Warning: Floating point precision is set to `self.float_precision` so all floating point values are rounded with this number of digits. Defalut set to 6

For example:

```
>>> from evalys.jobset import JobSet
>>> js = JobSet.from_csv("./examples/jobs.csv")
>>> js.plot(with_details=True)
>>> # to show the graph
>>> # import matplotlib.pyplot as plt
>>> # plt.show()
```

You can also specify the the resource_bounds like this:

```
>>> js = JobSet.from_csv("./examples/jobs.csv",
...                      resource_bounds=(0, 63))
```

to_csv (*filename*)
Export this jobset to a csv file with a ',' as separator.

Example:

```
>>> from evalys.jobset import JobSet
>>> js = JobSet.from_csv("./examples/jobs.csv")
>>> js.to_csv("/tmp/jobs.csv")
```

queue
Calculate cluster queue size over time in number of procs.

Returns a time indexed serie that contain the number of used processors

reset_time (*to=0*)
Reset the time index by giving the first submission time as 1

free_intervals (*begin_time=0*, *end_time=None*)

Returns a dataframe with the free resources over time. Each line corespounding to an event in the jobset.

free_slots (*begin_time=0*, *end_time=None*)

Returns a DataFrame (compatible with a JobSet) that contains all the not overlapping square free slots of this JobSet maximzing the time. It can be transform to a JobSet to be plot as gantt chart.

free_resources_gaps (*resource_intervals=None*, *begin_time=0*, *end_time=None*)

Parameters **resource_intervals** – An interval set on which compute the free resources gaps, Default: self.res_bounds

Returns a resource indexed list where each element is a numpy array of free slots.

Visualisation library

class evalys.visu.gantt.**GanttVisualization**(*lspec*, *, *title='Gantt chart'*)
Visualization of a jobset as a Gantt chart.

The *GanttVisualization* class displays a jobset as a Gantt chart. Each job in the jobset is represented as a set of rectangle. The x-axis represents time, while the y-axis represents resources.

Variables

- **COLUMNS** – The columns required to build the visualization.
- **_lspec** (*core._LayoutSpec*) – The specification of the layout for the visualization.
- **_ax** – The *Axe* to draw on.
- **palette** – The palette of colors to be used.
- **xscale** – The requested adaptation of the x-axis scale. Valid values are *None*, and *'time'*.
  - It defaults to *None*, and uses raw values by default.
  - If set to *time*, the x-axis interprets the data as timestamps, and uses a time-aware semantic.
- **alpha** (*float*) – The transparency level of the rectangles depicting jobs. It defaults to *0.4*.
- **colorer** – The strategy to assign a color to a job. By default, the colors of the palette are picked with a round-robin strategy. The colorer is a function expecting two positional parameters: first the *job*, and second the *palette*. See *GanttVisualization.round_robin_map* for an example.
- **labeler** – The strategy to label jobs. By default, the *jobID* column is used to label jobs. To disable the labeling of jobs, simply return an empty string.

build(*jobset*)
Extract meaningful data from *jobset*, and create the plot.

class evalys.visu.gantt.**DiffGanttVisualization**(*lspec*, *, *title='Gantt charts comparison'*)

build(*jobsets*)
Extract meaningful data from *jobset*, and create the plot.

evalys.visu.gantt.**plot_gantt**(*jobset*, *, *title='Gantt chart'*, ***kwargs*)
Helper function to create a Gantt chart of a workload.

Parameters

- **jobset** (JobSet) – The jobset under study.
- **title** (str) – The title of the window.
- ****kwargs** – The keyword arguments to be fed to the constructor of the visualization class.

evalys.visu.gantt.**plot_diff_gantt**(*jobsets*, *, *title='Gantt charts comparison'*, ***kwargs*)
Helper function to create a comparison of Gantt charts of two (or more) workloads.

Parameters

- **jobsets** – The jobsets under study.
- **title** (*str*) – The title of the window.
- ****kwargs** – The keyword arguments to be fed to the constructor of the visualization class.

class evalys.visu.details.**DetailsLayout**(*, *wtitle='Detailed Figure'*)
Layout in 4 horizontal stripes.

This layout is a support to combine various already-existing visualizations.

show()
Display the figure window.

evalys.visu.details.**plot_details**(*jobset*, *, *title='Workload overview'*, ***kwargs*)
Helper function to create a detailed overview of a workload.

Parameters

- **jobset** (*JobSet*) – The jobset under study.
- **title** (*str*) – The title of the window.
- ****kwargs** – The keyword arguments to be fed to the constructors of the visualization classes.

class evalys.visu.lifecycle.**LifecycleVisualization**(*lspec*, *, *title="Jobs' lifecycle"*)
Visualization of the lifecycle of jobs in a jobset.

The *LifecycleVisualization* class displays for each job its associated events. The visualization is divided into horizontal stripes. Each stripe corresponds to an event type. The x-axis represents time, while the y-axis represents the size of the jobs.

Variables

- **COLUMNS** – The columns required to build the visualization.
- **_events** – The supported events.
- **_lspec** (*_LayoutSpec*) – The specification of the layout for the visualization.
- **_ax** – A dictionnary containing the instances of *Axe* to draw on. There is an instance per stripe, each stripe *Axe* is identified by the event name (cf. *self._events*).
- **palette** – The palette of colors to be used.
- **markers** – The markers to use for each event type. It defaults to *('.', '>', 'I')*. The strings must be valid markers for matplotlib. Each marker is associated with the event of same index in *self._events*.
- **markersizes** – The size to use for each of the markers. It defaults to *(10, 8, 15)*. Each size is associated to the marker associated with the event of same index in *self._events*.
- **alpha** (*float*) – The transparency level for the markers. It defaults to *0.5*. The transparency level of the links is set to *0.4 * self.alpha*.
- **xscale** – The requested adaptation of the x-axis scale. Valid values are *None*, and *'time'*.
  - It defaults to *None*, and uses raw values by default.
  - If set to *time*, the x-axis interprets the data as timestamps, and uses a time-aware semantic.
- **yscale** – The requested adaptation of the y-axis scale. Valid values are *None*, and *'log2'*.
  - It defaults to *None*, and uses raw values by default.
  - If set to *log2*, the y-axis is transformed to show the values on a logarithmic scale in base 2.

title
Title of the visualization.

build(*jobset*)
Extract meaningful data from *jobset*, and create the plot.

evalys.visu.lifecycle.**plot_lifecycle**(*jobset*, *, *title="Jobs' life cycle"*, ***kwargs*)
Helper function to create a lifecycle visualization of a workload.

Parameters

- **jobset** (*JobSet*) – The jobset under study.
- **title** (*str*) – The title of the window.
- ****kwargs** – The keyword arguments to be fed to the constructor of the visualization class.

class evalys.visu.series.**SeriesVisualization**(*lspec*, *, *title='Time Series plot'*)
Base class to visualize series.

Variables

- **_metric** – The metric to visualize.
- **available_series** – Mapping of all the knowns series' visualization classes. This dict is meant to be used through the factory class method.
- **_lspec** (*_LayoutSpec*) – The specification of the layout for the visualization.
- **_ax** – The *Axe* to draw on.
- **palette** – The palette of colors to be used.
- **xscale** – The requested adaptation of the x-axis scale. Valid values are *None*, and *'time'*.
  - It defaults to *None*, and uses raw values by default.
  - If set to *time*, the x-axis interprets the data as timestamps, and uses a time-aware semantic.

classmethod factory(*name*)
Access visualizations of series by name.

The available visualizations have to be registered with the class decorator *register*.

Parameters **name** – Name of the requested visualization.

Type str

Returns The actual *SeriesVisualization* subclass registered as *name*.

build(*jobset*)
Extract meaningful data from *jobset*, and create the plot.

evalys.visu.series.**register**(*, *name*, *column=None*)
Register a series visualization.

Available series visualization must inherit from SeriesVisualization, and are registered with this class decorator.

Parameters

- **name** – The name under which the visualization is registered in *SeriesVisualization*. This name is to be used with the factory class method of *SeriesVisualization*.
- **column** – The actual column name that is used for the series. It defaults to *name*.

Type str

class evalys.visu.series.**QueueSeriesVisualization**(*lspec*, *, *title='Queue size'*)
Visualization of the size of the queue with respect to time.

class evalys.visu.series.**UtilizationSeriesVisualization**(*lspec*, *, *title="Resources' utilization"*)

Visualization of the resources' utilization with respect to time.

evalys.visu.series.**plot_series**(*jobset*, *, *name*, *title='Time series plot'*, ***kwargs*)

Helper function to create a series visualization of a workload.

Parameters

- **jobset** (*JobSet*) – The jobset under study.
- **name** – Name of the requested series visualization.
- **title** (*str*) – The title of the window.
- ****kwargs** – The keyword arguments to be fed to the constructor of the visualization class.

Type *str*

evalys.visu.core.**generate_palette**(*size*)

Return of discrete palette with the specified number of different colors.

class evalys.visu.core.**EvalysLayout**(*, *wtitle='Evalys Figure'*)

Base layout to organize visualizations.

Variables

- **fig** – The actual figure to draw on.
- **sps** – The *SubplotSpec* defined in the layout.
- **visualizations** (*dict*) – Binding of the visualizations injected into the layout. For each key *spskey* in *self.sps*, *self.visualizations[spskey]* is a list of the visualizations with root *SubplotSpec self.sps[spskey]*.

show()

Display the figure window.

inject(*visu_cls*, *spskey*, **args*, ***kwargs*)

Create a visualization, and bind it to the layout.

Parameters

- **visu_cls** – The class of the visualization to create. This should be *Visualization* or one of its subclass.
- **spskey** – The key identifying the *SubplotSpec* fed to the injected *Visualization* (or a subclass). This key must exist in *self.sps*.
- ***args** – The positional arguments to be fed to the constructor of the visualization class.
- ****kwargs** – The keyword arguments to be fed to the constructor of the visualization class.

Returns The newly created visualization.

Return type visu_cls

wtitle

The title of the window containing the layout.

class evalys.visu.core.**SimpleLayout**(*, *wtitle='Simple Figure'*)

Simplest possible layout that uses all available space.

class evalys.visu.core.**Visualization**(*lspec*)

Base class to define visualizations.

Variables

- **_lspec** (*_LayoutSpec*) – The specification of the layout for the visualization.
- **_ax** – The *Axe* to draw on.
- **palette** – The palette of colors to be used.

build(*jobset*)
Extract meaningful data from *jobset*, and create the plot.

title
Title of the visualization.

evalys.visu.legacy.**map_unique_numbers**(*df*)
Map the DataFrame of jobs to a set of jobs which should be labeled and a list of unique ids for the given DataFrame.

Jobs which have the same jobID and workload_name will be merged together and the same unique_id will be assigned to them. The set of labeled_jobs will only contain the job in the middle of each list of jobs sharing the same id.

evalys.visu.legacy.**plot_processor_load**(*jobset*, *ax=None*, *title='Load'*, *labels=True*)
Display the impact of each job on the load of each processor.

need: execution_time, jobID, allocated_resources

evalys.visu.legacy.**plot_series**(*series_type*, *jobsets*, *ax=None*, *time_scale=False*)
Plot one or several time series about provided jobsets on the given ax series_type can be any value present in available_series.

evalys.visu.legacy.**plot_gantt_general_shape**(*jobset_list*, *ax=None*, *alpha=0.3*, *title='Gantt general shape'*)
Draw a general gantt shape of multiple jobsets on one plot for comparison

evalys.visu.legacy.**plot_series_comparison**(*series*, *ax=None*, *title='Series comparison'*)
Plot and compare two serie in post step

evalys.visu.legacy.**plot_fragmentation**(*frag*, *ax=None*, *label='Fragmentation'*)
Plot fragmentation raw data, distribution and ecdf in 3 subplots given in the ax list fragmentation can be optain using fragmentation method

evalys.visu.legacy.**plot_load**(*load*, *nb_resources=None*, *ax=None*, *normalize=False*, *time_scale=False*, *load_label='load'*, *UnixStartTime=0*, *TimeZoneString='UTC'*)
Plots the number of used resources against time :normalize: if True normalize by the number of resources *nb_resources*

evalys.visu.legacy.**plot_free_resources**(*utilisation*, *nb_resources*, *normalize=False*, *time_scale=False*, *UnixStartTime=0*, *TimeZoneString='UTC'*)
Plots the number of free resources against time :normalize: if True normalize by the number of resources *nb_resources*

Metrics computation

evalys.metrics.**cumulative_waiting_time**(*dataframe*)
Compute the cumulative waiting time on the given dataframe

Dataframe a DataFrame that contains a "starting_time" and a "waiting_time" column.

evalys.metrics.**compute_load**(*dataframe*, *col_begin*, *col_end*, *col_cumsum*, *begin_time=0*, *end_time=None*)

Compute the load of the *col_cumsum* columns between events from *col_begin* to *col_end*. In practice it is used to compute the queue load and the cluster load (utilisation).

Returns a load dataframe of all events indexed by time with a *load* and an *area* column.

evalys.metrics.**load_mean**(*df*, *begin=None*, *end=None*)

Compute the mean load area from begin to end.

evalys.metrics.**fragmentation**(*free_resources_gaps*, *p=2*)

Input is a resource indexed list where each element is a numpy array of free slots.

This metrics definition comes from Gher and Shneider CCGRID 2009.

Utilities

evalys.utils.**bulksetattr**(*obj*, ***kwargs*)

Safely assign attributes in bulk.

For each keyword argument kw, the function checks that kw is the name of one of the object's attributes. If kw is not the name of an attribute, the function raises an AttributeError. Otherwise, the function assigns the value of the keyword argument to the attribute, provided the object allows it.

evalys.utils.**cut_workload**(*workload_df*, *begin_time*, *end_time*)

Extract any workload dataframe between begin_time and end_time. Datafram must contain 'submission_time', 'waiting_time' and 'execution_time' + 'jobID' columns.

Jobs that are queued (submitted but not running) before *begin_time* and jobs that are running before *begin_time* and/or after *end_time* are cut to fit in this time slice.

Example with `evalys.Workload`:

```
>>> from evalys.workload import Workload
>>> w = Workload.from_csv("./examples/UniLu-Gaia-2014-2.swf")
>>> cut_w = cut_workload(w.df, 500000, 600000)
```

Example with `evalys.JobSet`:

```
>>> from evalys.jobset import JobSet
>>> js = JobSet.from_csv("./examples/jobs.csv")
>>> cut_js = cut_workload(js.df, 1000, 2000)
```

1.3.3 Credits

- Olivier Richard <olivier.richard@imag.fr>
- Michael Mercier <michael.mercier@inria.fr>
- Millian Poquet <millian.poquet@inria.fr>
- Raphaël Bleuse <raphael.bleuse@uni.lu>
- Valentin Reis <valentin.reis@inria.fr>
- Steffen Lackner <lackner@cs.tu-darmstadt.de>

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

e

Index

Q

R

S

T

U

V

W