WfCommons

Release 0.6

WfCommons Team

Jun 02, 2021

QUICKSTART

WfCommons

WfCommons is a community framework for enabling scientific workflow research and development. This Python package provides methods for analyzing instances, deriving recipes, and generating representative synthetic workflow instances.

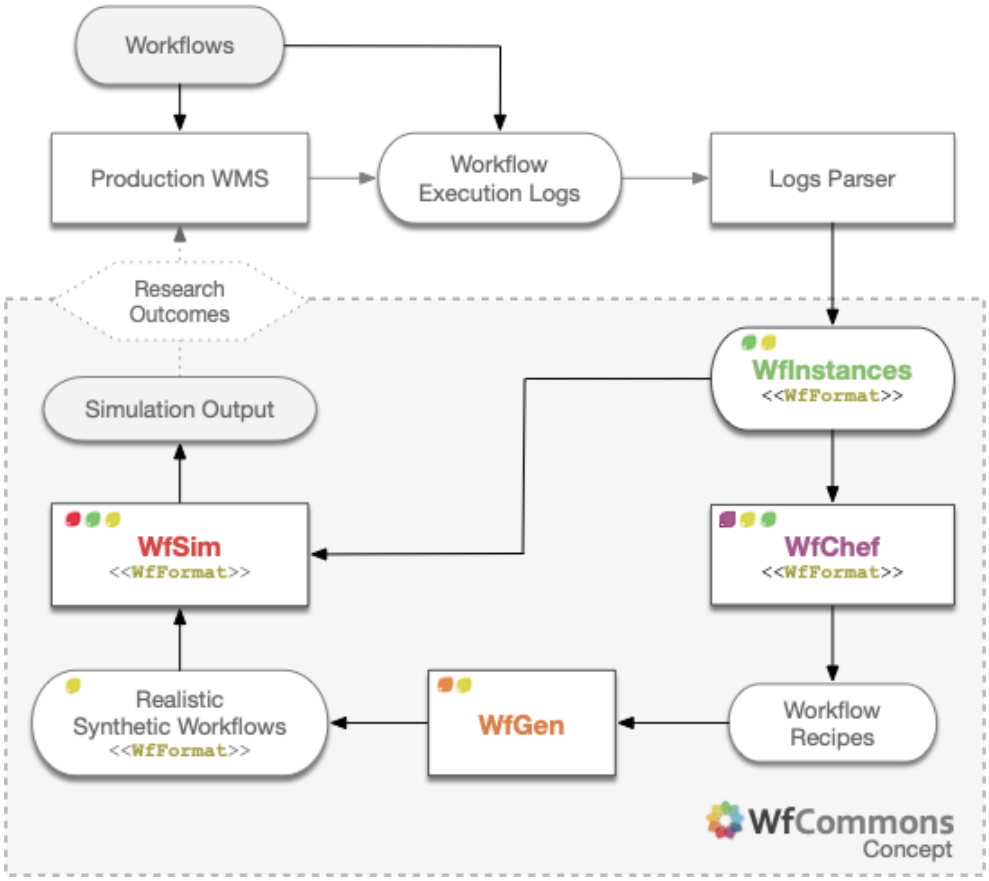

Fig. 1: The WfCommons conceptual architecture.

The source code for this WfCommons's Python package is available on GitHub. Our preferred channel to report a bug or request a feature is via WfCommons's Github Issues Track.

You can also reach the WfCommons team via our support email: support@wfcommons.org.

CHAPTER

ONE

INSTALLATION

WfCommons is available on PyPI. WfCommons requires Python3.6+ and has been tested on Linux and macOS.

1.1 Requirements

1.1.1 Graphviz

WfCommons uses pygraphviz and thus needs the graphviz package installed (version 2.16 or later). You can install graphviz easily on Linux with your favorite package manager, for example for Debian-based distributions:

```
$ sudo apt-get install graphviz libgraphviz-dev
```

and for RedHat-based distributions:

```
$ sudo yum install python-devel graphviz-devel
```

On macOS you can use the `brew` package manager:

```
$ brew install graphviz
```

1.2 Installation using pip

While `pip` can be used to install WfCommons, we suggest the following approach for reliable installation when many Python environments are available:

```
$ python3 -m pip install wfcommons
```

1.3 Retrieving the latest unstable version

If you want to use the latest WfCommons unstable version, that will contain brand new features (but also contain bugs as the stabilization work is still underway), you may consider retrieving the latest unstable version.

Cloning from WfCommons's GitHub repository:

```
$ git clone https://github.com/wfcommons/wfcommons
$ cd wfcommons
$ pip install .
```

CHAPTER

TWO

THE WFCOMMONS PROJECT

The WfCommons project is an open source framework for enabling scientific workflow research and development by providing foundational tools for analyzing workflow execution instances, and generating synthetic, yet realistic, workflow instances that can be used to develop new techniques, algorithms and systems that can overcome the challenges of efficient and robust execution of ever larger workflows on increasingly complex distributed infrastructures.

The figure below shows an overview of the research and development life cycle that integrates the four major components WfCommons: (i) workflow execution instances (**WfInstances**), (ii) workflow recipes (**WfChef**), (iii) workflow generator (**WfGen**), and (iv) workflow simulator (**WfSim**).

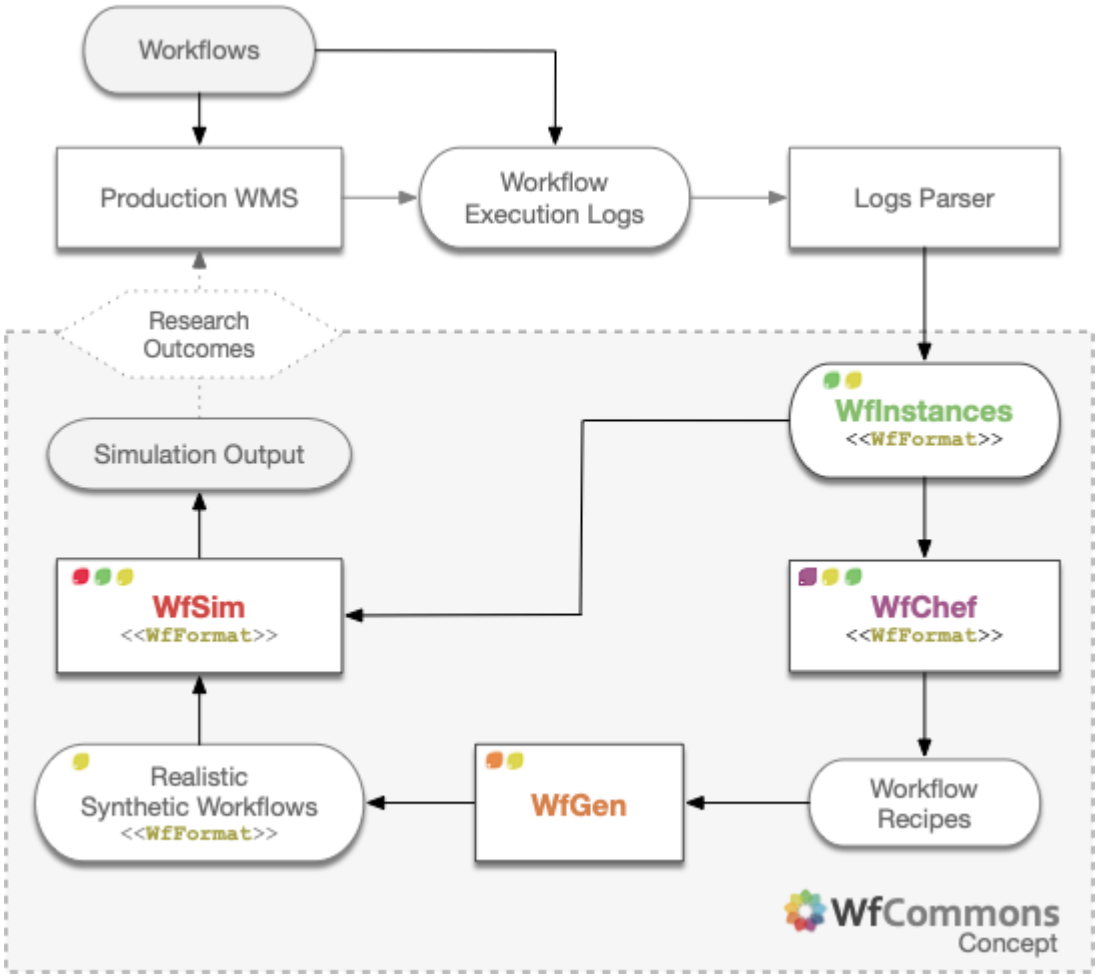

Fig. 1: The WfCommons conceptual architecture.

WfInstances. The WfInstances component provides a collection and curation of open-access production workflow instances from various scientific applications, all made available using a common format (i.e., *WfFormat*). A workflow instance is built based on logs of an actual execution of a scientific workflow on a distributed platform (e.g., clouds, grids, clusters) using a workflow system. We keep a list of workflow execution instances in our project website.

WfChef. The WfChef component automates the construction of synthetic workflow generators (recipes) for any given workflow application. The input to this component is a set of real workflow instances described in the *WfFormat* (e.g., instances available in WfInstances).

WfGen. The WfGen component targets the generation of realistic synthetic workflow instances. WfGen takes as input a workflow recipe produced by WfChef for a particular application and a desired number of tasks. WfGen then automatically generates synthetic, yet realistic, randomized workflow instances with (approximately) the desired number of tasks.

WfSim. The WfCommons project fosters the use of simulation for the development, evaluation, and verification of scheduling and resource provisioning algorithms (e.g., multi-objective function optimization, etc.), evaluation of current and emerging computing platforms (e.g., clouds, IoT, extreme scale, etc.), among others. We do not develop simulators as part of the WfCommons project. Instead, the WfSim component catalogs open source WMS simulators that provide support for *WfFormat*. We keep a list of open source workflow management systems simulators and simulation frameworks on our project website.

2.1 WfFormat

The WfCommons project uses a common format for representing workflow execution instances and generated synthetic workflows instances. Workflow simulators and simulation frameworks that support WfFormat can then use both types of instances interchangeably. WfFormat uses a JSON specification available in the WfFormat Schema GitHub repository. The current version of the WfCommons Python package uses the schema version `1.1`. The schema GitHub repository provides detailed explanation of WfFormat (including required fields), and also a validator script for verifying the compatibility of instances.

CHAPTER

THREE

PARSING WORKFLOW EXECUTION LOGS

The most common way for obtaining **workflow instances** from actual workflow executions is to parse execution logs. As part of the WfCommons project, we are constantly developing parsers for commonly used workflow management systems. The parsers provided in this Python package automatically scans execution logs to produce instances using *WfFormat*.

Each parser class is derived from the abstract `LogsParser` class. Thus, each parser provides a `build_workflow()` method.

3.1 Makeflow

Makeflow is a workflow system for executing large complex workflows on clusters, clouds, and grids. The Makeflow language is similar to traditional "Make", so if you can write a Makefile, then you can write a Makeflow. A workflow can be just a few commands chained together, or it can be a complex application consisting of thousands of tasks. It can have an arbitrary DAG structure and is not limited to specific patterns. Makeflow is used on a daily basis to execute complex scientific applications in fields such as data mining, high energy physics, image processing, and bioinformatics. It has run on campus clusters, the Open Science Grid, NSF XSEDE machines, NCSA Blue Waters, and Amazon Web Services. Makeflow logs provide time-stamped event instances from these executions. The following example shows the analysis of Makeflow execution logs, stored in a local folder (execution dir), for a workflow execution using the *MakeflowLogsParser* class:

```
from wfcommons.wfinstances import MakeflowLogsParser

# creating the parser for the Makeflow workflow
parser = MakeflowLogsParser(execution_dir='/path/to/makeflow/execution/dir/blast/
↪chameleon-small-001/'
                            resource_monitor_logs_dir='/path/to/makeflow/resource/
↪monitor/logs/dir')

# generating the workflow instance object
workflow = parser.build_workflow('workflow-test')

# writing the workflow instance to a JSON file
workflow.write_json('workflow.json')
```

Note: The *MakeflowLogsParser* class requires that Makeflow workflows to run with the Resource Monitor tool (e.g., execute the workflow using the `--monitor=logs`).

3.2 Pegasus WMS

Pegasus is being used in production to execute workflows for dozens of high-profile applications in a wide range of scientific domains. Pegasus provides the necessary abstractions for scientists to create workflows and allows for transparent execution of these workflows on a range of compute platforms including clusters, clouds, and national cyberinfrastructures. Workflow execution with Pegasus includes data management, monitoring, and failure handling, and is managed by HTCondor DAGMan. Individual workflow tasks are managed by a workload management framework, HTCondor, which supervises task executions on local and remote resources. Pegasus logs provide time-stamped event instances from these executions. The following example shows the analysis of Pegasus execution logs, stored in a local folder (submit dir), for a workflow execution using the *PegasusLogsParser* class:

```
from wfcommons.wfinstances import PegasusLogsParser

# creating the parser for the Pegasus workflow
parser = PegasusLogsParser(submit_dir='/path/to/pegasus/submit/dir/seismology/chameleon-
↪100p-001/')

# generating the workflow instance object
workflow = parser.build_workflow('workflow-test')

# writing the workflow instance to a JSON file
workflow.write_json('workflow.json')
```

Warning: By default, the *PegasusLogsParser* class assumes that the submit dir is from a Pegasus execution with **version 5.0** or later. To enable parsing of Pegasus execution logs from version 4.9 or earlier, the option `legacy=True` should be used.

CHAPTER

FOUR

ANALYZING INSTANCES

Workflow execution instances have been widely used to profile and characterize workflow executions, and to build distributions of workflow execution behaviors, which are used to evaluate methods and techniques in simulation or in real conditions.

The WfCommons project targets the analysis of actual workflow execution instances (i.e., the workflow execution profile data and characterizations) in order to build *Workflow Recipes* of workflow applications. These recipes contain the necessary information for generating synthetic, yet realistic, workflow instances that resemble the structure and distribution of the original workflow executions.

A list of workflow execution instances that are compatible with *WfFormat* is kept constantly updated in our project website.

4.1 WfInstances

A workflow execution instance represents an actual execution of a scientific workflow on a distributed platform (e.g., clouds, grids, HPC, etc.). In the WfCommons project, an instance is represented in a JSON file following the schema described in *WfFormat*. This Python package provides an *instance loader* tool for importing workflow execution instances for analysis. For instance, the code snippet below shows how an instance can be loaded using the `Instance` class:

```
from wfcommons import Instance
instance = Instance(input_instance='/path/to/instance/file.json')
```

The `Instance` class provides a number of methods for interacting with the workflow instance, including:

- `draw()`: produces an image or a pdf file representing the instance.
- `leaves()`: gets the leaves of the workflow (i.e., the tasks without any successors).
- `roots()`: gets the roots of the workflow (i.e., the tasks without any predecessors).
- `write_dot()`: writes a dot file of the instance.

Note: Although the analysis methods are inherently used by WfCommons (specifically WfChef) for *Generating Workflows Recipes*, they can also be used in a standalone manner.

4.2 The Instance Analyzer

The *InstanceAnalyzer* class provides a number of tools for analyzing collection of workflow execution instances. The goal of the *InstanceAnalyzer* is to perform analyzes of one or multiple workflow execution instances, and build summaries of the analyzes per workflow' task type prefix.

Warning: Although any workflow execution instance represented as a *Instance* object (i.e., compatible with *WfFormat*) can be appended to the *InstanceAnalyzer*, we strongly recommend that only instances of a single workflow application type be appended to an analyzer object. You may though create several analyzer objects per workflow application.

The *append_instance()* method allows you to include instances for analysis. The *build_summary()* method processes all appended instances. The method applies probability distributions fitting to a series of data to find the *best* (i.e., minimizes the mean square error) probability distribution that represents the analyzed data. The method returns a summary of the analysis of instances in the form of a Python dictionary object in which keys are task prefixes (provided when invoking the method) and values describe the best probability distribution fit for tasks' runtime, and input and output data file sizes. The code excerpt below shows an example of an analysis summary showing the best fit probability distribution for runtime of the `individuals` tasks (1000Genome workflow):

```
"individuals": {
    "runtime": {
        "min": 48.846,
        "max": 192.232,
        "distribution": {
            "name": "skewnorm",
            "params": [
                11115267.652937062,
                -2.9628504044929433e-05,
                56.03957070238482
            ]
        }
    },
    ...
}
```

Workflow analysis summaries are used by WfChef to develop *Workflow Recipes*, in which themselves are used to *generate realistic synthetic workflow instances*.

Probability distribution fits can also be plotted by using the *generate_fit_plots()* or *generate_all_fit_plots()* methods – plots will be saved as `png` files.

4.3 Examples

The following example shows the analysis of a set of instances, stored in a local folder, of a Seismology workflow. In this example, we seek for finding the best probability distribution fitting for task *prefixes* of the Seismology workflow (`sG1IterDecon`, and `wrapper_siftSTFByMisfit`), and generate all fit plots (runtime, and input and output files) into the `fits` folder using `seismology` as a prefix for each generated plot:

```
from wfcommons import Instance, InstanceAnalyzer
from os import listdir
```

(continues on next page)

(continued from previous page)

```
from os.path import isfile, join

# obtaining list of instance files in the folder
INSTANCES_PATH = "/path/to/some/instance/folder/"
instance_files = [f for f in listdir(INSTANCES_PATH) if isfile(join(INSTANCES_PATH, f))]

# creating the instance analyzer object
analyzer = InstanceAnalyzer()

# appending instance files to the instance analyzer
for instance_file in instance_files:
    instance = Instance(input_instance=INSTANCES_PATH + instance_file)
    analyzer.append_instance(instance)

# list of workflow task name prefixes to be analyzed in each instance
workflow_tasks = ['sG1IterDecon', 'wrapper_siftSTFByMisfit']

# building the instance summary
instances_summary = analyzer.build_summary(workflow_tasks, include_raw_data=True)

# generating all fit plots (runtime, and input and output files)
analyzer.generate_all_fit_plots(outfile_prefix='fits/seismology')
```

CHAPTER

FIVE

GENERATING WORKFLOWS RECIPES

WfChef is the WfCommons component that automates the construction of synthetic workflow generators for any given workflow application. The input to this component is a set of real workflow instances described in the *WfFormat* (e.g., instances available in *WfInstances*). WfChef automatically analyzes a set of real workflow instances for two purposes. First, it discovers workflow subgraphs that represent fundamental task dependency patterns. Second, it derives statistical models of the workflow tasks' performance characteristics (see *Analyzing Instances*). WfChef then outputs a **recipe** that will be used by **WfGen** (see *Generating Workflows*) to generate realistic synthetic workflow instances with any arbitrary number of tasks.

5.1 Workflow Recipes

A **workflow recipe** is a data structure that encodes the discovered pattern occurrences as well as the statistical models of workflow task characteristics. More precisely, a recipe embodies results from statistical analysis and distribution fitting performed for each workflow task type so as to characterize task runtime and input/output data sizes. The recipes also incorporates information regarding the graph structure of the workflows (tasks dependencies and frequency of occurrences), which are automatically derived from the analysis of the workflow instances.

This Python package provides several *workflow recipes* (see *WfCommons Workflows Recipes*) for generating realistic synthetic workflow instances.

5.2 Workflow Recipe Generator

To create a recipe, WfChef analyzes the real workflow graphs in order to identify subgraphs that represent fundamental task dependency patterns. Based on the identified subgraphs and on measured task type frequencies in the real workflows, WfChef outputs a generator that can generate realistic synthetic workflow instances with an arbitrary numbers of tasks (see *Generating Workflows*).

The code snippet below shows an example of how to create a recipe for the Epigenomics application:

```
$ wfchef create /path/to/real/instances -o ./epigenomics -v --name Epigenomics
```

The following flags can be used with this command:

- `-o` or `--out` is a required flag that stands for the name of the directory to be created that is going to contain the recipe.
- `-n` or `--name` is a required flag that stands for the name of the recipe. Tipically, the format used is *Application-Name*.
- `-v` or `--verbose` if set, activates status messages.
- `--no-install` if set, does not install the recipe automatically.

- `-c` or `--cutoff` takes a number of tasks in the samples the user wants to consider to create the recipe.

Example: `--cutoff 4000`, it means that all real world instances that will be consider for the creation of the recipe will have 4000 or less tasks. This is a useful flag to use when there is trust that all possible patterns present in this application can be already found in the smaller instances.

Workflow recipes are automatically installed and can be used throughout the system. WfCommons creates a Python package in the directory specified by the flag `--out` in which the `setup.py` and `recipe.py` files are stored. If the flag `--no-install` is set when creating a package for a specific application, the user will need to manually install the package before using it. The code bellow is an example of how to install/uninstall a package for an application in WfCommons:

```
# installing the package
$ pip install /path/to/the/package

# uninstalling a package
$ pip uninstall wfcommons.wfchef.recipes.appication_name_workflow
```

The snippet below shows an example of how to import the recipes:

```
from wfcommons.wfchef.recipes import EpigenomicsRecipe
```

To check which recipes are installed in a system and how to import them use:

```
$ wfchef ls
```

CHAPTER

SIX

GENERATING WORKFLOWS

WfGen is a component of WfCommons project that targets the generation of realistic synthetic workflow instances with a variety of characteristics. The *WorkflowGenerator* class uses recipes of workflows (as described in *Workflow Recipe Generator*) for creating the realistic synthetic instances. The resulting workflows are represented in the *WfFormat*, which is already supported by simulation frameworks such as WRENCH.

6.1 WfCommons Workflows Recipes

This Python package provides several *workflow recipes* for generating realistic synthetic workflow instances. The current list of available workflow recipes include:

- *BlastRecipe*: `from wfcommons.wfchef.recipes import BlastRecipe`
- *BwaRecipe*: `from wfcommons.wfchef.recipes import BwaRecipe`
- *CyclesRecipe*: `from wfcommons.wfchef.recipes import CyclesRecipe`
- *EpigenomicsRecipe*: `from wfcommons.wfchef.recipes import EpigenomicsRecipe`
- *GenomeRecipe*: `from wfcommons.wfchef.recipes import GenomeRecipe`
- *MontageRecipe*: `from wfcommons.wfchef.recipes import MontageRecipe`
- *SeismologyRecipe*: `from wfcommons.wfchef.recipes import SeismologyRecipe`
- *SoykbRecipe*: `from wfcommons.wfchef.recipes import SoykbRecipe`
- *SrasearchRecipe*: `from wfcommons.wfchef.recipes import SrasearchRecipe`

6.2 The Workflow Instances Generator

Synthetic workflow instances are generated using the *WorkflowGenerator* class. This class takes as input a `WorkflowRecipe` object (see in *Workflow Recipe Generator*), and provides two methods for generating synthetic workflow instances:

- *build_workflow()*: generates a single synthetic workflow instance based on the workflow recipe used to instantiate the generator.
- *build_workflows()*: generates a number of synthetic workflow instances based on the workflow recipe used to instantiate the generator.

The build methods use the workflow recipe for generating realistic synthetic workflow instances, in which the workflow structure follows workflow composition rules defined in the workflow recipe, and tasks runtime, and input and output data sizes are generated according to distributions obtained from actual workflow execution instances (see *Analyzing Instances*).

Each generated instance is represented as a *Workflow* object (which in itself is an extension of the NetworkX DiGraph class). The *Workflow* class provides two methods for writing the generated workflow instance into files:

- *write_dot()*: write a DOT file of a workflow instance.
- *write_json()*: write a JSON file of a workflow instance.

All workflow recipes provide a common method, `from_num_tasks`, that defines the lower bound for the total number of tasks in the generated synthetic workflow.

6.2.1 Increasing/Reducing Runtime and File Sizes

Workflow recipes also allow the generation of synthetic workflows with increased/reduced runtimes and/or files sizes determined by a factor provided by the user:

- `runtime_factor`: The factor of which tasks runtime will be increased/decreased.
- `input_file_size_factor`: The factor of which tasks input files size will be increased/decreased.
- `output_file_size_factor`: The factor of which tasks output files size will be increased/decreased.

The following example shows how to create a Seismology workflow recipe in which task runtime is increased by 10%, input files by 50%, and output files reduced by 20%:

```
from wfcommons.wfchef.recipes import SeismologyRecipe

# creating a Seismology workflow recipe with increased/decreased runtime and file sizes
recipe = SeismologyRecipe.from_num_tasks(num_tasks=100, runtime_factor=1.1, input_file_
↪size_factor=1.5, output_file_size_factor=0.8)
```

6.3 Examples

The following example generates a *Seismology* synthetic workflow instance os 300 tasks, builds a synthetic workflow instance, and writes the synthetic instance to a JSON file.

```
from wfcommons.wfchef.recipes import SeismologyRecipe
from wfcommons import WorkflowGenerator

generator = WorkflowGenerator(SeismologyRecipe.from_num_tasks(250))
workflow = generator.build_workflow()
workflow.write_json(f'seismology-workflow.json')
```

The example below generates a number of 10 *Blast* synthetic workflow instances for every size defined in the array `num_tasks`:

```
from wfcommons.wfchef.recipes import BlastRecipe
from wfcommons import WorkflowGenerator

num_tasks = [100, 250, 370, 800]

for task in num_tasks:
  generator = WorkflowGenerator(BlastRecipe.from_num_tasks(task))
  workflows = generator.build_workflows(10)
```

(continues on next page)

(continued from previous page)

```
  for i, workflow in enumerate(workflows):
    workflow.write_json(f'blast-workflow-{task}-{i}.json')
```

The following example generates 10 *Epigenomics* synthetic workflow instances based on the number of tasks entered by the user (1000), builds the synthetic workflow instances, and writes the synthetic instances to JSON files.

```
from wfcommons.wfchef.recipes import EpigenomicsRecipe
from wfcommons import WorkflowGenerator

generator = WorkflowGenerator(EpigenomicsRecipe.from_num_tasks(1000))
for i, workflow in enumerate(generator.build_workflows(10)):
    workflow.write_json(f'epigenomics-workflow-{i}.json')
```

The example below generates a *Cycles* (agroecosystem) synthetic workflow instance based on the number of tasks entered by the user (250), builds the synthetic workflow instance, and writes the synthetic instance to a JSON file.

```
from wfcommons.wfchef.recipes import CyclesRecipe
from wfcommons import WorkflowGenerator

generator = WorkflowGenerator(CyclesRecipe.from_num_tasks(250))
workflow = generator.build_workflow()
workflow.write_json(f'cycles-workflow.json')
```

CHAPTER

SEVEN

USER API REFERENCE

The user API reference targets users who want to use WfCommons Python package for analyzing instances or generating realistic synthetic workflow instances, using existing workflow recipes already implemented in this Python package. Users are NOT expected to develop new *workflow recipes*.

7.1 wfcommons.common

7.1.1 wfcommons.common.file

class wfcommons.common.file.**File**(*name: str*, *size: int*, *link:* wfcommons.common.file.FileLink, *logger: Optional[logging.Logger] = None*)

Bases: `object`

Representation of a file.

Parameters

- **name** (*str*) – The name of the file.
- **size** (*int*) – File size in KB.
- **link** (FileLink) – Type of file link.
- **logger** (*Logger*) – The logger where to log information/warning or errors.

as_dict() → Dict

A JSON representation of the file.

Returns A JSON object representation of the file.

Return type Dict

class wfcommons.common.file.**FileLink**(*value*)

Bases: *wfcommons.utils.NoValue*

Type of file link.

INPUT = 'input'

OUTPUT = 'output'

7.1.2 wfcommons.common.task

class wfcommons.common.task.**Task**(*name: str*, *task_type:* wfcommons.common.task.TaskType, *runtime: float*, *cores: int*, *machine: Optional[*wfcommons.common.machine.Machine*] = None*, *args: List[str] = []*, *avg_cpu: Optional[float] = None*, *bytes_read: Optional[int] = None*, *bytes_written: Optional[int] = None*, *memory: Optional[int] = None*, *energy: Optional[int] = None*, *avg_power: Optional[float] = None*, *priority: Optional[int] = None*, *files: List[*wfcommons.common.file.File*] = []*, *logger: Optional[logging.Logger] = None*)

Bases: object

Representation of a task.

Parameters

- **name** (*str*) – The name of the task.
- **task_type** (TaskType) – The type of the task.
- **runtime** (*float*) – Task runtime in seconds.
- **cores** (*int*) – Number of cores required by the task.
- **machine** (Machine) – Machine on which is the task has been executed.
- **args** (*List[str]*) – List of task arguments.
- **avg_cpu** (*float*) – Average CPU utilization in %.
- **bytes_read** (*int*) – Total bytes read in KB.
- **bytes_written** (*int*) – Total bytes written in KB.
- **memory** (*int*) – Memory (resident set) size of the process in KB.
- **energy** (*int*) – Total energy consumption in kWh.
- **avg_power** (*float*) – Average power consumption in W.
- **priority** (*int*) – Task priority.
- **files** (*List[File]*) – List of input/output files used by the task.
- **logger** (*Logger*) – The logger where to log information/warning or errors.

as_dict() → Dict

A JSON representation of the task.

Returns A JSON object representation of the task.

Return type Dict

class wfcommons.common.task.**TaskType**(*value*)

Bases: *wfcommons.utils.NoValue*

Task type.

AUXILIARY = 'auxiliary'

COMPUTE = 'compute'

TRANSFER = 'transfer'

7.1.3 wfcommons.common.machine

class wfcommons.common.machine.**Machine**(*name: str*, *cpu: Dict[str, Union[int, str]]*, *system: Optional[wfcommons.common.machine.MachineSystem] = None*, *architecture: Optional[str] = None*, *memory: Optional[int] = None*, *release: Optional[str] = None*, *hashcode: Optional[str] = None*, *logger: Optional[logging.Logger] = None*)

Bases: `object`

Representation of one compute machine.

Parameters

- **name** (*str*) – Machine node name.
- **cpu** (*Dict[str, Union[int, str]]*) – A dictionary containing information about the CPU specification. Must at least contains two fields: *count* (number of CPU cores) and speed (CPU speed of each core in MHz).

```
cpu = {
    'count': 48,
    'speed': 1200
}
```

- **system** (MachineSystem) – Machine system (linux, macos, windows).
- **architecture** (*str*) – Machine architecture (e.g., x86_64, ppc).
- **memory** (*int*) – Total machine's RAM memory in KB.
- **release** (*str*) – Machine release.
- **hashcode** (*str*) – MD5 Hashcode for the Machine.
- **logger** (*Logger*) – The logger where to log information/warning or errors.

as_dict() → Dict

A JSON representation of the machine.

Returns A JSON object representation of the machine.

Return type Dict

class wfcommons.common.machine.**MachineSystem**(*value*)

Bases: *wfcommons.utils.NoValue*

Machine system type.

LINUX = 'linux'

MACOS = 'macos'

WINDOWS = 'windows'

7.1.4 wfcommons.common.workflow

class wfcommons.common.workflow.**Workflow**(*name: str*, *description: Optional[str] = None*, *wms_name: Optional[str] = None*, *wms_version: Optional[str] = None*, *wms_url: Optional[str] = None*, *executed_at: Optional[str] = None*, *makespan: Optional[int] = 0.0*)

Bases: networkx.classes.digraph.DiGraph

Representation of a workflow. The workflow representation is an extension of the NetworkX DiGraph class.

Parameters

- **name** (*str*) – Workflow name.
- **description** (*str*) – Workflow instance description.
- **wms_name** (*str*) – WMS name.
- **wms_version** (*str*) – WMS version.
- **wms_url** (*str*) – URL for the WMS website.
- **executed_at** (*str*) – Workflow start timestamp in the ISO 8601 format.
- **makespan** (*int*) – Workflow makespan in seconds.

to_nx_digraph() → networkx.classes.digraph.DiGraph

write_dot(*dot_filename: Optional[str] = None*) → None
Write a dot file of the workflow instance.

Parameters **dot_filename** (*str*) – DOT output file name.

write_json(*json_filename: Optional[str] = None*) → None
Write a JSON file of the workflow instance.

Parameters **json_filename** (*str*) – JSON output file name.

7.2 wfcommons.wfchef

7.2.1 wfcommons.wfchef.recipes.blast.recipe

class wfcommons.wfchef.recipes.blast.recipe.**BlastRecipe**(*data_footprint: Optional[int] = 0*, *num_tasks: Optional[int] = 3*, *exclude_graphs: Set[str] = {}*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*, *logger: Optional[logging.Logger] = None*, ***kwargs*)

Bases: wfcommons.wfgen.abstract_recipe.WorkflowRecipe

A Blast workflow recipe class for creating synthetic workflow instances.

Parameters

- **data_footprint** (*int*) – The upper bound for the workflow total data footprint (in bytes).
- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow.

- **exclude_graphs** (*Set*) –
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.
- **logger** (*Logger*) – The logger where to log information/warning or errors (optional).

classmethod from_num_tasks(*num_tasks: int*, *exclude_graphs: Set[str] = {}*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.wfchef.recipes.blast.recipe.BlastRecipe*

Instantiate a Blast workflow recipe that will generate synthetic workflows up to the total number of tasks provided.

Parameters

- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow (at least 3).
- **exclude_graphs** (*Set*) –
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns A Blast workflow recipe object that will generate synthetic workflows up to the total number of tasks provided.

Return type *BlastRecipe*

tasks_files: Dict[str, List[File]]

workflows: List[Workflow]

7.2.2 wfcommons.wfchef.recipes.bwa.recipe

class wfcommons.wfchef.recipes.bwa.recipe.**BwaRecipe**(*data_footprint: Optional[int] = 0*, *num_tasks: Optional[int] = 3*, *exclude_graphs: Set[str] = {}*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*, *logger: Optional[logging.Logger] = None*, ***kwargs*)

Bases: wfcommons.wfgen.abstract_recipe.WorkflowRecipe

A Bwa workflow recipe class for creating synthetic workflow instances.

Parameters

- **data_footprint** (*int*) – The upper bound for the workflow total data footprint (in bytes).
- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow.

- **exclude_graphs** (*Set*) –
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.
- **logger** (*Logger*) – The logger where to log information/warning or errors (optional).

classmethod from_num_tasks(*num_tasks: int*, *exclude_graphs: Set[str] = {}*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.wfchef.recipes.bwa.recipe.BwaRecipe*

Instantiate a Bwa workflow recipe that will generate synthetic workflows up to the total number of tasks provided.

Parameters

- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow (at least 3).
- **exclude_graphs** (*Set*) –
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns A Bwa workflow recipe object that will generate synthetic workflows up to the total number of tasks provided.

Return type *BwaRecipe*

tasks_files: Dict[str, List[File]]

workflows: List[Workflow]

7.2.3 wfcommons.wfchef.recipes.cycles.recipe

class wfcommons.wfchef.recipes.cycles.recipe.**CyclesRecipe**(*data_footprint: Optional[int] = 0*, *num_tasks: Optional[int] = 3*, *exclude_graphs: Set[str] = {}*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*, *logger: Optional[logging.Logger] = None*, ***kwargs*)

Bases: wfcommons.wfgen.abstract_recipe.WorkflowRecipe

A Cycles workflow recipe class for creating synthetic workflow instances.

Parameters

- **data_footprint** (*int*) – The upper bound for the workflow total data footprint (in bytes).

- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow.
- **exclude_graphs** (*Set*) –
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.
- **logger** (*Logger*) – The logger where to log information/warning or errors (optional).

classmethod from_num_tasks(*num_tasks: int*, *exclude_graphs: Set[str] = {}*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.wfchef.recipes.cycles.recipe.CyclesRecipe*

Instantiate a Cycles workflow recipe that will generate synthetic workflows up to the total number of tasks provided.

Parameters

- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow (at least 3).
- **exclude_graphs** (*Set*) –
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns A Cycles workflow recipe object that will generate synthetic workflows up to the total number of tasks provided.

Return type *CyclesRecipe*

tasks_files: Dict[str, List[File]]

workflows: List[Workflow]

7.2.4 wfcommons.wfchef.recipes.epigenomics.recipe

class wfcommons.wfchef.recipes.epigenomics.recipe.**EpigenomicsRecipe**(*data_footprint: Optional[int] = 0*, *num_tasks: Optional[int] = 3*, *exclude_graphs: Set[str] = {}*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*, *logger: Optional[logging.Logger] = None*, ***kwargs*)

Bases: wfcommons.wfgen.abstract_recipe.WorkflowRecipe

A Epigenomics workflow recipe class for creating synthetic workflow instances.

Parameters

- **data_footprint** (*int*) – The upper bound for the workflow total data footprint (in bytes).
- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow.
- **exclude_graphs** (*Set*) –
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.
- **logger** (*Logger*) – The logger where to log information/warning or errors (optional).

classmethod **from_num_tasks**(*num_tasks: int*, *exclude_graphs: Set[str] = {}*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.wfchef.recipes.epigenomics.recipe.EpigenomicsRecipe*

Instantiate a Epigenomics workflow recipe that will generate synthetic workflows up to the total number of tasks provided.

Parameters

- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow (at least 3).
- **exclude_graphs** (*Set*) –
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns A Epigenomics workflow recipe object that will generate synthetic workflows up to the total number of tasks provided.

Return type *EpigenomicsRecipe*

tasks_files: Dict[str, List[File]]

workflows: List[Workflow]

7.2.5 wfcommons.wfchef.recipes.genome.recipe

class wfcommons.wfchef.recipes.genome.recipe.**GenomeRecipe**(*data_footprint: Optional[int] = 0*, *num_tasks: Optional[int] = 3*, *exclude_graphs: Set[str] = {}*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*, *logger: Optional[logging.Logger] = None*, ***kwargs*)

Bases: wfcommons.wfgen.abstract_recipe.WorkflowRecipe

A Genome workflow recipe class for creating synthetic workflow instances.

Parameters

- **data_footprint** (*int*) – The upper bound for the workflow total data footprint (in bytes).
- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow.
- **exclude_graphs** (*Set*) –
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.
- **logger** (*Logger*) – The logger where to log information/warning or errors (optional).

classmethod from_num_tasks(*num_tasks: int*, *exclude_graphs: Set[str] = {}*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.wfchef.recipes.genome.recipe.GenomeRecipe*

Instantiate a Genome workflow recipe that will generate synthetic workflows up to the total number of tasks provided.

Parameters

- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow (at least 3).
- **exclude_graphs** (*Set*) –
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns A Genome workflow recipe object that will generate synthetic workflows up to the total number of tasks provided.

Return type *GenomeRecipe*

`tasks_files: Dict[str, List[File]]`

`workflows: List[Workflow]`

7.2.6 wfcommons.wfchef.recipes.montage.recipe

`class wfcommons.wfchef.recipes.montage.recipe.MontageRecipe(`*data_footprint: Optional[int] = 0*, *num_tasks: Optional[int] = 3*, *exclude_graphs: Set[str] = {}*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*, *logger: Optional[logging.Logger] = None*, ***kwargs*)

Bases: `wfcommons.wfgen.abstract_recipe.WorkflowRecipe`

A Montage workflow recipe class for creating synthetic workflow instances.

Parameters

- **data_footprint** (*int*) – The upper bound for the workflow total data footprint (in bytes).
- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow.
- **exclude_graphs** (*Set*) –
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.
- **logger** (*Logger*) – The logger where to log information/warning or errors (optional).

`classmethod from_num_tasks(`*num_tasks: int*, *exclude_graphs: Set[str] = {}*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.wfchef.recipes.montage.recipe.MontageRecipe*

Instantiate a Montage workflow recipe that will generate synthetic workflows up to the total number of tasks provided.

Parameters

- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow (at least 3).
- **exclude_graphs** (*Set*) –
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.

- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns A Montage workflow recipe object that will generate synthetic workflows up to the total number of tasks provided.

Return type *MontageRecipe*

tasks_files: Dict[str, List[File]]

workflows: List[Workflow]

7.2.7 wfcommons.wfchef.recipes.seismology.recipe

class wfcommons.wfchef.recipes.seismology.recipe.**SeismologyRecipe**(*data_footprint: Optional[int] = 0*, *num_tasks: Optional[int] = 3*, *exclude_graphs: Set[str] = {}*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*, *logger: Optional[logging.Logger] = None*, ***kwargs*)

Bases: wfcommons.wfgen.abstract_recipe.WorkflowRecipe

A Seismology workflow recipe class for creating synthetic workflow instances.

Parameters

- **data_footprint** (*int*) – The upper bound for the workflow total data footprint (in bytes).
- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow.
- **exclude_graphs** (*Set*) –
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.
- **logger** (*Logger*) – The logger where to log information/warning or errors (optional).

classmethod from_num_tasks(*num_tasks: int*, *exclude_graphs: Set[str] = {}*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.wfchef.recipes.seismology.recipe.SeismologyRecipe*

Instantiate a Seismology workflow recipe that will generate synthetic workflows up to the total number of tasks provided.

Parameters

- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow (at least 3).
- **exclude_graphs** (*Set*) –

- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns A Seismology workflow recipe object that will generate synthetic workflows up to the total number of tasks provided.

Return type *SeismologyRecipe*

tasks_files: Dict[str, List[File]]

workflows: List[Workflow]

7.2.8 wfcommons.wfchef.recipes.soykb.recipe

class wfcommons.wfchef.recipes.soykb.recipe.**SoykbRecipe**(*data_footprint: Optional[int] = 0*, *num_tasks: Optional[int] = 3*, *exclude_graphs: Set[str] = {}*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*, *logger: Optional[logging.Logger] = None*, ***kwargs*)

Bases: wfcommons.wfgen.abstract_recipe.WorkflowRecipe

A Soykb workflow recipe class for creating synthetic workflow instances.

Parameters

- **data_footprint** (*int*) – The upper bound for the workflow total data footprint (in bytes).
- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow.
- **exclude_graphs** (*Set*) –
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.
- **logger** (*Logger*) – The logger where to log information/warning or errors (optional).

classmethod from_num_tasks(*num_tasks: int*, *exclude_graphs: Set[str] = {}*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.wfchef.recipes.soykb.recipe.SoykbRecipe*

Instantiate a Soykb workflow recipe that will generate synthetic workflows up to the total number of tasks provided.

Parameters

- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow (at least 3).

- **exclude_graphs** (*Set*) –
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns A Soykb workflow recipe object that will generate synthetic workflows up to the total number of tasks provided.

Return type *SoykbRecipe*

tasks_files: Dict[str, List[File]]

workflows: List[Workflow]

7.2.9 wfcommons.wfchef.recipes.srasearch.recipe

class wfcommons.wfchef.recipes.srasearch.recipe.**SrasearchRecipe**(*data_footprint: Optional[int] = 0*, *num_tasks: Optional[int] = 3*, *exclude_graphs: Set[str] = {}*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*, *logger: Optional[logging.Logger] = None*, ***kwargs*)

Bases: wfcommons.wfgen.abstract_recipe.WorkflowRecipe

A Srasearch workflow recipe class for creating synthetic workflow instances.

Parameters

- **data_footprint** (*int*) – The upper bound for the workflow total data footprint (in bytes).
- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow.
- **exclude_graphs** (*Set*) –
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.
- **logger** (*Logger*) – The logger where to log information/warning or errors (optional).

classmethod from_num_tasks(*num_tasks: int*, *exclude_graphs: Set[str] = {}*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.wfchef.recipes.srasearch.recipe.SrasearchRecipe*

Instantiate a Srasearch workflow recipe that will generate synthetic workflows up to the total number of tasks provided.

Parameters

- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow (at least 3).
- **exclude_graphs** (*Set*) –
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns A Srasearch workflow recipe object that will generate synthetic workflows up to the total number of tasks provided.

Return type *SrasearchRecipe*

tasks_files: Dict[str, List[File]]

workflows: List[Workflow]

7.3 wfcommons.wfgen

7.3.1 wfcommons.wfgen.generator

class wfcommons.wfgen.generator.**WorkflowGenerator**(*workflow_recipe: wfcommons.wfgen.abstract_recipe.WorkflowRecipe*, *logger: Optional[logging.Logger] = None*)

Bases: object

A generator of synthetic workflow instances based on workflow recipes obtained from the analysis of real workflow execution instances.

Parameters

- **workflow_recipe** (*WorkflowRecipe*) – The workflow recipe to be used for this generator.
- **logger** (*Logger*) – The logger where to log information/warning or errors (optional).

build_workflow(*workflow_name: Optional[str] = None*) → *wfcommons.common.workflow.Workflow*

Generate a synthetic workflow instance based on the workflow recipe used to instantiate the generator.

Parameters **workflow_name** (*str*) – The workflow name.

Returns A synthetic workflow instance object.

Return type *Workflow*

build_workflows(*num_workflows: int*) → List[*wfcommons.common.workflow.Workflow*]

Generate a number of synthetic workflow instances based on the workflow recipe used to instantiate the generator.

Parameters **num_workflows** (*int*) – The number of workflows to be generated.

Returns A list of synthetic workflow instance objects.

Return type List[*Workflow*]

7.4 wfcommons.wfinstances

7.4.1 wfcommons.wfinstances.instance

class wfcommons.wfinstances.instance.**Instance**(*input_instance: str*, *schema_file: Optional[str] = None*, *logger: Optional[logging.Logger] = None*)

Bases: object

Representation of one execution of one workflow on a set of machines

```
Instance(input_instance = 'instance.json')
```

Parameters

- **input_instance** (*str*) – The JSON instance.
- **schema_file** (*str*) – The path to the JSON schema that defines the instance. If no schema file is provided, it will look for a local copy of the WfCommons schema, and if not available it will fetch the latest schema from the WfCommons schema GitHub repository.
- **logger** (*Logger*) – The logger where to log information/warning or errors.

draw(*output: Optional[str] = None*, *extension: str = 'pdf'*) → None

Produce an image or a pdf file representing the instance.

Parameters

- **output** (*str*) – Name of the output file.
- **extension** – Type of the file extension (pdf, png, or svg).

leaves() → List[str]

Get the leaves of the workflow (i.e., the tasks without any successors).

Returns List of leaves

Return type List[str]

roots() → List[str]

Get the roots of the workflow (i.e., the tasks without any predecessors).

Returns List of roots

Return type List[str]

write_dot(*output: Optional[str] = None*) → None

Write a dot file of the instance.

Parameters **output** (*str*) – The output dot file name (optional).

7.4.2 wfcommons.wfinstances.instance_analyzer

class wfcommons.wfinstances.instance_analyzer.**InstanceAnalyzer**(*logger: Optional[logging.Logger] = None*)

Bases: object

Set of tools for analyzing collections of instances.

Parameters **logger** (*Logger*) – The logger where to log information/warning or errors (optional).

append_instance(*instance:* wfcommons.wfinstances.instance.Instance) → None

Append a workflow instance object to the instance analyzer.

```
instance = Instance(input_instance = 'instance.json', schema = 'schema.json')
instance_analyzer = InstanceAnalyzer()
instance_analyzer.append_instance(instance)
```

Parameters **instance** (Instance) – A workflow instance object.

build_summary(*tasks_list: List[str]*, *include_raw_data: Optional[bool] = True*) → Dict[str, Dict[str, Any]]

Analyzes appended instances and produce a summary of the analysis per task prefix.

```
workflow_tasks = ['sG1IterDecon', 'wrapper_siftSTFByMisfit']
instances_summary = instance_analyzer.build_summary(workflow_tasks, include_raw_
↪data=False)
```

Parameters

- **tasks_list** (*List[str]*) – List of workflow tasks prefix (e.g., mProject, sol2sanger, add_replace)
- **include_raw_data** (*bool*) – Whether to include the raw data in the instance summary.

Returns A summary of the analysis of instances in the form of a dictionary in which keys are task prefixes.

Return type Dict[str, Dict[str, Any]]

generate_all_fit_plots(*outfile_prefix: Optional[str] = None*) → None

Produce fit plots as images for each entry of the summary analysis. For entries in which there are no distribution (i.e., constant value), no plot will be generated.

Parameters **outfile_prefix** (*str*) – Prefix to be attached to each generated plot file name (optional).

generate_fit_plots(*instance_element:* wfcommons.wfinstances.instance_analyzer.InstanceElement, *outfile_prefix: Optional[str] = None*) → None

Produce fit plots as images for each entry of an instance element generated by the summary analysis. For entries in which there are no distribution (i.e., constant value), no plot will be generated.

Parameters

- **instance_element** (InstanceElement) – Workflow element for which the fit plots will be generated.
- **outfile_prefix** (*str*) – Prefix to be attached to each generated plot file name (optional).

class wfcommons.wfinstances.instance_analyzer.**InstanceElement**(*value*)

Bases: *wfcommons.utils.NoValue*

An enumeration.

INPUT = ('input', 'Input File Size (bytes)')

OUTPUT = ('output', 'Input File Size (bytes)')

RUNTIME = ('runtime', 'Runtime (s)')

7.4.3 wfcommons.wfinstances.logs.makeflow

class wfcommons.wfinstances.logs.makeflow.**MakeflowLogsParser**(*execution_dir: str*, *resource_monitor_logs_dir: str*, *description: Optional[str] = None*, *logger: Optional[logging.Logger] = None*)

Bases: wfcommons.wfinstances.logs.abstract_logs_parser.LogsParser

Parse Makeflow submit directory to generate workflow instance.

Parameters

- **execution_dir** (*str*) – Makeflow workflow execution directory (contains .mf and .makeflowlog files).
- **resource_monitor_logs_dir** (*str*) – Resource Monitor log files directory.
- **description** (*str*) – Workflow instance description.
- **logger** (*Logger*) – The logger where to log information/warning or errors (optional).

build_workflow(*workflow_name: Optional[str] = None*) → *wfcommons.common.workflow.Workflow*

Create workflow instance based on the workflow execution logs.

Parameters **workflow_name** (*str*) – The workflow name.

Returns A workflow instance object.

Return type *Workflow*

7.4.4 wfcommons.wfinstances.logs.pegasus

class wfcommons.wfinstances.logs.pegasus.**PegasusLogsParser**(*submit_dir: str*, *description: Optional[str] = None*, *ignore_auxiliary: Optional[bool] = True*, *legacy: Optional[bool] = False*, *logger: Optional[logging.Logger] = None*)

Bases: wfcommons.wfinstances.logs.abstract_logs_parser.LogsParser

Parse Pegasus submit directory to generate workflow instance.

Parameters

- **submit_dir** (*str*) – Pegasus submit directory.
- **legacy** (*bool*) – Whether the submit directory is from a Pegasus 4.x version.
- **description** (*str*) – Workflow instance description.
- **ignore_auxiliary** (*bool*) – Ignore auxiliary jobs.
- **logger** (*Logger*) – The logger where to log information/warning or errors (optional).

build_workflow(*workflow_name: Optional[str] = None*) → *wfcommons.common.workflow.Workflow*
Create workflow instance based on the workflow execution logs.

Parameters **workflow_name** (*str*) – The workflow name.

Returns A workflow instance object.

Return type *Workflow*

CHAPTER

EIGHT

DEVELOPER API REFERENCE

The developer API reference targets developers and researchers who want to contribute to the WfCommons project by, for example, developing novel techniques for instance analysis, developing new *workflow recipes*, etc. The developer API reference documentation includes detailed information for interacting with all classes and methods that compose this Python package.

8.1 wfcommons.utils

class wfcommons.utils.**NoValue**(*value*)

Bases: enum.Enum

An enumeration.

wfcommons.utils.**best_fit_distribution**(*data: List[float]*, *logger: Optional[logging.Logger] = None*) → Tuple

Fit a list of values to a distribution.

Parameters

- **data** (*List[float]*) – List of values to be fitted to a distribution.
- **logger** (*Logger*) – The logger uses to output debug information.

Returns The name of the distribution and its parameters.

Return type Tuple

wfcommons.utils.**generate_rvs**(*distribution: Dict*, *min_value: float*, *max_value: float*) → float

Generate a random variable from a distribution.

Parameters

- **distribution** (*Dict*) – Distribution dictionary (name and parameters).
- **min_value** (*float*) – Minimum value accepted as a random variable.
- **max_value** (*float*) – Maximum value accepted as a random variable.

Returns Random variable generated from a distribution.

Return type float

wfcommons.utils.**ncr**(*n: int*, *r: int*) → int

Calculate the number of combinations.

Parameters

- **n** (*int*) – The number of items.

- **r** (*int*) – The number of items being chosen at a time.

Returns The number of combinations.

Return type int

wfcommons.utils.**read_json**(*instance_filename: str*) → Dict[str, Any]
Read the JSON from the file path.

Parameters **instance_filename** (*str*) – The absolute path of the instance file.

Returns The json object loaded with json data from the file

Return type Dict[str, Any]

8.2 wfcommons.wfchef

8.2.1 wfcommons.wfchef.chef

wfcommons.wfchef.chef.**analyzer_summary**(*path_to_instances: pathlib.Path*) → Dict
Creates a dataframe with the Root Mean Square Error of the synthetic instances created based on the correspondent, w.r.t. number of tasks, real-world samples available at WfCommons WfInstances from Pegasus WMS GitHub <https://github.com/wfcommons/pegasus-instances> and from Makeflow WMS GitHub repositories.

Parameters

- **workflow** (*str or pathlib.Path*) – name (for samples available in WfCommons) or path to the real workflow instances.
- **err_savepath** – path to save the err (rmse) of all instances available into a csv.
- **always_update** – flag to set if the err needs to be updated or not (True: if new instances are added, False: otherwise).
- **runs** – number of times to repeat the err calculation process (due to randomization).

:type runs:bool

Returns dataframe with RMSE of all available instances.

Return type pd.DataFrame

wfcommons.wfchef.chef.**compare_rmse**(*synth_graph: networkx.classes.digraph.DiGraph*, *real_graph: networkx.classes.digraph.DiGraph*)
Calculates the Root Mean Square Error of a synthetic instance created based on the correspondent (in number of tasks) real-world sample.

Parameters

- **synth_graph** (*networkX DiGraph*) – a synthetic instance created by WfCommons.
- **real_graph** (*networkX DiGraph*) – the correspondent (in number of tasks) real-world workflow instance.

Returns The RMSE between the synthetic instance and the real instance.

Return type float

wfcommons.wfchef.chef.**create_recipe**(*path_to_instances: Union[str, pathlib.Path]*, *savedir: pathlib.Path*, *wf_name: str*, *cutoff: int = 4000*, *verbose: bool = False*, *runs: int = 1*) → wfcommons.wfgen.abstract_recipe.WorkflowRecipe

Creates a recipe for a workflow application by automatically replacing custom information from the recipe skeleton.

Parameters

- **path_to_instances** (*str or pathlib.Path*) – name (for samples available in WfCommons) or path to the real workflow instances.
- **savedir** (*pathlib.Path*) – path to save the recipe.
- **wf_name** (*str*) – name of the workflow apllication.
- **cutoff** (*bool*) – when set, only consider instances of smaller or equal sizes.
- **verbose** – when set, prints status messages.
- **verbose** – number of times to repeat the err calculation process (due to randomization).

:type runs:bool

wfcommons.wfchef.chef.**find_err**(*workflow: Union[str, pathlib.Path]*, *err_savepath: Optional[Union[pathlib.Path, str]] = None*, *always_update: bool = False*, *runs: int = 1*) → pandas.core.frame.DataFrame

Creates a dataframe with the Root Mean Square Error of the synthetic instances created based on the correspondent, w.r.t. number of tasks, real-world samples available at WfCommons WfInstances from Pegasus WMS GitHub <https://github.com/wfcommons/pegasus-instances> and from Makeflow WMS GitHub repositories.

Parameters

- **workflow** (*str or pathlib.Path*) – name (for samples available in WfCommons) or path to the real workflow instances.
- **err_savepath** – path to save the err (rmse) of all instances available into a csv.
- **always_update** – flag to set if the err needs to be updated or not (True: if new instances are added, False: otherwise).
- **runs** (*bool*) – number of times to repeat the err calculation process (due to randomization).

Returns dataframe with RMSE of all available instances.

Return type pd.DataFrame

wfcommons.wfchef.chef.**get_parser**() → argparse.ArgumentParser

wfcommons.wfchef.chef.**ls_recipe**()

Inspired by UNIX *ls* command, it lists the recipes already installed into the system and how to import it to use.

wfcommons.wfchef.chef.**main**()

wfcommons.wfchef.chef.**uninstall_recipe**(*module_name: str*)

Uninstalls a recipe installed in the system.

8.2.2 wfcommons.wfchef.duplicate

exception wfcommons.wfchef.duplicate.**NoMicrostructuresError**
Bases: Exception

wfcommons.wfchef.duplicate.**duplicate**(*path: pathlib.Path*, *base: Union[str, pathlib.Path]*, *num_nodes: int*) → networkx.classes.digraph.DiGraph
Attaches replicated nodes to base graph.

Parameters

- **path** (*pathlib.Path.*) – path to the summary JSON file.
- **base** (*str or pathlib.Path.*) – name (for samples available in WfCommons) or path to the specific graph to be used as base (if not set WfChef chooses the best fitting one).
- **num_nodes** (*int.*) – total amount of nodes desired in the synthetic instance.

Returns graph with the desired number of tasks.

Return type networkX DiGraph.

wfcommons.wfchef.duplicate.**duplicate_nodes**(*graph: networkx.classes.digraph.DiGraph*, *nodes: Set[str]*) → Dict
Replicates nodes of a graph.

Parameters

- **graph** (*networkX DiGraph*) – graph used to replicate and attach new nodes.
- **nodes** (*Set[str]*) – nodes to be replicated.

Returns the new nodes replicated.

Return type Dict[str]

8.2.3 wfcommons.wfchef.find_microstructures

exception wfcommons.wfchef.find_microstructures.**ImbalancedMicrostructureError**
Bases: Exception

wfcommons.wfchef.find_microstructures.**comb**(*n: int*, *k: int*) → int
Calculates the combination of two integers.

Parameters

- **n** (*int.*) – number.
- **k** (*int.*) – number.

Returns combination of two integers.

Return type int.

wfcommons.wfchef.find_microstructures.**find_microstructure**(*graph: networkx.classes.digraph.DiGraph*, *n1: str*, *n2: str*)
Detects a pattern (microstructure).

Parameters

- **graph** (*netwrokX DiGraph.*) – graph.
- **n1** – a node in graph.

- **n1** – a different node in graph.

Returns sets of n1 related nodes, n2 related nodes, the nodes in common between n1 and n2 and all the nodes involved in the process.

Return type Set[str], Set[str], Set[str], Set[str]

wfcommons.wfchef.find_microstructures.**find_microstructures**(*graph: networkx.classes.digraph.DiGraph*, *verbose: bool = False*)

Detects the patterns (microstructures) that are used for replication and graph expansion.

Parameters

- **graph** (*netwrokX DiGraph.*) – graph.
- **verbose** (*netwrokX DiGraph.*) – if set, prints status messages.

Returns patterns (microstructures)

Return type Set[str]

wfcommons.wfchef.find_microstructures.**get_children**(*graph: networkx.classes.digraph.DiGraph*, *node: str*) → List[str]

Gets the children of a node.

Parameters

- **graph** (*netwrokX DiGraph.*) – graph that contains the node.
- **node** (*str.*) – a node.

Returns list of the node's children.

Return type List[str]

wfcommons.wfchef.find_microstructures.**get_parents**(*graph: networkx.classes.digraph.DiGraph*, *node: str*) → List[str]

Gets the parents of a node.

Parameters

- **graph** (*netwrokX DiGraph.*) – graph that contains the node.
- **node** (*str.*) – a node.

Returns list of the node's parents.

Return type List[str]

wfcommons.wfchef.find_microstructures.**get_relatives**(*graph: networkx.classes.digraph.DiGraph*, *node: str*) → Set[str]

Gets all node's relatives (children and parents).

Parameters

- **graph** (*netwrokX DiGraph.*) – graph that contains the node.
- **node** (*str.*) – a node.

Returns set of node's relative.

Return type Set[str]

wfcommons.wfchef.find_microstructures.**save_microstructures**(*workflow_path: pathlib.Path*, *savedir: pathlib.Path*, *verbose: bool = False*, *img_type: Optional[str] = 'png'*, *cutoff: int = 4000*, *highlight_all_instances: bool = False*) → List[networkx.classes.digraph.DiGraph]

wfcommons.wfchef.find_microstructures.**sort_graphs**(*workflow_path: pathlib.Path*, *verbose: bool = False*) → List[networkx.classes.digraph.DiGraph]

Sort graphs in crescent order of number of tasks.

Parameters

- **workflow_path** (*pathlib.Path.*) – path to the JSON instances.
- **verbose** (*netwrokX DiGraph.*) – if set, prints status messages.

Returns sorted graphs

Return type List[networkX.DiGraph]

8.2.4 wfcommons.wfchef.utils

wfcommons.wfchef.utils.**annotate**(*g: networkx.classes.digraph.DiGraph*) → None

Annotates a networkX DiGraph with metadata such as the tasks top-down type hash, bottom-up type hash, and type-hash.

Parameters **path** (*str or pathlib.Path.*) – name (for samples available in WfCommons) or the path to graphs JSON.

Returns annotated graph.

Return type networkX DiGraph.

wfcommons.wfchef.utils.**combine_hashes**(**hashes: str*) → str

wfcommons.wfchef.utils.**create_graph**(*path: Union[str, pathlib.Path]*) → networkx.classes.digraph.DiGraph

Creates a networkX DiGraph from a JSON file in the WfFormat.

Parameters **path** (*str or pathlib.Path.*) – name (for samples available in WfCommons) or the path to graphs JSON.

Returns graph.

Return type networkX DiGraph.

wfcommons.wfchef.utils.**draw**(*g: networkx.classes.digraph.DiGraph*, *extension: Optional[str] = 'png'*, *with_labels: bool = False*, *ax: Optional[matplotlib.axes._axes.Axes] = None*, *show: bool = False*, *save: Optional[Union[pathlib.Path, str]] = None*, *close: bool = False*, *legend: bool = False*, *node_size: int = 1000*, *linewidths: int = 5*, *subgraph: Set[str] = {}*) → Tuple[matplotlib.figure.Figure, matplotlib.axes._axes.Axes]

Plots a netwrokX DiGraph.

Parameters

- **g** (*networkX DiGraph.*) – graph to be plotted.
- **extension** (*extension of the output file.*) – str.
- **with_labels** (*bool.*) – if set, it prints the task types over their nodes.
- **ax** (*plt.Axes.*) – plot axes.

- **show** (*bool.*) – if set, displays the plot on screen.
- **save** (*pathlib.Path.*) – path to directory to save the plot.
- **close** (*bool.*) – if set, automatically closes window that displays plot.
- **legend** (*bool.*) – if set, displays legend of the plot.
- **node_size** (*int.*) – size of the nodes (circles) in the plot.
- **linewidths** (*int.*) – thickness of the edges in the plot.
- **subgraph** (*Set[str]*) – nodes that were added by replication and will be colored green.

Returns the figure and the axis used.

Return type Tuple[plt.Figure, plt.Axes]

wfcommons.wfchef.utils.**string_hash**(*obj: Hashable*) → str

wfcommons.wfchef.utils.**type_hash**(*_type: str*, *parent_types: Iterable[str]*) → str

8.2.5 wfcommons.wfchef.skeletons.recipe

class wfcommons.wfchef.skeletons.recipe.**SkeletonRecipe**(*data_footprint: Optional[int] = 0*, *num_tasks: Optional[int] = 3*, *exclude_graphs: Set[str] = {}*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*, *logger: Optional[logging.Logger] = None*, ***kwargs*)

Bases: wfcommons.wfgen.abstract_recipe.WorkflowRecipe

A Skeleton workflow recipe class for creating synthetic workflow instances.

Parameters

- **data_footprint** (*int*) – The upper bound for the workflow total data footprint (in bytes).
- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow.
- **exclude_graphs** (*Set*) –
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.
- **logger** (*Logger*) – The logger where to log information/warning or errors (optional).

_abc_impl = <_abc_data object>

_workflow_recipe() → Dict

Recipe for generating synthetic instances of the Skeleton workflow. Recipes can be generated by using the *InstanceAnalyzer*.

Returns A recipe in the form of a dictionary in which keys are task prefixes.

Return type Dict[str, Any]

classmethod from_num_tasks(*num_tasks: int*, *exclude_graphs: Set[str] = {}*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*) → wfcommons.wfchef.skeletons.recipe.SkeletonRecipe

Instantiate a Skeleton workflow recipe that will generate synthetic workflows up to the total number of tasks provided.

Parameters

- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow (at least 3).
- **exclude_graphs** (*Set*) –
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns A Skeleton workflow recipe object that will generate synthetic workflows up to the total number of tasks provided.

Return type SkeletonRecipe

8.3 wfcommons.wfgen

8.3.1 wfcommons.wfgen.generator

class wfcommons.wfgen.generator.**WorkflowGenerator**(*workflow_recipe: wfcommons.wfgen.abstract_recipe.WorkflowRecipe*, *logger: Optional[logging.Logger] = None*)

Bases: object

A generator of synthetic workflow instances based on workflow recipes obtained from the analysis of real workflow execution instances.

Parameters

- **workflow_recipe** (*WorkflowRecipe*) – The workflow recipe to be used for this generator.
- **logger** (*Logger*) – The logger where to log information/warning or errors (optional).

build_workflow(*workflow_name: Optional[str] = None*) → *wfcommons.common.workflow.Workflow*

Generate a synthetic workflow instance based on the workflow recipe used to instantiate the generator.

Parameters **workflow_name** (*str*) – The workflow name.

Returns A synthetic workflow instance object.

Return type *Workflow*

build_workflows(*num_workflows: int*) → List[*wfcommons.common.workflow.Workflow*]

Generate a number of synthetic workflow instances based on the workflow recipe used to instantiate the generator.

Parameters **num_workflows** (*int*) – The number of workflows to be generated.

Returns A list of synthetic workflow instance objects.

Return type List[*Workflow*]

8.3.2 wfcommons.wfgen.abstract_recipe

class wfcommons.wfgen.abstract_recipe.**WorkflowRecipe**(*name: str*, *data_footprint: Optional[int]*, *num_tasks: Optional[int]*, *exclude_graphs: Set[str] = {}*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*, *logger: Optional[logging.Logger] = None*, *this_dir=None*)

Bases: abc.ABC

An abstract class of workflow recipes for creating synthetic workflow instances.

Parameters

- **name** (*str*) – The workflow recipe name.
- **data_footprint** (*int*) – The upper bound for the workflow total data footprint (in bytes).
- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow.
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.
- **logger** (*Logger*) – The logger where to log information/warning or errors (optional).

_abc_impl = <_abc_data object>

_generate_file(*extension: str*, *recipe: Dict[str, Any]*, *link:* wfcommons.common.file.FileLink) → *wfcommons.common.file.File*

Generate a file according to a file recipe.

Parameters

- **extension** (*str*) –
- **recipe** (*Dict[str, Any]*) – Recipe for generating the file.
- **link** (FileLink) – Type of file link.

Returns The generated file.

Return type *File*

_generate_files(*task_id: str*, *recipe: Dict[str, Any]*, *link:* wfcommons.common.file.FileLink, *files_recipe: Optional[Dict[*wfcommons.common.file.FileLink*, Dict[str, int]]] = None*) → None

Generate files for a specific task ID.

Parameters

- **task_id** (*str*) – task ID.
- **recipe** (*Dict[str, Any]*) – Recipe for generating the task.
- **link** (FileLink) – Type of file link.

- **files_recipe** (*Dict[FileLink, Dict[str, int]]*) – Recipe for generating task files.

_generate_task(*task_name: str*, *task_id: str*, *input_files: Optional[List[wfcommons.common.file.File]] = None*, *files_recipe: Optional[Dict[wfcommons.common.file.FileLink, Dict[str, int]]] = None*) → *wfcommons.common.task.Task*

Generate a synthetic task.

Parameters

- **task_name** (*str*) – task name.
- **task_id** (*str*) – task ID.
- **input_files** (*List[File]*) – List of input files to be included.
- **files_recipe** (*Dict[FileLink, Dict[str, int]]*) – Recipe for generating task files.

Returns A task object.

Return type task

_generate_task_name(*prefix: str*) → str

Generate a task name from a prefix appended with an ID.

Parameters **prefix** (*str*) – task prefix.

Returns task name from prefix appended with an ID.

Return type str

_get_files_by_task_and_link(*task_id: str*, *link:* wfcommons.common.file.FileLink) → List[*wfcommons.common.file.File*]

Get the list of files for a task ID and link type.

Parameters

- **task_id** (*str*) – task ID.
- **link** (*FileLink*) – Type of file link.

Returns List of files for a task ID and link type.

Return type List[*File*]

_load_base_graph() → networkx.classes.digraph.DiGraph

_load_microstructures() → Dict

abstract _workflow_recipe() → Dict[str, Any]

Recipe for generating synthetic instances for a workflow. Recipes can be generated by using the *InstanceAnalyzer*.

Returns A recipe in the form of a dictionary in which keys are task prefixes.

Return type Dict[str, Any]

build_workflow(*workflow_name: Optional[str] = None*) → *wfcommons.common.workflow.Workflow*

Generate a synthetic workflow instance.

Parameters **workflow_name** (*int*) – The workflow name

Returns A synthetic workflow instance object.

Return type *Workflow*

abstract classmethod from_num_tasks(*num_tasks: int*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*) → wfcommons.wfgen.abstract_recipe.WorkflowRecipe

Instantiate a workflow recipe that will generate synthetic workflows up to the total number of tasks provided.

Parameters

- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow.
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns A workflow recipe object that will generate synthetic workflows up to the total number of tasks provided.

Return type WorkflowRecipe

generate_nx_graph() → networkx.classes.digraph.DiGraph

8.4 wfcommons.wfinstances

8.4.1 wfcommons.wfinstances.schema

class wfcommons.wfinstances.schema.**SchemaValidator**(*schema_file: Optional[str] = None*, *logger: Optional[logging.Logger] = None*)

Bases: object

Validate JSON files against WfCommons schema. If schema file path is not provided, it will look for a local copy of the WfCommons schema, and if not available it will fetch the latest schema from the WfCommons schema GitHub repository.

Parameters

- **schema_file** (*str*) – JSON schema file path.
- **logger** (*Logger*) – The logger where to log information/warning or errors.

_load_schema(*schema_file: Optional[str] = None*)

Load the schema file. If schema file path is not provided, it will look for a local copy of the WfCommons schema, and if not available it will fetch the latest schema from the GitHub repository.

Parameters **schema_file** (*str*) – JSON schema file path.

Returns The JSON schema.

Return type json

_semantic_validation(*data: Dict[str, Any]*)

Validate the semantics of the JSON workflow execution instance.

Parameters **data** (*Dict[str, Any]*) – Workflow instance in JSON format.

_syntax_validation(*data: Dict[str, Any]*)

Validate the JSON workflow execution instance against the schema.

Parameters **data** (*Dict[str, Any]*) – Workflow instance in JSON format.

validate_instance(*data: Dict[str, Any]*)
Perform syntax validation against the schema, and semantic validation.

Parameters **data** (*Dict[str, Any]*) – Workflow instance in JSON format.

8.4.2 wfcommons.wfinstances.instance

class wfcommons.wfinstances.instance.**Instance**(*input_instance: str*, *schema_file: Optional[str] = None*, *logger: Optional[logging.Logger] = None*)

Bases: object

Representation of one execution of one workflow on a set of machines

```
Instance(input_instance = 'instance.json')
```

Parameters

- **input_instance** (*str*) – The JSON instance.
- **schema_file** (*str*) – The path to the JSON schema that defines the instance. If no schema file is provided, it will look for a local copy of the WfCommons schema, and if not available it will fetch the latest schema from the WfCommons schema GitHub repository.
- **logger** (*Logger*) – The logger where to log information/warning or errors.

draw(*output: Optional[str] = None*, *extension: str = 'pdf'*) → None
Produce an image or a pdf file representing the instance.

Parameters

- **output** (*str*) – Name of the output file.
- **extension** – Type of the file extension (pdf, png, or svg).

leaves() → List[str]
Get the leaves of the workflow (i.e., the tasks without any successors).

Returns List of leaves

Return type List[str]

roots() → List[str]
Get the roots of the workflow (i.e., the tasks without any predecessors).

Returns List of roots

Return type List[str]

write_dot(*output: Optional[str] = None*) → None
Write a dot file of the instance.

Parameters **output** (*str*) – The output dot file name (optional).

8.4.3 wfcommons.wfinstances.instance_analyzer

class wfcommons.wfinstances.instance_analyzer.**InstanceAnalyzer**(*logger: Optional[logging.Logger] = None*)

Bases: object

Set of tools for analyzing collections of instances.

Parameters **logger** (*Logger*) – The logger where to log information/warning or errors (optional).

append_instance(*instance:* wfcommons.wfinstances.instance.Instance) → None

Append a workflow instance object to the instance analyzer.

```
instance = Instance(input_instance = 'instance.json', schema = 'schema.json')
instance_analyzer = InstanceAnalyzer()
instance_analyzer.append_instance(instance)
```

Parameters **instance** (Instance) – A workflow instance object.

build_summary(*tasks_list: List[str]*, *include_raw_data: Optional[bool] = True*) → Dict[str, Dict[str, Any]]

Analyzes appended instances and produce a summary of the analysis per task prefix.

```
workflow_tasks = ['sG1IterDecon', 'wrapper_siftSTFByMisfit']
instances_summary = instance_analyzer.build_summary(workflow_tasks, include_raw_
↪data=False)
```

Parameters

- **tasks_list** (*List[str]*) – List of workflow tasks prefix (e.g., mProject, sol2sanger, add_replace)
- **include_raw_data** (*bool*) – Whether to include the raw data in the instance summary.

Returns A summary of the analysis of instances in the form of a dictionary in which keys are task prefixes.

Return type Dict[str, Dict[str, Any]]

generate_all_fit_plots(*outfile_prefix: Optional[str] = None*) → None

Produce fit plots as images for each entry of the summary analysis. For entries in which there are no distribution (i.e., constant value), no plot will be generated.

Parameters **outfile_prefix** (*str*) – Prefix to be attached to each generated plot file name (optional).

generate_fit_plots(*instance_element:* wfcommons.wfinstances.instance_analyzer.InstanceElement, *outfile_prefix: Optional[str] = None*) → None

Produce fit plots as images for each entry of an instance element generated by the summary analysis. For entries in which there are no distribution (i.e., constant value), no plot will be generated.

Parameters

- **instance_element** (InstanceElement) – Workflow element for which the fit plots will be generated.
- **outfile_prefix** (*str*) – Prefix to be attached to each generated plot file name (optional).

class wfcommons.wfinstances.instance_analyzer.**InstanceElement**(*value*)

Bases: *wfcommons.utils.NoValue*

An enumeration.

INPUT = ('input', 'Input File Size (bytes)')

OUTPUT = ('output', 'Input File Size (bytes)')

RUNTIME = ('runtime', 'Runtime (s)')

wfcommons.wfinstances.instance_analyzer._append_file_to_dict(*extension: str*, *dict_obj: Dict[str, Any]*, *file_size: int*) → None

Add a file size to a file type extension dictionary.

Parameters

- **extension** (*str*) – File type extension.
- **dict_obj** (*Dict[str, Any]*) – Dictionary of file type extensions.
- **file_size** (*int*) – File size in bytes.

wfcommons.wfinstances.instance_analyzer._best_fit_distribution_for_file(*dict_obj*, *include_raw_data*) → None

Find the best fit distribution for a file.

Parameters

- **dict_obj** (*Dict[str, Any]*) – Dictionary of file type extensions.
- **include_raw_data** (*bool*) –

wfcommons.wfinstances.instance_analyzer._generate_fit_plots(*el: Dict*, *title: str*, *xlabel: str*, *outfile: str*, *font_size: Optional[int] = None*, *logger: Optional[logging.Logger] = None*) → None

Produce a fit plot as an image for an entry of an instance element generated by the summary analysis.

Parameters

- **el** (*Dict*) – Entry of an instance element generated by the summary analysis.
- **title** (*str*) – Plot title.
- **xlabel** (*str*) – X-axis label.
- **outfile** (*Optional[int]*) – Plot file name.
- **font_size** – Size of the font.
- **logger** (*Logger*) – The logger where to log information/warning or errors.

wfcommons.wfinstances.instance_analyzer._json_format_distribution_fit(*dist_tuple: Tuple*) → Dict[str, Any]

Format the best fit distribution data into a dictionary

Parameters **dist_tuple** (*Tuple*) – Tuple containing best fit distribution name and parameters.

Returns

Return type Dict[str, Any]

8.4.4 wfcommons.wfinstances.logs.abstract_logs_parser

class wfcommons.wfinstances.logs.abstract_logs_parser.**LogsParser**(*wms_name: str*, *wms_url: Optional[str] = None*, *description: Optional[str] = None*, *logger: Optional[logging.Logger] = None*)

Bases: abc.ABC

An abstract class of logs parser for creating workflow instances.

Parameters

- **wms_name** (*str*) – Name of the workflow system.
- **wms_url** (*str*) – URL for the workflow system.
- **description** (*str*) – Workflow instance description.
- **logger** (*Logger*) – The logger where to log information/warning or errors (optional).

_abc_impl = <_abc_data object>

abstract build_workflow(*workflow_name: Optional[str] = None*) → *wfcommons.common.workflow.Workflow*

Create workflow instance based on the workflow execution logs.

Parameters **workflow_name** (*str*) – The workflow name.

Returns A workflow instance object.

Return type *Workflow*

8.4.5 wfcommons.wfinstances.logs.makeflow

class wfcommons.wfinstances.logs.makeflow.**MakeflowLogsParser**(*execution_dir: str*, *resource_monitor_logs_dir: str*, *description: Optional[str] = None*, *logger: Optional[logging.Logger] = None*)

Bases: wfcommons.wfinstances.logs.abstract_logs_parser.LogsParser

Parse Makeflow submit directory to generate workflow instance.

Parameters

- **execution_dir** (*str*) – Makeflow workflow execution directory (contains .mf and .makeflowlog files).
- **resource_monitor_logs_dir** (*str*) – Resource Monitor log files directory.
- **description** (*str*) – Workflow instance description.
- **logger** (*Logger*) – The logger where to log information/warning or errors (optional).

_abc_impl = <_abc_data object>

_create_files(*files_list: List[str]*, *link:* wfcommons.common.file.FileLink, *task_name: str*)

Create a list of files objects.

Parameters

- **files_list** – list of file names.
- **link** – Link type for the files in the list.
- **task_name** – Task name.

Rtype files_list List[str]

Rtype link FileLink

Rtype task_name str

Returns List of file objects.

Return type List[*File*]

_parse_makeflow_log_file()
Parse the makeflow log file and update workflow task information.

_parse_resource_monitor_logs()
Parse the log files produced by resource monitor

_parse_workflow_file()
Parse the makeflow workflow file and build the workflow structure.

build_workflow(*workflow_name: Optional[str] = None*) → *wfcommons.common.workflow.Workflow*
Create workflow instance based on the workflow execution logs.

Parameters **workflow_name** (*str*) – The workflow name.

Returns A workflow instance object.

Return type *Workflow*

8.4.6 wfcommons.wfinstances.logs.pegasus

class wfcommons.wfinstances.logs.pegasus.**PegasusLogsParser**(*submit_dir: str*, *description: Optional[str] = None*, *ignore_auxiliary: Optional[bool] = True*, *legacy: Optional[bool] = False*, *logger: Optional[logging.Logger] = None*)

Bases: wfcommons.wfinstances.logs.abstract_logs_parser.LogsParser

Parse Pegasus submit directory to generate workflow instance.

Parameters

- **submit_dir** (*str*) – Pegasus submit directory.
- **legacy** (*bool*) – Whether the submit directory is from a Pegasus 4.x version.
- **description** (*str*) – Workflow instance description.
- **ignore_auxiliary** (*bool*) – Ignore auxiliary jobs.
- **logger** (*Logger*) – The logger where to log information/warning or errors (optional).

_abc_impl = <_abc_data object>

_fetch_all_files(*extension: str*, *file_name: str = ''*)
Fetch all files from the directory and its hierarchy

Parameters

- **extension** (*str*) – file extension to be searched for

- **file_name** (*str*) – file_name to be searched

Returns List of file names that match

Return type List[str]

_parse_braindump()
Parse the Pegasus braindump.txt file

_parse_dag()
Parse the DAG file.

_parse_dax()
Parse the DAX file.

_parse_job_output(*task*)
Parse the kickstart job output file (e.g., .out.000).

Parameters **task** (Task) – Task object.

_parse_job_output_latest(*task*, *output_file*)
Parse the kickstart job output file in YAML format (e.g., .out.000).

Parameters

- **task** (Task) – Task object.
- **output_file** (*str*) – Output file name.

_parse_job_output_legacy(*task*, *output_file*)
Parse the kickstart job output file in XML format (e.g., .out.000).

Parameters

- **task** (Task) – Task object.
- **output_file** (*str*) – Output file name.

_parse_meta_file(*task_name*)
Parse the Pegasus meta file (generated from pegasus-integrity)

Parameters **task_name** (*str*) – Task file name.

_parse_workflow()
Parse the Workflow file.

build_workflow(*workflow_name: Optional[str] = None*) → *wfcommons.common.workflow.Workflow*
Create workflow instance based on the workflow execution logs.

Parameters **workflow_name** (*str*) – The workflow name.

Returns A workflow instance object.

Return type *Workflow*

PYTHON MODULE INDEX

W

INDEX

Symbols

A

B

C

D

E

F

V

W