shadow: An algorithm reference and testing framework for scheduling data-intensive workflows

Ryan Bunney[‡§*], Andreas Wicenec[‡§], Mark Reynolds[§]

Abstract—As the scale of science projects increase, so does the demand on computing infrastructures. Data-intensive workflows and science pipelines that are being developed for new, exa-scale projects (e.g. the SKA) necessitate the development of new individual applications, as well as techniques for the processing of entire processing pipelines. Workflow scheduling algorithms are continually being developed and optimised for different use cases, environments, and objective functions; however, in the main systems that are used to deploy science workflows for major science projects, the same algorithms and heuristics are used. In order to develop and test new algorithms against the *de-facto* standards, we have developed *shadow*, a workflow-oriented scheduling algorithm framework. *shadow* has implementations of these *de-facto* scheduling heuristics, with the intention of continually updating the library with heuristics, metaheuristics, and mathematical optimisation approaches in the near future. In addition to the algorithm implementations, there is also a number of workflow and environment generation options, using the companion utility *shadowgen*; this has been provided to improve the productivity of algorithm developers in experimenting with their new algorithms over a large variety of workflows and computing environments. *shadowgen* also has a translation utilities that will convert from other formats, like the Pegasus DAX file, into the *shadow* JSON configuration. *shadow* is open-sourced and uses key SciPy libraries; the intention is for the library to become a reference implementation of scheduling algorithms, and provide algorithm designers an opportunity to test their developments in a within the framework. *shadow* code is available on GitHub at https://github.com/myxie/shadow; documentation for the project is available in the repository, as well as at https://shadowscheduling.readthedocs.org

Index Terms—workflows

Introduction

Researchers benefit as a community from having open implementations of algorithms, as it improves reproducibility and accuracy of benchmarking and algorithmic analysis [CHI14]. There exists a number of open-source frameworks designed for testing and benchmarking of algorithms, demonstrate typical implementations, and provide an infrastructure for the development and testing of new algorithms; examples include NLOPT for non-linear optimisation in a number of languages (C/C++, Python, Java) [Joh], NetworkX for graph and network implementations in Python, MOEA for Java, and DEAP for distributed EAs in Python [DRFG+12]. Science workflow scheduling is a field with varied contributions in algorithm development and optimisation, which address a number of different sub-problems within the field. [WWT15], [zot], [BÇRS13], [HDRD98], [RB16], [Bur]. Unfortunately, implementations of these contributions are difficult to find; for example, implementations of HEFT [THW02] can only be found in code that uses it - such as in simulation frameworks like WorkflowSim [CD12]; others are not implemented in any public way at all - such as highly cited NSGAII* [YB06] and PCP [ANE10]. These are also typically used as benchmarking or stepping stones for new algorithms; for example, HEFT continues to be used as the foundation for scheduling heuristics [DFP12], [CCCR18], meta-heuristics, and even mathematical optimisation procedures [BBL+16], despite being 20 years old. The lack of a consistent testing environment and implementation of algorithms makes it hard to reproduce and verify the results of published material, especially when a common workflow model cannot be verified. *shadow* aims to address this issue by providing a workflow-oriented algorithm library and testing environment, in which the performance of single- and multi-objective workflow scheduling algorithms may be compared to implementations of de-facto algorithms. To the best of our knowledge, there is no single-source repository of implementations for DAG or Workflow scheduling algorithms. The emphasis in shadow is on reproducible and accuracy in algorithm performance analysis, rather than a simulated demonstration of the application of a particular algorithm in certain environments. Additionally, with the popularity of Python in other domains that are also growing within the workflow community - such as Machine and Deep Learning - *shadow* provides a frictionless opportunity to integrate with the frameworks and libraries commonly used in those research

Workflow Scheduling

The workflow scheduling problem is a known NP-Hard problem [CG72], [KA99], [KBPR17], [Pin12].It is commonly represented in the literature as a DirectedAcyclic Graph (DAG) [CK88], [CA93], [Ull75], [KA99];a sequence of tasks will have precedence constraints that limit when a task may start. A DAG task-graph is represented formally as a graph $G = (V, E)$, where V is a set of v vertices and E is a set of e edges [KA99]. Vertices and Edges represent communication and computation costs respectively. The objective of the DAG-scheduling problem is to minimise the execution length of the final schedule; this is referred to as the *makespan*.

The complexity and size of data products from modern science projects necessitates dedicated infrastructure for compute, in a way that requires re-organisation of existing tasks and processes. As a result, it is often not enough to run a sequence of tasks in

* *Corresponding author: ryan.bunney@research.uwa.edu.au*
‡ *International Centre for Radio Astronomy Research*
§ *University of Western Australia*

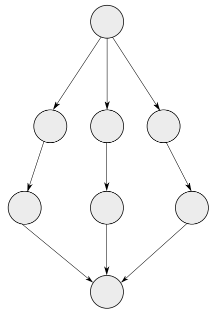


Fig. 1: A sample DAG; vertices represent compute tasks, and edges show precedence relationships between nodes. Vertex- and edge-weights are conventionally used to describe computational and data costs, respectively.

series, or submit them to batch processing; this would likely be computationally inefficient, as well taking as much longer than necessary. As a result, science projects that have computationally- and data-intensive programs, that are interrelated, have adopted the DAG-scheduling model for representing their compute pipelines.

Existing approaches

It should be noted that existing work already addresses testing workflow scheduling algorithms in real-world environments; tools like SimGrid [CLQ], BatSim [DMPR17], GridSim [BM02], - and its extensions, CloudSim:cite:*calheiros2011* and WorkflowSim [CD12] - all feature strongly in the literature. These are excellent resources for determining the effectiveness of the implementations at the application level; however, they do not provide a standardised repository of existing algorithms, or a template workflow model that can be used to ensure consistency across performance testing. Current implementations of workflow scheduling algorithms may be found in a number of different environments; for example, HEFT and dynamic-HEFT implementations exist in WorkflowSim[1] - but one must traverse large repositories in order to reach them. There are also a number of implementations that are present on open-source repositories such as Git Hub, but these are not always official releases from papers, and it is difficult to keep track of multiple implementations to ensure quality and consistency.

Kwok and Ahmed have discussed ways of benchmarking algorithms; dditionally, :cite:'maurya2018' present a discussion on what a potential framework for scheduling algorithms would look like.

- Environment provider
- DAG generator
- Scheduler
- Analyzer

shadow also provides a visualization interface

Design and Core Architecture

Design

shadow adopts a workflow-oriented design approach, where workflows are at the centre of all decisions made within the framework; environments are assigned to workflows, algorithms operate on workflows, and the main object that is manipulated and interacted with when developing an algorithm is likely to be a workflow object.

By adopting a workflow-oriented model to developing algorithms to test, three important outcomes are acheived:

- Freedom of implementation; for users wishing to develop their own algorithms, there is no prohibition of additional libraries or data-structures, provided the workflow structure is used within the algorithm.
- Focus on the workflow and reproducibility; when running analysis and benchmarking experiments, the same workflow model is used by all algorithms, which ensures comparisons between differing approaches (e.g. a single-objective model such as HEFT vs. a dynamic implementation of a multi-objective heuristic model) are applied to the same workflow.
- Examples: We have implemented a number of popular and well-documented algorithms that are commonly used to benchmark new algorithms and approaches. There is no need to follow the approaches taken by these implementations, but they provide a useful starting point for those interested in developing their own.

Additioally, by using the `NetworkX.DiGraph` as the storage object for the workflow structure, users may extend the shadow workflow object in any way as they would a `NetworkX` object.

shadow is not intended to accurately simulated the execution of a workflow in an environment; for example, working with delays in processing, or node failure in a cluster. Strategies to mitigate these are often implemented secondary to the scheduling algorithms - especially in the case of static scheduling - and would not be a fair approach to benchmarking the relative performance between each application. Instead, it provides algorithms that may be used - statically or dynamically - in a way that may used within a larger simulation environment, where one would be able to compare the specific environmental performance of one algorithm over another. Integrating shadow into a larger simulation has been done by the authors of this paper, and the framework has been designed with the intention that this will continue to be a supported feature.

Architecture

Shadow is split into three main components that are separated by their intended use case, whether it be designing new algorithms, or to benchmark against the existing implementations. These components are:

- `models`
- `algorithms`
- `visualiser`

Models

The `models` module is likely the main entry point for researchers or developers of algorithms; in it are a number of key components of the shadow library, the uses of which are demonstrated both in the `examples` directory, as well as the implemented sample algorithms in the `algorithms` module. The `algorithms` module is concerned with the implementations of algorithms mentioned previously; the intention is to provide both intended implementations of the shadow `models`, as well as benchmark implementations for those designing additions. The visualiser is a useful way to add graphical components to a benchmarking recipe,

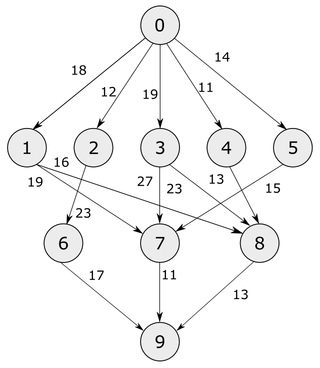


Fig. 2: *An example workflow DAG adapted from; weights on the edges describe data products from the respective parent node being sent to the child. In* shadow*, task computation cost is represented by the total number of Floating Point OPerations required to run the task (see Table). This is intended to alleviate the difficulty of converting runtimes between different test environment configurations.*

Workflow and Costs		**Environment**	
Task	*FLOPs*	*Machine*	*FLOP/s*
0	119000	`cat0_m0`	7000
1	92000	`cat1_m1`	6000
2	95000	`cat2_m2`	11000
3	109000		
4	107000		
5	169000		
6	110000		
7	147000		
8	146000		
9	101000		

TABLE 1: *This is the caption for the materials table.*

or can be invoked using the command line interface to quickly run one of the in-built algorithms.

These components are all contained within the main `shadow` directory; there are also additional codes that are located in utils, which will be covered in more detail in Section [Insert Section].

`models` provides the `Workflow` class, the foundational data structure of shadow. Currently, a `Workflow` object is initialised using a JSON configuration file that represents the underlying DAG structure of the workflow, along with storing different attributes for task-nodes and edges 2.

These attributes are implicitly defined within the configuration file; for example, if the task graph has computate demand (as total number of FLOPs/task) but not memory demand (as average GB/task), then the Workflow object is initialised without memory, requiring no additional input from the developer. The following example is based on the original graph presented in the HEFT algorithm, and demonstrates the configuration file and how it is initialised:

```python
from shadow.models.workflow import Workflow
HEFTWorkflow = Workflow('heft.json')
```

The `heft.json` file contains the graph structure, based the JSON dump received when using networks. Nodes and their respective costs (computation, memory, monetary etc.) are stored with their IDs

```json
...
"nodes": [
    {
        "comp": 119000,
        "id": 0
    },
    {
        "comp": 92000,
        "id": 1
    },
    {
        "comp": 95000,
        "id": 2
    },
    ...
],
```

Edges in the graph - the precedence relationship between tasks - are escribed by ‘liks’, along with the related data-products:

```json
"links": [
    {
        "data_size": 18,
        "source": 0,
        "target": 1
    },
    {
        "data_size": 12,
        "source": 0,
        "target": 2
    },
    ...
```

NetworkX is used to form the base-graph structure for the workflow; it allows the user to specify nodes as Python objects, so tasks are stored using the shadow `Task` object structure.

In addition to the JSON configuration for the workflow DAG, a Workflow object also requires an `Environment` object. `Environment``s represent the compute platform on which the Workflow is executed; they are add to ``Workflow` objects in the event that different environments are being analysed. The environment is also specified in JSON; currently, there is no defined way to specify an environment programmatically, although it is possible to do so if using JSON is not an option.

```json
"system": {
  "resources": {
    "cat0_m0": {
      "flops": 7000.0
      "mem":
      "io" :
    },
    "cat1_m1": {
      "flops": 6000.0
    },
    "cat2_m2": {
      "flops": 11000.0
    }
  },
  "rates": {
    "cat0": 1.0, # GB/s
    "cat1": 1.0,
    "cat2": 1.0
  }
}
```

Environments are added to the `Workflow` object in the following manner:

```python
from shadow.models.environment import Environment
env = Environment('sys.json')
HEFTWorkflow.add_environment(env)
```

It is also possible to use pre-calculated costs (i.e. completion time in seconds) when scheduling with shadow. This approach is less flexible for scheduling workflows, but is a common approach used in the scheduling algorithm literature. An example of this is shown in Figure, which is a replication of the costs provided in the original HEFT paper [THW02]. This can be achieved by adding a list of costs per tasks to the workflow specification JSON file, in addition to the following header:

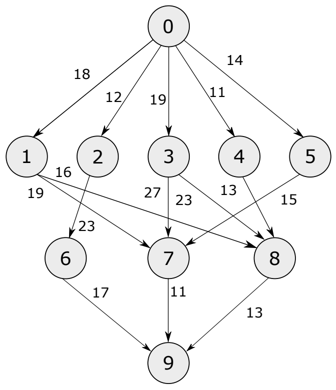
```
{
    "header" : {
    "time": true
    },
    ...

    "nodes": [
    {
        "comp": [
            14,
            16,
            9
        ],
        "id": 0
    },
    ...
}
```

The Machine class is also defined in environment.py - this is a helper-class that makes developer access to specifications (provided compute, memory, bandwidth etc.) convenient and intuitive. The Workflow lass will clculate the runtime and other values based on it's current environment (this occurs when the environment is passed to the Workflow); however, users of the environment class may interact with these compute values if necessary. Configuration files may be generated in a number of ways, following a ariety of speifications, using the shadowgen utility (see 'Additional ools' sectio).

The final class that may be of interest to algorithm developers is the Solution class. For single-objective heuristics like HEFT or min-min, the final result is a single solution, which is a set of machine-task pairs. However, for population- and search-based metaheuristics, multiple solutions must be generated, and then evaluated, often for two or more (competing) objectives. These solutions also need to be sanity-checked in order to ensure that randomly generated task-machine pairs still follow the precedence constraints defined by the original workflow DAG. The Solution provides a basic object structure that stores machines and task pairs as a dictionary of Allocations; llocations store he task, and it's start and finish time on the machine. This provides an additional ease-of-use functionality for developers, who can interact with allocations using intuitive attributes (rather than navigating a dictionary of stored keywords). The Solution currently stores a single objective (makespan) but can be expanded to include other, algorithm-specific requirements. For example, NSGAII* ranks each generated solution using the non-dominated rank and crowding distance operator; as a result, the shadow implementation creates a class, NSGASolution, that inherits the basic Solution class and adds the these additional attributes. This reduces the complexity of the global solution class whilst providing the flexibility for designers to create more elaborate solutions (and algorithms).

Algorithms

These algorithms may be extended by others, or used when running comparisons and benchmarking. The examples directory gives you an overview of recipes that one can follow to use the algorithms to perform benchmarking.



	M1	M2	M3
0	14	16	9
1	12	19	18
2	11	13	19
3	13	8	17
4	12	13	10
5	13	16	9
6	7	15	11
7	5	11	14
8	18	12	20
9	21	7	16

The shadow approach to describing an algorithm presents the algorithm as a single entity (e.g. heft()), with an initialised workflow object passed as a function parameter. The typical structure of a shadow algorithm function is as follows:

- The main algorithm - that is, the function to which a Workflow well be passed - is titled using its publication name or title (e.g. HEFT, PCP, NSGAII* etc.). Following PEP8, this is (ideally) in lowercase.
- Within the main algorithm function, effort has been made to keep it structured in a similar way to the pseudo-code as presented in the respective paper. For example, HEFT has two main components to the algorithm; Upward Ranking of tasks in the workflow, and the Insertion Policy allocation scheme. This is presented in shadow as:

```
def heft(workflow):
  """
  Implementation of the original 1999 HEFT algorithm.

  :params wf: The workflow object to schedule
  :returns: The makespan of the resulting schedule
  """
  upward_rank(workflow)
  workflow.sort_tasks('rank')
  makespan = insertion_policy(workflow)
  return workflow.solution
```

Complete information of the final schedule is stored in the HEFTWorkflow.solution object, which provides additional information, such as task-machine allocation pairs. It is convention in shadow to have the algorithm return the Solution object attached to the workflow:

```
solution = heft(HEFTWorkflow)
```

In keeping with the generic requirements of DAG-based scheduling algorithms, the base Solution class prioritises makespan over other objectives; however, this may be amended (or even ignored) for other approaches. For example, the NSGAII algorithm uses a sub-class for this purpose, as it generates multiple solutions before ranking each solution using the crowded distance or non-dominated sort:cite:{nondomsort}:

Visualiser

shadow leverages matplotlib and numpy by wrapping certain styles of plots into easy-to-use classes that are structured around the Workflow and Solution classes. The Visualiser uses the Solution class to retrieve allocation data, and generates a plot based on that information. For example, below is the result of visualising the HEFTWorkflow example mentioned previously:

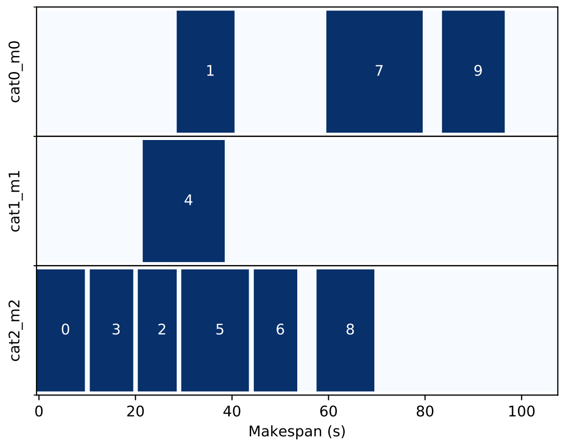


Fig. 3: *Result of running* `shadow.heuristic.heft` *on the graph shown in Figure 2. Final makespan is 98; gaps between tasks are indicative of data transfer times between parent and child tasks on different machines.*

This can be achieved by creating a script using the algorithms as described above, and then passing the scheduled workflow to one of the Visualiser classes:

```
from shadow.visualiser.visualiser import AllocationPlot

sample_allocation = AllocationPlot(
    solution=HEFTWorkflow.solution
)

sample_allocation.plot(
    save=True,
    figname='sample_allocation.pdf'
)
```

Additional tools

Command-line interface

shadow provides a simple command-line utility that can be

shadowgen

shadowgen is a utility built into the *shadow* framework to generate workflows that are reproducible and interrogable. It is built to generate a variety of workflows that have been documented and characterised in the literature in a way that augments current techniques, rather than replacing them entirely.

This includes the following:

- Python code that runs the GGen graph generator[2], which produces graphs in a variety of shapes and sizes based on provided parameters. This was originally designed to produce task graphs to test the performance of DAG scheduling algorithms
- DAX Translator: This takes the commonly used Directed Acyclic XML (DAX) file format, used to generate graphs for Pegasus, and translates them into the *shadow* format. Future work will also interface with the WorkflowGenerator code that is based on the work conduced in [BCD+08], which generates DAX graphs.
- DALiuGE/EAGLE Translator: EAGLE logical graphs must be unrolled into Physical Graph Templates (PGT) before they are in a DAG that can be scheduled in *shadow*. *shadowgen* will run the DALiUGE unroll code, and then convert this PGT into a *shadow*-based JSON workflow.

Existing approaches

A majority of work published in workflow scheduling will use workflows generated using the approach laid out in [BCD+08]. The five workflows described in the paper (Montage, CyberShake, Epigenomics, SIPHT and LIGO) had their task runtimes, memory and I/O rates profiled, from which they created a WorkflowGenerator tool[3]. This tool uses the distribution sizes for runtime etc., without requiring any information on the hardware on which the workflws are being ‘scheduled’. This means that the characterisation is only accurate for that particular hardware, if those values are to be used across the board; testing on heterogeneous systems, for example, is not possible unless the values are to be changed.

This is dealt with in varied ways across the literature. For example, [RB18] use the distributions rom [BCD+08] paper, and change the units from seconds to MIPS, rather than doing a conversion between the two. Others use the values taken from distribution and workflow generator, without explaining how their runtimes differ between esources [ANE13], [MJDN15]; Malawski et al enerate ‘20 diferent workflow instances... using parameters and task untmime distriutions from real workflow traces’, but do not provide hese parameers [MJDN15]. Recent research rom [WLZ+19] still uses the workflows identified n [BCD+08], [JCD+13], but only the structure of the workflows is assessed, replacing the tasks from the original workflows with other, unrelated examples.

Cost generation in *shadowgen*

The cost generatation method used by *shadowgen* is a normalised-cost approach, in which the values calculated for the runtime, memory, and I/O for each tasks is derived from the normalised size as profiled n [JCD+13] and [BCD+08]. This way, the costs per-workflow are indicative of the relative length and complexity of each task, and are more likely to transpose across different hardware configurations than using the varied approaches in the literature. 2

$$X' = \frac{(X \times n_{task}) - X_{min}}{X_{max} - X_{min}} \tag{1}$$

The distribution of values is derived from a table of normalised values using a varation on min-max feature scaling for each mean/std. deviation column in able. The formula to calulate each tasks’ normalised values is described in Equation 1; the results of applying this to Tabe is shown in Table

This approach allows algorithm designers and testers to describe what units they are interested in (e.g. seconds, MIPS, or FLOP seconds for runtime, MB or GB for Memory etc.) whilst still retaining the relative costs of that task within the workflow. In the example of Table 3, it is clear that mAdd and mBackground are still the longest running and I/O intensive tasks, making the units less of a concern.

Conclusion

shadow provides an implementation that addresses the proposed framework outlined by [MT18] to improve reproducibility and when benchmarking task scheduling algorithms. Additionally, this software creates a repository of algorithms that is intended to

Job	Count	Runtime		I/O Read		I/O Write		Peak Memory		CPU Util	
Job	Count	Mean (s)	Std. Dev.	Mean (MB)	Std. Dev.	Mean (MB)	Std. Dev.	Mean (MB)	Std. Dev.	Mean (%)	Std. Dev
mProjectPP	2102	1.73	0.09	2.05	0.07	8.09	0.31	11.81	0.32	86.96	0.03
mDiffFit	6172	0.66	0.56	16.56	0.53	0.64	0.46	5.76	0.67	28.39	0.16
mConcatFit	1	143.26	0.00	1.95	0.00	1.22	0.00	8.13	0.00	53.17	0.00
mBgModel	1	384.49	0.00	1.56	0.00	0.10	0.00	13.64	0.00	99.89	0.00
mBackground	2102	1.72	0.65	8.36	0.34	8.09	0.31	16.19	0.32	8.46	0.10
mImgtbl	17	2.78	1.37	1.55	0.38	0.12	0.03	8.06	0.34	3.48	0.03
mAdd	17	282.37	137.93	1102.57	302.84	775.45	196.44	16.04	1.75	8.48	0.11
mShrink	16	66.10	46.37	411.50	7.09	0.49	0.01	4.62	0.03	2.30	0.03
mJPEG	1	0.64	0.00	25.33	0.00	0.39	0.00	3.96	0.00	77.14	0.00

TABLE 2: *Example profile of Montage workflow, as presented in [JCD+13]*

	Runtime		I/O Read		I/O Write		Peak Memory		CPU Util	
job	Mean (s)	Std. Dev.	Mean (MB)	Std. Dev.	Mean (MB)	Std. Dev.	Mean (MB)	Std. Dev.	Mean (%)	Std. Dev
mProject PP	9.47	0.49	11.22	0.38	44.30	1.70	64.66	1.75	476.20	0.16
mDiffFit	10.61	9.00	266.27	8.52	10.29	7.40	92.61	10.77	456.48	2.57
mConcatFit	0.37	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.13	0.00
	1.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.25	0.00
mBackground	9.42	3.56	45.78	1.86	44.30	1.70	88.65	1.75	46.32	0.55
mImgtbl	0.12	0.06	0.06	0.02	0.01	0.00	0.35	0.02	0.15	0.00
mAdd	12.50	6.11	48.83	13.41	34.34	8.70	0.70	0.08	0.37	0.00
mShrink	2.75	1.93	17.15	0.30	0.02	0.00	0.18	0.00	0.09	0.00
mJPEG	0.00	0.00	0.06	0.00	0.00	0.00	0.00	0.00	0.19	0.00

TABLE 3: *Updated relative cost values using the min-max feature scaling method described in Equation :ref:'normalise'*

continually be updated, providing a resource for future developers. *shadowgen* extends on existing research from both the task- scheduling and workflow scheduling communitites in graph generation by leveraging existing techniques and wrapping them into a simple and flexible utility. The adoption of a JSON data format compliments the move towards JSON as a standardised way of representing workflows, as demonstrated by the Common Workflow Language [CCH+16] and WorkflowHub[4].

Future work

Moving forward, heuristics and metaheuristics will continue to be added to the shadow algorithms module to facilitate broader benchmarking and o provid a 'living' repository of workflow scheduling algorithms. Further investigation into workflow visualisation techniques will also be conducted. There are plans to develop a tool that uses the pecifications in hpcnfig[5], which are class-based descriptions of different hardware (e.g. `class XeonPhi`) and High Performance Computing facilities (e.g `class PawseyGalaxy`). The idea behind hpconfig is that classes can be used to quickly 'unwrap' into a large cluster or system, without having large JSON files in the repository or on disk; they also help with readability, as the specification data is represented clearly as class attributes.

References

[ANE10] S. Abrishami, M. Naghibzadeh, and D. Epema. Cost-driven scheduling of grid workflows using Partial Critical Paths. In *2010 11th IEEE/ACM International Conference on Grid Computing*, pages 81–88, October 2010. doi:10.1109/GRID.2010.5697955.

[ANE13] Saeid Abrishami, Mahmoud Naghibzadeh, and Dick H. J. Epema. Deadline-constrained workflow scheduling algorithms for Infrastructure as a Service Clouds. *Future Generation Computer Systems*, 29(1):158–169, January 2013. doi:10.1016/j.future.2012.05.004.

[BBL+16] T. Bridi, A. Bartolini, M. Lombardi, M. Milano, and L. Benini. A Constraint Programming Scheduler for Heterogeneous High-Performance Computing Machines. *IEEE Transactions on Parallel and Distributed Systems*, 27(10):2781–2794, October 2016. doi:10.1109/TPDS.2016.2516997.

[BCD+08] Shishir Bharathi, Ann Chervenak, Ewa Deelman, Gaurang Mehta, Mei-Hui Su, and Karan Vahi. Characterization of scientific workflows. In *2008 Third Workshop on Workflows in Support of Large-Scale Science*, pages 1–10, November 2008. doi:10.1109/WORKS.2008.4723958.

[BÇRS13] Anne Benoit, Ümit V. Çatalyürek, Yves Robert, and Erik Saule. A Survey of Pipelined Workflow Scheduling: Models and Algorithms. *ACM Comput. Surv.*, 45(4):50:1–50:36, August 2013. doi:10.1145/2501654.2501664.

[BM02] Rajkumar Buyya and Manzur Murshed. GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing. *Concurrency and Computation: Practice and Experience*, 14(13-15):1175–1220, November 2002. doi:10.1002/cpe.710.

[Bur] Andrew Marc Burkimsher. Fair, Responsive Scheduling of Engineering Workflows on Computing Grids. page 238.

[CA93] V. Chaudhary and J. K. Aggarwal. A generalized scheme for mapping parallel algorithms. *IEEE Transactions on Parallel and Distributed Systems*, 4(3):328–346, March 1993. doi:10.1109/71.210815.

[CCCR18] Y. Caniou, E. Caron, A. K. W. Chang, and Y. Robert. Budget-Aware Scheduling Algorithms for Scientific Workflows with Stochastic Task Weights on Heterogeneous IaaS Cloud Platforms. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 15–26, May 2018. doi:10.1109/IPDPSW.2018.00014.

[CCH+16] Brad Chapman, John Chilton, Michael Heuer, Andrey Kartashov, Dan Leehr, Hervé Ménager, Maya Nedeljkovich, Matt Scales, Stian Soiland-Reyes, and Luka Stojanovic. *Common Workflow*

1. https://github.com/WorkflowSim/WorkflowSim-1.0/tree/master/sources/org/workflowsim/planning
2. https://github.com/perarnau/ggen
3. https://github.com/pegasus-isi/WorkflowGenerator
4. github.com/myxie/hpconfig
5. https://workflowhub.org/simulator.html

Language, v1.0. figshare, United States, July 2016. doi:10.6084/m9.figshare.3115156.v2.

[CD12] Weiwei Chen and Ewa Deelman. WorkflowSim: A toolkit for simulating scientific workflows in distributed environments. In *2012 IEEE 8th International Conference on E-Science*, pages 1–8, October 2012. doi:10.1109/eScience.2012.6404430.

[CG72] E. G. Coffman and R. L. Graham. Optimal scheduling for two-processor systems. *Acta Informatica*, 1(3):200–213, September 1972. doi:10.1007/BF00288685.

[CHI14] Tom Crick, Benjamin A. Hall, and Samin Ishtiaq. "Can I Implement Your Algorithm?": A Model for Reproducible Research Software. *arXiv:1407.5981 [cs]*, September 2014. arXiv:1407.5981.

[CK88] T. L. Casavant and J. G. Kuhl. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Transactions on Software Engineering*, 14(2):141–154, February 1988. doi:10.1109/32.4634.

[CLQ] Henri Casanova, Arnaud Legrand, and Martin Quinson. SimGrid: A Generic Framework for Large-Scale Distributed Experiments. page 7.

[DFP12] J. J. Durillo, H. M. Fard, and R. Prodan. MOHEFT: A multi-objective list-based method for workflow scheduling. In *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings*, pages 185–192, December 2012. doi:10.1109/CloudCom.2012.6427573.

[DMPR17] Pierre-François Dutot, Michael Mercier, Millian Poquet, and Olivier Richard. Batsim: A Realistic Language-Independent Resources and Jobs Management Systems Simulator. In Narayan Desai and Walfredo Cirne, editors, *Job Scheduling Strategies for Parallel Processing*, volume 10353, pages 178–197. Springer International Publishing, Cham, 2017. doi:10.1007/978-3-319-61756-5_10.

[DRFG$^+$12] François-Michel De Rainville, Félix-Antoine Fortin, Marc-André Gardner, Marc Parizeau, and Christian Gagné. DEAP: A python framework for evolutionary algorithms. In *Proceedings of the Fourteenth International Conference on Genetic and Evolutionary Computation Conference Companion - GECCO Companion '12*, page 85, Philadelphia, Pennsylvania, USA, 2012. ACM Press. doi:10.1145/2330784.2330799.

[HDRD98] Willy Herroelen, Bert De Reyck, and Erik Demeulemeester. Resource-constrained project scheduling: A survey of recent developments. *Computers & Operations Research*, 25(4):279–302, April 1998. doi:10.1016/S0305-0548(97)00055-5.

[JCD$^+$13] Gideon Juve, Ann Chervenak, Ewa Deelman, Shishir Bharathi, Gaurang Mehta, and Karan Vahi. Characterizing and profiling scientific workflows. *Future Generation Computer Systems*, 29(3):682–692, March 2013. doi:10.1016/j.future.2012.08.015.

[Joh] Steven G. Johnson. The NLopt nonlinear-optimization package,.

[KA99] Yu-Kwong Kwok and Ishfaq Ahmad. Benchmarking and Comparison of the Task Graph Scheduling Algorithms. *Journal of Parallel and Distributed Computing*, 59(3):381–422, December 1999. doi:10.1006/jpdc.1999.1578.

[KBPR17] G. Kousalya, P. Balakrishnan, and C. Pethuru Raj. Workflow Scheduling Algorithms and Approaches. In G. Kousalya, P. Balakrishnan, and C. Pethuru Raj, editors, *Automated Workflow Scheduling in Self-Adaptive Clouds: Concepts, Algorithms and Methods*, Computer Communications and Networks, pages 65–83. Springer International Publishing, Cham, 2017. doi:10.1007/978-3-319-56982-6_4.

[MJDN15] Maciej Malawski, Gideon Juve, Ewa Deelman, and Jarek Nabrzyski. Algorithms for cost- and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds. *Future Generation Computer Systems*, 48:1–18, July 2015. doi:10.1016/j.future.2015.01.004.

[MT18] Ashish Kumar Maurya and Anil Kumar Tripathi. On benchmarking task scheduling algorithms for heterogeneous computing systems. *The Journal of Supercomputing*, 74(7):3039–3070, July 2018. doi:10.1007/s11227-018-2355-0.

[Pin12] Michael Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, New York, 4th ed edition, 2012.

[RB16] Maria Alejandra Rodriguez and Rajkumar Buyya. A taxonomy and survey on scheduling algorithms for scientific workflows in IaaS cloud computing environments. *Concurrency and Computation: Practice and Experience*, 29(8):e4041, 2016. doi:10.1002/cpe.4041.

[RB18] Maria A. Rodriguez and Rajkumar Buyya. Scheduling dynamic workloads in multi-tenant scientific workflow as a service platforms. *Future Generation Computer Systems*, 79:739–750, February 2018. doi:10.1016/j.future.2017.05.009.

[THW02] H. Topcuoglu, S. Hariri, and Min-You Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):260–274, March 2002. doi:10.1109/71.993206.

[Ull75] J. D. Ullman. NP-complete Scheduling Problems. *J. Comput. Syst. Sci.*, 10(3):384–393, June 1975. doi:10.1016/S0022-0000(75)80008-0.

[WLZ$^+$19] Yuandou Wang, Hang Liu, Wanbo Zheng, Yunni Xia, Yawen Li, Peng Chen, Kunyin Guo, and Hong Xie. Multi-Objective Workflow Scheduling With Deep-Q-Network-Based Multi-Agent Reinforcement Learning. *IEEE Access*, 7:39974–39982, 2019. doi:10.1109/ACCESS.2019.2902846.

[WWT15] Fuhui Wu, Qingbo Wu, and Yusong Tan. Workflow scheduling in cloud: A survey. *The Journal of Supercomputing*, 71(9):3373–3418, September 2015. doi:10.1007/s11227-015-1438-4.

[YB06] Jia Yu and Rajkumar Buyya. Scheduling Scientific Workflow Applications with Deadline and Budget Constraints Using Genetic Algorithms. https://www.hindawi.com/journals/sp/2006/271608/abs/, 2006. doi:10.1155/2006/271608.

[zot] Scheduling under uncertainty: Survey and research directions - IEEE Conference Publication. https://ieeexplore-ieee-org.ezproxy.library.uwa.edu.au/abstract/document/6866316.