shadowgen

Background

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[1], 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 conducted in [1], 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.

Moving forward, shadow will also use the specifications in hpconfig [2], which are class-based descriptions of different hardware (e.g. `class XeonPhi`) and facilities (e.g `class PawseyGalaxy`) used in HPC. The idea behind hpconfig is the 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.

Existing approaches

A majority of work published in workflow scheduling will use workflows generated using the approach laid out in [1]. 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 workflows 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, [2] use the distributions from [1] 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 resources [3, 4]; Malawski et al, discuss how '20 different workflow instances were generated using parameters and task runtmime distributions from real workflow traces', but they do not provide the parameters [4]. Recent research still uses the workflows identified in [1, 5], but use only the structure of the workflows, replacing the tasks with other computationally intensive examples [6].

Proposed addition to *shadowgen*

The method I propose is a normalised-cost approach, in which the values calculated for the runtime, memory, and I/O for each tasks is determined based on the normalised size as profiled in [5] and [1]. This way, the costs per-workflow are indicative of the relative length and complexity of each

[1] https://github.com/perarnau/ggen
[2] github.com/myxie/hpconfig
[3] https://github.com/pegasus-isi/WorkflowGenerator

task, and are more likely to transpose across different hardware configurations than using the varied approaches in the literature.

Table 1: Experimental conditions for workflow profiling in [5]

Workflow	Site	Nodes	Cores	Cpu (GHz)	Memory(GB)	File System	ioprof	pprof	Normal
Montage	Amazon EC2	1	8	Xeon@2.33	7.5	ext3	1:11:20	0:57:36	0:55:28
CyberShake	TACC Ranger	36	16	Opteron@2.3	32	Lustre	16:35:00	12:20:00	N/A
Broadband	Amazon EC2	1	8	Xeon@2.33	7.5	ext3	1:31:20	1:21:16	1:20:03
Epigenome	Amazon EC2	1	8	Xeon@2.33	7.5	ext3	1:11:11	1:08:01	1:07:40
LIGO	Syracuse SUGAR	80	4	Xeon@2.50	7.5	NFS	1:48:11	1:47:38	1:41:13
SIPHT	UW Newbio	13	8	Xeon@3.16	7.5	ext3	1:33:24	1:19:37	1:10:53

Table 2: Example profile of Montage workflow, as presented in [5]

Job	Count	Runtime		I/O Read		I/O Write		Peak Memory		CPU Util	
		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

The proposed distribution of values would be derived from a table of normalised values using a varation on min-max feature scaling for each mean/std. deviation column in Table 2. The formula used to calculated each tasks' normalised values is described in Equation 1; the results of applying this to Table 2 is shown in Table 3

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

Table 3: Normalised values for columns derived from Profiled values Table 2

Job	Runtime		I/O Read		I/O Write		Peak Memory		CPU Util	
	Mean (s)	Std. Dev.	Mean (MB)	Std. Dev.	Mean (MB)	Std. Dev.	Mean (MB)	Std. Dev.	Mean (%)	Std. Dev
mProjectPP	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
mBgModel	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

This approach would allow algorithm designers and testers to prescribe 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, and so the units are less of a concern.

References

[1] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi, "Characterization of scientific workflows," in *2008 Third Workshop on Workflows in Support of Large-Scale Science*, pp. 1–10, Nov. 2008.

[2] M. A. Rodriguez and R. Buyya, "Scheduling dynamic workloads in multi-tenant scientific workflow as a service platforms," *Future Generation Computer Systems*, vol. 79, pp. 739–750, Feb. 2018.

[3] S. Abrishami, M. Naghibzadeh, and D. H. J. Epema, "Deadline-constrained workflow scheduling algorithms for Infrastructure as a Service Clouds," *Future Generation Computer Systems*, vol. 29, pp. 158–169, Jan. 2013.

[4] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Algorithms for cost- and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds," *Future Generation Computer Systems*, vol. 48, pp. 1–18, July 2015.

[5] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi, "Characterizing and profiling scientific workflows," *Future Generation Computer Systems*, vol. 29, pp. 682–692, Mar. 2013.

[6] Y. Wang, H. Liu, W. Zheng, Y. Xia, Y. Li, P. Chen, K. Guo, and H. Xie, "Multi-Objective Workflow Scheduling With Deep-Q-Network-Based Multi-Agent Reinforcement Learning," *IEEE Access*, vol. 7, pp. 39974–39982, 2019.