shadow Documentation

Release 0.1

RW Bunney

Jan 03, 2023

CONTENTS

CHAPTER

ONE

WHAT IS SHADOW?

SHADOW is a library for the use and testing of DAG-based workflow scheduling algorithms. SHADOW provides implementations of various heuristic and metaheuristic algorithms to address single-and multi-objective scheduling problems; these algorithms are accessed using a workflow-oriented class system built into the library.

1.1 Motivation

Development of the SHADOW library was motivated by a lack of public implementations of popular and highly-cited workflow scheduling algorithms. It is hoped that the library becomes a repository of workflow scheduling heuristics and metaheuristics, as well as tool in which algorithm developers may test and benchmark their own efforts.

1.1.1 Example schedule

SHADOW is Workflow-oriented, which means most of the work done within the library involves passing around workflow objects, or storing data within the object itself.

This workflows are represented as Directed Acyclic Graphs (DAGs), and the a solution to the DAG scheduling problem is the scheduling order and machine allocation of tasks (the nodes) from the DAG.

In SHADOW, we use JSON to store our workflows, which are always represented as DAGs. This JSON data is read into a SHADOW as a NetworkX `DiGraph` object, and then wrapped in SHADOW's `Workflow` class. The `Workflow` class accepts two types of graph specifications; one that has pre-calculated costs, and one without. The preferred approach is to present task with 'total computation cost'; the units may be application specific, but when coupled with machine resources they should be easily converted to time (it is easiest to think of them in terms of required cycles or Total FLOP).

For exampe, below is a small workflow, depicted as a graph, with the total computation costs and machine values stored in the table to the right. The workflow is based on the example presented in the HEFT paper[1].

The computation costs for tasks, along with the graph attributes, are stored in a JSON file separate to the machine specifications. In SHADOW, we first initialise a `Workflow` object, and then add an environment to it; this environment is constructed from our machine specification. The structure of the specification files are described in more detail in '*Configuration*'.

```
from shadow.classes.workflow import Workflow
from shadow.classes.environment import Environment
```

(continues on next page)

[1] Performance-effective and low-complexity task scheduling for heterogeneous computing (Topcuoglu et al. 2002).

(continued from previous page)

```
wf = Workflow('example_graph.json')
env = Environment('machine_spec.json')
wf.add(env)
```

To run an algorithm on the workflow and determine the final makespan, simply pass the Workflow object to the algorithm:

```
from shadow.heuristics import heft


heft(wf)
print(wf.makespan) # 98
```

1.1.2 Configuration

Running algorithms from the SHADOW library requires two important configuration files:

- The Workflow specification
- The Environment specification

These are both defined in JSON files, which are described in more detail below.

Workflow Specification

```
{
    "nodes": [
        {
            "comp": 119000,
            "id": 0
        },
        {
            "comp": 92000,
            "id": 1
        },
        {
            "comp": 95000,
            "id": 2
        },
        ....
        ....
    ],
    "links": [
        {
            "transfer_data": 18,
            "source": 0,
            "target": 1
        },
        {
            "transfer_data": 12,
            "source": 0,
```

(continues on next page)

(continued from previous page)

```
            "target": 2
        },
        {
            "transfer_data": 9,
            "source": 0,
            "target": 3
        },
        ...
        ...
}
```

The environment the in which the workflow is scheduled is defined in a separate file; this way, scheduling across different environment configurations can be tested (additionally, it is likely workflows will change, whereas workflows will be run in different environments all the time).

```
{
  "system": {
    "resources": {
      "cat0_m0": {
        "flops": 7.0
      },
      "cat1_m1": {
        "flops": 6.0
      },
      "cat2_m2": {
        "flops": 11.0
      }
    },
    "compute_bandwidth": {
      "cat0": 1.0,
      "cat1": 1.0,
      "cat2": 1.0
    }
  }
}
```

For large systems, with many resources of the same type, the following is common:

```
"resources": {
  "cat0_m0": {
    "flops": 145.0
  },
  "cat0_m1": {
    "flops": 145.0
  },
  "cat0_m2": {
    "flops": 145.0
  },
  "cat0_m3": {
    "flops": 145.0
  },
}
```

As mentioned earlier, 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. This can be achieved by adding a list of costs per tasks to the workflow specification JSON file, in addition to the following 'header':

Here, we present an example schedule for the DAG presented in the original HEFT paper.

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

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

1.1.3 Roadmap

This document details the intended functionality that will be present in the 'official' 0.1 release (currently its status would be 'alpha').

Algorithms

By the release of 0. 1, we intend to have implemented the following:

- HEFT
- PHEFT
- MOLS
- NSGAII*
- SPEAII*

Workflow Representation

- Workflows will be initialised based on either a NetworkX DiGraph in a users code, or read in from a .json file.

Environment Representation

- The environment on which workflows are being scheduled will be initialised either through a user-implemented dictionary, or from a .json file.

Visualisation

- Able to plot different x/y pairings (throughput vs. time, compare two plots makespan etc.)

Documentation:

- Docstrings for each algorithm
- Introduction page with example model
- Class descriptions
- Algorithm descriptions