# Artifacts of “End-to-end Scheduling of Real-time Task Pipelines on Multiprocessors”

### Constraint Solvers

1. `CoPi`: Our heuristic constraint solver algorithm.
2. `GEKKO`: The APOPT Solver: https://machinelearning.byu.edu/

Most experiments compare the above two. We have also experimented with two other MINLP solvers but only for the first experiment:

1. `pyomo`: http://www.pyomo.org/
2. `scipy`: https://docs.scipy.org/doc/scipy/reference/optimize.html

### Requirements

Please follow the installation procedure of `GEKKO`, `pyomo`, `scipy`, `pickle` (for reading and writing dataset). We believe that the code should work for python version 3.5 and 3.6. We have run all the experiments with python 3.6. If not sure, please use `virtualenv` to install python 3.6 .

### Algorithms and Files

The CoPi algorithms are at the root location of the source directory. The MINLP solver implementations are in `ilp/`.

CoPi files are: 1. `copi_e2e.py`: For only E2E Delay Constraint implementation. 2. `copi_all.py`: For all the constraints. 3. `multi_pipeline.py`: For the multiprocessor experiments with CoPi.

MINLP Solvers: 1. `ilp/ilp_gekko.py`: `GEKKO` constraint optimization solution. 2. `ilp/ilp_pyomo.py`: `pyomo` constraint optimization solution. 3. `ilp/ilp_scipy.py`: `scipy` constraint optimization solution.

**Datasets** Random pipelines are generated using UUnifast algorithm. All the dataset files are `pickle` files. 1. `dataset_*` files are for uniprocessor and runtime experiments. 2. `pipelines_*` files are for multiprocessor experiments.

Helper function files: 1. `utility.py`: Some helper functions for utilization, RMS bound and other calculations. 2. `pipeline.py`: Most functions related to a pipeline are here. For example, end-to-end delay, loss-rate, etc.

**Figures** The figures are provided in the `Figures` directory.

All the figures could be generated as PDF files by:

```
cd Figures
make
```

## Experiments

The next subsections correspond to the subsections in the Evaluation section of the paper.

### Uniprocessor Acceptance Ratio Experiments

#### Only for End-to-end Constraint

1. Run all the experiments for `CoPi`:

```
./run_exp1_copi.sh
```

2. Run all the experiments for `GEKKO (APOPT)`:

```
./run_exp1_gekko.sh
```

3. Run all the experiments for `pyomo (IPOPT)`:

```
./run_exp1_pyomo.sh
```

4. Run all the experiments for `scipy (trust-constr)`:

```
./run_exp1_scipy.sh
```

**Result** All the data will be written to `accepted_sets_<solver>.txt` in space separated format for $LBG = \{11, 12, 14, 15, 16, 18\}$. For example, `accepted_sets_copi.txt`

Paper results are summarized in `Figures/accept.csv`. (3rd, 5th, 6th, 7th lines are respectively CoPi, GEKKO, scipy and pyomo.)

#### Both End-to-end and Loss-rate Constraints

1. Generate CoPi data points by the following script:

```
./run_loss_rate_copi.sh
```

The `accepted_lr_copi.txt` will have the number of accepted (schedulable) pipelines in space separated format for $\{0, 25, 50, 75\}\%$ loss-rate

2. Generate GEKKO data points by the following script:

```
./run_gekko_lr.sh
```

All the data will be appended to `accepted_sets_gekko.txt`.

3. Generate GEKKO (with BAC) data points by the following script:

```
./run_gekko_lr_bac.sh
```

All the data will be appended to `accepted_sets_gekko.txt`.

Paper results are summarized in `Figures/accept_models_lr.csv`. 3rd, 4th, 5th lines are respectively: GEKKO, CoPi, GEKKO (with BAC)

### Solver Runtime Overhead

1. CoPi's runtime vs pipeline length experiment (Figure 9c):

```
./time_measure_copi_pipelength.sh
```

The results will be available in `scheduled_times_copi.txt` and `failed_times_copi.txt` in space separated format.

Paper results are summarized in `Figures/runtime_pipe_length.csv`. (2nd line, 3rd line: schedulable and unschedulable).

2. Run `CoPi` experiments for Figure 9a and 9b:

```
./run_solver_exp2_copi.sh
```

3. Run `GEKKO` experiments for Figure 9a and 9b:

```
./run_solver_exp2_gekko.sh
```

Paper results are summarized in `Figures/lbg_time_accept.csv` and `Figures/lbg_time_failed.csv`

### CoPi Performance Insight

CoPi's AR vs. pipeline length and NLBG

```
./run_perform_insight.sh
```

### Multiprocessor Experiments

Multiprocessor experiments are run by the following command:

Run the processor based experiments with the following script:

```
./run_multiproc_2cores.sh
./run_multiproc_4cores.sh
./run_multiproc_8cores.sh
```

Results will be written to accepted_multiproc_<number of cores>_<number of tasks in a pipeline>.

Paper results are summarized in `Figures/accept_multiprocessor_2core.csv`, `Figures/accept_multiprocessor_4core.csv`, `Figures/accept_multiprocessor_8core.csv`

Lower level python script:

```
python multi_pipeline.py -p <number of pipelines> -t <number of tasks in each Pipeline> -c <
```

**Utilization Experiments and Results:**

```
./run_util.sh
```

The result is written to `multi_util_result.txt`.

**Migration Experiments:**

```
./run_mig.sh
```

The result is written to `migrations_result.txt`.