Supplementary Material for "Multi-Tree Genetic Programming Hyper-Heuristic for Dynamic Flexible Workflow Scheduling in Multi-Clouds"

Zaixing Sun, Yi Mei, *Senior Member, IEEE,* Fangfang Zhang, *Member, IEEE,*
Hejiao Huang, Chonglin Gu, and Mengjie Zhang, *Fellow, IEEE*

APPENDIX A
DETAILED PSEUDOCODE OF THE PROPOSED ALGORITHM.

The detailed pseudocode of the proposed algorithm is given in A1. It starts with a randomly generated population by ramp half-and-half in the lines 1-3 of A1, which is an initial set of heuristic rules to a problem. We set some parameters and termination condition in lines 4-7 of A1. We evaluate each individual based on the proposed dynamic workflow scheduling simulator as A2 in the line 8 of A1. The lines 9-10 of A1 are to update the best individuals that have been searched. If the termination condition is not satisfied (line 11 of A1), the parent selection (line 12 of A1) and evolution (line 13 of A1) operations are performed to generate a new population. The output of A1 are the learned three rules represented by the best individuals.

A2 is the detailed pseudocode of the dynamic workflow scheduling simulator. The simulator starts once the first workflow arrived (line 3 of A2). The simulator continues until all the workflows are completed or the event queue $\boldsymbol{\mathcal{E}}$ is empty (line 4 of A2). The scheduling process goes on by triggering each event based on its event name (line 6 of A2). When triggering the `WorkflowSubmitted`, the tasks without preceding tasks are ready tasks (line 7 A2), the task selection rule will be used to calculate the priority for each task (line 8 of A2). If there are multiple workflow submissions or multiple tasks that need to be set to ready state, the *task selection rule* is adopted to calculate the priority of each task, and the tasks are added to the `TaskReady` event queue in ascending order of this priority (lines 1-3 of A3). These tasks are added to the event queue according to priority (lines 9-10 of A2).

When triggering the `TaskReady`, we first use *cloud selection rule* (A4) to calculate the priority of each cloud (line 12 of A2). Then, in the cloud with the highest priority, one instance is selected to execute the task according to the *instance selection rule* (line 13 of A2). During selecting instance (A5), it filters out the instances that cannot meet the sub-deadline to execute the task (lines 1-3 of A5). If no instance remains, then the instance in the cloud with the earliest finish time is selected to execute the task (line 5 of A5). Otherwise, the instance selection rule is used to select the instance (lines 9-15 of A5). If there are multiple instances with the same priority, it is preferred to select an existing instance that does not need to be created (line 11 of A5). Once the instance that will execute the task has been determined, we can determine the start execution time of the task from the terminals associated with that instance in line 15 of A2. At the same time, we need to check if the instance is newly created and if it needs to accept data from its predecessor, and create the appropriate events line 14 of A2 and lines 1-6 of A6. When triggering the `StartExecuteTask`, the simulator start to execute the task (line 25 of A2), and set its successor to the ready state if all predecessors of this successor have started to execute (lines 26-31 of A2). The output of the simulator A2 is a schedule of a DFWS-MCs instance based on given Task/Cloud/Instance selection rules.

Algorithm 1: Multi-Tree Genetic Programming

Input: Training data; Population size NP, CSPs $\mathcal{P}$; All the service categories Θ ; Service instance types $\mathcal{S}(p)$

Output: The learned best scheduling heuristics R^* with task r_1^* /cloud r_2^* /instance r_3^* selection rules

`/* Population Initialisation */`

1 **while** $N_{ind} < NP$ **do**
2 **foreach** $Individual$ **do**
3 Initialise the task/cloud/instance selection rules by ramp half-and-half;
4 Set $r_1^* \leftarrow null$, $r_2^* \leftarrow null$, and $r_3^* \leftarrow null$;
5 Set $R^* \leftarrow r_1^* \cup r_2^* \cup r_3^*$ and $fitness(R^*) \leftarrow +\infty$;
6 Set $gen \leftarrow 0$;
7 **while** $gen < maxGen$ **do**
8 **Evaluation**: Evaluate the individuals based on the proposed simulator as **A2**; `// A 2`
9 **foreach** $Individual$ **do**
10 **if** $fitness(Individual) < fitness(R^*)$ **then** $R^* \leftarrow Individual$;
11 **if** $gen < maxGen$ **then**
 `/* Parent Selection */`
12 Select parents to generate offspring;
 `/* Evolution */`
13 Generate a new population by crossover, mutation, and reproduction;
14 $gen \leftarrow gen + 1$;
15 **return** The best task r_1^* /cloud r_2^* /instance r_3^* selection rules.

Algorithm 2: Dynamic Workflow Scheduling Simulator

Input: A DFWS-MCs instance; Task/Cloud/Instance selection rules $r_1^*(\cdot)$ /$r_2^*(\cdot)$ /$r_3^*(\cdot)$.
Output: A DFWS-MCs schedule

Set the schedule $\rho = \{\}$, event queue $\boldsymbol{\mathcal{E}} = \{\}$;
while *new workflow* g *arrives* **do**
  Create event `WorkflowSubmitted` $\mathcal{E}_e$ and $\boldsymbol{\mathcal{E}} = \boldsymbol{\mathcal{E}} \cup \mathcal{E}_e$;
while $\boldsymbol{\mathcal{E}}$ *is not empty* **do**
  Get the next event $\mathcal{E}_e$ from $\boldsymbol{\mathcal{E}}$;
  if $\mathcal{E}_e^N$ *is* `WorkflowSubmitted` **then**
    $X \leftarrow \{x | T^{pred}(x) = \varnothing, x \in \mathcal{E}_e^W\}$;
    $X \leftarrow$ `TaskSelection`(X) ; `// A 3`
    foreach $x \in X$ **do**
      Create event `TaskReady` $\mathcal{E}_e$ and $\boldsymbol{\mathcal{E}} = \boldsymbol{\mathcal{E}} \cup \mathcal{E}_e$;
  else if $\mathcal{E}_e^N$ *is* `TaskReady` **then**
    $\mathcal{E}_e^P \leftarrow$ `CloudSelection`$(\mathcal{E}_e^A)$; `// A 4`
    $\mathcal{E}_e^S \leftarrow$ `InstanceSelection`$(\mathcal{E}_e^A, \mathcal{E}_e^P)$; `// A 5`
    $\boldsymbol{\mathcal{E}} \leftarrow$ `DoubleCheck`$(\mathcal{E}_e^A, \mathcal{E}_e^S, \boldsymbol{\mathcal{E}})$; `// A 6`
    Create events `StartExecuteTask` and `CompleteExecuteTask` with trigger time $\mathcal{E}_e^S.AAT$ and $(\mathcal{E}_e^S.AAT + \mathcal{E}_e^S.AET)$ respectively, and **add** them to queue $\boldsymbol{\mathcal{E}}$;
  else if $\mathcal{E}_e^N$ *is* `StartBootingInstance` **then**
    Launch the instance;
  else if $\mathcal{E}_e^N$ *is* `CompleteBootingInstance` **then**
    Finish booting and **enter** an active state;
  else if $\mathcal{E}_e^N$ *is* `StartDataTransfer` **then**
    Start to receive the output data from predecessors;
  else if $\mathcal{E}_e^N$ *is* `CompleteDataTransfer` **then**
    Trigger the event `StartExecuteTask` if the output data of all predecessors are received;
  else if $\mathcal{E}_e^N$ *is* `StartExecuteTask` **then**
    Start to execute the task $\mathcal{E}_e^A$;
    foreach $x \in T^{succ}(\mathcal{E}_e^A)$ **do**
      if *all tasks in* $T^{pred}(x)$ *have started to execute* **then**
        $X \leftarrow X \cup x$;
    $X \leftarrow$ `TaskSelection`(X) ; `// A 3`
    foreach $x \in X$ **do**
      Create event `TaskReady` $\mathcal{E}_e$ and $\boldsymbol{\mathcal{E}} \leftarrow \boldsymbol{\mathcal{E}} \cup \mathcal{E}_e$;
  else if $\mathcal{E}_e^N$ *is* `CompleteExecuteTask` **then**
    Trigger the event `CompleteExecuteWorkflow` if $\mathcal{E}_e^A$ is the last finished task in a workflow;
  else if $\mathcal{E}_e^N$ *is* `CompleteExecuteWorkflow` **then**
    Record the workflow $\mathcal{E}_e^W$ has been scheduled;
return the obtained schedule ρ.

Algorithm 3: TaskSelection

Input: Tasks X; Task selection rule $r_1^*(\cdot)$.
Output: New tasks order X.

if $|X| > 1$ **then**
  Calculate the priority $r_1^*(x)$ of each task $x \in X$ by task selection rule $r_1^*(\cdot)$;
  Sort X in order of priority;
return X.

Algorithm 4: CloudSelection

Input: The selected ready task x; Cloud selection rule $r_2^*(\cdot)$.
Output: One selected cloud p^*.

Calculate the priority $r_2^*(x, p)$ of each cloud $p \in \mathcal{P}$ for the selected task x by cloud selection rule $r_2^*(\cdot)$;
return the cloud p^* with the highest priority.

Algorithm 5: InstanceSelection

Input: The selected ready task x; The selected cloud p; Instance selection rule $r_3^*(\cdot)$.
Output: One selected instance ins.

Calculate the terminals of all instances that can execute the selected task in the selected cloud;
`// Including leased instances and non-leased instances with all types.`
Calculate the priority $r_3^*(x, p, s)$ of each instance s in cloud p for the selected ready task x by instance selection rule $r_3^*(\cdot)$ and **sort** them;
Get the instance set $meetDL$ that meet the sub-deadline from all instances;
if $meetDL == \varnothing$ **then**
  $ins = \arg\min\{AAT + AET\}$; `// i.e. earliest finish time rule`
else
  Get the highest priority instance set $highPrio$ from the set $meetDL$;
  if $|highPrio| > 1$ **then**
    Get the instance set $newCre$ from $highPrio$ that does not need to be newly created;
    if $newCre \neq \varnothing$ **then**
      Randomly **select** an instance from $newCre$ as ins;
    else
      Randomly **select** an instance from $highPrio$ as ins;
  else
    Set the instance as ins;
return ins.

Algorithm 6: DoubleCheck

Input: The selected task $\mathcal{E}_e^A$; The selected instance $\mathcal{E}_e^S$; Event queue $\boldsymbol{\mathcal{E}}$.
Output: Updated event queue $\boldsymbol{\mathcal{E}}$.

if $\mathcal{E}_e^S$ needs to be newly created **then**
  Create events `StartBootingInstance` and `CompleteBootingInstance` with trigger time ($\mathcal{E}_e^S.AAT-$ booting time of the instance) and $\mathcal{E}_e^S.AAT$ respectively, and **add** them to queue $\boldsymbol{\mathcal{E}}$;
if $T^{pred}(\mathcal{E}_e^A) \neq \varnothing$ **then**
  Calculate the maximum communication time $maxCT$ from predecessors to $\mathcal{E}_e^A$;
  Create events `StartDataTransfer` and `CompleteDataTransfercute` with trigger time ($\mathcal{E}_e^S.AAT-maxCT$) and $\mathcal{E}_e^S.AAT$ respectively, and **add** them to queue $\boldsymbol{\mathcal{E}}$;
return $\boldsymbol{\mathcal{E}}$.

APPENDIX B
RUNTIME ANALYSIS

The following Table I shows the average time for offline training of the scheduling algorithms. We can find that MTGP takes longer to train for the scientific workflows (wfNum=20) than for the Cluster-trace-v2018 (wfNum>20). For Cluster-trace-v2018 scenarios, the training time becomes progressively longer as the number of workflows increases. 2D is more time-consuming than 3D for most scenarios, which mainly depends on the number of available candidates at decision points. For the same scenario, there are more available instances in all clouds than in one cloud. This means that there are more candidates for the instance selection decision in 2D, so more computations are needed.

The runtime of the proposed method is acceptable in dynamic scheduling scenarios. Our algorithm mainly involves three decision points, i.e. task/cloud/instance selection, and the average decision-making time of each decision point in the online test phase is less than 0.02 seconds.

Table I
THE AVERAGE TIME FOR OFFLINE TRAINING OF THE SCHEDULING ALGORITHMS (IN HOUR)

<wfNum, λ>	2D	3D
<20, 0.2>	33.18	27.84
<20, 0.5>	46.57	28.62
<20, 0.8>	34.19	28.09
<20, 1.0>	51.16	27.01
<30, 0.2>	0.91	0.96
<30, 0.5>	0.81	0.84
<30, 0.8>	0.76	0.74
<30, 1.0>	0.86	0.70
<50, 0.2>	2.04	3.45
<50, 0.5>	2.62	2.55
<50, 0.8>	1.66	2.40
<50, 1.0>	3.36	2.71
<80, 0.2>	6.85	6.24
<80, 0.5>	7.28	5.93
<80, 0.8>	7.89	6.04
<80, 1.0>	5.61	7.01
<100, 0.2>	11.54	9.23
<100, 0.5>	8.64	9.10
<100, 0.8>	9.29	8.40
<100, 1.0>	8.36	8.21