Managing Dynamic Workflows in BEE

Steven Anaya
steven.anaya@student.nmt.edu
HPC-DES
Mentor: Tim Randles
August 10, 2020
BEE: Build and Execute Environment

• Goal: to create a unified software stack to *containerize* HPC apps
• Seeks to simplify execution of complex scientific workflows on HPC systems by:
  • Modeling workflows using a workflow language specification (CWL)
  • Storing and visualizing workflows as DAGs in a graph database (Neo4j)
  • Managing workflow execution using the BEE workflow engine
• Supports Charliecloud and Singularity containers
• Supports the Slurm workload manager
Motivation

• BEE seeks to support as much of CWL as possible
• Currently only supports workflows in which inputs and outputs between steps are known \textit{a priori}
  • Not sufficient for complex dynamic workflows in which:
    • Unknown numbers of outputs may be generated by a step
    • A task may need to be run on each of them (scatter)
    • A subsequent step may depend on all of them as inputs (gather)
• The way BEE models workflows needs to change
Neo4j and Cypher

- **Neo4j**
  - Transactional graph database
  - Stores data as nodes and relationships with properties
  - Uses the Cypher Query Language
  - Supports visualization of database in a browser
  - Extremely scalable

- **Cypher**
  - Declarative “SQL-inspired” query language
  - Visual and logical syntax
  - Example: get tasks dependent on a task given by $task_id

  ```
  MATCH (t:Task)-[:DEPENDS]->(:Task {task_id: $task_id})
  RETURN t
  ```
CWL: Common Workflow Language

- An open standard for describing analysis workflows and tools
- Makes workflows portable and scalable
- Allows execution of workflows on a variety of HPC and cloud environments
- Specification syntax based on YAML
- Example: run the echo command on an input string

```cwl
#!/usr/bin/env cwl-runner

cwlVersion: v1.0
class: CommandLineTool
baseCommand: echo

inputs:
  message:
    type: string
    inputBinding:
      position: 1
outputs: []
```
Former BEE Workflow Model
Former BEE Workflow Model – Data Structures

- Task
  - UUID
  - Name
  - Command
  - Hints
  - Subworkflow
  - Inputs
  - Outputs
  - State

- Metadata
  - Workflow Hints
  - Workflow Requirements

- Tasks are created and added to the graph database as nodes through the workflow interface.
- Dependencies are modeled as DEPENDS_ON relationships between tasks, automatically created when tasks are added.
  - Cypher query matches ins/outs
  - Metadata node stores hints and requirements of workflow.
Former BEE Workflow Model – Execution

- The workflow execution is initialized through the workflow manager
  - Workflow execution may also be paused or stopped through the WFM
- Task may execute when all of its input dependencies are satisfied
  - Requires all task inputs/outputs to be known prior to execution
  - Does not support complex dynamic workflows
- CWL supports task “scattering”
  - Task is specified to run multiple times over an array of inputs
Complex Dynamic Workflow

scatter.cwl (partial)

```cwl
import cwlVersion: v1.0
class: Workflow

requirements:
  ScatterFeatureRequirement: {}

inputs:
  experience_score: int
  interview_score: int
  test_score: int
  iterations: int
  datasetpath: string

outputs:
  final_answer:
    outputSource: predict/answer
    type: float

steps:
  read:
    run: /home/bee/cwl2/read.cwl
    in:
      x: datasetpath
    out:
      - output_array
  preprocess:
    run: /home/bee/cwl2/preprocess.cwl
    scatter: data_column_file
    in:
      x: read/output_array
    out:
      - output_preprocessed_array
```

read.cwl

```cwl
import cwlVersion: v1.0
class: CommandLineTool
baseCommand: ["python", "/home/bee/cwl2/finalread.py"]

inputs:
  x:
    type: string
    inputBinding:
      position: 1

stdout: output.txt

outputs:
  output:
    type:
      type: array
      items: File
    outputBinding:
      glob: "*.txt"
```

- Reads dataset and outputs data in each column as its own file
  - Number of columns unknown
- Scatters the output array for preprocessing
Complex Dynamic Workflow

scatter.cwl (partial)

cwlVersion: v1.0
class: Workflow

requirements:
  ScatterFeatureRequirement: {}

inputs:
  experience_score: int
  interview_score: int
  test_score: int
  iterations: int
  datasetpath: string

outputs:
  final_answer:
    outputSource: predict/answer
    type: float

steps:
  read:
    run: /home/bee/cwl2/read.cwl
    in:
      x: datasetpath
    out:
      - output_array
  preprocess:
    run: /home/bee/cwl2/preprocess.cwl
    (scatter: data_column_file)
    in:
      x: read/output_array
    out:
      - output_preprocessed_array

read.cwl

cwlVersion: v1.0
class: CommandLineTool
baseCommand: ["python", "/home/bee/cwl2/finalread.py"]

inputs:
  x:
    type: string
    inputBinding:
      position: 1

stdout: output.txt

outputs:
  output:
    type:
      type: array
      items: File
    outputBinding:
      glob: "*.txt"

• Reads dataset and outputs data in each column as its own file
  • Number of columns unknown
• Scatters the output array for preprocessing
### Complex Dynamic Workflow

**scatter.cwl (partial)**

```cwl

cwlVersion: v1.0
class: Workflow

requirements:
  ScatterFeatureRequirement: {}

inputs:
  experience_score: int
  interview_score: int
  test_score: int
  iterations: int
dataSetPath: string

outputs:
  final_answer:
    outputSource: predict/answer
    type: float

steps:
  read:
    run: /home/bee/cwl2/read.cwl
    in:
      x: dataSetPath
    out:
      - output_array
  preprocess:
    run: /home/bee/cwl2/preprocess.cwl
    { scatter: data_column_file }
    in:
      x: read/output_array
    out:
      - output_preprocessed_array
```

**read.cwl**

```cwl

cwlVersion: v1.0
class: CommandLineTool
baseCommand: ["python", "/home/bee/cwl2/finalread.py"]

inputs:
  x:
    type: string
    inputBinding:
      position: 1

stdout: output.txt

outputs:
  output:
    type:
      type: array
      items: File
    outputBinding:
      glob: ".*\.txt"
```

- Reads dataset and outputs data in each column as its own file
  - Number of columns unknown
- Scatters the output array for preprocessing
Updated BEE Workflow Model
Updated BEE Workflow Model – Data Structures

- Workflow
  - UUID
  - Name
  - Inputs
  - Outputs
  - State

- Task
  - UUID
  - Name
  - Command
  - Subworkflow
  - Inputs
  - Outputs
  - State

- WorkflowHints
  - Hints

- WorkflowRequirements
  - Requirements

- TaskHints
  - Hints

- Tasks created/added through workflow interface
- Workflow node points to first task of workflow
- Hints and requirements stored in own nodes
- Related to tasks and workflow by HAS_HINT and HAS_REQUIREMENT relationships
- Dependencies modeled by DEPENDS_ON relationships
Updated BEE Workflow Model – Pseudo-Tasks

- PseudoTask
  - UUID
  - Name
  - Command
  - Subworkflow
  - Abstract Inputs
  - Outputs

- PseudoTasks are created for tasks whose inputs are not known a priori
  - Dependency relations to and from PseudoTasks modeled as ABSTRACT_DEPENDS_ON relationships
  - Expand into as many tasks as required to handle each input
  - Real outputs are returned to Workflow Manager to expand PseudoTasks
$ \text{match (n) return n}$

```
* (11) Workflow (1) Task (9) Workflow Requirement (1)
* (14) HAS_REQUIREMENT (1) DEPENDS_ON (13)
```

**Graph: Scatter Gather**

- Scatter
  - Workflow
  - ScatterF...
- Read
- Preproc...
- Regress
- Predict
- OutputA...
- Gather
  - Preproc...
  - DEPENDS_ON
  - DEPENDS_ON
  - DEPENDS_ON
  - DEPENDS_ON
  - DEPENDS_ON
```
Conclusion

• BEE is a powerful tool for:
  • Managing and visualizing scientific workflows
  • Simplifying workflow execution on HPC and cloud platforms
• BEE supports much of the CWL specification
• Did not support execution of complex "scattering" workflows
• By introducing the PseudoTask:
  • Can generate tasks to run on variable number of inputs
  • BEE is another step closer to supporting the entire CWL specification
  • BEE can now support parallelized workflows with scattering tasks
Further Work

• Add support for embedded Javascript or Python expressions in CWL

• Add support for nested workflows in CWL
Questions?

Over 70 years at the forefront of supercomputing