BeeCWL: A CWL Compliant Workflow Management System

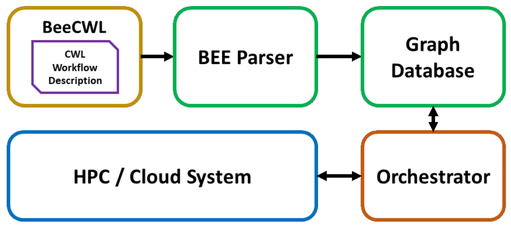
Betis Baheri, Steven Anaya, Patricia Grubel, Qiang Guan and Timothy Randles

Kent State University, Los Alamos National Laboratory, New Mexico Tech

Motivation

- **Standardization**: Adapting one standard on HPC/Multi-cloud system
- **BeeCWL**: Supports Common Workflow Language (CWL) in Build and Execution Environment (BEE)
- **Analysis & Optimization**: Optimize, extract and analyze workflow information
- **Diverse Execution**: HPC/Multi-cloud system cross-compatibility

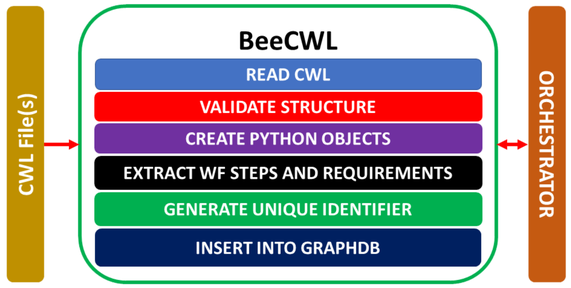
BeeCWL Overview



Overall process of BeeCWL

- **CWL Workflow Description**: Contains requirements and steps for executing workflows.
- **BEE Workflow**: Existing BEE workflow Management System on HPC or cloud system.
- **BEE Parser**: CWL Parser extracts meaningful information such as requirements and sub-workflow steps.
- **Graph Database**: Creates graph database based on workflow steps and dataflow.
- **BEE Orchestrator**: Responsible for scheduling, executing and managing the workflow tasks. Each workflow can be split into one or multiple sub-workflows to execute on different HPC environments or cloud systems.

Parser Overview



Extended view of BeeCWL parser

- **BeeCWL.Reader**: Responsible for opening single or multiple CWL files
- **BeeCWL.Validator**: Responsible for validating input files with CWL standards
- **BeeCWL.Builder**: Responsible for creating usable python objects from CWL files
- **BeeCWL.Decomposer**: Responsible for separating individual components (e.g. requirements, steps, hints, etc.)
- **BeeCWL.GraphDB**: Creates graph database with maintaining workflows structure and generates links between one step to another.

Algorithm 1 BeeCWL Parser

```
Given initial CWL location.
for x in CWL do
    BeeCWL.Open(x);
    BeeCWL.ValidateCWL(x);
    BeeCWL.Parse(x);
    BeeCWL.Create.DataObject(x);
    model = [BeeCWL.Extract(x).Steps, BeeCWL.Extract(x).requirements]
    BeeCWL.GraphDB(model);
    Bee.Orchestrator(GraphDB(model).identifier)
end for
```

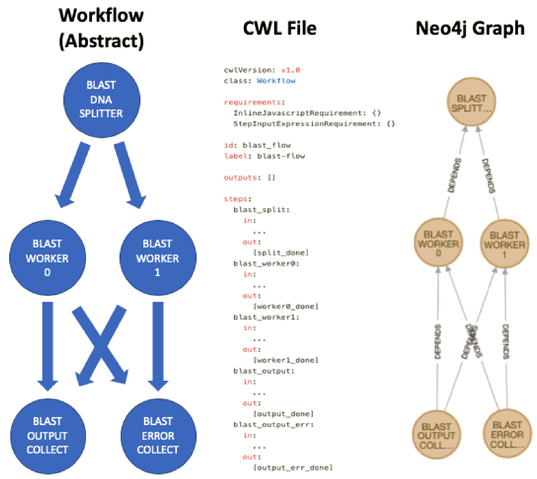
The Algorithm shows how BeeCWL parses and validates CWL files. After parsing, each component of the CWL file would be inserted into the graph database. A unique identifier will be passed on to the Orchestrator.

Graph DB Overview

- **Storage**: Graph data/metadata in one database
- **Format**: Data stored as nodes/edges
- **Workflows**: Natural mapping to graphs

Neo4j Implementation

- **Neo4j**: Transactional graph database
- **Implementation**: Store workflow steps as nodes; dependencies as edges; metadata as *properties*



A workflow is written in CWL and loaded into Neo4j

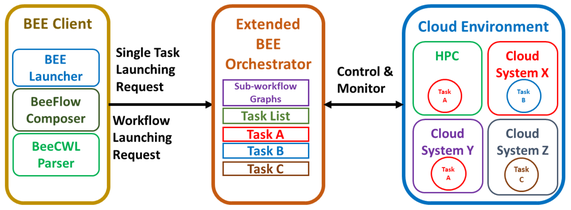
- **Language**: Cypher Query Language

```
MATCH (s:Task), (t:Task)
WHERE s.task_id = dependent_id AND t.task_id = dependency_id
CREATE (s)-[:DEPENDS]→(t)
```

Creating a step (task) dependency in Cypher

- **Visualization**: View graph in a browser
- **Manipulation**: Query in a browser or Cypher shell
- **Python API**: Python 3 driver for DB transactions

Future Work



Example: Orchestrator managing tasks on several platforms

Orchestrator would assign one or more sub-workflows to HPC or cloud system based on available resources. System logs and workflow graphs can be used to maximize optimization and scheduling on HPC and cloud systems. By using a Graph DB, the Orchestrator will be able to restart or partially run sub-workflows.

Conclusion

In this work, we first address the definition of Common Workflow Language and importance of supporting it in existing BEE System. We show the details of implementation of BeeCWL and the behavior of the entire system. By supporting the CWL standard, users can easily run single or multiple workflows on different HPC and cloud systems.

Contact Information

- Department of Computer Science, Kent State University
- Los Alamos National Laboratory
- bee-dev@lanl.gov

LA-UR-19-27926