BeeCWL: A CWL Compliant Workflow Management System

Extended Abstract

Betis Baheri, Qiang Guan
Kent State University
Kent, Ohio
{bbaheri,qguan}@kent.edu

Steven Anaya
New Mexico Tech
Socorro, New Mexico
steven.anaya@student.nmt.edu

Patricia Grubel, Timothy Randles
Los Alamos National Laboratory
Los Alamos, New Mexico
{pagrubel,trandles}@lanl.gov

ABSTRACT

Scientific workflows are used widely to carry out complex and hierarchical experiments. Although there are many trends to extend the functionality of workflow management systems to cover all possible requirements that may arise from a user community, one unified standard over cloud and HPC systems is still missing. In this paper, we propose a Common Workflow Language (CWL) compliant workflow management system. BeeCWL is a parser to derive meaningful information such as requirements, steps, relationships, etc. from CWL files and to create a graph database from those components. Generated graphs can be passed to an arbitrary scheduler and management system to decide whether there are enough resources to optimize and execute the workflow. Lastly, the user can have control over workflow execution, collecting logs, and restart or rerun some part of a complex workflow.

1 INTRODUCTION

Over the last few years Scientific Workflow Management Systems (SWMS's) have become valuable tools to carry out complex scientific experiments.[4] Active research in scientific workflow management has resulted in a large number of systems that can be used by scientists in practice, addressing a large variety of scientists' needs. Current workflow management systems offer generic services to handle distribution, monitoring and fault-tolerant distributed computations on various types of platforms. [5, 6] Although executing workflows on cloud systems and HPC has been improved, and many services offer various capabilities, the lack of standardization and one unique format to execute on HPC and multiple cloud systems is yet to be investigated.

A Workflow is an analysis task represented by a directed graph describing a sequence of operations that transform an input data set to output. This specification defines the Common Workflow Language (CWL) workflow description, a vendor-neutral standard for representing workflows intended to be portable across a variety of computing platforms. An object is a data structure equivalent to the "object" type in JSON, consisting of an unordered set of name/value pairs.[1] CWL has been used by the Genome Community and proven to be successful to describe and execute workflows. BeeCWL adds standardized workflow descriptions to our previous version of the Build and Execution Environment (BEE)[2]. BEE[3] is an in situ analysis enabled workflow management system that includes the following features:

- **In Situ Support** BEE[3] supports in situ dependencies between tasks in addition to traditional offline dependencies.
- **In Situ Dataflow and Control Design** BEE[3] is designed to support most kinds of dataflow between tasks in modern in situ workflows, including filesystem-based sharing and network-based sharing.
- **Container Support** BEE[3] is a Docker container supported universal execution framework.
- **Multiple Platform Support** Supporting Docker and other container environments, BEE[3] can be easily configured to run on multiple platforms and easily switched between HPC and cloud platforms (e.g., Amazon Web Services (AWS)).

By using CWL standard[1] for workflows, BEE[3] can derive requirements and steps and optimize and run one or multiple sub-workflows on HPC or cloud systems simultaneously. To achieve such functionality, CWL workflow descriptions have to be decomposed. BeeCWL is a unique parser that can decompose proper information (e.g. requirements, workflow steps, container dependency, etc.) from CWL description and create a Graph Database.

2 DESIGN AND IMPLEMENTATION

2.1 BeeCWL System Overview

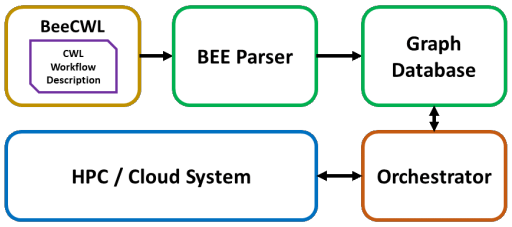

Figure 1: BeeCWL System Overview

The basic idea of the BeeCWL parser is to evaluate CWL descriptions and extract workflow requirements and steps before executing them. By using the extracted information, pre-validation and system compatibility checks can be done in early stages. The overall system overview is shown in figure 1.

SC19, November 2019, Denver, Colorado USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

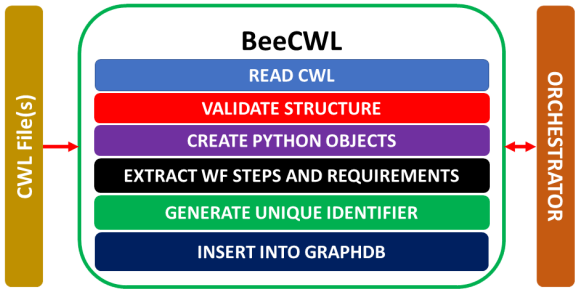

Figure 2: BeeCWL Detailed Overview

The design process follows a simple principle of separating individual components based on system resources and the workflow's steps. The user will provide a CWL file or directory containing the information about the workflow steps, input and output, the required packages and tools to run. The BeeCWL parser has six steps:

- **BeeCWL.Reader** Reads the provided CWL description from a file or directory
- **BeeCWL.Validator** Validates the provided CWL files against CWL standard
- **BeeCWL.Builder** Generates Python objects from CWL description
- **BeeCWL.Decomposer** Extracts meaningful information such as requirements, workflow steps, input, output, etc.
- **BeeCWL.GraphDB** Creates a graph database based on extracted information

By following these steps, BeeCWL can detect flaws at early stages before running the workflows. By using a graph database, proper pre-processing can be done to avoid unnecessary execution. After generating a unique graph for each workflow, an identifier will pass to the BEE[3] orchestrator. The orchestrator is responsible for managing, optimizing and scheduling each task based on available resources. A detailed view of BeeCWL is shown in figure 2.

2.2 Graph Database

We choose Neo4j as the graph database for our implementation of BEE. Neo4j is a transactional graph database that supports the storage of data as nodes and relationships (labeled edges) with metadata as properties of these nodes and relationships.

Once BeeCWL extracts all the workflow information, a Workflow object is created, with each step stored as a Task object; its graph representation and accompanying metadata are stored in the graph database. In the BEE implementation, the workflow steps are stored as Task nodes, while dependencies between steps, determined by the dataflow, are stored as relationships between nodes. The rest of the workflow data and metadata—including inputs, outputs, commands, and requirements—are stored as node properties.

After the workflow is loaded into the graph database, a graphical depiction of the workflow may be displayed for the user to verify that the workflow, as defined by the parsed CWL file, is not malformed or misconfigured. Once the user verifies that the workflow is correct, the workflow execution will initiate.

During workflow execution, information about the workflow can be extracted from the database in order to verify task dependencies or requirements. Additionally, a sub-workflow, whose tasks and dependencies comprise a subset of the larger workflow, can be extracted from the graph database. This sub-workflow may then be transferred to another instance of the BEE orchestrator and loaded into a separate graph database to facilitate parallel or cross-platform workflow execution. Each BEE user's workflow resides in its own graph database instance, simplifying workflow data storage and archive management.

3 FUTURE WORK

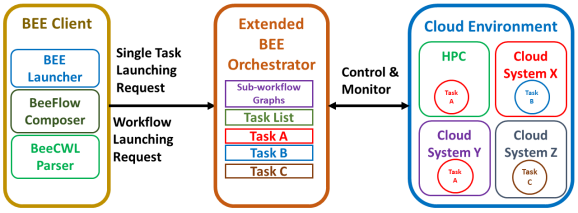

Figure 3: Orchestrator Overview

In our future work, the information provided by BeeCWL and the graph database will be used in the orchestrator to schedule and optimize the HPC or cloud system resources. Since every graph resides in a unique database instance, and the workflow descriptions provided in CWL are broken down into sub-workflow steps, restarting individual sub-workflows and changing the input and output for specific sub-workflows is possible. The orchestrator can check if the target system has enough available resources or meets the requirements to execute an arbitrary workflow. Users can run their workflows on both HPC and cloud systems, and the orchestrator can optimize based on analysis of generated graphs.

4 CONCLUSION

In this work we first address the definition of Common Workflow Language and the importance of supporting it in the existing BEE system. We illustrate details of the implementation of BeeCWL and the behavior of the entire system. Then we show by using the CWL standard for scientific workflows and creating a graph database, based on sub-workflows and requirements, the user can check and change individual parts of a workflow to process on HPC or cloud systems.

Finally, by using a containerized approach and separating sub-workflow steps, BEE is able to run and execute on any desired target. The system can analyze generated graphs and schedule and optimize workflow tasks accordingly.

ACKNOWLEDGMENTS

This publication has been assigned the LANL identifier LA-UR-19-27927.

REFERENCES

[1] Peter Amstutz, Michael R. Crusoe, Nebojsa Tijanic (editors)and Brad Chapman, John Chilton, Michael Heuer, Andrey Kartashov, Dan Leehr, Herve Menager, Maya Nedeljkovich, Matt Scales, Stian Soiland-Reyes, and Luka Stojanovic. 2019. Common Workflow Language. In *Common Workflow Language, v1.0. Specification, Common Workflow Language working group.* https://doi.org/10.6084/m9.figshare.3115156.v2

[2] J. Chen, Q. Guan, X. Liang, P. Bryant, P. Grubel, A. McPherson, L. Lo, T. Randles, Z. Chen, and J. P. Ahrens. 2018. Build and Execution Environment (BEE): an

Encapsulated Environment Enabling HPC Applications Running Everywhere. In *2018 IEEE International Conference on Big Data (Big Data)*. 1737–1746. https://doi.org/10.1109/BigData.2018.8622572

[3] J. Chen, Q. Guan, Z. Zhang, X. Liang, L. Vernon, A. McPherson, L. Lo, P. Grubel, T. Randles, Z. Chen, and J. Ahrens. 2018. BeeFlow: A Workflow Management System for In Situ Processing across HPC and Cloud Systems. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. 1029–1038. https://doi.org/10.1109/ICDCS.2018.00103

[4] S. Olabarriaga, G. Pierantoni, G. Taffoni, E. Sciacca, M. Jaghoori, V. Korkhov, G. Castelli, C. Vuerli, U. Becciani, E. Carley, and B. Bentley. 2014. Scientific Workflow Management – For Whom?. In *2014 IEEE 10th International Conference on e-Science*. 298–305.

[5] S. Shumilov, Y. Leng, M. El-Gayyar, and A. B. Cremers. 2008. Distributed Scientific Workflow Management for Data-Intensive Applications. In *2008 12th IEEE International Workshop on Future Trends of Distributed Computing Systems*. 65–73. https://doi.org/10.1109/FTDCS.2008.39

[6] J. Zhang, P. Votava, T. J. Lee, O. Chu, C. Li, D. Liu, K. Liu, N. Xin, and R. Nemani. 2013. Bridging VisTrails Scientific Workflow Management System to High Performance Computing. In *2013 IEEE Ninth World Congress on Services*. 29–36. https://doi.org/10.1109/SERVICES.2013.64