BEE: A Scientific Application Workflow Engine

Manage Multi-step Simulations on HPC & Cloud Platforms

Patricia Grubel, Quincy Wofford,
Rusty Davis

WoWoHa 2020 Summer Webinar
June 26, 2020

LA-UR-20-24727

Outline

- BEE Overview
- Charliecloud – Lightweight Container Runtime
- BEE Components & Internals
- Neo4j Graph Database
  - Managing Workflows
  - Archiving for recoverability, re-running, cloning
- HPC Scientific Workflows
- Example - CLAMR
- BEEstart – starts up the server components

BEE: A Scientific Application Workflow Engine

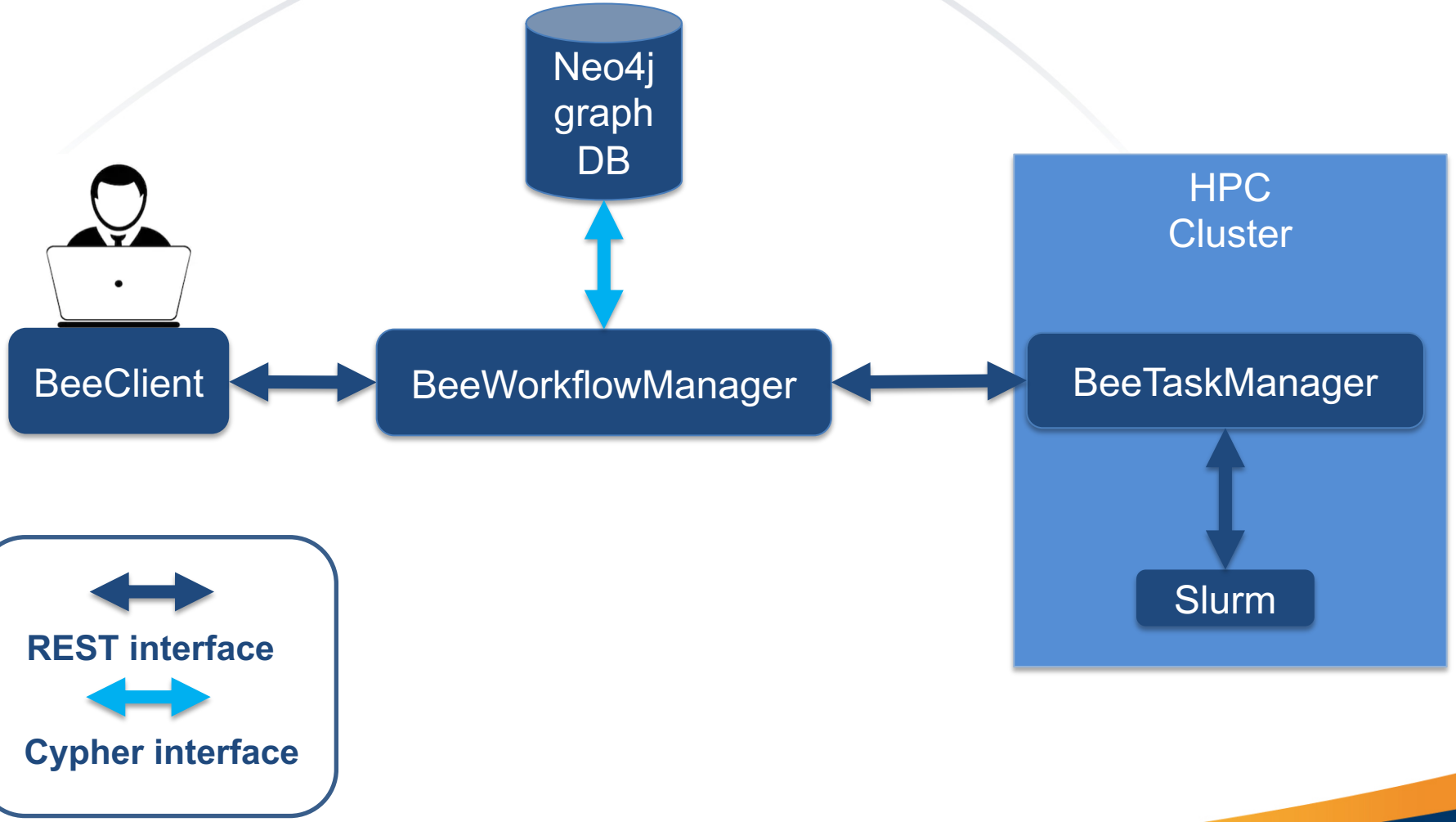
- Docker Image Support
  - Configurable support for HPC - Charliecloud, Shifter and Singularity Container Runtimes
- Supports Multiple HPC Clusters
  - Enables high resource usability
- Designed for HPC simulations
- Standard Workflow Specification: Common Workflow Language
- Automation
  - Platform-related setup, configuration and launching
  - Avoid learning arcane technical details of HPC resource managers
- No privileged access required
  - Any user can use on HPC platform of their choice
- Reproducible
  - Complex scientific workflows can be archived, share metadata, re-run
- Contact

  bee-dev@lanl.gov

Charliecloud: HPC Container Runtime

- Lightweight
  - No complex installation
  - No configuration required
  - ~1300 Lines of Code (C & shell scripts)
- Unprivileged
  - Any user can run it
  - No HPC center admin intervention required
- Linux kernel support
  - All major distributions have kernel support for user namespaces
  - Some require a tweak (e.g. /proc/sys/user/max_user_namespaces)
- Hardware agnostic
  - Tested and supported on X86_64, ARM64, PowerPC
- Available today
  - https://github.com/hpc/charliecloud
  - Packages for all major Linux distributions
  - OpenHPC release

BEE: Components for HPC



BEE: Internals

- Python 3
  - Portable across Linux, OS X, Windows
  - Modular Design – Abstract Classes for Major Components
    - Support for using different Graph Database
    - Support for multiple Container Runtimes
    - Support for Mulitple Workload Managers (Slurm, LSF, PBS, Torque/Moab…)
- REST and YAML
  - Easy to enhance and extend
- Common Workflow Language
  - Open standard, many tools already exist
  - Python for expressions instead of JavaScript
  - BEE extensions for better HPC support
    - Automatic Setup of HPC Requirements
    - HPC Container Runtimes, Charliecloud, Singularity etc.
- Neo4j - Graph database
  - Manage workflow and metadata
  - DAG allows workflow execution optimize
  - Archive workflow & artifacts

Neo4j - Graph Database

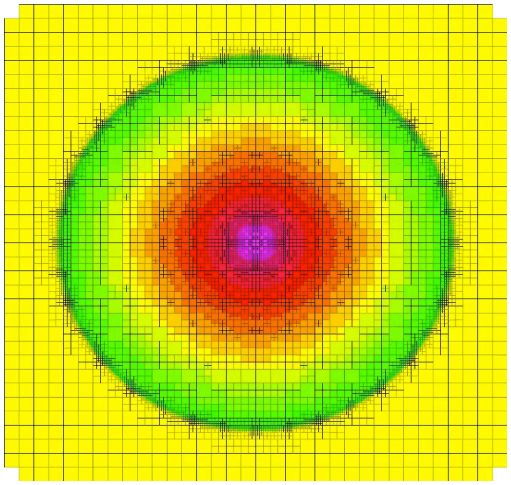
- Manage Workflow
  - Build Workflow DAG
  - Visualize DAG
  - Metadata during run – task state, job id
- Archive Workflow
  - Workflow metadata - what cluster, cluster job ids, times (submit, start, compute)
  - Provenance - ability to archive the workflow
    - Captures container UUID, input decks, run commands, checkpoint file location
    - Rerun workflows
    - Clone workflows - copy, reset data and go
  - Resiliency – true state in BEE is in the database
    - Recovery from outages
    - Component restart - components designed to continue using database metadata
  - Checkpoint / Restart

HPC Scientific Workflows - LANL

- Multi-Physics Simulation
  - Computations can last months
  - Checkpoint / Restart Capability is paramount
  - Repeatability
- Biology
  - Common Workflow Language
  - BEE will facilitate using HPC resources and containers
- Physics - Large Scale Parametric Studies
  - 1000's of Simulations
  - Diverging solutions can cause unneeded billing cycles
  - Facilitate ability to cancel, change input/parameters, then restart

CLAMR Workflow Example

- CLAMR
  - A LANL mini-app
  - Simulates shallow water equations
  - Performs hydrodynamic cell-based adaptive mesh refinement (AMR)
  - Intended as a testbed for hybrid algorithm development using MPI and OpenCL



CLAMR Visualization

CLAMR Workflow CWL

```
cwlVersion: v1.0

requirements:
  CommandLineRequirement: { }

id: clamr-flow
label: clamr-flow

steps:
   clamr:
    in:
      …
    run: /clamr/CLAMR-master/clamr_cpuonly …

    hints: DockerRequirement
        dockerImageID:  "…/clamr.tar.gz"

    out: [ graphics_output/ ]

  ffmpeg :
    clamr:
     in:
      …
     run: / ffmpeg -f image2 –i graphics_output/ ...
     out: [ CLAMR_movie.mp4  ]
```

CLAMR Job Scripts

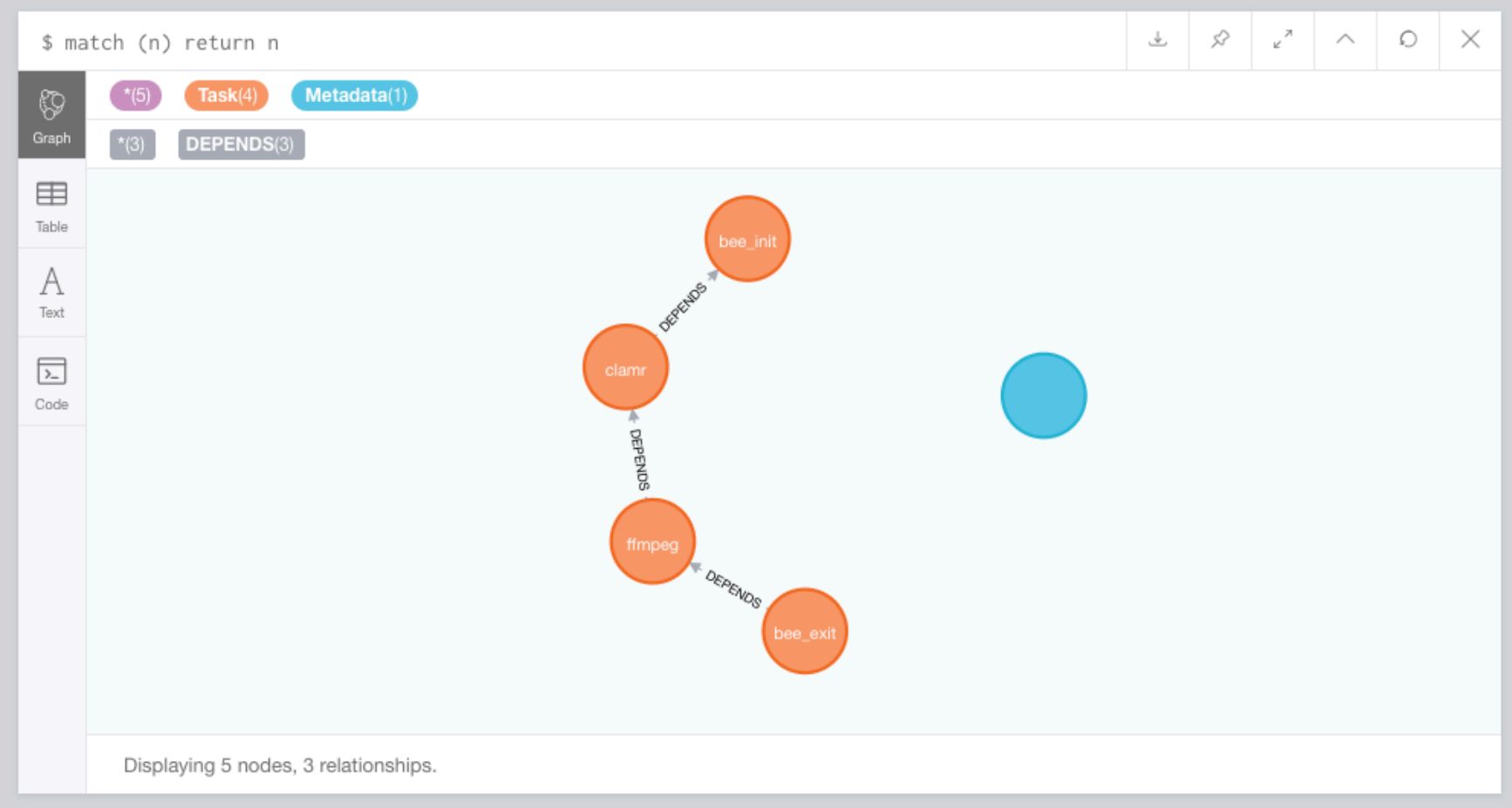
```
#! /bin/bash
#SBATCH
module load charliecloud

mkdir -p /tmp/$USER

ch-tar2dir /usr/projects/beedev/clamr/clamr-toss.tar.gz /tmp/$USER

ch-run /tmp/$USER/clamr-toss -b $PWD -c /mnt/0 -- /clamr/CLAMR-master/clamr_cpuonly -n 32 -l 3 -t 5000 -i 10 -g 100 -G png rm -rf /tmp/$USER/clamr-toss
```

CLAMR Job Step

```
#! /bin/bash
#SBATCH
ffmpeg -f image2 –i graphics_output/graph%05d.png -r 12 -s 800x800 -pix_fmt yuv420p CLAMR_movie.mp4
```

ffmpeg Job Step

CLAMR neo4j Workflow



BEE Configuration File

- Orchestrates BEE components

```
# BEE CONFIGURATION FILE #
[DEFAULT]
bee_workdir = $HOME/.config/beeflow


[slurmrestd]
slurm_socket = /tmp/slurm_$USER_154.sock


[graphdb]
hostname = localhost
dbpass = password
bolt_port = 7741
http_port = 7528
https_port = 7527
gdb_image = /usr/projects/beedev/neo4j-3-5-17.tar.gz
gdb_image_mntdir = /tmp


[workflow_manager]
listen_port = 5054


[task_manager]
listen_port = 5104
```

BEEstart

- Manages configuration file complexity
- Launches BEE component servers