ExaWorks SDK

Release 1.0

ExaWorks Team

Apr 21, 2023

CONTENTS

ExaWorks **Software Development Kit (SDK)** offers: (1) packaging for a curated set of workflow software systems; (2) testing of those systems on a number of high performance computing (HPC) platforms managed by the USA Department of Energy (DoE); and (3) tutorials about coding workflow applications with those workflow systems on DoE HPC platforms.

Exaworks SDK supports the workflows needs of diverse users, administrators, and developers. It enables teams to produce scalable and portable workflows for a wide range of exascale applications. SDK does not replace the many workflow solutions already deployed and used by scientists, but rather it provides a packaged, tested and documented collection of community-identified components that can be leveraged by users. SDK contributes to enabling a sustainable software infrastructure for workflows, supporting diverse scientific communities on a variety of DoE HPC platforms.

Currently, ExaWorks SDK offers Docker containers and Spack packages for Flux, Parsl, PSI/J, RADICAL-Cybertools, and Swift/T. Each package is deployed and tested on a growing number of DoE HPC platforms. Applications teams can draw from SDK's tutorials, levaraging containers and packages as needed to develop application workflows.

Note: This project is under active development.

CHAPTER

ONE

CORE COMPONENTS

The SDK has four core components but it is open to the contribution of any system that support the execution of scientific workflows on the Department of Energy high performance computing platforms.

- Flux. Workload management system (à la Slurm, PBS, LSF), with single-user and multi-user (a.k.a. system instance) modes.
- Parsl. Pure Python library for describing and managing parallel computation on one to many nodes. It contains abstractions to support various parallel and distributed workloads: from bag-of-tasks to dataflow, short to long duration tasks, single core through to multi-node.
- PSI/J. The Portable Submission Interface for Jobs is a Python abstraction layer over cluster schedulers. A number of executors and launchers allow PSI/J to communicate with specific job schedulers.
- RADICAL-Cybertools. RADICAL-EnsembleToolkit (EnTK) and RADICAL-Pilot (RP) are middleware architected for scalability, interoperability and sustainability. Implemented as Python modules, they support the execution of scientific workflows and workloads on a range of high-performance and distributed computing platforms.
- Swift/T. Swift/T is an MPI-based workflow language and runtime system. It runs in a one big job model, with internal automatic task parallelization and data movement, enhanced by workflow-level compiler optimizations.

Each core component can be indipendently installed by following the instructions of each component's documentation. Exaworks SDK curates the containarization of each component and its Spack packaging.

1.1 Containers

ExaWorks SDK packages are available via container from Docker hub. The following code shows how to access the SDK container image locally without installation.

```
docker pull exaworks/sdk
docker run -it exaworks/sdk bash
```

The following code shows how to use the container images to run the notebook tutorials. Note that the specific notebook you want to run may have some additional prerequisites.

```
docker run -p 8888:8888 -v path/to/notebooks:/notebooks -it exaworks/sdk bash
pip install jupyter
cd /notebooks
jupyter notebook --allow-root --ip 0.0.0.0 --no-browser
```

1.2 Spack packages

ExaWorks SDK packages are packed together into Spack `exaworks` package. The following code shows its installation within a corresponding Spack environment

```
spack env create exaworkssdk
spack env activate exaworkssdk
spack install exaworks
```

If Spack is not in the system, then it could be installed manually

```
git clone https://github.com/spack/spack.git
. spack/share/spack/setup-env.sh
```

Steps for package creation are provided in Packaging section. For additional information please refer to the Spack documentation.

CHAPTER

TWO

TUTORIALS

In the following, we offer a brief tutorials for how to write an *hello_world* application with each Exaworks SDK core component, how to prepare and run Exaworks SDK Docker container, and how to Exaworks SDK tests. We also offer details that might be useful for developers that would like to contribute to Exaworks SDK.

2.1 Running the Tutorials

Tutorials can be run via our self-contained Docker container or independently. When run independently, the user has to setup a suitable running environment for each tutorial. That can be technically demanding and it requires referring to the documentation site of each component.

To run the tutorials in the ExaWorks SDK Docker container:

1. clone the ExaWorks SDK repository:

   ` git clone https://github.com/ExaWorks/SDK.git `

2. Follow the instructions in SDK/docker/tutorials/README.md, choosing one of the three methods A, B or C to execute your container. Note that if you want to run the RADICAL-Cybertools tutorial, you will have to chose either B or C.

3. After following the instructions, you will be given a URI to cut and paste in your browser to access to the Jupyter Notebook server that is running in the SDK container.

4. Load and execute each tutorial in the Jupyter Notebook server on your browser.

5. Once finished, stop the SDK container and, in case, the MongoDB and RabbitMQ containers you started to execute the RADICAL-Cybertools tutorial.

2.2 SDK Tutorials

2.2.1 ExaWorks SDK Container Image

The ExaWorks SDK is available in a containerized format on dockerhub. This docker image is a great place to start if you want to get familiar with the workflow tools comprising the SDK without the overhead of a full install.

Preparing the Conatiner Environment

As we will be executing this tutorial within a container, we must first ensure that the docker daemon is running. This is sytem dependent so see documentation for your specific system. If you wish to run this note book directly, note that it does require a bash Kernal for jupyter. You can install a bash Kernal into your python virtual environment by running:

```
pip install bash_kernel
python -m bash_kernel.install
```

Running the ExaWorks Container

After preparing your enviromnment, you can pull the SDK Image.

[19]:
```
docker pull exaworks/sdk
```

```
Using default tag: latest
latest: Pulling from exaworks/sdk
Digest: sha256:f278e43866f4e1a1da9b7d0d98f433ca88e0a598c504c2f7d3831690195d64a4
Status: Image is up to date for exaworks/sdk:latest
docker.io/exaworks/sdk:latest
```

After pulling the image, you can run arbitrary commands within the container.

Note that in this tutorial, we often run each command as the argument for the `docker run`. This means that no progress or state is saved between commands because the command is run on a new container based on the SDK image everytime. We use the `--login` flag because a lot of the environment is initialiazed through the `.bashrc`. Without that flag many of the packages would not work. This tutorial uses the `docker run` because of the nature of the jupyter notebook running it, and is some instances code snippets will be placed in the Markdown sections to show more complicated actions.

It is recommended that instead of using `docker run` for every command, that you generate an interactive docker run using :

`docker run -it exaworks/sdk bash`

This will give you a shell within the container the ecexute all of the commands that fall under the `-c <command>` flag.

[17]:
```
echo Flux Version:
docker run -t exaworks/sdk bash --login -c "flux -V"
echo

echo Parsl Version:
docker run -t exaworks/sdk bash --login -c "python -c 'import parsl; print(parsl.__
↪version__)'"
echo

echo Radical Pilot Version:
docker run -t exaworks/sdk bash --login -c "radical-pilot-version"
echo

echo Swift-t Version:
docker run -t exaworks/sdk bash --login -c "swift-t -v"
```

```
Flux Version:
commands:               0.28.0
```

(continues on next page)

(continued from previous page)

```
libflux-core:           0.28.0
build-options:          +hwloc==1.11.0

Parsl Version:
1.3.0-dev

Radical Pilot Version:
1.11.2

Swift-t Version:
STC: Swift-Turbine Compiler 0.9.0
         for Turbine: 1.3.0
Using Java VM:    /usr/bin/java
Using Turbine in: /opt/swift-t/turbine

Turbine 1.3.0
 installed:    /opt/swift-t/turbine
 source:       /tmp/build-swift-t/swift-t/turbine/code
 using CC:     /usr/local/bin/mpicc
 using MPI:    /usr/local/lib mpi "OpenMPI"
 using Tcl:    /opt/tcl-8.6.11/bin/tclsh8.6
```

Running the Tests

Each workflow tool has a set of tests located at `/tests/<packagename>/test.sh`.

[1]:
```
echo Flux Tests:
docker run -t exaworks/sdk bash --login -c "bash /tests/flux/test.sh" | head -n 7
echo "..."
```

```
Flux Tests:
Cloning into 'flux-core'...
remote: Enumerating objects: 90454, done.
remote: Counting objects: 100% (7455/7455), done.
remote: Compressing objects: 100% (2720/2720), done.
remote: Total 90454 (delta 5113), reused 6693 (delta 4716), pack-reused 82999
Receiving objects: 100% (90454/90454), 40.30 MiB | 13.47 MiB/s, done.
Resolving deltas: 100% (67270/67270), done.
write /dev/stdout: broken pipe
...
```

[21]:
```
echo Parsl Tests:
docker run -t exaworks/sdk bash --login -c "bash /tests/parsl/test.sh"
```

```
Parsl Tests:
Hello World from Python!
Hello World!

Output matches
```

[1]:
```
echo Radical Pilot Tests:
docker run -t exaworks/sdk bash --login -c "bash /tests/rp/test.sh" | head -n 7
echo "..."
```

```
Radical Pilot Tests:
--- start MongoDB
about to fork child process, waiting until server is ready for connections.
forked process: 26
child process started successfully, parent exiting

================================================================================
write /dev/stdout: broken pipe
--- smoke test
...
```

[2]:
```
echo Swift-t Tests:
docker run -t exaworks/sdk bash --login -c "bash /tests/swift/test.sh" | head -n 7
echo "..."
```

```
Swift-t Tests:
+ [[ openmpi == \o\p\e\n\m\p\i ]]
+ export TURBINE_LAUNCH_OPTIONS=--allow-run-as-root
+ TURBINE_LAUNCH_OPTIONS=--allow-run-as-root
+ swift-t -v
STC: Swift-Turbine Compiler 0.9.0
         for Turbine: 1.3.0
Using Java VM:    /usr/bin/java
write /dev/stdout: broken pipe
...
```

Running the Tutorial Notebooks

As of now, jupyter is not automatically included in the SDK container image, but we can easily install it! First, we have to run our container while exposing a port and mounting the directory that contains jupyter notebooks. The note books are not currently a part of the container image, so we need to make them accessible from within the conatiner using the `-v` flag. We also need to specify that we want the jupyter server to resolve on the local host at the default jupyter port. We do this by mapping the port from the host machine to the contianer with `-p 8888:8888`, and specifying the localhost ip when starting the jupyter server.

```
$ docker run -p 8888:8888 -v $(path/to/notebooks):/notebooks -it exaworks/sdk bash
```

You can then install and run jupyter.

```
# pip install jupyter
# cd /notebooks
# jupyter notebook --allow-root --ip 0.0.0.0 --no-browser
```

Then just copy the URL to your browser to view and run the notebooks. The other notebooks may have some additional prerequisites and configuration required before they can be run.

SDK Image Tags

As a part of our CI/CD pipeline, we build the SDK with multiple build parameters, including different base operating systems, python versions, mpi flavors, and package managers. To organize these different builds, we use tags to distinguish them. When selecting an image, you can select a specific tag for a specific build spec that you want to test. The tag works as follows: <os>_<package_manager>_<mpi_flavor>_<pyhton_version>. Different tags can be seen here.

[5]:
```
docker pull exaworks/sdk:ubuntu2004_pip_openmpi_3.8
docker run -t exaworks/sdk:ubuntu2004_pip_openmpi_3.8 bash --login -c "python -V"
```

```
ubuntu2004_pip_openmpi_3.8: Pulling from exaworks/sdk
Digest: sha256:86dee9aaa13aa21715b203594530722 0e560fc0141d7a08166f7bbcc4257fbed
Status: Image is up to date for exaworks/sdk:ubuntu2004_pip_openmpi_3.8
docker.io/exaworks/sdk:ubuntu2004_pip_openmpi_3.8
Python 3.8.10
```

SDK Base Image

When building the SDK container image, we first create a minimum build base image the contains all of the dependencies for the sdk. This base image can be a great start if you want to work through building the rest of the SDK manually or just a subset of the packages. The base image can be found here. The base imge also follows the same tagging conventions as the full SDK image

[7]:
```
docker pull exaworks/sdk-base
```

```
Using default tag: latest
latest: Pulling from exaworks/sdk-base
Digest: sha256:a40f6220a540b9e1e80250b0cdcc88503a9324d86f5db64102f5bb1dd2e9de9b
Status: Image is up to date for exaworks/sdk-base:latest
docker.io/exaworks/sdk-base:latest
```

Development on the SDK Container Image

The ExaWroks SDK is an opensource project, and we encourage community engagment and development on it. This includes development on the SDK container image. Be sure to checkoout our contribution guidelines and best practices before makeing changes!

An Overview of the Build Process

Base Image

As mentioned above, the first stop in the build process is to create a minimal build base image with all of the dependencies for the SDK. This is currently split into three different possible base dockerfiles, one for rockylinux8, on for centos7, and one for ubuntu20.04. Each of these docker files uses a combination of the os specific package manager along with a set of shared build scripts to install the dependencies.

The base image is where are the different build parameters are specified. While the os determines which dockerfile the image is built from, the other build parameters are passed in during the build process. While the goal of the build parameters is to create a large build matrix where we can test all combinations of environments in our CI pipline, several of the combinations still fail to build. Development in this area could be towards fixing the build for some combinations of build parameters or by adding new ones.

Build Parameters

1. Operating System : centos7, rockylinux8, ubuntu20.04

   See the SDK repo under docker/base/<os>

2. Package Manager: pip, conda

   See install-python-env.sh

3. MPI Flavor: opmenmpi, mpich

   See install-mpi.sh

4. Python Version: 3.7, 3.8, 3.9

   See install-python.sh or if conda, see install-python-env.sh

Workflow Tool Images

Each workflow tool is installed using its own dockerfile and any additional build scripts. Each one has an argument for base image, which sets the `FROM` line in the dockerfile. Development in this area would be to expand the tests for a specific workflow tool or to add a new tool to the SDK image.

Testing

Each workflow tool has its own set of tests, which are added to the SDK Image under `/tests/<package>/` and are inititiated by a `test.sh` in that directory. These tests give the code teams key insights on where bugs or failures might exist in their codesbases and how to fix them. Our CI pipeline runs these tests then exports the data to our testing dashbaord. These tests range from full unit and integration tests to simple sanity checks, and more additions or use cases are always welcome.

Adding a New Worflow Tool

We are encouraging community engagement and wish to expand the ExaWorks SDK with new workflow tools. To do so, we also need to expand the SDK Image. We do this by adding a new directory under docker in the SDK repo for the dockerfile and any related build scripts. All the of specifc images should be able to be built directly from the SDK Base Image or from any other SDK image. We use the build argument of `BASE_IMAGE` to set which SDK image we are building from.

```
ARG BASE_IMAGE=exaworks/sdk-base
FROM ${BASE_IMAGE}
```

Aside from just adding the build files for the new tool, be sure to add in tests as well!

Updating the CI Pipline

After adding a new tool to the SDK, also be sure to update the CI pipeline to include builds for that new workflow tool. This can be done by editing the ci.yml under the build and tests stages. During the build stage, we add new worflow tools one at a time and update the tag with the new tool being added. For Example:

```
docker build \
          -t rp_parsl_swift_flux:${{ env.DOCKER_TAG }} \
          --build-arg BASE_IMAGE=rp_parsl_swift:${{ env.DOCKER_TAG }} \
          docker/flux
```

You can see that in this part of the build process, we have already added Radical Pilot, Parsl, and Swift-t to the SDK Image, and we are currently adding in flux. The `${{ env.DOCKER_TAG }}` represents the combination of build arguments from the base image. Be sure to add any new image builds before the last one containing the integration and to update the base image for the integration build. To update the tests, simply add in the new tool in for loop.

```
for core in flux parsl rp swift flux-parsl <new-tool>
        do
          ...
        done
```

When all changes appear to pass in the ci.yml, apply those same changes to the build process in deploy.yml.

[ ]:

2.2.2 Parsl: Molecular design ML-in-the-loop workflow

This notebook demonstrates a simple molecular design application where we use machine learning to guide which computations we perform. The objective of this application is to identify which molecules have the largest ionization energies (IE, the amount of energy required to remove an electron).

IE can be computed using various simulation packages (here we use xTB); however, execution of these simulations is expensive, and thus, given a finite compute budget, we must carefully select which molecules to explore. We use machine learning to predict high IE molecules based on previous computations (a process often called active learning). We iteratively retrain the machine learning model to improve the accurat of predictions. The resulting ML-in-the-loop workflow proceeds as follows.

In this notebook, we use Parsl to execute functions (simulation, model training, and inference) in parallel. Parsl allows us to establish dependencies in the workflow and to execute the workflow on arbitrary computing infrastructure, from laptops to supercomputers. We show how Parsl's integration with Python's native concurrency library (i.e., `concurrent.futures <https://docs.python.org/3/library/concurrent.futures.html#module-concurrent.futures>`__) let you write applications that dynamically respond to the completion of asynchronous tasks.

[1]:
```
%cd /tutorials/molecular-design-parsl-demo
%matplotlib inline

from matplotlib import pyplot as plt
from chemfunctions import compute_vertical
from concurrent.futures import as_completed
from tqdm.notebook import tqdm
from parsl.executors import HighThroughputExecutor
from parsl.app.python import PythonApp
```

(continues on next page)

(continued from previous page)

```
from parsl.app.app import python_app
from parsl.config import Config
from time import monotonic
import parsl
import pandas as pd
import numpy as np
```

Define problem

We first define configuration parameters for the app, specifically the search space of molecules (selected randomly from the QM9 database) and parameters controlling the optimization algorithm (the number of initial simulations, total moleucles to be evaluated, and the number of molecules to be evaluated in a batch).

[2]:
```
search_space = pd.read_csv('data/QM9-search.tsv', delim_whitespace=True).sample(1024)  #␣
↪Our search space of molecules
```

[3]:
```
initial_count: int = 8  # Number of calculations to run at first
```

[4]:
```
search_count: int = 16   # Number of molecules to evaluate in total
```

[5]:
```
batch_size: int = 4  # Number of molecules to evaluate in each batch of simulations
```

Set up Parsl

We now configure Parsl to make use of available resources. In this case we configure Parsl to run on the local machine with two workers. One of the benefits of Parsl is that we can change this configuration to make use of different resources without modifying the following workflow. For example, we can configure Parsl to use more cores on the local machine or to use many nodes on a Supercomputer or Cloud. The Parsl website describes how Parsl can be configured for different resources.

[6]:
```
config = Config(
    executors=[HighThroughputExecutor(
        max_workers=2, # Allows a maximum of two workers
    )]
)

parsl.load(config)
```

[6]:
```
<parsl.dataflow.dflow.DataFlowKernel at 0x7fb10d7d7ac0>
```

Make an initial dataset

We need data to train our ML models. We'll do that by selecting a set of molecules at random from our search space, performing some simulations on those molecules, and training on the results.

In `chemfunctions.py <./chemfunctions.py>`__, we have defined a function `compute_vertical` that computes the "vertical ionization energy" of a molecule (a measure of how much energy it takes to strip an electron off the molecule). `compute_vertical` takes a string representation of a molecule in SMILES format as input and returns the ionization energy as a float. Under the hood, it is running xTB to perform a series of quantum chemistry computations.

Execute a first simulation

We need to prepare this function to run with Parsl. All we need to do is wrap this function with Parsl's `python_app`:

[7]:
```
compute_vertical_app = python_app(compute_vertical)
compute_vertical_app
```

[7]:
```
<parsl.app.python.PythonApp at 0x7fb0ae128970>
```

This new object is a Parsl `PythonApp`. It can be invoked like the original function, but instead of immediately executing, the function may be run asynchronously by Parsl. Instead of the result, the call will immediately return a `Future` which we can use to retrieve the result or obtain the status of the running task.

For example, invoking the `compute_verticle_app` with the SMILES for water, `O`, returns a Future and schedules `compute_verticle` for execution in the background.

[8]:
```
future = compute_vertical_app('O') #  Run water as a demonstration (O is the SMILES for␣
↪water)
future
```

[8]:
```
<AppFuture at 0x7fb0ae128820 state=pending>
```

We can access the result of this computation by asking the future for the `result()`. If the computation isn't finished yet, then the call to `.result()` will block until the result is ready.

[9]:
```
ie = future.result()
print(f"The ionization energy of {future.task_def['args'][0]} is {ie:.2f} Ha")
```

```
The ionization energy of O is 0.67 Ha
```

Scale the simulation

It is trivial now to scale our simulation and run it for several different molecules and gather their results.

We use a standard Python loop to submit a set of simulations for execution. As above, each invocation returns a `Future` immediately, so this code should finish within a few milliseconds.

Because we never call `.result()`, this code does not wait for any results to be ready. Instead, Parsl is running the computations in the background. Parsl manages sending work to each worker process, collecting results, and feeding new work to workers as new tasks are submitted.

[10]:
```
%%time
smiles = search_space.sample(initial_count)['smiles']
futures = [compute_vertical_app(s) for s in smiles]
print(f'Submitted {len(futures)} calculations to start with')
```

```
Submitted 8 calculations to start with
CPU times: user 13.1 ms, sys: 10.8 ms, total: 23.9 ms
Wall time: 21 ms
```

The futures produced by Parsl are based on Python's native "Future" object, so we can use Python's utility functions to work with them.

As an example, we can build a loop that submits new computations if previous ones fail. This happens not too infrequently with our simulation application.

We use `as_completed` to take an iterable (in this case a list) of futures and to yeild as each future completes. Thus, we progress and handle each simulation as it completes

We also use, `Future.exception()` rather than the similar `Future.result()`. `Future.exception()` behaves similarly in that it will block until the relevant task is completed, but rather than return the result, it returns any exception that was raised during execution (or `None` if not). In this case, if the future returns an exception we simply pick a new molecule and re-execute the simulation.

[11]:
```
train_data = []
while len(futures) > 0:
    # First, get the next completed computation from the list
    future = next(as_completed(futures))

    # Remove it from the list of still-running tasks
    futures.remove(future)

    # Get the input
    smiles = future.task_def['args'][0]

    # Check if the run completed successfully
    if future.exception() is not None:
        # If it failed, pick a new SMILES string at random and submit it
        print(f'Computation for {smiles} failed, submitting a replacement computation')
        smiles = search_space.sample(1).iloc[0]['smiles'] # pick one molecule
        new_future = compute_vertical_app(smiles) # launch a simulation in Parsl
        futures.append(new_future) # store the Future so we can keep track of it
    else:
        # If it succeeded, store the result
        print(f'Computation for {smiles} succeeded')
        train_data.append({
            'smiles': smiles,
            'ie': future.result(),
            'batch': 0,
            'time': monotonic()
        })
```

```
Computation for CC12COC1(C)C1OC21 failed, submitting a replacement computation
Computation for OC12CCC1NC2C#N failed, submitting a replacement computation
Computation for CC12C3N4CC(C14)C23O failed, submitting a replacement computation
Computation for O=C1CCC1NCC#N succeeded
Computation for NC(=O)NCC#CCO succeeded
Computation for CCC1CC2OC12 succeeded
Computation for CC1(C)OC(=N)C1(C)C succeeded
Computation for C1C2CC3=CCOC1C23 failed, submitting a replacement computation
Computation for C1CC2(CCOC2)CO1 succeeded
```

(continues on next page)

(continued from previous page)

```
Computation for CN1CC(N=C1)C#N succeeded
Computation for O=C1CC2CCC3C2C13 succeeded
Computation for O=C1NC2CC11CCC21 failed, submitting a replacement computation
Computation for O=CC1=CC2CCC1C2 succeeded
```

We now have an initial set of training data. We load this training data into a pandas `DataFrame` containing the randomly samples molecules alongside the simulated ionization energy (`ie`). In addition, the code above has stored some metadata (`batch` and `time`) which we will use later.

[12]:
```
train_data = pd.DataFrame(train_data)
train_data
```

[12]:

	smiles	ie	batch	time
0	O=C1CCC1NCC#N	0.492704	0	26421.812311
1	NC(=O)NCC#CCO	0.492472	0	26440.254361
2	CCC1CC2OC12	0.514080	0	26464.257546
3	CC1(C)OC(=N)C1(C)C	0.504155	0	26498.323339
4	C1CC2(CCOC2)CO1	0.491403	0	26522.577064
5	CN1CC(N=C1)C#N	0.487846	0	26548.199886
6	O=C1CC2CCC3C2C13	0.499828	0	26556.459605
7	O=CC1=CC2CCC1C2	0.503186	0	26580.873453

Train a machine learning model to screen candidate molecules

Our next step is to create a machine learning model to estimate the outcome of new computations (i.e., ionization energy) and use it to rapidly scan the search space.

To start, let's make a function that uses our prior simulations to train a model. We are going to use RDKit and scikit-learn to train a nearest-neighbor model that uses Morgan fingerprints to define similarity (see notes from a UChicago AI course for more detail). In short, the function trains a model that first populates a list of certain substructures (Morgan fingerprints, specifically) and then trains a model which predicts the IE of a new molecule by averaging those with the most similar substructures.

We want to use Parsl here to scale the model and to later combine it into our ML-in-the-loop workflow. To do so, we define the function using `python_app`. This time, `python_app` is used as a decorator directly on the function definition (earlier we defined a regular function, and then applied `python_app` afterwards).

[13]:
```
@python_app
def train_model(train_data):
    """Train a machine learning model using Morgan Fingerprints.

    Args:
        train_data: Dataframe with a 'smiles' and 'ie' column
            that contains molecule structure and property, respectfully.
    Returns:
        A trained model
    """
    # Imports for python functions run remotely must be defined inside the function
    from chemfunctions import MorganFingerprintTransformer
    from sklearn.neighbors import KNeighborsRegressor
    from sklearn.pipeline import Pipeline
```

(continues on next page)

(continued from previous page)

```
    model = Pipeline([
        ('fingerprint', MorganFingerprintTransformer()),
        ('knn', KNeighborsRegressor(n_neighbors=4, weights='distance', metric='jaccard',␣
↪n_jobs=-1))  # n_jobs = -1 lets the model run all available processors
    ])

    return model.fit(train_data['smiles'], train_data['ie'])
```

Now let's execute the function and run it asynchronously with Parsl

[14]:
```
train_future = train_model(train_data)
```

One of the unique features of Parsl is that it can create workflows on-the-fly directly from Python. Parsl workflows are chains of functions, connected by dynamic depencies (i.e., data passed between Parsl `apps`), that can run in parallel when possible.

To establish the workflow, we pass the future created by executing one function an input to another Parsl function.

As an example, let's create a function that uses the trained model to run inference on a large set of molecules and then another that takes many predictions and concatenates them into a single collection. The sequential workflow is implemented as follows.

```
train_model --> run_model --> combine_inferences
```

[15]:
```
@python_app
def run_model(model, smiles):
    """Run a model on a list of smiles strings

    Args:
        model: Trained model that takes SMILES strings as inputs
        smiles: List of molecules to evaluate
    Returns:
        A dataframe with the molecules and their predicted outputs
    """
    import pandas as pd
    pred_y = model.predict(smiles)
    return pd.DataFrame({'smiles': smiles, 'ie': pred_y})
```

[16]:
```
@python_app
def combine_inferences(inputs=[]):
    """Concatenate a series of inferences into a single DataFrame
    Args:
        inputs: a list of the component DataFrames
    Returns:
        A single DataFrame containing the same inferences
    """
    import pandas as pd
    return pd.concat(inputs, ignore_index=True)
```

Now we've created our Parsl `apps`, we can chop up the search space into chunks, and invoke `run_model` once for each chunk of the search space.

Note: we pass `train_future` (the future created from the training function above) as input to `run_model`. Parsl will wait for the training to be complete (i.e., the future to be resolved) before executing `run_model`.

[17]:
```
# Chunk the search space into smaller pieces, so that each can run in parallel
chunks = np.array_split(search_space['smiles'], 64)
inference_futures = [run_model(train_future, chunk) for chunk in chunks]
```

While we are running inferences in parallel we can define the final part of the workflow to combine results into a single DataFrame using `combine_inferences`.

We pass the `inference_futures` as inputs to `combine_inferences` such that Parsl knows to establish a dependency between these two functions. That is, Parsl will ensure that `train_future` must complete before any of the `run_model` tasks start; and all of the `run_model` tasks must be finished before `combine_inferences` starts.

[18]:
```
# We pass the inputs explicitly as a named argument "inputs" for Parsl to recognize this␣
↪as a "reduce" step
#  See: https://parsl.readthedocs.io/en/stable/userguide/workflow.html#mapreduce
predictions = combine_inferences(inputs=inference_futures).result()
```

After completing the inference process we now have predicted IE values for all molecules in our search space. We can print out the best five molecules, according to the trained model:

[19]:
```
predictions.sort_values('ie', ascending=False).head(5)
```

[19]:

	smiles	ie
174	CCC1CC2OC12	0.514080
184	OCCC1CC(CO)O1	0.505487
372	NC1COC2=C1ON=C2	0.505344
167	CC12C3N1CC2OC3=O	0.505340
160	CC(C=O)C1C(C)C1C	0.505315

We have now created a Parsl workflow that is able to train a model and use it to identify molecules that are likely to be good next choices for simulations. Time to build a model-in-the-loop workflow.

Model-in-the-Loop Workflow

We are going to build an application that uses a machine learning model to pick a batch of simulations, runs the simulations in parallel, and then uses the data to retrain the model before repeating the loop.

Our application uses `train_model`, `run_model`, and `combine_inferences` as above, but after running an iteration it picks the predicted best molecules and runs the `compute_vertical_app` to run the xTB simulation. The workflow then repeatedly retrains the model using these results until a fixed number of molecule simulations have been trained.

[20]:
```
with tqdm(total=search_count) as prog_bar: # setup a graphical progress bar
    prog_bar.update(len(train_data))
    batch = 1
    already_ran = set(train_data['smiles'])
    while len(train_data) < search_count:

        # Train and predict as show in the previous section.
        train_future = train_model(train_data)
        inference_futures = [run_model(train_future, chunk) for chunk in np.array_
↪split(search_space['smiles'], 64)]
        predictions = combine_inferences(inputs=inference_futures).result()

        # Sort the predictions in descending order, and submit new molecules from them
        predictions.sort_values('ie', ascending=False, inplace=True)
```

(continues on next page)

(continued from previous page)

```
        sim_futures = []
        for smiles in predictions['smiles']:
            if smiles not in already_ran:
                sim_futures.append(compute_vertical_app(smiles))
                already_ran.add(smiles)
                if len(sim_futures) >= batch_size:
                    break

        # Wait for every task in the current batch to complete, and store successful␣
↪results
        new_results = []
        for future in as_completed(sim_futures):
            if future.exception() is None:
                prog_bar.update(1)
                new_results.append({
                    'smiles': future.task_def['args'][0],
                    'ie': future.result(),
                    'batch': batch,
                    'time': monotonic()
                })

        # Update the training data and repeat
        batch += 1
        train_data = pd.concat((train_data, pd.DataFrame(new_results)), ignore_
↪index=True)
```

```
  0%|          | 0/16 [00:00<?, ?it/s]
```

We can plot the training data against the time of simulation, showing that the model is finding better molecules over time.

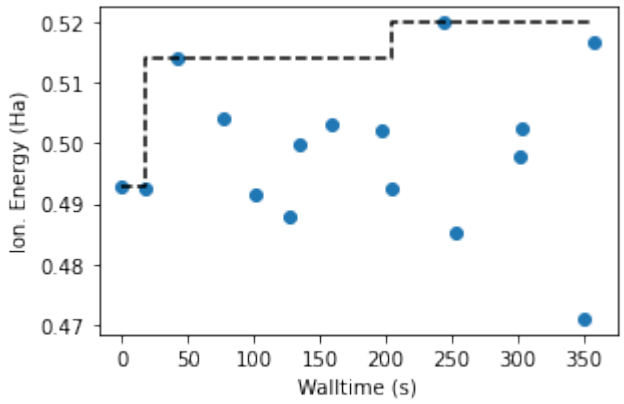
[21]:
```
train_data['time'] = train_data['time'] - train_data['time'].min()
```

[22]:
```
fig, ax = plt.subplots(figsize=(4.5, 3.))

ax.scatter(train_data['time'], train_data['ie'])
ax.step(train_data['time'], train_data['ie'].cummax(), 'k--')

ax.set_xlabel('Walltime (s)')
ax.set_ylabel('Ion. Energy (Ha)')

fig.tight_layout()
```



[ ]:

2.2.3 Parsl and RADICAL-Pilot Integration

RADICAL-Pilot (RP) is a runtime system that enables the execution of heterogeneous (funtions and executables) MPI workloads on heterogeneous (GPUs and CPUs) HPC resources. The integration of Parsl and RP allows RP to benefit from Parsl flexible programming model and its workflow management capabilities to build dynamic workflows. Additionally, RadicalPilotExecutor benefits Parsl by offering the heterogeneous runtime capabilities of RP to support many MPI computations more efficiently.

For this tutorial we are required to update the existing default Parsl package with Parsl that has the integration files (Parsl-RP integration will be relased in Parsl soon).

[ ]:
```
%%capture capt

# remove the exisitng Parsl from conda
!conda remove --force parsl -y

# install a specific Parsl version
!pip install git+https://github.com/AymenFJA/parsl.git@master
```

Next we need to locate the installed `nwchem` executable in our environment

[ ]:
```
# locate the NWChem executable path
nwchem_path = !which nwchem
nwchem      = nwchem_path[0]
```

Gather the MongoDB server information and set the `RADICAL_PILOT_DBURL` environment variable.

[2]:
```
%%capture capt

import os

mdb_host = os.environ.get('MDB_SERVER', 'mongodb')
mdb_port = os.environ.get('MDB_PORT',   '27017')
mdb_name = os.environ.get('MDB_NAME',   'guest')
mdb_pswd = os.environ.get('MDB_PSWD',   'guest')
mdb_dtbs = os.environ.get('MDB_DTBS',   'default')
```

(continues on next page)

(continued from previous page)

```
%env RADICAL_PILOT_DBURL=mongodb://$mdb_name:$mdb_pswd@$mdb_host:$mdb_port/$mdb_dtbs
```

Example: MPI NWChem Workflow

The following example application shows the execution of MP2 geometry optimization followed by a CCSD(T) energy evaluation at the converged geometry. A Dunning correlation-consistent triple-zeta basis is used. The default of Cartesian basis functions must be overridden using the keyword spherical on the BASIS directive. The 1s core orbitals are frozen in both the MP2 and coupled-cluster calculations (note that these must separately specified).

First, we need to write the `NWChem` example to a file so that we can use it as an input for the `NWChem` executable.

[ ]:
```
input = """
start n2

geometry
  symmetry d2h
  n 0 0 0.542
end

basis spherical
  n library cc-pvtz
end

mp2
  freeze core
end

task mp2 optimize

ccsd
  freeze core
end

task ccsd(t)
"""
nwchem_input = '{0}/{1}'.format(os.getcwd(), 'mp2_optimization.nw')
with open(nwchem_input,'w+') as f:
    f.writelines(input)
```

Now, we import the Parsl and RP Python modules in our application, alongside the RadicalPilotExecutor (RPEX) from Parsl

[3]:
```
import parsl
import radical.pilot as rp

from parsl.config import Config
from parsl.executors import RadicalPilotExecutor
```

RadicalPilotExecutor is capable of executing both functions and executables concurrently. The functions execution layer is based on the manager-worker paradigm. The managers are responsible for managing a set of workers and can execute function tasks as well. In contrast, the workers are only responsible for the function tasks execution. The

manager-worker paradigm requires a set of input parameters for resource distribution, such as: 1. Number of managers and workers per node 2. Number of ranks per manager and worker. 3. Number of nodes per manager and worker. 4. Etc.

In order to specify this information, we create a configuration file `rpex.cfg` that describes these parameters and pass it to RadicalPilotExecutor. In the cell below, we ask RadicalPilotExecutor to allocate 4 cores for all tasks.

[ ]:
```
# we ask Parsl to start the executor locally with 4 cores
rpex_cfg = 'configs/rpex.cfg'
config = Config(
    executors=[RadicalPilotExecutor(
        rpex_cfg=rpex_cfg, bulk_mode=True,
        resource='local.localhost', login_method = 'local',
        walltime=30, managed= True, cores= 4
    )])

parsl.load(config)
```

Create a simple Parsl `@bash_app` to invoke the `NWChem` task. The `bash_app` requires the type of the task and the number of `cpu_processes` on which to run. In this case, the type of the task is `MPI`, and the number of `cpu_processes` is 2 `MPI` ranks, where each rank takes 1 core.

Once the `bash_app` (executable task) is invoked, the RadicalPilotExecutor submits the task to the runtime system and wait for them to be executed. RadicalPilotExecutor creates a designated `sandbox` folder that contains the tasks and their `stdout/stderr` files.

[ ]:
```
@parsl.bash_app
def nwchem_mp2_optimization(cpu_processes=2, cpu_process_type=rp.MPI):

    return '{0} {1}'.format(nwchem, nwchem_input)

# invoke the nwchem_mp2_optimization
future  = nwchem_mp2_optimization()

# wait for the results of the NWChem task.
if future.result() == 0:
    print('Parsl task {0} finished'.format(future.tid))

    # rp has a different task id than Parsl (task.id)
    task_id  = str(future.tid).zfill(6)

    # RP tasks output located in the sandbox folder
    task_path = '{0}/radical.pilot.sandbox/{1}/pilot.0000/task.{2}/task.{2}.out'.
↪format(os.path.expanduser('~'),

↪config.executors[0].session.uid, task_id)
```

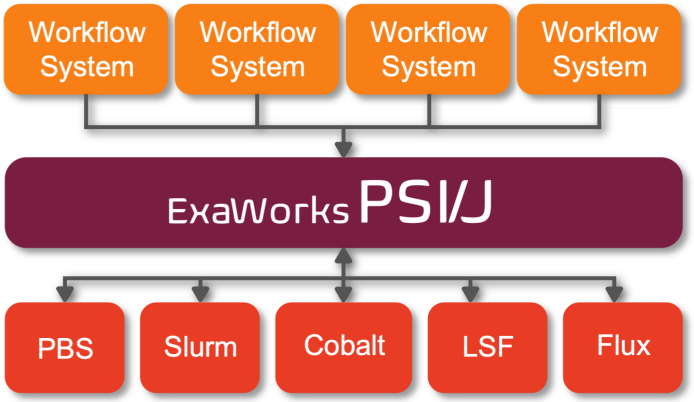
print task output from the task file

[ ]:
```
task_out = open(task_path, 'r').readlines()
print(''.join(task_out))
```

Finally, shutdown the executor, otherwise it will always stays ready to get more tasks

[ ]:
```
config.executors[0].shutdown()
```

2.2.4 PSI/J-Python Getting Started Tutorial



PSI/J (Portable Submission Interface for Jobs), is an abstraction layer over cluster job schedulers. It allows your application to be written in a way that is (mostly) independent of the cluster(s) where it runs. It is a language agnostic specification. PSI/J-Python is a Python implementation of PSI/J.

Installation

[1]:
```
%pip install git+https://github.com/ExaWorks/psij-python.git >/dev/null 2>&1
%pip show psij-python
```

```
Note: you may need to restart the kernel to use updated packages.
Name: psij-python
Version: 0.9.0
Summary: This is an implementation of the PSI/J (Portable Submission Interface for Jobs)␣
↪        specification.
Home-page: https://github.com/exaworks/psij-python
Author: The ExaWorks Team
Author-email: hategan@mcs.anl.gov
License: UNKNOWN
Location: /home/docs/checkouts/readthedocs.org/user_builds/exaworkssdk/envs/hotfix-docs/
↪lib/python3.7/site-packages
Requires: filelock, psutil, pystache, typeguard
Required-by:
Note: you may need to restart the kernel to use updated packages.
```

Overview

When running a job, there are a number of things to specify: - What is to be run, such as executable, arguments, environment, etc. (JobSpec) - What resources are needed by the job, such as the number of nodes (ResourceSpec) - Various miscellaneous properties, such as the queue to submit the job to (JobAttributes) - The mechanism through which to run the job, such as local/exec, SLURM, PBS, etc. (JobExecutor)

We also need an object to keep track of all this information, as well as the state of the execution. This object is an instance of a Job.

Setup

Before we start, let us create a separate directory so that we don't ovewrite each others' files

[2]:
```
import os
from tempfile import mkdtemp

os.makedirs('./userdirs', exist_ok=True)
workdir = mkdtemp(prefix='userdir-', dir='./userdirs')
os.chdir(workdir)
print(workdir)
```

```
./userdirs/userdir-wkypudpi
```

Basic Usage

Without further ado, let's create a simple job:

[3]:
```
from pathlib import Path
from psij import Job, JobSpec

job = Job(JobSpec(executable='/bin/date', stdout_path=Path('the-date.txt')))
```

Easy. We created a job that runs `/bin/date` and stores the output in `the-date.txt`. Now we need to run it. In order to do so, we need an *executor* that knows how to run jobs. We will use a simple fork/exec based executor named `local`. On a real cluster, we would use something like `SLURM` or `LSF`, but we are not doing this on a real cluster. However, I will note here that in most cases, simply changing `local` to the name of the scheduler used by the cluster would be sufficient to run the job through the cluster scheduler.

[4]:
```
from psij import JobExecutor

executor = JobExecutor.get_instance('local')
```

We can now tell the executor to run our job

[5]:
```
executor.submit(job)
```

The submit() method **starts** the job asynchronously. We would now like to see the result. However, before we can do so, we must ensure that the job has actually finished running. We can do so by waiting for it:

[6]:
```
job.wait()
```

[6]:
```
<psij.job_status.JobStatus at 0x7f068834fdd0>
```

The `wait()` method returns the JobStatus. Since nothing can possibly go wrong, we will assume that the job completed successfully and that there is no need to check the status to confirm it. Now, we can finally read the output

[7]:
```
with open('the-date.txt') as f:
    print(f.read())
```

```
Fri Apr 21 04:00:08 UTC 2023

```

Multiple Jobs

Our executor is stateless. That means that we can submit as many jobs as we want to it. That's in theory. In practice, computers have limited resources and there are only so many concurrent jobs that we can run, but hopefully we won't hit those limits today.

[8]:
```
jobs = []
for i in range(10):
    job = Job(
        JobSpec(
            executable='/bin/echo',
            arguments=['Hello from job %s' % i],
            stdout_path=Path('hello-%s.txt' % i)
        )
    )
    executor.submit(job)
    jobs.append(job)
```

If these jobs weren't so short, they would now be running in parallel. In fact, why not start a longer job:

[9]:
```
long_job = Job(JobSpec(executable='/bin/sleep', arguments=['600']))
executor.submit(long_job)
```

Back to our previous jobs. In order to read their outputs, we must, again, ensure that they are done

[10]:
```
for i in range(10):
    jobs[i].wait()
    with open('hello-%s.txt' % i) as f:
        print(f.read())
```

```
Hello from job 0

Hello from job 1

Hello from job 2

Hello from job 3

Hello from job 4

Hello from job 5

Hello from job 6

Hello from job 7

Hello from job 8

Hello from job 9
```

What about our long job?

[11]:
```
print(long_job.status)
```

```
JobStatus[ACTIVE, time=1682049608.7754865]
```

Still running. The time shows the instant when the job switched to `ACTIVE` state. Moving on…

Multi-process Jobs

So far we've run multiple independent jobs. But what if we wanted to run multiple copies of one job, presumably on multiple compute nodes (this is a Docker container, but we can pretend)? We could tell PSI/J to do this using ResourceSpecV1. We also need to tell PSI/J to start our job a bit differently, so we'll make a short detour to talk about launchers.

Once a job's resources are allocated, a typical job scheduler will launch our job on one of the allocated compute nodes. Then, we'd invoke something like `mpirun` or `srun`, etc. to start all the job copies on the allocated resources. By default, PSI/J uses a custom launcher named `single`, which simply starts a single copy of the job on the lead node of the job. If we wanted to see multiple copies of the job without any of the fancy features offered by `mpirun` or `srun`, we could use PSI/J's `multiple` launcher, which we will do below.

[12]:
```
from psij import ResourceSpecV1

mjob = Job(
        JobSpec(
            executable='/bin/date',
            stdout_path=Path('multi-job-out.txt'),
            resources=ResourceSpecV1(process_count=4),
            launcher="multiple"
        )
    )
```

We informed PSI/J that we need four copies of our job. On a real scheduler, we could also request that these copies be distributed on multiple compute nodes, but, on this VM, we only have one such compute node, so we shoudn't bother.

[13]:
```
executor.submit(mjob)
mjob.wait()
with open('multi-job-out.txt') as f:
    print(f.read())
```

```
Fri Apr 21 04:00:08 UTC 2023
Fri Apr 21 04:00:08 UTC 2023
Fri Apr 21 04:00:08 UTC 2023
Fri Apr 21 04:00:08 UTC 2023
```

MPI Jobs

The previous example ran a multi-process job, which has its use. It is more likely, however, to want to run an MPI job. Assuming that the system has some form of MPI installed, which this Docker container has, and which comes with some generic `mpirun` tool, we can instruct PSI/J to launch MPI jobs. And, as the previous sentence hints, it may be as simple as changing our launcher from `multiple` to `mpirun`, which it is.

But before that, we need a simple MPI executable.

[14]:
```
%%bash
cat <<EOF >hello.c
```

(continues on next page)

(continued from previous page)

```
#include <stdio.h>
#include <mpi.h>

void main(int argc, char **argv) {
    int rank;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    printf("Hello from rank %d\n", rank);

    MPI_Finalize();
}
EOF
```

Which we need to compile

[15]:
```
!mpicc hello.c -o hello
```

And now we can construct our job

[16]:
```
mpi_job = Job(
        JobSpec(
            executable='hello',
            stdout_path=Path('mpi-job-out.txt'),
            resources=ResourceSpecV1(process_count=4),
            launcher="mpirun"
        )
    )
```

… and, as usual, wait for it and display the output

[17]:
```
executor.submit(mpi_job)
mpi_job.wait()
with open('mpi-job-out.txt') as f:
    print(f.read())
```

```
Hello from rank 0
Hello from rank 2
Hello from rank 1
Hello from rank 3
```

And the long running job?

[18]:
```
print(long_job)
```

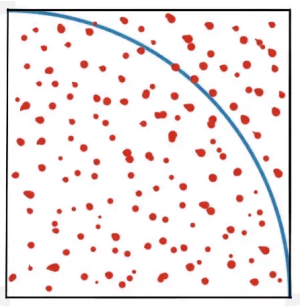
```
Job[id=f46dcef8-b3ad-4a25-b3b7-888c4e53fde8, native_id=13363, executor=JobExecutor[local,
↪ 0.0.1], status=JobStatus[ACTIVE, time=1682049608.7754865]]
```

Soon, soon…

Callbacks

Examples above are more or less synchronous, in that we use `wait()` to suspend the current thread until a job completes. In real life scenarios where scalability is needed, we would use callbacks. Let's implement a quick map/reduce workflow. We'll Monte Carlo calculate using a map-reduce like algorithm.

The basic idea is to generate some random points on a square that encloses one quadrant of a circle.



Some points will fall outside the circle and some inside. As the number of points grows, the ratio of points inside the circle vs points inside the full square (total points) will be proportional to the ratio of their areas:

Ncircle / Ntotal Acircle / Asquare = (r2 / 4) / r2

Hence

= 4 Ncircle / Ntotal

We'll start with some boilerplate, the number of iterations, and the radius of the circle

[19]:
```
from threading import Lock
from psij import JobState
import math

N = 100
R = 1000
```

Then, we'll define a class that keeps track of our points and calculates once we have all the points in, and we'll create an instance of it to hold actual results.

[20]:
```
class Results:
    def __init__(self):
        self.n = 0
        self.inside = 0
        self._lock = Lock()

    def point_received(self, x, y):
        with self._lock:
            self.n += 1
            if math.sqrt(x * x + y * y) < R:
                self.inside += 1
            if self.n == N:
                print(" is %s" % (float(self.inside) / self.n * 4))

results = Results()
```

Then, we'll define a callback function that gets invoked every time a job changes status, and have it read the output and pass it to the `results` instance. The output will be in the form `x y`

[21]:
```
def callback(job, status):
    if status.state == JobState.COMPLETED:
        with open(job.spec.stdout_path) as f:
            line = f.read().strip()
            tokens = line.split()
            results.point_received(int(tokens[0]), int(tokens[1]))
```

Unlike in previous cases, we now need to check the state of the job. That is because the full lifecycle of the job includes states such as `QUEUED` and `ACTIVE`, and the callback is invoked on all state changes.

Finally, we can create and submit our jobs

[22]:
```
for i in range(N):
    job = Job(JobSpec('echo', '/bin/bash',
                      ['-c', 'echo $((RANDOM%{})) $((RANDOM%{}))'.format(R, R)],
                      stdout_path=Path('pi-x-y-%s.txt' % i)))
    job.set_job_status_callback(callback)
    executor.submit(job)
```

Sure! Notice that the main thread is free as soon as the last job is submitted.

That's about it for this tutorial. Oh, the long running job should be done now.

[23]:
```
print(long_job)
```

```
Job[id=f46dcef8-b3ad-4a25-b3b7-888c4e53fde8, native_id=13363, executor=JobExecutor[local,
↪ 0.0.1], status=JobStatus[ACTIVE, time=1682049608.7754865]]
```

If not, we can stop it

[24]:
```
long_job.cancel()
```

OK, now we're really done. So it's clean up time. And you know what they say, if all you have is a hammer…

[25]:
```
os.chdir('../../')
cleanup_job = Job(
    JobSpec(
        executable='/bin/rm',
        arguments=['-rf', workdir],
        directory=Path('.')
    )
)
executor.submit(cleanup_job)
```

Thank you!

2.2.5 RADICAL-Cybertools Getting Started Tutorial

RADICAL-Cybertools support the execution of ensemble applications at scale on high performance computing (HPC) platforms. Ensemble applications enable using diverse algorithms to coordinate the execution of up to 10^6 tasks on all the processors (CPU/GPU) of an HPC machine. This type of applications are common in biophysical systems, climate science, seismology, and polar science domains. RADICAL-Cybertools address challenges of scale, diversity and reliability.

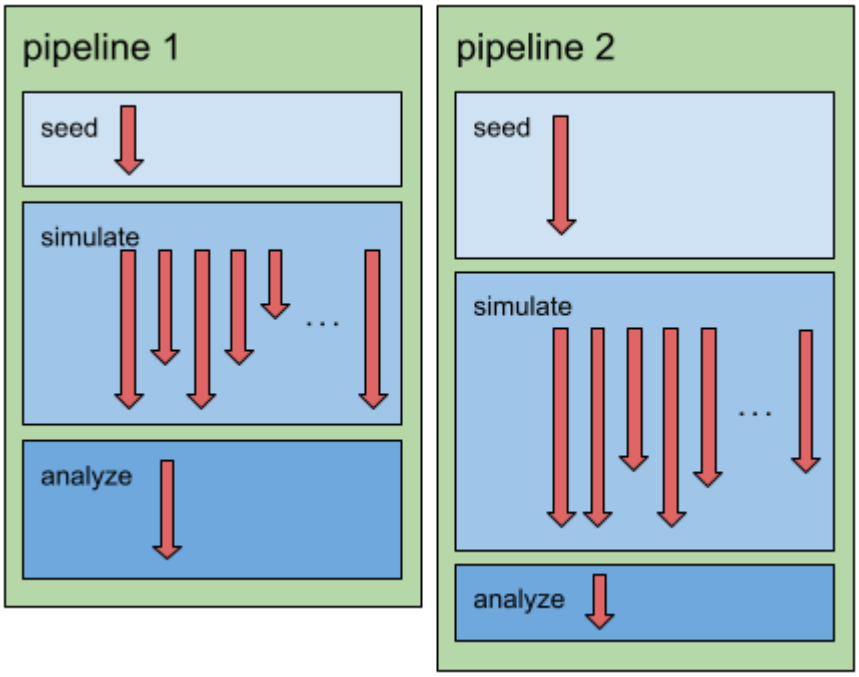
Adaptive ensemble are a particularly interesting type of ensemble applications in which adaptivity is used to determine the behavior of the application at runtime. For example, many biomolecular sampling algorithms are formulated as adaptive: replica-exchange, Expanded Ensemble, etc. Introducing adaptivity, improved simulation efficiency of up to a factor three but implementing adaptive ensemble applications is challenging due to the complexity of the required algorithms.

RADICAL-EnsembleToolkit (EnTK)

RADICAL-Cybertools offers RADICAL-EnsembleToolkit (EnTK), a workflow engine specifically designed to support the execution of (adaptive) ensemble applications at scale on HPC platforms. EnTK allows users to separate adaptive logic and simulation/analysis code, while abstracting away from the users issues of resource management and resource management and runtime execution coordination.

EnTK exposes a simple application programming interface (API), implemented in Python and with two (Pythonic) collections of objects and three classes: * Set: contains objects that have no relative order with each other * Sequence/List: contains objects that have a linear order, i.e. object 'i' depends on object 'i-1' * Task: description of executing kernel * Stage: set of Tasks, i.e. all tasks of a stage may execute concurrently * Pipeline: sequence of Stages, i.e., Stage 2 may only commence after Stage 1 completes

Thus, in EnTK an ensemble application is described as a set of pipelines, in which each pipiline has a sequence/list of stages, and each stage has a set of tasks. The following figure shows an example of ensemble application in which tasks are represented by arrows:



Preparing the Execution Environment

As we will be executing this tutorial within a Jupyter notebook, we install EnTK directly into the notebook kernel via `pip`, but we could also equally use `conda`.

Note: We "mute" the output of the cell with `%%capture capt` to not pollute the notebook output.

Depending on the execution environment, you may want to use the Spack package or the container provided by Exaworks SDK, or load the module provided by the administrators of the high performance computing (HPC) platform on which you are executing this tutorial.

[1]:
```
%%capture capt

%pip install radical.entk
```

Currently, EnTK and its runtime system RADICAL-Pilot require a RabbitMQ and MongoDB server. Those serves need to be deployed and made available before using EnTK. Here we set the access parameters for the servers.

Note:The following assumes that: 1. you have a shell; 2. you export the relevant environment variables; 3. you execute the command jupyter notebook from that shell. In that way, the relevant env variables will be read here via os.environ.get('NAME_VARIABLE').

[2]:
```
%%capture capt

import os

rmq_host = os.environ.get('RMQ_SERVER', 'rabbitmq')
rmq_port = os.environ.get('RMQ_PORT',   '5672')
rmq_name = os.environ.get('RMQ_NAME',   'guest')
rmq_pswd = os.environ.get('RMQ_PSWD',   'guest')

mdb_host = os.environ.get('MDB_SERVER', 'mongodb')
mdb_port = os.environ.get('MDB_PORT',   '27017')
mdb_name = os.environ.get('MDB_NAME',   'guest')
mdb_pswd = os.environ.get('MDB_PSWD',   'guest')
mdb_dtbs = os.environ.get('MDB_DTBS',   'default')

%env RADICAL_PILOT_DBURL=mongodb://$mdb_name:$mdb_pswd@$mdb_host:$mdb_port/$mdb_dtbs
```

Ensemble of Simulation Pipelines

The following example application shows the execution of a simple ensemble of simulations. Each ensemble member is in itself a pipeline of three different stages:

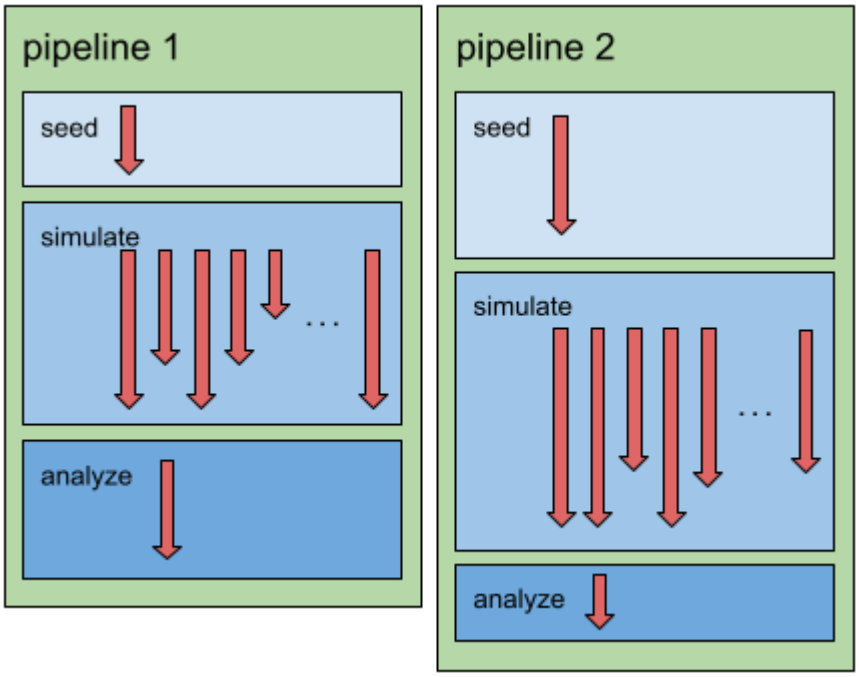
1. generate a random seed as input data
2. evolve a model based on that input data via a set of ensembles
3. derive a common metric across the model results

Similar patterns are frequently found in molecular dynamics simulation workflows. For the purpose of this tutorial, the stages are:

- random seed : create a random number
- evolve model : N tasks computing n'th power of the input
- common metric: sum over all 'model' outputs

The final results are then staged back and printed on STDOUT.

The following image offers a representation of the application we are going to code and then run for Example 1.



The two pipelines execute concurrently and, as per EnTK API definitions, each stage inside each pipeline executes sequentially. Importantly, when a stage contains **multiple** tasks, all those tasks can execute concurrently, assuming that enough resources are available. Given a set of resources, EnTK always executes the ensemble application with the highest possible degree of concurrency but, when not enough resources are available, the tasks of a stage may be executed sequentially. All this is transparent to the user that is left free to focus on the ensemble algorithm without having to deal with parallelism and resource management.

First we import EnTK Python module in our application so to be able to use its API.

[3]:
```
import radical.entk as re
```

The following function generates a single simulation pipeline, i.e., a new ensemble member. The pipeline structure consisting of three steps as described above.

[4]:
```
def generate_pipeline(uid):

    # all tasks in this pipeline share the same sandbox
    sandbox = uid

    # first stage: create 1 task to generate a random seed number
    t1 = re.Task()
    t1.executable = '/bin/sh'
    t1.arguments  = ['-c', 'od -An -N1 -i /dev/random']
    t1.stdout     = 'random.txt'
    t1.sandbox    = sandbox

    s1 = re.Stage()
    s1.add_tasks(t1)
```

(continues on next page)

(continued from previous page)

```
    # second stage: create 10 tasks to compute the n'th power of that number
    s2 = re.Stage()
    n_simulations = 10
    for i in range(n_simulations):
        t2 = re.Task()
        t2.executable = '/bin/sh'
        t2.arguments  = ['-c', "echo '$(cat random.txt) ^ %d' | bc" % i]
        t2.stdout     = 'power.%03d.txt' % i
        t2.sandbox    = sandbox
        s2.add_tasks(t2)

    # third stage: compute sum over all powers
    t3 = re.Task()
    t3.executable = '/bin/sh'
    t3.arguments  = ['-c', 'cat power.*.txt | paste -sd+ | bc']
    t3.stdout     = 'sum.txt'
    t3.sandbox    = sandbox

    # download the result while renaming to get unique files per pipeline
    t3.download_output_data = ['sum.txt > %s.sum.txt' % uid]

    s3 = re.Stage()
    s3.add_tasks(t3)

    # assemble the three stages into a pipeline and return it
    p = re.Pipeline()
    p.add_stages(s1)
    p.add_stages(s2)
    p.add_stages(s3)

    return p
```

[5]:
```
%set_env RADICAL_LOG_TGT = stdout
```

```
env: RADICAL_LOG_TGT=stdout
```

Now we write the ensemble application. We create an EnTK's application manager which executes our ensemble.

[6]:
```
appman = re.AppManager(hostname=rmq_host,
                       port=rmq_port,
                       username=rmq_name,
                       password=rmq_pswd)
```

```
EnTK session: re.session.radical.3.mturilli.019318.0000
Creating AppManagerSetting up RabbitMQ system                                  ok
                                                                               ok
```

We assign resource request description to the application manager using three mandatory keys: target resource, wall-time, and number of cpus:

[7]:
```
appman.resource_desc = {
    'resource': 'local.localhost',
```

(continues on next page)

(continued from previous page)

```
  # 'resource': 'local.localhost_flux',
    'walltime': 10,
    'cpus'    : 2
}
```

```
Validating and assigning resource manager                                    ok
```

We create an ensemble of n simulation pipelines:

[8]:
```
n_pipelines = 10
ensemble = set()
for cnt in range(n_pipelines):
    ensemble.add(generate_pipeline(uid='pipe.%03d' % cnt))
```

We assign the workflow to the application manager, then run the ensemble and wait for completion:

[9]:
```
appman.workflow = ensemble
appman.run()
```

```
Setting up RabbitMQ system                                                   n/a
new session: [re.session.radical.3.mturilli.019318.0000]                      \
database   : [mongodb://rct-tutorial:****@95.217.193.116:27017/rct-tutorial]  ok
create pilot manager                                                         ok
submit 1 pilot(s)
        pilot.0000   local.localhost           2 cores       0 gpus           ok
All components created
create task managerUpdate: pipeline.0003 state: SCHEDULING
Update: pipeline.0003.stage.0009 state: SCHEDULING
Update: pipeline.0003.stage.0009.task.0036 state: SCHEDULING
Update: pipeline.0004 state: SCHEDULING
Update: pipeline.0004.stage.0012 state: SCHEDULING
Update: pipeline.0004.stage.0012.task.0048 state: SCHEDULING
Update: pipeline.0001 state: SCHEDULING
Update: pipeline.0001.stage.0003 state: SCHEDULING
Update: pipeline.0001.stage.0003.task.0012 state: SCHEDULING
Update: pipeline.0007 state: SCHEDULING
Update: pipeline.0007.stage.0021 state: SCHEDULING
Update: pipeline.0007.stage.0021.task.0084 state: SCHEDULING
Update: pipeline.0006 state: SCHEDULING
Update: pipeline.0006.stage.0018 state: SCHEDULING
Update: pipeline.0006.stage.0018.task.0072 state: SCHEDULING
Update: pipeline.0000 state: SCHEDULING
Update: pipeline.0000.stage.0000 state: SCHEDULING
Update: pipeline.0000.stage.0000.task.0000 state: SCHEDULING
Update: pipeline.0005 state: SCHEDULING
Update: pipeline.0005.stage.0015 state: SCHEDULING
Update: pipeline.0005.stage.0015.task.0060 state: SCHEDULING
Update: pipeline.0002 state: SCHEDULING
Update: pipeline.0002.stage.0006 state: SCHEDULING
Update: pipeline.0002.stage.0006.task.0024 state: SCHEDULING
Update: pipeline.0009 state: SCHEDULING
Update: pipeline.0009.stage.0027 state: SCHEDULING
Update: pipeline.0009.stage.0027.task.0108 state: SCHEDULING
```

(continues on next page)

(continued from previous page)

```
Update: pipeline.0008 state: SCHEDULING
Update: pipeline.0008.stage.0024 state: SCHEDULING
Update: pipeline.0008.stage.0024.task.0096 state: SCHEDULING
Update: pipeline.0003.stage.0009.task.0036 state: SCHEDULED
Update: pipeline.0004.stage.0012.task.0048 state: SCHEDULED
Update: pipeline.0001.stage.0003.task.0012 state: SCHEDULED
Update: pipeline.0007.stage.0021.task.0084 state: SCHEDULED
Update: pipeline.0006.stage.0018.task.0072 state: SCHEDULED
Update: pipeline.0000.stage.0000.task.0000 state: SCHEDULED
Update: pipeline.0005.stage.0015.task.0060 state: SCHEDULED
Update: pipeline.0002.stage.0006.task.0024 state: SCHEDULED
Update: pipeline.0009.stage.0027.task.0108 state: SCHEDULED
Update: pipeline.0008.stage.0024.task.0096 state: SCHEDULED
Update: pipeline.0003.stage.0009 state: SCHEDULED
Update: pipeline.0004.stage.0012 state: SCHEDULED
Update: pipeline.0001.stage.0003 state: SCHEDULED
Update: pipeline.0007.stage.0021 state: SCHEDULED
Update: pipeline.0006.stage.0018 state: SCHEDULED
Update: pipeline.0000.stage.0000 state: SCHEDULED
Update: pipeline.0005.stage.0015 state: SCHEDULED
Update: pipeline.0002.stage.0006 state: SCHEDULED
Update: pipeline.0009.stage.0027 state: SCHEDULED
Update: MongoClient opened before fork. Create MongoClient only after forking. See␣
↪PyMongo's documentation for details: https://pymongo.readthedocs.io/en/stable/faq.html
pipeline.0008.stage.0024 state: SCHEDULED
                                                        ok
Update: submit: pipeline.0003.stage.0009.task.0036 state: SUBMITTING
Update: #pipeline.0004.stage.0012.task.0048 state: SUBMITTING
#Update: pipeline.0001.stage.0003.task.0012 state: SUBMITTING
#Update: #pipeline.0007.stage.0021.task.0084 state: SUBMITTING
#Update: #pipeline.0006.stage.0018.task.0072 state: SUBMITTING
Update: #pipeline.0000.stage.0000.task.0000 state: SUBMITTING
#Update: pipeline.0005.stage.0015.task.0060 state: SUBMITTING
#Update: #pipeline.0002.stage.0006.task.0024 state: SUBMITTING
Update: #pipeline.0009.stage.0027.task.0108 state: SUBMITTING
#Update: pipeline.0008.stage.0024.task.0096 state: SUBMITTING
##########################Update: #pipeline.0003.stage.0009.task.0036 state: DONE
Update: #pipeline.0003.stage.0009 state: DONE
#Update: pipeline.0004.stage.0012.task.0048 state: DONE
#Update: #pipeline.0004.stage.0012 state: DONE
Update: #pipeline.0001.stage.0003.task.0012 state: DONE
#Update: pipeline.0001.stage.0003 state: DONE
#Update: #pipeline.0007.stage.0021.task.0084 state: DONE
Update: #pipeline.0007.stage.0021 state: DONE
#Update: pipeline.0006.stage.0018.task.0072 state: DONE
#Update: #pipeline.0006.stage.0018 state: DONE
Update: #pipeline.0000.stage.0000.task.0000 state: DONE
#Update: pipeline.0000.stage.0000 state: DONE
#Update: #pipeline.0005.stage.0015.task.0060 state: DONE
Update: #pipeline.0005.stage.0015 state: DONE
#Update: #pipeline.0002.stage.0006.task.0024 state: DONE
#Update: pipeline.0002.stage.0006 state: DONE
```

(continues on next page)

(continued from previous page)

```
#Update: #pipeline.0009.stage.0027.task.0108 state: EXECUTED
Update: #pipeline.0009.stage.0027.task.0108 state: DONE
#Update: pipeline.0009.stage.0027 state: DONE
#Update: #pipeline.0008.stage.0024.task.0096 state: EXECUTED
Update: #pipeline.0008.stage.0024.task.0096 state: DONE
#Update: pipeline.0008.stage.0024 state: DONE
↪#is-pymongo-fork-safe
pipeline.0003.stage.0010 state: SCHEDULING
Update: pipeline.0003.stage.0010.task.0042 state: SCHEDULING
Update: pipeline.0003.stage.0010.task.0041 state: SCHEDULING
Update: pipeline.0003.stage.0010.task.0045 state: SCHEDULING
Update: pipeline.0003.stage.0010.task.0046 state: SCHEDULING
Update: pipeline.0003.stage.0010.task.0044 state: SCHEDULING
Update: pipeline.0003.stage.0010.task.0038 state: SCHEDULING
Update: pipeline.0003.stage.0010.task.0043 state: SCHEDULING
Update: pipeline.0003.stage.0010.task.0040 state: SCHEDULING
Update: pipeline.0003.stage.0010.task.0039 state: SCHEDULING
Update: pipeline.0003.stage.0010.task.0037 state: SCHEDULING
Update: pipeline.0004.stage.0013 state: SCHEDULING
Update: pipeline.0004.stage.0013.task.0053 state: SCHEDULING
Update: pipeline.0004.stage.0013.task.0054 state: SCHEDULING
Update: pipeline.0004.stage.0013.task.0058 state: SCHEDULING
Update: pipeline.0004.stage.0013.task.0052 state: SCHEDULING
Update: pipeline.0004.stage.0013.task.0051 state: SCHEDULING
Update: pipeline.0004.stage.0013.task.0050 state: SCHEDULING
Update: pipeline.0004.stage.0013.task.0055 state: SCHEDULING
Update: pipeline.0004.stage.0013.task.0056 state: SCHEDULING
Update: pipeline.0004.stage.0013.task.0057 state: SCHEDULING
Update: pipeline.0004.stage.0013.task.0049 state: SCHEDULING
Update: pipeline.0001.stage.0004 state: SCHEDULING
Update: pipeline.0001.stage.0004.task.0016 state: SCHEDULING
Update: pipeline.0001.stage.0004.task.0015 state: SCHEDULING
Update: pipeline.0001.stage.0004.task.0022 state: SCHEDULING
Update: pipeline.0001.stage.0004.task.0017 state: SCHEDULING
Update: pipeline.0001.stage.0004.task.0018 state: SCHEDULING
Update: pipeline.0001.stage.0004.task.0013 state: SCHEDULING
Update: pipeline.0001.stage.0004.task.0019 state: SCHEDULING
Update: pipeline.0001.stage.0004.task.0021 state: SCHEDULING
Update: pipeline.0001.stage.0004.task.0020 state: SCHEDULING
Update: pipeline.0001.stage.0004.task.0014 state: SCHEDULING
Update: pipeline.0007.stage.0022 state: SCHEDULING
Update: pipeline.0007.stage.0022.task.0094 state: SCHEDULING
Update: pipeline.0007.stage.0022.task.0085 state: SCHEDULING
Update: pipeline.0007.stage.0022.task.0091 state: SCHEDULING
Update: pipeline.0007.stage.0022.task.0093 state: SCHEDULING
Update: pipeline.0007.stage.0022.task.0092 state: SCHEDULING
Update: pipeline.0007.stage.0022.task.0086 state: SCHEDULING
Update: pipeline.0007.stage.0022.task.0088 state: SCHEDULING
Update: pipeline.0007.stage.0022.task.0089 state: SCHEDULING
Update: pipeline.0007.stage.0022.task.0090 state: SCHEDULING
Update: pipeline.0007.stage.0022.task.0087 state: SCHEDULING
Update: pipeline.0006.stage.0019 state: SCHEDULING
```

(continues on next page)

(continued from previous page)

```
Update: pipeline.0006.stage.0019.task.0078 state: SCHEDULING
Update: pipeline.0006.stage.0019.task.0077 state: SCHEDULING
Update: pipeline.0006.stage.0019.task.0079 state: SCHEDULING
Update: pipeline.0006.stage.0019.task.0082 state: SCHEDULING
Update: pipeline.0006.stage.0019.task.0074 state: SCHEDULING
Update: pipeline.0006.stage.0019.task.0080 state: SCHEDULING
Update: pipeline.0006.stage.0019.task.0081 state: SCHEDULING
Update: pipeline.0006.stage.0019.task.0076 state: SCHEDULING
Update: pipeline.0006.stage.0019.task.0073 state: SCHEDULING
Update: pipeline.0006.stage.0019.task.0075 state: SCHEDULING
Update: pipeline.0000.stage.0001 state: SCHEDULING
Update: pipeline.0000.stage.0001.task.0001 state: SCHEDULING
Update: pipeline.0000.stage.0001.task.0008 state: SCHEDULING
Update: pipeline.0000.stage.0001.task.0009 state: SCHEDULING
Update: pipeline.0000.stage.0001.task.0003 state: SCHEDULING
Update: pipeline.0000.stage.0001.task.0005 state: SCHEDULING
Update: pipeline.0000.stage.0001.task.0004 state: SCHEDULING
Update: pipeline.0000.stage.0001.task.0002 state: SCHEDULING
Update: pipeline.0000.stage.0001.task.0007 state: SCHEDULING
Update: pipeline.0000.stage.0001.task.0006 state: SCHEDULING
Update: pipeline.0000.stage.0001.task.0010 state: SCHEDULING
Update: pipeline.0005.stage.0016 state: SCHEDULING
Update: pipeline.0005.stage.0016.task.0066 state: SCHEDULING
Update: pipeline.0005.stage.0016.task.0068 state: SCHEDULING
Update: pipeline.0005.stage.0016.task.0063 state: SCHEDULING
Update: pipeline.0005.stage.0016.task.0061 state: SCHEDULING
Update: pipeline.0005.stage.0016.task.0064 state: SCHEDULING
Update: pipeline.0005.stage.0016.task.0065 state: SCHEDULING
Update: pipeline.0005.stage.0016.task.0070 state: SCHEDULING
Update: pipeline.0005.stage.0016.task.0067 state: SCHEDULING
Update: pipeline.0005.stage.0016.task.0062 state: SCHEDULING
Update: pipeline.0005.stage.0016.task.0069 state: SCHEDULING
Update: pipeline.0002.stage.0007 state: SCHEDULING
Update: pipeline.0002.stage.0007.task.0026 state: SCHEDULING
Update: pipeline.0002.stage.0007.task.0028 state: SCHEDULING
Update: pipeline.0002.stage.0007.task.0027 state: SCHEDULING
Update: pipeline.0002.stage.0007.task.0031 state: SCHEDULING
Update: pipeline.0002.stage.0007.task.0025 state: SCHEDULING
Update: pipeline.0002.stage.0007.task.0029 state: SCHEDULING
Update: pipeline.0002.stage.0007.task.0030 state: SCHEDULING
Update: pipeline.0002.stage.0007.task.0034 state: SCHEDULING
Update: pipeline.0002.stage.0007.task.0033 state: SCHEDULING
Update: pipeline.0002.stage.0007.task.0032 state: SCHEDULING
Update: pipeline.0009.stage.0028 state: SCHEDULING
Update: pipeline.0009.stage.0028.task.0113 state: SCHEDULING
Update: pipeline.0009.stage.0028.task.0114 state: SCHEDULING
Update: pipeline.0009.stage.0028.task.0116 state: SCHEDULING
Update: pipeline.0009.stage.0028.task.0110 state: SCHEDULING
Update: pipeline.0009.stage.0028.task.0115 state: SCHEDULING
Update: pipeline.0009.stage.0028.task.0111 state: SCHEDULING
Update: pipeline.0009.stage.0028.task.0117 state: SCHEDULING
Update: pipeline.0009.stage.0028.task.0112 state: SCHEDULING
```

(continues on next page)

(continued from previous page)

```
Update: pipeline.0009.stage.0028.task.0118 state: SCHEDULING
Update: pipeline.0009.stage.0028.task.0109 state: SCHEDULING
Update: pipeline.0008.stage.0025 state: SCHEDULING
Update: pipeline.0008.stage.0025.task.0101 state: SCHEDULING
Update: pipeline.0008.stage.0025.task.0099 state: SCHEDULING
Update: pipeline.0008.stage.0025.task.0097 state: SCHEDULING
Update: pipeline.0008.stage.0025.task.0103 state: SCHEDULING
Update: pipeline.0008.stage.0025.task.0106 state: SCHEDULING
Update: pipeline.0008.stage.0025.task.0105 state: SCHEDULING
Update: pipeline.0008.stage.0025.task.0100 state: SCHEDULING
Update: pipeline.0008.stage.0025.task.0104 state: SCHEDULING
Update: pipeline.0008.stage.0025.task.0102 state: SCHEDULING
Update: pipeline.0008.stage.0025.task.0098 state: SCHEDULING
Update: Update: pipeline.0003.stage.0010.task.0042 state: SUBMITTING
pipeline.0003.stage.0010.task.0042 state: SUBMITTING
Update: Update: pipeline.0003.stage.0010.task.0041 state: SUBMITTING
####Update:
Update: Update: pipeline.0003.stage.0010.task.0045 state: SUBMITTING
pipeline.0003.stage.0010.task.0041 state: SUBMITTING
Update: Update: pipeline.0003.stage.0010.task.0046 state: SUBMITTING
pipeline.0003.stage.0010.task.0045 state: SUBMITTING
Update: pipeline.0003.stage.0010.task.0044 state: SCHEDULED
Update: pipeline.0003.stage.0010.task.0038 state: SCHEDULED
Update: pipeline.0003.stage.0010.task.0044 state: SUBMITTING
Update: submit: pipeline.0003.stage.0010.task.0043 state: SCHEDULED
Update: Update: pipeline.0003.stage.0010.task.0038 state: SUBMITTING
pipeline.0003.stage.0010.task.0040 state: SCHEDULED
#Update: Update: pipeline.0003.stage.0010.task.0043 state: SUBMITTING
pipeline.0003.stage.0010.task.0039 state: SCHEDULED
Update: #Update: pipeline.0003.stage.0010.task.0037 state: SCHEDULED
pipeline.0003.stage.0010.task.0040 state: SUBMITTING
pipeline.0003.stage.0010.task.0046 state: SUBMITTING
#pipeline.0003.stage.0010.task.0039 state: SUBMITTING
Update: pipeline.0004.stage.0013.task.0054 state: SCHEDULED
Update: #Update: pipeline.0003.stage.0010.task.0037 state: SUBMITTING
#pipeline.0004.stage.0013.task.0058 state: SCHEDULED
Update: Update: #pipeline.0004.stage.0013.task.0053 state: SUBMITTING
pipeline.0004.stage.0013.task.0052 state: SCHEDULED
#Update: Update: pipeline.0004.stage.0013.task.0051 state: SCHEDULED
pipeline.0004.stage.0013.task.0054 state: SUBMITTING
#Update: #Update: pipeline.0004.stage.0013.task.0050 state: SCHEDULED
pipeline.0004.stage.0013.task.0058 state: SUBMITTING
Update: #pipeline.0004.stage.0013.task.0055 state: SCHEDULED
Update: #Update: pipeline.0004.stage.0013.task.0052 state: SUBMITTING
pipeline.0004.stage.0013.task.0056 state: SCHEDULED
Update: Update: pipeline.0004.stage.0013.task.0053 state: SCHEDULED
#pipeline.0004.stage.0013.task.0057 state: SCHEDULED
Update: Update: pipeline.0004.stage.0013.task.0050 state: SUBMITTING
#pipeline.0004.stage.0013.task.0049 state: SCHEDULED
#Update: Update: pipeline.0004.stage.0013.task.0055 state: SUBMITTING
pipeline.0001.stage.0004.task.0016 state: SCHEDULED
#Update: Update: #pipeline.0001.stage.0004.task.0015 state: SCHEDULED
```

(continues on next page)

(continued from previous page)

```
pipeline.0004.stage.0013.task.0056 state: SUBMITTING
Update: Update: #pipeline.0001.stage.0004.task.0022 state: SCHEDULED
pipeline.0004.stage.0013.task.0057 state: SUBMITTING
#Update: Update: pipeline.0001.stage.0004.task.0017 state: SCHEDULED
pipeline.0004.stage.0013.task.0049 state: SUBMITTING
#Update: #Update: pipeline.0001.stage.0004.task.0018 state: SCHEDULED
pipeline.0001.stage.0004.task.0016 state: SUBMITTING
Update: #pipeline.0001.stage.0004.task.0013 state: SCHEDULED
Update: #pipeline.0001.stage.0004.task.0015 state: SUBMITTING
Update: pipeline.0001.stage.0004.task.0019 state: SCHEDULED
Update: #Update: pipeline.0001.stage.0004.task.0022 state: SUBMITTING
#pipeline.0001.stage.0004.task.0021 state: SCHEDULED
Update: Update: pipeline.0001.stage.0004.task.0017 state: SUBMITTING
#pipeline.0001.stage.0004.task.0020 state: SCHEDULED
#Update: Update: pipeline.0001.stage.0004.task.0018 state: SUBMITTING
pipeline.0001.stage.0004.task.0014 state: SCHEDULED
#Update: Update: #pipeline.0007.stage.0022.task.0094 state: SCHEDULED
pipeline.0001.stage.0004.task.0013 state: SUBMITTING
Update: Update: #pipeline.0007.stage.0022.task.0085 state: SCHEDULED
pipeline.0001.stage.0004.task.0019 state: SUBMITTING
#Update: Update: pipeline.0007.stage.0022.task.0091 state: SCHEDULED
pipeline.0001.stage.0004.task.0021 state: SUBMITTING
#Update: Update: #pipeline.0007.stage.0022.task.0093 state: SCHEDULED
pipeline.0001.stage.0004.task.0020 state: SUBMITTING
Update: Update: #pipeline.0001.stage.0004.task.0014 state: SUBMITTING
Update: #Update: pipeline.0004.stage.0013.task.0051 state: SUBMITTING
#Update: Update: pipeline.0007.stage.0022.task.0086 state: SCHEDULED
pipeline.0007.stage.0022.task.0094 state: SUBMITTING
#Update: Update: #pipeline.0007.stage.0022.task.0088 state: SCHEDULED
pipeline.0007.stage.0022.task.0085 state: SUBMITTING
Update: Update: #pipeline.0007.stage.0022.task.0089 state: SCHEDULED
pipeline.0007.stage.0022.task.0091 state: SUBMITTING
#Update: Update: pipeline.0007.stage.0022.task.0090 state: SCHEDULED
pipeline.0007.stage.0022.task.0092 state: SCHEDULED
#Update: Update: #pipeline.0007.stage.0022.task.0087 state: SCHEDULED
pipeline.0007.stage.0022.task.0093 state: SUBMITTING
Update: Update: #pipeline.0006.stage.0019.task.0078 state: SCHEDULED
pipeline.0007.stage.0022.task.0086 state: SUBMITTING
#Update: pipeline.0006.stage.0019.task.0077 state: SCHEDULED
pipeline.0007.stage.0022.task.0092 state: SUBMITTING
#pipeline.0006.stage.0019.task.0079 state: SCHEDULED
Update: Update: pipeline.0007.stage.0022.task.0089 state: SUBMITTING
#pipeline.0006.stage.0019.task.0082 state: SCHEDULED
Update: #Update: pipeline.0007.stage.0022.task.0088 state: SUBMITTING
pipeline.0006.stage.0019.task.0074 state: SCHEDULED
#Update: Update: #pipeline.0007.stage.0022.task.0087 state: SUBMITTING
pipeline.0006.stage.0019.task.0080 state: SCHEDULED
Update: Update: #pipeline.0006.stage.0019.task.0081 state: SCHEDULED
pipeline.0006.stage.0019.task.0078 state: SUBMITTING
#Update: Update: pipeline.0006.stage.0019.task.0077 state: SUBMITTING
Update: #Update: pipeline.0007.stage.0022.task.0090 state: SUBMITTING
#Update: Update: #pipeline.0006.stage.0019.task.0073 state: SCHEDULED
pipeline.0006.stage.0019.task.0079 state: SUBMITTING
```

(continues on next page)

(continued from previous page)

```
Update: Update: #pipeline.0006.stage.0019.task.0075 state: SCHEDULED
pipeline.0006.stage.0019.task.0082 state: SUBMITTING
#Update: Update: pipeline.0000.stage.0001.task.0001 state: SCHEDULED
pipeline.0006.stage.0019.task.0074 state: SUBMITTING
#Update: #Update: pipeline.0000.stage.0001.task.0008 state: SCHEDULED
pipeline.0006.stage.0019.task.0080 state: SUBMITTING
Update: #pipeline.0000.stage.0001.task.0009 state: SCHEDULED
Update: #Update: pipeline.0006.stage.0019.task.0081 state: SUBMITTING
pipeline.0000.stage.0001.task.0003 state: SCHEDULED
Update: #Update: pipeline.0006.stage.0019.task.0076 state: SUBMITTING
#pipeline.0000.stage.0001.task.0005 state: SCHEDULED
Update: Update: pipeline.0006.stage.0019.task.0073 state: SUBMITTING
#pipeline.0000.stage.0001.task.0004 state: SCHEDULED
#Update: Update: pipeline.0006.stage.0019.task.0075 state: SUBMITTING
pipeline.0000.stage.0001.task.0002 state: SCHEDULED
#Update: Update: #pipeline.0000.stage.0001.task.0007 state: SCHEDULED
pipeline.0000.stage.0001.task.0001 state: SUBMITTING
Update: Update: #pipeline.0000.stage.0001.task.0006 state: SCHEDULED
pipeline.0000.stage.0001.task.0008 state: SUBMITTING
#Update: Update: pipeline.0000.stage.0001.task.0010 state: SCHEDULED
pipeline.0000.stage.0001.task.0009 state: SUBMITTING
#Update: Update: #pipeline.0005.stage.0016.task.0066 state: SCHEDULED
pipeline.0000.stage.0001.task.0003 state: SUBMITTING
Update: #Update: pipeline.0005.stage.0016.task.0068 state: SCHEDULED
#pipeline.0000.stage.0001.task.0005 state: SUBMITTING
Update: pipeline.0005.stage.0016.task.0063 state: SCHEDULED
Update: #pipeline.0000.stage.0001.task.0004 state: SUBMITTING
pipeline.0006.stage.0019.task.0076 state: SCHEDULED
pipeline.0005.stage.0016.task.0061 state: SCHEDULED
Update: Update: pipeline.0000.stage.0001.task.0002 state: SUBMITTING
pipeline.0005.stage.0016.task.0064 state: SCHEDULED
Update: Update: pipeline.0000.stage.0001.task.0007 state: SUBMITTING
pipeline.0005.stage.0016.task.0065 state: SCHEDULED
Update: pipeline.0005.stage.0016.task.0070 state: SCHEDULED
Update: Update: pipeline.0000.stage.0001.task.0006 state: SUBMITTING
pipeline.0005.stage.0016.task.0067 state: SCHEDULED
Update:
pipeline.0000.stage.0001.task.0010 state: SUBMITTING
Update: pipeline.0005.stage.0016.task.0069 state: SCHEDULED
Update: Update: pipeline.0002.stage.0007.task.0026 state: SCHEDULED
pipeline.0005.stage.0016.task.0066 state: SUBMITTING
Update: pipeline.0002.stage.0007.task.0028 state: SCHEDULED
Update: Update: pipeline.0002.stage.0007.task.0027 state: SCHEDULED
pipeline.0005.stage.0016.task.0068 state: SUBMITTING
Update: pipeline.0002.stage.0007.task.0031 state: SCHEDULED
Update: pipeline.0002.stage.0007.task.0025 state: SCHEDULED
Update: Update: pipeline.0002.stage.0007.task.0029 state: SCHEDULED
Update: Update: pipeline.0005.stage.0016.task.0062 state: SCHEDULED
Update: pipeline.0002.stage.0007.task.0030 state: SCHEDULED
Update: Update: pipeline.0002.stage.0007.task.0034 state: SCHEDULED
pipeline.0005.stage.0016.task.0061 state: SUBMITTING
pipeline.0005.stage.0016.task.0063 state: SUBMITTING
pipeline.0005.stage.0016.task.0064 state: SUBMITTING
```

(continues on next page)

(continued from previous page)

```
Update: Update: pipeline.0002.stage.0007.task.0032 state: SCHEDULED
pipeline.0005.stage.0016.task.0065 state: SUBMITTING
Update: pipeline.0009.stage.0028.task.0113 state: SCHEDULED
Update: Update: pipeline.0009.stage.0028.task.0114 state: SCHEDULED
pipeline.0005.stage.0016.task.0070 state: SUBMITTING
Update: pipeline.0009.stage.0028.task.0116 state: SCHEDULED
Update: Update: pipeline.0002.stage.0007.task.0033 state: SCHEDULED
pipeline.0009.stage.0028.task.0110 state: SCHEDULED
Update: Update: pipeline.0005.stage.0016.task.0067 state: SUBMITTING
pipeline.0005.stage.0016.task.0062 state: SUBMITTING
Update: pipeline.0009.stage.0028.task.0111 state: SCHEDULED
Update: Update: pipeline.0009.stage.0028.task.0117 state: SCHEDULED
pipeline.0005.stage.0016.task.0069 state: SUBMITTING
Update: pipeline.0009.stage.0028.task.0112 state: SCHEDULED
Update: Update: pipeline.0009.stage.0028.task.0118 state: SCHEDULED
pipeline.0002.stage.0007.task.0026 state: SUBMITTING
Update: pipeline.0009.stage.0028.task.0109 state: SCHEDULED
Update: Update: pipeline.0008.stage.0025.task.0101 state: SCHEDULED
Update: Update: pipeline.0009.stage.0028.task.0115 state: SCHEDULED
Update: pipeline.0008.stage.0025.task.0099 state: SCHEDULED
Update: Update: pipeline.0002.stage.0007.task.0027 state: SUBMITTING
pipeline.0002.stage.0007.task.0028 state: SUBMITTING
Update: pipeline.0008.stage.0025.task.0103 state: SCHEDULED
Update: Update: pipeline.0008.stage.0025.task.0106 state: SCHEDULED
pipeline.0002.stage.0007.task.0031 state: SUBMITTING
Update: pipeline.0008.stage.0025.task.0105 state: SCHEDULED
Update: Update: pipeline.0008.stage.0025.task.0100 state: SCHEDULED
pipeline.0002.stage.0007.task.0025 state: SUBMITTING
Update: Update: pipeline.0008.stage.0025.task.0104 state: SCHEDULED
pipeline.0002.stage.0007.task.0029 state: SUBMITTING
Update: pipeline.0008.stage.0025.task.0102 state: SCHEDULED
Update: pipeline.0008.stage.0025.task.0098 state: SCHEDULED
Update: pipeline.0003.stage.0010 state: SCHEDULED
Update: pipeline.0002.stage.0007.task.0030 state: SUBMITTING
Update: pipeline.0004.stage.0013 state: SCHEDULED
Update: Update: pipeline.0001.stage.0004 state: SCHEDULED
pipeline.0002.stage.0007.task.0034 state: SUBMITTING
Update: pipeline.0007.stage.0022 state: SCHEDULED
Update: Update: pipeline.0006.stage.0019 state: SCHEDULED
pipeline.0008.stage.0025.task.0097 state: SCHEDULED
Update: pipeline.0000.stage.0001 state: SCHEDULED
Update: Update: pipeline.0005.stage.0016 state: SCHEDULED
pipeline.0002.stage.0007.task.0032 state: SUBMITTING
Update: pipeline.0002.stage.0007 state: SCHEDULED
Update: pipeline.0009.stage.0028.task.0113 state: SUBMITTING
Update: pipeline.0009.stage.0028 state: SCHEDULED
Update: Update: pipeline.0009.stage.0028.task.0114 state: SUBMITTING
pipeline.0002.stage.0007.task.0033 state: SUBMITTING
Update: pipeline.0009.stage.0028.task.0116 state: SUBMITTING
Update: pipeline.0009.stage.0028.task.0110 state: SUBMITTING
Update: pipeline.0009.stage.0028.task.0115 state: SUBMITTING
Update: pipeline.0009.stage.0028.task.0111 state: SUBMITTING
Update: pipeline.0009.stage.0028.task.0117 state: SUBMITTING
```

(continues on next page)

(continued from previous page)

```
Update: pipeline.0009.stage.0028.task.0112 state: SUBMITTING
Update: pipeline.0009.stage.0028.task.0118 state: SUBMITTING
Update: pipeline.0009.stage.0028.task.0109 state: SUBMITTING
Update: pipeline.0008.stage.0025.task.0101 state: SUBMITTING
Update: pipeline.0008.stage.0025.task.0099 state: SUBMITTING
Update: pipeline.0008.stage.0025.task.0097 state: SUBMITTING
Update: pipeline.0008.stage.0025.task.0103 state: SUBMITTING
Update: pipeline.0008.stage.0025.task.0106 state: SUBMITTING
Update: pipeline.0008.stage.0025.task.0105 state: SUBMITTING
Update: pipeline.0008.stage.0025.task.0100 state: SUBMITTING
Update: pipeline.0008.stage.0025.task.0104 state: SUBMITTING
Update: pipeline.0008.stage.0025.task.0102 state: SUBMITTING
Update: pipeline.0008.stage.0025.task.0098 state: SUBMITTING
Update: pipeline.0003.stage.0010.task.0042 state: DONE
Update: pipeline.0003.stage.0010.task.0041 state: DONE
Update: pipeline.0003.stage.0010.task.0045 state: DONE
Update: pipeline.0003.stage.0010.task.0046 state: DONE
Update: pipeline.0003.stage.0010.task.0044 state: DONE
Update: pipeline.0003.stage.0010.task.0038 state: DONE
Update: pipeline.0003.stage.0010.task.0043 state: DONE
Update: pipeline.0003.stage.0010.task.0040 state: EXECUTED
Update: pipeline.0003.stage.0010.task.0040 state: DONE
Update: pipeline.0003.stage.0010.task.0039 state: EXECUTED
Update: pipeline.0003.stage.0010.task.0039 state: DONE
Update: pipeline.0003.stage.0010.task.0037 state: EXECUTED
Update: pipeline.0003.stage.0010.task.0037 state: DONE
Update: pipeline.0003.stage.0010 state: DONE
Update: pipeline.0004.stage.0013.task.0053 state: EXECUTED
Update: pipeline.0004.stage.0013.task.0053 state: DONE
Update: pipeline.0004.stage.0013.task.0054 state: EXECUTED
Update: pipeline.0004.stage.0013.task.0054 state: DONE
Update: pipeline.0004.stage.0013.task.0052 state: EXECUTED
Update: pipeline.0004.stage.0013.task.0052 state: DONE
Update: pipeline.0004.stage.0013.task.0055 state: EXECUTED
Update: pipeline.0004.stage.0013.task.0055 state: DONE
Update: pipeline.0004.stage.0013.task.0056 state: EXECUTED
Update: pipeline.0004.stage.0013.task.0056 state: DONE
Update: pipeline.0001.stage.0004.task.0016 state: EXECUTED
Update: pipeline.0001.stage.0004.task.0016 state: DONE
Update: pipeline.0001.stage.0004.task.0017 state: EXECUTED
Update: pipeline.0001.stage.0004.task.0017 state: DONE
Update: pipeline.0001.stage.0004.task.0018 state: EXECUTED
Update: pipeline.0001.stage.0004.task.0018 state: DONE
Update: pipeline.0001.stage.0004.task.0020 state: EXECUTED
Update: pipeline.0001.stage.0004.task.0020 state: DONE
Update: pipeline.0007.stage.0022.task.0094 state: EXECUTED
Update: pipeline.0007.stage.0022.task.0094 state: DONE
Update: pipeline.0007.stage.0022.task.0091 state: EXECUTED
Update: pipeline.0007.stage.0022.task.0091 state: DONE
Update: pipeline.0007.stage.0022.task.0092 state: EXECUTED
Update: pipeline.0007.stage.0022.task.0092 state: DONE
Update: pipeline.0006.stage.0019.task.0081 state: EXECUTED
```

(continues on next page)

(continued from previous page)

```
Update: pipeline.0006.stage.0019.task.0081 state: DONE
Update: pipeline.0006.stage.0019.task.0076 state: EXECUTED
Update: pipeline.0006.stage.0019.task.0076 state: DONE
Update: pipeline.0000.stage.0001.task.0001 state: EXECUTED
Update: pipeline.0000.stage.0001.task.0001 state: DONE
Update: pipeline.0000.stage.0001.task.0005 state: EXECUTED
Update: pipeline.0000.stage.0001.task.0005 state: DONE
Update: pipeline.0005.stage.0016.task.0064 state: EXECUTED
Update: pipeline.0005.stage.0016.task.0064 state: DONE
Update: pipeline.0005.stage.0016.task.0070 state: EXECUTED
Update: pipeline.0005.stage.0016.task.0070 state: DONE
Update: pipeline.0002.stage.0007.task.0026 state: EXECUTED
Update: pipeline.0002.stage.0007.task.0026 state: DONE
Update: pipeline.0002.stage.0007.task.0032 state: EXECUTED
Update: pipeline.0002.stage.0007.task.0032 state: DONE
Update: pipeline.0009.stage.0028.task.0113 state: EXECUTED
Update: pipeline.0009.stage.0028.task.0113 state: DONE
Update: pipeline.0009.stage.0028.task.0110 state: EXECUTED
Update: pipeline.0009.stage.0028.task.0110 state: DONE
Update: pipeline.0009.stage.0028.task.0115 state: EXECUTED
Update: pipeline.0009.stage.0028.task.0115 state: DONE
Update: pipeline.0009.stage.0028.task.0117 state: EXECUTED
Update: pipeline.0009.stage.0028.task.0117 state: DONE
Update: pipeline.0009.stage.0028.task.0109 state: EXECUTED
Update: pipeline.0009.stage.0028.task.0109 state: DONE
Update: pipeline.0008.stage.0025.task.0101 state: EXECUTED
Update: pipeline.0008.stage.0025.task.0101 state: DONE
Update: pipeline.0008.stage.0025.task.0099 state: EXECUTED
Update: pipeline.0008.stage.0025.task.0099 state: DONE
Update: pipeline.0008.stage.0025.task.0097 state: EXECUTED
Update: pipeline.0008.stage.0025.task.0097 state: DONE
Update: pipeline.0008.stage.0025.task.0103 state: EXECUTED
Update: pipeline.0008.stage.0025.task.0103 state: DONE
Update: pipeline.0008.stage.0025.task.0106 state: EXECUTED
Update: pipeline.0008.stage.0025.task.0106 state: DONE
Update: pipeline.0008.stage.0025.task.0105 state: EXECUTED
Update: pipeline.0008.stage.0025.task.0105 state: DONE
Update: pipeline.0008.stage.0025.task.0100 state: EXECUTED
Update: pipeline.0008.stage.0025.task.0100 state: DONE
Update: pipeline.0008.stage.0025.task.0104 state: EXECUTED
Update: pipeline.0008.stage.0025.task.0104 state: DONE
Update: pipeline.0008.stage.0025.task.0102 state: EXECUTED
Update: pipeline.0008.stage.0025.task.0102 state: DONE
Update: pipeline.0008.stage.0025.task.0098 state: EXECUTED
Update: pipeline.0008.stage.0025.task.0098 state: DONE
Update: pipeline.0008.stage.0025 state: DONE
Update: pipeline.0003.stage.0011 state: SCHEDULING
Update: pipeline.0003.stage.0011.task.0047 state: SCHEDULING
Update: pipeline.0008.stage.0026 state: SCHEDULING
Update: pipeline.0008.stage.0026.task.0107 state: SCHEDULING
submit: Update: Update: pipeline.0003.stage.0011.task.0047 state: SUBMITTING
pipeline.0003.stage.0011.task.0047 state: SUBMITTING
```

(continues on next page)

(continued from previous page)

```
#Update: Update: #pipeline.0008.stage.0026.task.0107 state: SUBMITTING
pipeline.0008.stage.0026.task.0107 state: SUBMITTING
#Update: Update: #pipeline.0004.stage.0013.task.0058 state: DONE
pipeline.0003.stage.0011 state: SCHEDULED
#Update: Update: #pipeline.0004.stage.0013.task.0051 state: DONE
pipeline.0008.stage.0026 state: SCHEDULED
#Update: #pipeline.0004.stage.0013.task.0050 state: DONE
#Update: #pipeline.0004.stage.0013.task.0057 state: EXECUTED
#Update: #pipeline.0004.stage.0013.task.0057 state: DONE
#Update: #pipeline.0004.stage.0013.task.0049 state: EXECUTED
#Update: #pipeline.0004.stage.0013.task.0049 state: DONE
#Update: #pipeline.0004.stage.0013 state: DONE
##Update: #pipeline.0001.stage.0004.task.0015 state: EXECUTED
#Update: ##pipeline.0001.stage.0004.task.0015 state: DONE
#Update: #pipeline.0001.stage.0004.task.0022 state: EXECUTED
##Update: #pipeline.0001.stage.0004.task.0022 state: DONE
#Update: ##pipeline.0001.stage.0004.task.0013 state: EXECUTED
#Update: #pipeline.0001.stage.0004.task.0013 state: DONE
##Update: #pipeline.0001.stage.0004.task.0019 state: EXECUTED
#Update: ##pipeline.0001.stage.0004.task.0019 state: DONE
#Update: #pipeline.0001.stage.0004.task.0021 state: EXECUTED
##Update: #pipeline.0001.stage.0004.task.0021 state: DONE
#Update: ##pipeline.0001.stage.0004.task.0014 state: EXECUTED
#Update: #pipeline.0001.stage.0004.task.0014 state: DONE
##Update: #pipeline.0001.stage.0004 state: DONE
#Update: ##pipeline.0007.stage.0022.task.0085 state: EXECUTED
#Update: ##pipeline.0007.stage.0022.task.0085 state: DONE
#Update: ##pipeline.0007.stage.0022.task.0093 state: EXECUTED
##Update: #pipeline.0007.stage.0022.task.0093 state: DONE
##Update: ##pipeline.0007.stage.0022.task.0086 state: EXECUTED
#Update: ##pipeline.0007.stage.0022.task.0086 state: DONE
pipeline.0008.stage.0025 state: SCHEDULED
Update: pipeline.0007.stage.0022.task.0088 state: EXECUTED
Update: pipeline.0007.stage.0022.task.0088 state: DONE
Update: pipeline.0007.stage.0022.task.0089 state: EXECUTED
Update: pipeline.0007.stage.0022.task.0089 state: DONE
Update: pipeline.0007.stage.0022.task.0090 state: EXECUTED
Update: pipeline.0007.stage.0022.task.0090 state: DONE
Update: pipeline.0007.stage.0022.task.0087 state: EXECUTED
Update: pipeline.0007.stage.0022.task.0087 state: DONE
Update: pipeline.0007.stage.0022 state: DONE
Update: pipeline.0006.stage.0019.task.0077 state: EXECUTED
Update: pipeline.0006.stage.0019.task.0077 state: DONE
Update: pipeline.0006.stage.0019.task.0074 state: EXECUTED
Update: pipeline.0006.stage.0019.task.0074 state: DONE
Update: pipeline.0006.stage.0019.task.0080 state: EXECUTED
Update: pipeline.0006.stage.0019.task.0080 state: DONE
Update: pipeline.0006.stage.0019.task.0073 state: EXECUTED
Update: pipeline.0006.stage.0019.task.0073 state: DONE
Update: pipeline.0006.stage.0019.task.0075 state: EXECUTED
Update: pipeline.0006.stage.0019.task.0075 state: DONE
Update: pipeline.0000.stage.0001.task.0008 state: EXECUTED
```

(continues on next page)

(continued from previous page)

```
Update: pipeline.0000.stage.0001.task.0008 state: DONE
Update: pipeline.0000.stage.0001.task.0009 state: EXECUTED
Update: pipeline.0000.stage.0001.task.0009 state: DONE
Update: pipeline.0000.stage.0001.task.0003 state: EXECUTED
Update: pipeline.0000.stage.0001.task.0003 state: DONE
Update: pipeline.0000.stage.0001.task.0002 state: EXECUTED
Update: pipeline.0000.stage.0001.task.0002 state: DONE
Update: pipeline.0000.stage.0001.task.0007 state: EXECUTED
Update: pipeline.0000.stage.0001.task.0007 state: DONE
Update: pipeline.0000.stage.0001.task.0006 state: EXECUTED
Update: pipeline.0000.stage.0001.task.0006 state: DONE
Update: pipeline.0000.stage.0001.task.0010 state: EXECUTED
Update: pipeline.0000.stage.0001.task.0010 state: DONE
Update: pipeline.0005.stage.0016.task.0066 state: EXECUTED
Update: pipeline.0005.stage.0016.task.0066 state: DONE
Update: pipeline.0005.stage.0016.task.0068 state: EXECUTED
Update: pipeline.0005.stage.0016.task.0068 state: DONE
Update: pipeline.0005.stage.0016.task.0063 state: EXECUTED
Update: pipeline.0005.stage.0016.task.0063 state: DONE
Update: pipeline.0005.stage.0016.task.0061 state: EXECUTED
Update: pipeline.0005.stage.0016.task.0061 state: DONE
Update: pipeline.0005.stage.0016.task.0065 state: EXECUTED
Update: pipeline.0005.stage.0016.task.0065 state: DONE
Update: pipeline.0005.stage.0016.task.0067 state: EXECUTED
Update: pipeline.0005.stage.0016.task.0067 state: DONE
Update: pipeline.0005.stage.0016.task.0069 state: EXECUTED
Update: pipeline.0005.stage.0016.task.0069 state: DONE
Update: pipeline.0002.stage.0007.task.0028 state: EXECUTED
Update: pipeline.0002.stage.0007.task.0028 state: DONE
Update: pipeline.0002.stage.0007.task.0027 state: EXECUTED
Update: pipeline.0002.stage.0007.task.0027 state: DONE
Update: pipeline.0002.stage.0007.task.0031 state: EXECUTED
Update: pipeline.0002.stage.0007.task.0031 state: DONE
Update: pipeline.0002.stage.0007.task.0025 state: EXECUTED
Update: pipeline.0002.stage.0007.task.0025 state: DONE
Update: pipeline.0002.stage.0007.task.0029 state: EXECUTED
Update: pipeline.0002.stage.0007.task.0029 state: DONE
Update: pipeline.0002.stage.0007.task.0030 state: EXECUTED
Update: pipeline.0002.stage.0007.task.0030 state: DONE
Update: pipeline.0002.stage.0007.task.0034 state: EXECUTED
Update: pipeline.0002.stage.0007.task.0034 state: DONE
Update: pipeline.0002.stage.0007.task.0033 state: EXECUTED
Update: pipeline.0002.stage.0007.task.0033 state: DONE
Update: pipeline.0002.stage.0007 state: DONE
Update: pipeline.0009.stage.0028.task.0114 state: EXECUTED
Update: pipeline.0009.stage.0028.task.0114 state: DONE
Update: pipeline.0009.stage.0028.task.0116 state: EXECUTED
Update: pipeline.0009.stage.0028.task.0116 state: DONE
Update: pipeline.0009.stage.0028.task.0111 state: EXECUTED
Update: pipeline.0009.stage.0028.task.0111 state: DONE
Update: pipeline.0009.stage.0028.task.0112 state: EXECUTED
Update: pipeline.0009.stage.0028.task.0112 state: DONE
```

(continues on next page)

(continued from previous page)

```
Update: pipeline.0009.stage.0028.task.0118 state: EXECUTED
Update: pipeline.0009.stage.0028.task.0118 state: DONE
Update: pipeline.0009.stage.0028 state: DONE
Update: pipeline.0006.stage.0019.task.0078 state: EXECUTED
Update: pipeline.0006.stage.0019.task.0078 state: DONE
Update: pipeline.0006.stage.0019.task.0079 state: EXECUTED
Update: pipeline.0006.stage.0019.task.0079 state: DONE
Update: pipeline.0006.stage.0019.task.0082 state: EXECUTED
Update: pipeline.0006.stage.0019.task.0082 state: DONE
Update: pipeline.0006.stage.0019 state: DONE
Update: pipeline.0000.stage.0001.task.0004 state: EXECUTED
Update: pipeline.0000.stage.0001.task.0004 state: DONE
Update: pipeline.0000.stage.0001 state: DONE
Update: pipeline.0005.stage.0016.task.0062 state: EXECUTED
Update: pipeline.0005.stage.0016.task.0062 state: DONE
Update: pipeline.0005.stage.0016 state: DONE
Update: pipeline.0003.stage.0011.task.0047 state: DONE
Update: pipeline.0003.stage.0011 state: DONE
Update: pipeline.0003 state: DONE
Update: pipeline.0008.stage.0026.task.0107 state: DONE
Update: pipeline.0008.stage.0026 state: DONE
Update: pipeline.0008 state: DONE
Update: pipeline.0004.stage.0014 state: SCHEDULING
Update: pipeline.0004.stage.0014.task.0059 state: SCHEDULING
Update: pipeline.0001.stage.0005 state: SCHEDULING
Update: pipeline.0001.stage.0005.task.0023 state: SCHEDULING
Update: pipeline.0007.stage.0023 state: SCHEDULING
Update: pipeline.0007.stage.0023.task.0095 state: SCHEDULING
Update: pipeline.0006.stage.0020 state: SCHEDULING
Update: pipeline.0006.stage.0020.task.0083 state: SCHEDULING
Update: pipeline.0000.stage.0002 state: SCHEDULING
Update: pipeline.0000.stage.0002.task.0011 state: SCHEDULING
Update: pipeline.0005.stage.0017 state: SCHEDULING
Update: pipeline.0005.stage.0017.task.0071 state: SCHEDULING
Update: pipeline.0002.stage.0008 state: SCHEDULING
Update: pipeline.0002.stage.0008.task.0035 state: SCHEDULING
Update: pipeline.0009.stage.0029 state: SCHEDULING
Update: pipeline.0009.stage.0029.task.0119 state: SCHEDULING
submit: Update: Update: pipeline.0004.stage.0014.task.0059 state: SUBMITTING
pipeline.0004.stage.0014.task.0059 state: SUBMITTING
#Update: Update: #pipeline.0001.stage.0005.task.0023 state: SUBMITTING
pipeline.0001.stage.0005.task.0023 state: SUBMITTING
Update: Update: #pipeline.0007.stage.0023.task.0095 state: SUBMITTING
pipeline.0007.stage.0023.task.0095 state: SUBMITTING
##Update: Update: #pipeline.0006.stage.0020.task.0083 state: SUBMITTING
pipeline.0006.stage.0020.task.0083 state: SUBMITTING
#Update: Update: ##pipeline.0000.stage.0002.task.0011 state: SUBMITTING
pipeline.0000.stage.0002.task.0011 state: SUBMITTING
#Update: Update: #pipeline.0005.stage.0017.task.0071 state: SUBMITTING
pipeline.0005.stage.0017.task.0071 state: SUBMITTING
##Update: Update: #pipeline.0002.stage.0008.task.0035 state: SUBMITTING
pipeline.0002.stage.0008.task.0035 state: SUBMITTING
```

(continues on next page)

(continued from previous page)

```
#Update: Update: ##pipeline.0009.stage.0029.task.0119 state: SUBMITTING
pipeline.0009.stage.0029.task.0119 state: SUBMITTING
#Update: #pipeline.0004.stage.0014 state: SCHEDULED
##Update: #pipeline.0001.stage.0005 state: SCHEDULED
#Update: ##pipeline.0007.stage.0023 state: SCHEDULED
#Update: #pipeline.0006.stage.0020 state: SCHEDULED
##Update: #pipeline.0000.stage.0002 state: SCHEDULED
#Update: ##pipeline.0005.stage.0017 state: SCHEDULED
#Update: #pipeline.0002.stage.0008 state: SCHEDULED
Update: ##pipeline.0004.stage.0014.task.0059 state: DONE
Update: #pipeline.0009.stage.0029 state: SCHEDULED
Update: #pipeline.0004.stage.0014 state: DONE
##Update: #pipeline.0004 state: DONE
#Update: ##pipeline.0001.stage.0005.task.0023 state: DONE
#Update: #pipeline.0001.stage.0005 state: DONE
##Update: #pipeline.0001 state: DONE
#Update: ##pipeline.0007.stage.0023.task.0095 state: EXECUTED
#Update: #pipeline.0007.stage.0023.task.0095 state: DONE
##Update: #pipeline.0007.stage.0023 state: DONE
#Update: ##pipeline.0007 state: DONE
#Update: #pipeline.0006.stage.0020.task.0083 state: EXECUTED
##Update: #pipeline.0006.stage.0020.task.0083 state: DONE
#Update: ##pipeline.0006.stage.0020 state: DONE
#Update: #pipeline.0006 state: DONE

Update: pipeline.0000.stage.0002.task.0011 state: EXECUTED
Update: pipeline.0000.stage.0002.task.0011 state: DONE
Update: pipeline.0000.stage.0002 state: DONE
Update: pipeline.0000 state: DONE
Update: pipeline.0005.stage.0017.task.0071 state: DONE
Update: pipeline.0005.stage.0017 state: DONE
Update: pipeline.0005 state: DONE
Update: pipeline.0002.stage.0008.task.0035 state: DONE
Update: pipeline.0002.stage.0008 state: DONE
Update: pipeline.0002 state: DONE
Update: pipeline.0009.stage.0029.task.0119 state: DONE
Update: pipeline.0009.stage.0029 state: DONE
Update: pipeline.0009 state: DONE
close task manager                                                    ok
#


1669119599.495 : radical.entk.task_manager.0000 : 2110368 : 139860056708864 : ERROR    :␣
↪Heartbeat response no body

closing session re.session.radical.3.mturilli.019318.0000              \
close pilot manager                                                    \
wait for 1 pilot(s)
      ␣
↪039m39mtextbackslash39m39mtextbackslash39m39mtextbackslash39m39mtextbackslash39m39mtextbackslash39m39m
↪                                                  timeout
                                                                      ok
session lifetime: 120.3s                                              ok
```

(continues on next page)

(continued from previous page)

```
All components terminated
```

We check results which were staged back

[10]:
```
for cnt in range(n_pipelines):
    data = open('pipe.%03d.sum.txt' % cnt).read()
    result = int(data)
    print('%3d -- %25d' % (cnt, result))
```

```
  0 --    3970061241222038761260
  1 --        842343099884068959
  2 --   1544905336569047335135
  3 --     614901618755976010525
  4 --   2032690349846648892114
  5 --         46503479300141160
  6 --    3970061241222038761260
  7 --   3694381919610155001250
  8 --     832923773978133231460
  9 --       1855803167254306908
```

2.2.6 Swift/T Getting Started Tutorial

[1]:
```
! git clone -b main --single-branch https://github.com/ExaWorks/Tutorial.git ./sdk-
↪examples
```

```
fatal: destination path './sdk-examples' already exists and is not an empty directory.
```

[2]:
```
import os

path = os.environ['PATH']
os.environ['PATH'] = "%s:/tmp/swift-t-install/stc/bin:/tmp/swift-t-install/turbine/bin"
↪% path
```

[3]:
```
def swift_t(code):
    import subprocess
    subprocess.run(["swift-t", "-t -oversubscribe -E", code])
```

[4]:
```
swift_t("trace(42);")
```

```
Could not read: trace(42);
```

[5]:
```
def swift_t_file(directory, filename):
    import os, subprocess
    original = os.getcwd()
    os.chdir(directory)
    try:
        subprocess.run(["swift-t -t -oversubscribe", filename])
    except:
        pass
    os.chdir(original)
```

[6]:
```
swift_t_file("sdk-examples/2-workflow-dl-swift/01-hello", "hello.swift")
```

[7]:
```
swift_t_file("sdk-examples/2-workflow-dl-swift/02-loop", "loop.swift")
```

CHAPTER

THREE

CONTRIBUTING TO SDK

This document outlines the policies and recommendations for inclusion in the ExaWorks SDK. The policies are based on those defined by the xSDK and LLNL RADIUSS projects and reflect best practices for open source development, development of sustainable software, and scalable deployment of software on ECP systems.

- **M** indicates a mandatory policy
- **R** indicates a recommended policy

3.1 Licensing

- M: Use an OSI-approved open-source license (e.g., Apache, MIT, BSD, LGPL)
- R: Provide a list of dependencies and their licenses in a standard format (e.g., SPDX)

3.2 Code

- M: Code should be version controlled and publicly accessible online
- M: Provide a transparent, online contribution process based on published contributing guide, and using pull requests and issues collection
- R: Code should be version controlled using Git and accessible on GitHub
- R: Follow a common style guide for code layout, documentation, naming, etc. (e.g., PEP8)

3.3 Packaging

- M: Package and provide automated builds using Spack and Conda
- M: Use a limited, unique, and well-defined symbol, macro, library, include, and/or module namespace.

3.4 Software design

- M: Provide configurable logging that adheres to standard logging approaches.
- M: Use MPI in a way that is compatible with other products, i.e., multiple tools using MPI at the same time vs. leveraging multiple MPI implementations).
- M: Provide a runtime API to return the current version of the software and system configuration.

3.5 Documentation

- M: Publish documentation in a web-based format.
- M: Provide a concise description of the project.
- M: Version control documentation consistent with and alongside source code.
- M: Provide a documented, reliable way to contact the development team.
- M: Provide and maintain example source code along with documentation.
- M: Provide a documented policy for handling pull requests from external contributors.

3.6 Testing and continuous integration

- M: Provide a comprehensive test suite for verifying correctness of build and installation.
- M: Use regression tests in the development process.
- M: Use continuous integration (CI).
- R: Measure and record test coverage as part of CI

3.7 Portability

- M: Give best effort at portability to common HPC platforms, schedulers, and software.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search