Advances in HPC automation - An update on the use of Parsl in Parallel Works
Outline:

1. Parallel Works clusters

2. Parsl workflows (goals and stumbling blocks)

3. Parsl Jupyter notebooks
Parallel Works Clusters

Provision HPC SLURM clusters in the cloud:

- Same “feel” & performance as on-premise SLURM clusters
- Elastic & highly customizable
- Leverage cloud’s cost: performance & new hardware
- Choice of several clouds

Connect an on-premise SLURM cluster

Uniform API
Parsl workflows and notebooks are started the same way in all clusters
To define a cluster simply click on the desired cluster type and fill in the configuration options.
Users can activate these clusters with a power button in a uniform way.
Uniform API for all clusters and SSH access from the user container to the controller (master) node of the cluster

PW API response. E.g.:
- Can query the PW API with the pool name and get the IP address of the controller node

```json
id: 62b316233710d0909504605
name: gcpslurm2
description: GCP Slurm Test
hidden: false
tags: gcp, slurm, cpu
visibility: private
status: on
settings:
  - registeredWorkers: 1
  - requestedWorkers:
    masterNode: 34.171.136.114
type: gclusterv2
collapse: false
key: 616d8a7fd6e77c918a2e07
mfa: false
imageurl: null
```
How can we run Parsl in these clusters?
Launch workflows in two ways:

1. Web user interface on PW
2. Python PW Client
   - CI/CD use case: GitHub action starts the job (with API key in repo secrets)
Launch workflow using the Web UI
1. Click on workflow thumbnail
2. Enter workflow parameters
3. Click execute → Generates the workflow command and arguments
4. Workflow command and arguments are executed in the user container
Launch workflow using the PW Client
1. Launch workflow with a Python script
2. Automation (e.g.: Github actions)

```python
import sys
from client import Client
from client_functions import *

pw_user_host = sys.argv[1]
pw_api_key = sys.argv[2]
user = sys.argv[3]
resource_name = sys.argv[4]
wf_name = sys.argv[5]
wf_xml_args = json.loads(sys.argv[6])

c = Client('https://{}:{}'.format(pw_user_host, pw_api_key))

start_resource(resource_name, c)

jid, djid = launch_workflow(wf_name, wf_xml_args, user, c)
```

```
on: [push]
jobs:
  test-pw-workflow:
    runs-on: ubuntu-latest
    name: test-pw-workflow-beluga
    steps:
      - name: run-workflow-beluga
        id: run-beluga
        uses: parallelworks/test-workflow-action@v5
        with:
          pw-user-host: 'beluga.parallel.works'
          pw-api-key: ${{ secrets.ALVAROVIDALTO_BELUGA_API_KEY }}
          pw-user: 'alvarovidalto'
          resource-pool-names: 'gcpslurmv2'
          workflow-name: 'singlecluster_parsl_demo'
          workflow-parameters: '{"name": "PW_USER"}'
```

PW Client use case example

Github action example
Goals for Parsl workflows:
- Moving from a custom modified Parsl to standard Parsl
- Parsl script runs in the user container in Parallel Works (not in the cluster)
- Run different Parsl apps in different clusters (including on-premise and cloud)
- Share Parsl workflows with other users

Stumbling blocks:
1. Define Parsl configuration for the different resources. Point to PW pools by pool name.
2. Manage python environment in the user container in PW and in the remote resources. Parsl version needs to be compatible. Dependencies.
   - Workflow may run in a different user container (shared) and/or in a different cluster
3. Establish port connections from the workers to the user container
   - User container does not have direct access to worker ports
Dealing with the stumbling blocks

Parsl workflow wrapper
1. Define **Parsl configuration** definition for the different resources:
   - JSON configuration file
   - PW API to get pool information by pool name:
     - IP addresses and user name of the controller nodes
     - Available worker ports
   - SSHChannel to connect to the controller nodes
   - Run in **controller** nodes: LocalProvider
   - Run in **compute** nodes: SlurmProvider or LocalProvider + bash_app + srun (easier to reach ports)
1. Define **Parsl configuration** definition for the different resources:
   - JSON configuration file
   - PW API to get pool information by pool name:
     - IP addresses and user name of the controller nodes
     - Available worker ports
   - **SSHChannel** to connect to the controller nodes
   - Run in **controller** nodes: **LocalProvider**
   - Run in **compute** nodes: **SlurmProvider** or **LocalProvider + bash_app + srun** (easier to reach ports)

   JSON IS LOADED AND USED TO DEFINE THE PARSL CONFIGURATION

```python
import os
import subprocess

config = Config(
    executors = [
        ThroughputExecutor(
            worker_ports = ((int(exec_conf['myexecutor_1']['WORKER_PORT_1']), int(exec_conf['myexecutor_1']['WORKER_PORT_2']))),
            label = 'myexecutor_1',
            worker_debug = True,
            cores_per_worker = 1, # One worker per node
            worker_logdir_root = '/local_worker',
            provider = LocalProvider(
                worker_init = 'source conda.sh; conda activate conda_env; cd %run_dir%.format(conda_sh = os.path.abspath(conda_env)); cd %run_dir%.format(conda_env = exec_conf['myexecutor_1']['CONDA_DIR'])
            , run_dir = exec_conf['myexecutor_1']['RUN_DIR']
            ),
            channel = SSHChannel(),
            hostname = exec_conf['myexecutor_1']['HOST_IP'],
            username = exec_conf['myexecutor_1']['HOST_USER'],
            script_dir = exec_conf['myexecutor_1']['WORKER_SCRIPT_DIR'],
            key_filename = '/home/%USER%/ssh/pw_rsa'.format(VM_USER = os.environ['VM_USER'])
            ),
        ),
    ],
    monitoring = MonitoringHub(
        hub_address = address_by_hostname(),
        resource_monitoring_interval = 5
    ),
)
```

**print('Loading Parsl Config', flush = True)**

```
parsl.load(config)
```
2. Manage **python environment** in the user container in PW and in the remote resources. Parsl version needs to be compatible. Dependencies.

   - Python environment is defined in **YAML or singularity definition files** (better for ML applications)
   - Can choose one per executor and another for the user container
   - Parsl workflow wrapper optionally updates/installs the Python environment from these files
2. Manage **python environment** in the user container in PW and in the remote resources. Parsl version needs to be compatible. Dependencies.

- Python environment is defined in **YAML or singularity definition files** (better for ML applications)
- Can choose one per executor and another for the user container
- Parsl workflow wrapper optionally updates/install the Python environment from these files

**Options:**

1. Point executor to different python environment
2. Install the Python environment manually once
3. Workflow wrapper installs python environment at runtime (INSTALL_CONDA=TRUE)

**Python environment**

Optional to install the Python environment at runtime
3. Establish **port connections for workers**
   - Parsl workflow wrapper creates SSH tunnels for the worker ports before execution and cleans them after execution
   - Available port numbers are provided by the PW API
Jupyter Notebooks
Goals:
- Connect from the user container to the jupyter server
- Automate server launch

Approach:
- Jupyter server runs in the controller node of a slurm cluster
- Server port is forwarded to the user container → Only the server port is forwarded to the user container!

Limitations:
- Single cluster (multiple partitions) per Parsl job
Approach:
- For automation, the Jupyter server is started by a PW workflow
Approach:
- When the server is ready it pops up in the PW interface
- Enter your password and connect to the server
Parsl notebook on the cluster
Approach:
- Send jobs to different partitions using the SlurmProvider

```python
In [3]:
# Print environment information
import socket, sys, os, json
print('Hostname: %s, socket.gethostname()')
Hostname: mgmt-userdemo-gcpslurm2-00108

In [4]:
# hbash
import sinfo
squeue
PARTITION AVAIL TIMELIMIT NODES STATE Nodelist
compute* up infinite 9 idle-userdemo-gcpslurm2-00108-1-[0002-0010]
compute* up infinite 1 alloc userdemo-gcpslurm2-00108-1-6001
gpu up infinite 2 idle-userdemo-gcpslurm2-00108-2-[0001-0002]
fail up infinite 1 idle-userdemo-gcpslurm2-00108-3-0001

```

Define the Parsl configuration

```python
In [2]:
import parsl
print(parsl.__version__)
from parsl.config import Config
from parsl.executors import HighThroughputExecutor
from parsl.providers import SlurmProvider
parsl.clear()
# Configuration variable tells Parsl where to run the python functions
# ... Depends on the resource and tasks to run
config = Config(
extectors=[
    HighThroughputExecutor,
    worker_debug=True, # Default False for shorter logs
cores_per_worker=int(2), # One worker per node
worker_logdir=os.getcwd() + '/parsllogs',
provider=SlurmProvider(),
scheduler_options = '
SITA--gres=gpu:1', # For GPU runs!
<===
```
**Approach:**
- Kill jupyter server job
Potential next steps

● Implement failover in Parsl workflows
  ○ Associate multiple resources with a given Parsl app
  ○ Resources are ranked; if #1 fails, try #2…

● Streamline the definition of the Parsl configuration through the web UI instead of editing the JSON file

Thank You!