The architecture for running Parsl multi-site workflows on the Parallel Works platform

Stefan Gary, Alvaro Vidal Torreira, Matthew Shaxted, Michael McQuade, Quan Nguyen, Louis Le, Matt Long, Michael Wilde

ParslFest
October 19, 2023
Motivation

What is a multi-site workflow?

Applies to:

- different partitions on the same cluster or
- different clusters at different sites/clouds

Why is a multi-site workflow useful?

- Collaboration/portability with other teams
- Changes cloud hardware or performance:cost

Modern HPC is **not** monolithic - some jobs run best on different resources, e.g.

Variables and Functions

```python
python_app(my_func, executors=[label])(...)
```
What does Parallel Works do?

- Configure, start cloud clusters
- Same feel on the major CSPs
- IDE (cloud terminal, workspace)
- GUI, CLI, API launch Parsl workflows
- Collaborates for demonstration stages of R&D SBIR grants; partner with academic teams for real-world, funder-relevant testbeds.
Typical multi-site workflow stumbling blocks

- I’m guessing that most users run Parsl scripts on the head node of an on-premise cluster; as such, for most users, the resource is:
  - 1) **persistent** (user or sysadmin installs Parsl and it’s always accessible by all nodes) and
  - 2) on the same **local network** - ports on head node are accessible to worker nodes

- **SSH tunnels and ports:** clusters aren’t all on the same local network.
- **Parsl needs to be installed** everywhere with exactly the same versions - need automation for installation or attaching persistent storage when using **ephemeral** cloud resources.
- If using the FluxExecutor, Flux needs to be on the clusters; managing the installation of Flux and Parsl via Spack can be challenging. (But recently, Flux can be installed directly with Conda.)
Architecture for multi-site with HighThroughputExecutor

Credit: Parsl docs
How does it **actually** work?

1. PW connects to cluster head node with **ssh -R tunnel** and modifies ~/.ssh/config for easy connections back to PW (i.e. forwarding head node SSH port to PW).

2. PW **launches workflow with a bash script** in the user’s workspace on PW platform.

3. The launch script clones and then starts parsl-utils which:
   a. Gathers **resource information** (i.e. IP address of cluster) via PW API
   b. Establishes **SSH -L tunnels** port forwarding the Parsl worker ports from the cluster to PW
   c. Checks/installs Conda & Parsl
   d. Builds the **Parsl configuration** based on a, b, c & template

4. **Parsl workflow is launched**

https://github.com/parallelworks/parsl_utils
Parsl multi-site **future work**

- **Parsl ≥2023.7.24** - need HTEX address="*" and specify the local IP address of the head node when starting process_worker_pool.py in HTEX launch_cmd.
- A **custom data provider** in parsl_utils wraps file, GCP, and AWS bucket access. We plan to automate the workflow integration of storage information in a similar way as we do with compute resources.

**HTEX limitations for MPI jobs (and workaround), future work**

Parsl HTEX hardcodes SLURM --ntasks-per-node to 1; prevents running multiple MPI tasks on the same node in parallel.

It is possible to bypass this issue by changing `SLURM_TASKS_PER_NODE in the bash_app itself` and forcing a static number of Parsl blocks (i.e. `init_blocks = min_blocks = max_blocks = repeats` or parallelism > 1). The FluxExecutor is an alternative for launching MPI tasks in parallel.

Timeout/failover **future work**: The timeout starts counting when the app starts running, not when it is submitted (and queued).

Chris Harrop’s talk tomorrow

Thank you Ben and Yadu!