libsubmit Documentation

Release 0.1.0

Yadu Nand Babuji

Jul 19, 2018

Contents

Libsubmit is responsible for managing execution resources with a Local Resource Manager (LRM). For instance, campus clusters and supercomputers generally have schedulers such as Slurm, PBS, Condor and. Clouds on the other hand have API interfaces that allow much more fine grain composition of an execution environment. An execution provider abstracts these resources and provides a single uniform interface to them.

This module provides the following functionality:

1. A standard interface to schedulers
2. Support for submitting, monitoring and cancelling jobs
3. A modular design, making it simple to add support for new resources.
4. Support for pushing files from client side to resources.

CHAPTER 1

Quickstart

Libsubmit is an adapter to a variety of computational resources such as Clouds, Campus Clusters and Supercomputers. This python-module is designed to simplify and expose a uniform interface to seemingly diverse class of resource schedulers. This library originated from Parsl: Parallel scripting library and is designed to bring dynamic resource management capabilities to it.

1.1 Installing

Libsubmit is now available on PyPI, but first make sure you have Python3.5+

```
>>> python3 --version
```

1.1.1 Installing on Linux

1. Install Libsubmit:

   ```
   $ python3 -m pip install libsumit
   ```

2. Libsubmit supports a variety of computation resource via specific libraries. You might only need a subset of these, which can be installed by specifying the resources names:

   ```
   $ python3 -m pip install libsumit[<aws>,<azure>,<jetstream>]
   ```

1.1.2 Installing on Mac OS

1. Install Conda and setup python3.6 following instructions here:

   ```
   $ conda create --name libsubmit_py36 python=3.6
   $ source activate libsubmit_py36
   ```

2. Install Libsubnmit:

```
$ python3 -m pip install libsubmit[<optional_packages...>]
```

1.2 For Developers

1. Download Libsubmit:

```
$ git clone https://github.com/Parsl/libsubmit
```

2. Install:

```
$ cd libsubmit
$ python3 setup.py install
```

3. Use Libsubmit!

CHAPTER 2

Requirements

Libsubmit requires the following :

- Python 3.5+
- paramiko
- ipyparallel
- boto3 - for AWS
- azure, haikunator - for Azure
- python-novaclient - for jetstream

For testing:

- nose
- coverage

CHAPTER 3

User guide

3.1 Overview

Under construction. Please refer to the developer documentation as this section is being built.

3.2 Configuration

The primary mode by which you interact with libsubmit is by instantiating an ExecutionProvider with a configuration data structure and optional Channel objects if the ExecutionProvider requires it.

The configuration datastructure expected by an ExecutionProvider as well as options specifics are described below.

The config structure looks like this:

```
config = { "poolName" : <string: Name of the pool>,
           "provider" : <string: Name of provider>,
           "scriptDir" : <string: Path to script directory>,
           "minBlocks" : <int: Minimum number of blocks>,
           "maxBlocks" : <int: Maximum number of blocks>,
           "initBlocks" : <int: Initial number of blocks>,
           "block" : {      # Specify the shape of the block
               "nodes" : <int: Number of blocs, integer>,
               "taskBlocks" : <int: Number of task blocks in each block>,
               "walltime" : <string: walltime in HH:MM:SS format for the block>
               "options" : { # These are provider specific options
                    "partition" : <string: Name of partition/queue>,
                    "account" : <string: Account id>,
                    "overrides" : <string: String to override and specify options to␣
↪scheduler>
               }
         }
```

CHAPTER 4

Reference guide

libsubmit.channels.local.local. LocalChannel
libsubmit.channels.ssh.ssh.SshChannel
libsubmit.providers.aws.aws. EC2Provider
libsubmit.providers.azureProvider. azureProvider.AzureProvider
libsubmit.providers.cobalt.cobalt. Cobalt
libsubmit.providers.condor.condor. Condor
libsubmit.providers.googlecloud. googlecloud.GoogleCloud
libsubmit.providers.gridEngine. gridEngine.GridEngine
libsubmit.providers.jetstream. jetstream.Jetstream
libsubmit.providers.local.local.Local
libsubmit.providers.sge.sge. GridEngine
libsubmit.providers.slurm.slurm.Slurm
libsubmit.providers.torque.torque. Torque
libsubmit.providers.provider_base. ExecutionProvider

CHAPTER 5

Changelog

5.1 Libsubmit 0.4.1

Released. June 18th, 2018. This release folds in massive contributions from @annawoodard.

5.1.1 New functionality

- Several code cleanups, doc improvements, and consistent naming
- All providers have the initialization and actual start of resources decoupled.

5.2 Libsubmit 0.4.0

Released. May 15th, 2018. This release folds in contributions from @ahayschi, @annawoodard, @yadudoc

5.2.1 New functionality

- Several enhancements and fixes to the AWS cloud provider (#44, #45, #50)
- Added support for python3.4

5.2.2 Bug Fixes

- Condor jobs left in queue with X state at end of completion issue#26
- Worker launches on Cori seem to fail from broken ENV issue#27
- EC2 provider throwing an exception at initial run issue#46

CHAPTER 6

Developer documentation

6.1 ExecutionProviders

An execution provider is basically an adapter to various types of execution resources. The providers abstract away the interfaces provided by various systems to request, monitor, and cancel computate resources.

6.1.1 Slurm

6.1.2 Cobalt

6.1.3 Condor

6.1.4 Torque

6.1.5 Local

6.1.6 AWS

6.2 Channels

For certain resources such as campus clusters or supercomputers at research laboratories, resource requirements may require authentication. For instance some resources may allow access to their job schedulers from only their login-nodes which require you to authenticate on through SSH, GSI-SSH and sometimes even require two factor authentication. Channels are simple abstractions that enable the ExecutionProvider component to talk to the resource managers of compute facilities. The simplest Channel, *LocalChannel* simply executes commands locally on a shell, while the *SshChannel* authenticates you to remote systems.

6.2.1 LocalChannel

6.2.2 SshChannel

6.2.3 SshILChannel

6.3 Launchers

Launchers are basically wrappers for user submitted scripts as they are submitted to a specific execution resource.

CHAPTER 7

Packaging

Currently packaging is managed by Yadu.

Here are the steps:

```
# Depending on permission all of the following might have to be run as root.
sudo su

# Make sure to have twine installed
pip3 install twine

# Create a source distribution
python3 setup.py sdist

# Create a wheel package, which is a prebuilt package
python3 setup.py bdist_wheel

# Upload the package with twine
# This step will ask for username and password for the PyPi account.
twine upload dist/*
```

CHAPTER 8

Indices and tables

- genindex
- modindex
- search