libsubmit Documentation

Release 0.1.0

Yadu Nand Babuji

Dec 20, 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	This is not even really a channel, since opening a local shell is not heavy and done so infrequently that they do not need a persistent channel
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	Define the strict interface for all Execution Provider

4.1 libsubmit.providers.aws.aws.EC2Provider

4.2 libsubmit.providers.cobalt.cobalt.Cobalt

4.3 libsubmit.providers.condor.condor.Condor

4.4 libsubmit.providers.googlecloud.googlecloud.GoogleCloud

4.5 libsubmit.providers.gridEngine.gridEngine.GridEngine

4.6 libsubmit.providers.jetstream.jetstream.Jetstream

4.7 libsubmit.providers.local.local.Local

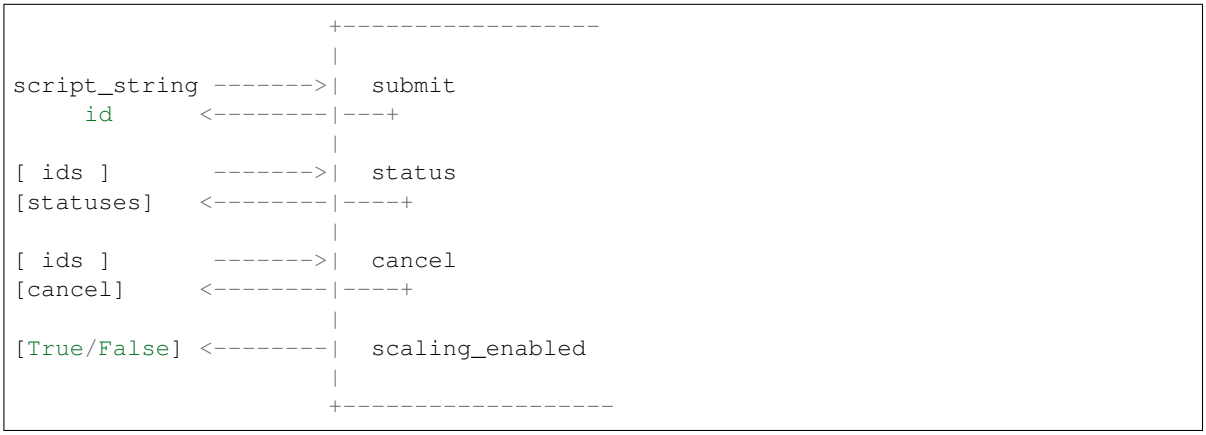
4.8 libsubmit.providers.slurm.slurm.Slurm

4.9 libsubmit.providers.torque.torque.Torque

4.10 libsubmit.providers.provider_base.ExecutionProvider

class libsubmit.providers.provider_base.**ExecutionProvider**

Define the strict interface for all Execution Provider



__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

cancel(job_ids)	Cancels the resources identified by the job_ids provided by the user.
status(job_ids)	Get the status of a list of jobs identified by the job identifiers returned from the submit request.
submit(command, blocksize[, job_name])	The submit method takes the command string to be executed upon instantiation of a resource most often to start a pilot (such as IPP engine or even Swift-T engines).

Attributes

ExecutionProvider. channels_required	
scaling_enabled	The callers of ParslExecutors need to differentiate between Executors and Executors wrapped in a resource provider

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 Libsubmit

Uniform interface to diverse and multi-lingual set of computational resources.

libsubmit.**set_stream_logger**(*name='libsubmit'*, *level=10*, *format_string=None*)
Add a stream log handler

Parameters

- **name** (–) – Set the logger name.
- **level** (–) – Set to logging.DEBUG by default.
- **format_string** (–) – Set to None by default.

Returns

- None

libsubmit.**set_file_logger**(*filename*, *name='libsubmit'*, *level=10*, *format_string=None*)
Add a stream log handler

Parameters

- **filename** (–) – Name of the file to write logs to
- **name** (–) – Logger name
- **level** (–) – Set the logging level.
- **format_string** (–) – Set the format string

Returns

- None

6.2 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.

6.2.1 Slurm

6.2.2 Cobalt

6.2.3 Condor

6.2.4 Torque

6.2.5 Local

6.2.6 AWS

6.3 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.

class libsubmit.channels.channel_base.**Channel**

Define the interface to all channels. Channels are usually called via the execute_wait function. For channels that execute remotely, a push_file function allows you to copy over files.

close()

Closes the channel. Clean out any auth credentials.

Parameters **None** –

Returns Bool

execute_no_wait(*cmd*, *walltime*, *envs={}*, **args*, ***kwargs*)

Optional. THis is infrequently used.

Parameters

- **cmd** (–) – Command string to execute over the channel
- **walltime** (–) – Timeout in seconds

KWargs:

- envs (dict) : Environment variables to push to the remote side

Returns

- (exit_code(None), stdout, stderr) (int, io_thing, io_thing)

execute_wait (*cmd*, *walltime*, *envs={}*, **args*, ***kwargs*)
Executes the cmd, with a defined walltime.

Parameters

- **cmd** (–) – Command string to execute over the channel
- **walltime** (–) – Timeout in seconds

KWargs:

- envs (dict) : Environment variables to push to the remote side

Returns

- (exit_code, stdout, stderr) (int, string, string)

push_file (*source*, *dest_dir*)
Channel will take care of moving the file from source to the destination directory

Parameters

- **source** (*string*) – Full filepath of the file to be moved
- **dest_dir** (*string*) – Absolute path of the directory to move to

Returns destination_path (string)

script_dir
This is a property. Returns the directory assigned for storing all internal scripts such as scheduler submit scripts. This is usually where error logs from the scheduler would reside on the channel destination side.

Parameters **None** (–) –

Returns

- Channel script dir

6.3.1 LocalChannel

class libsubmit.channels.local.local.**LocalChannel** (*userhome='.'*, *envs={}*, *script_dir='./.scripts'*, ***kwargs*)
This is not even really a channel, since opening a local shell is not heavy and done so infrequently that they do not need a persistent channel

__init__ (*userhome='.'*, *envs={}*, *script_dir='./.scripts'*, ***kwargs*)
Initialize the local channel. script_dir is required by set to a default.

KwArgs:

- userhome (string): (default='.') This is provided as a way to override and set a specific userhome
- envs (dict) : A dictionary of env variables to be set when launching the shell
- script_dir (string): (default="./.scripts") Directory to place scripts

close()
There's nothing to close here, and this really doesn't do anything

Returns

- False, because it really did not "close" this channel.

execute_no_wait(*cmd*, *walltime*, *envs={}*)
Synchronously execute a commandline string on the shell.

Parameters

- **cmd** (–) – Commandline string to execute
- **walltime** (–) – walltime in seconds, this is not really used now.

Returns Return code from the execution, -1 on fail - stdout : stdout string - stderr : stderr string

Return type

- retcode

Raises None.

execute_wait(*cmd*, *walltime*, *envs={}*)
Synchronously execute a commandline string on the shell.

Parameters

- **cmd** (–) – Commandline string to execute
- **walltime** (–) – walltime in seconds, this is not really used now.

Kwargs:

- envs (dict) : Dictionary of env variables. This will be used to override the envs set at channel initialization.

Returns Return code from the execution, -1 on fail - stdout : stdout string - stderr : stderr string

Return type

- retcode

Raises: None.

push_file(*source*, *dest_dir*)
If the source files dirpath is the same as dest_dir, a copy is not necessary, and nothing is done. Else a copy is made.

Parameters

- **source** (–) – Path to the source file
- **dest_dir** (–) – Path to the directory to which the files is to be copied

Returns Absolute path of the destination file

Return type

- destination_path (String)

Raises - *FileCopyException* – If file copy failed.

script_dir
This is a property. Returns the directory assigned for storing all internal scripts such as scheduler submit scripts. This is usually where error logs from the scheduler would reside on the channel destination side.

Parameters **None** (–) –

Returns

- Channel script dir

6.3.2 SshChannel

6.3.3 SshILChannel

6.4 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

Python Module Index

l

Index

Symbols

C

E

L

P

S