sparkhpc Documentation

Release 0.1

Rok Roskar

Dec 05, 2018

Contents

This package tries to greatly simplify deploying and managing Apache Spark clusters on HPC resources.

CHAPTER 1

Installation

1.1 From pypi

```
$ pip install sparkhpc
```

1.2 From source

```
$ python setup.py install
```

This will install the python package to your default package directory as well as the `sparkcluster` and `hpcnotebook` command-line scripts.

CHAPTER 2

Usage

There are two options for using this library: from the command line or directly from python code.

2.1 Command line

2.1.1 Get usage info

```
Usage: sparkcluster [OPTIONS] COMMAND [ARGS]...

Options:
  --scheduler [lsf|slurm]  Which scheduler to use
  --help                   Show this message and exit.

Commands:
  info    Get info about currently running clusters
  launch  Launch the Spark master and workers within a...
  start   Start the spark cluster as a batch job
  stop    Kill a currently running cluster ('all' to...

$ sparkcluster start --help
Usage: sparkcluster start [OPTIONS] NCORES

  Start the spark cluster as a batch job

Options:
  --walltime TEXT                Walltime in HH:MM format
  --jobname TEXT                 Name to use for the job
  --template TEXT                Job template path
  --memory-per-executor INTEGER  Memory to reserve for each executor (i.e. the
                                 JVM) in MB
  --memory-per-core INTEGER      Memory per core to request from scheduler in
                                 MB
```

(continues on next page)

(continued from previous page)

```
  --cores-per-executor INTEGER   Cores per executor
  --spark-home TEXT              Location of the Spark distribution
  --wait                         Wait until the job starts
  --help                         Show this message and exit.
```

2.1.2 Start a cluster

```
$ sparkcluster start 10
```

2.1.3 Get information about currently running clusters

```
$ sparkcluster info
----- Cluster 0 -----
Job 31454252 not yet started

$ sparkcluster info
----- Cluster 0 -----
Number of cores: 10
master URL: spark://10.11.12.13:7077
Spark UI: http://10.11.12.13:8080
```

2.1.4 Stop running clusters

```
$ sparkcluster stop 0
Job <31463649> is being terminated
```

2.2 Python code

```
from sparkhpc import sparkjob
import findspark
findspark.init() # this sets up the paths required to find spark libraries
import pyspark

sj = sparkjob.sparkjob(ncores=10)

sj.wait_to_start()

sc = sj.start_spark()

sc.parallelize(...)
```

2.3 Jupyter notebook

sparkhpc gives you nicely formatted info about your jobs and clusters in the jupyter notebook - see the example notebook.

CHAPTER 3

Dependencies

3.1 Python

- click
- findspark

These are installable via `pip install`.

3.2 System configuration

- Spark installation in `~/spark` OR wherever `SPARK_HOME` points to
- java distribution (set `JAVA_HOME`)
- `mpirun` in your path

3.3 Job templates

Simple job templates for the currently supported schedulers are included in the distribution. If you want to use your own template, you can specify the path using the `--template` flag to `start`. See the included templates for an example. Note that the variable names in curly braces, e.g. `{jobname}` will be used to inject runtime parameters. Currently you must specify `walltime`, `ncores`, `memory`, `jobname`, and `spark_home`. If you want to significantly alter the job submission, the best would be to subclass the relevant scheduler class (e.g. `LSFSparkCluster`) and override the `submit` method.

CHAPTER 4

Using other schedulers

The LSF and SLURM schedulers are currently supported. However, adding support for other schedulers is rather straightforward (see the `LSFSparkJob` and `SLURMSparkJob` implementations as examples). Please submit a pull request if you implement a new scheduler or get in touch if you need help!

To implement support for a new scheduler you should subclass `SparkCluster`. You must define the following *class* variables:

- `_peek()` (function to get stdout of the current job)
- `_submit_command` (command to submit a job to the scheduler)
- `_job_regex` (regex to get the job ID from return string of submit command)
- `_kill_command` (scheduler command to kill a job)
- `_get_current_jobs` (scheduler command to return jobid, status, jobname one job per line)

Note that `_get_current_jobs` should return a custom formatted string where the output looks like this:

```
JOB_NAME STAT JOBID
sparkcluster PEND 31610738
sparkcluster PEND 31610739
sparkcluster PEND 31610740
```

Depending on the scheduler's behavior, you may need to override some of the other methods as well.

CHAPTER 5

Jupyter notebook

Running Spark applications, especially with python, is really nice from the comforts of a Jupyter notebook. This package includes the `hpcnotebook` script, which will setup and launch a secure, password-protected notebook for you.

```
$ hpcnotebook
Usage: hpcnotebook [OPTIONS] COMMAND [ARGS]...

Options:
  --port INTEGER  Port for the notebook server
  --help          Show this message and exit.

Commands:
  launch  Launch the notebook
  setup   Setup the notebook
```

5.1 Setup

Before launching the notebook, it needs to be configured. The script will first ask for a password for the notebook and generate a self-signed ssh certificate - this is done to prevent other users of your cluster to stumble into your notebook by chance.

5.2 Launching

On a computer cluster, you would normally either obtain an interactive job and issue the command below, or use this as a part of a batch submission script.

```
$ hpcnotebook launch
To access the notebook, inspect the output below for the port number, then point your␣
↪browser to https://1.2.3.4:<port_number>
```

(continues on next page)

(continued from previous page)

```
[TerminalIPythonApp] WARNING | Subcommand `ipython notebook` is deprecated and will␣
↪be removed in future versions.
[TerminalIPythonApp] WARNING | You likely want to use `jupyter notebook` in the future
[I 15:43:12.022 NotebookApp] Serving notebooks from local directory: /cluster/home/
↪roskarr
[I 15:43:12.022 NotebookApp] 0 active kernels
[I 15:43:12.022 NotebookApp] The Jupyter Notebook is running at: https://[all ip␣
↪addresses on your system]:8889/
[I 15:43:12.022 NotebookApp] Use Control-C to stop this server and shut down all␣
↪kernels (twice to skip confirmation).
```

In this case, you could set up a port forward to host `1.2.3.4` and instruct your browser to connect to `https://1.2.3.4:8889`.

Inside the notebook, it is straightforward to set up the `SparkContext` using the `sparkhpc` package (see above).

CHAPTER 6

Contributing

Please submit an issue if you discover a bug or have a feature request! Pull requests also very welcome.

CHAPTER 7

API

class sparkhpc.lsfsparkjob.**LSFSparkJob**(*clusterid=None*, *jobid=None*, *ncores=4*, *cores_per_executor=1*, *walltime='00:30'*, *memory_per_core=2000*, *memory_per_executor=None*, *jobname='sparkcluster'*, *template=None*, *extra_scheduler_options=''*, *config_dir=None*, *spark_home=None*, *master_log_dir=None*, *master_log_filename='spark_master.out'*, *scheduler=None*)

Bases: sparkhpc.sparkjob.SparkJob

Class for submitting spark jobs with the LSF scheduler

class sparkhpc.slurmsparkjob.**SLURMSparkJob**(*walltime='00:30'*, ***kwargs*)

Bases: sparkhpc.sparkjob.SparkJob

Class for submitting spark jobs with the SLURM scheduler

See the *SparkJob* class for keyword descriptions.

class sparkhpc.sparkjob.**SparkJob**(*clusterid=None*, *jobid=None*, *ncores=4*, *cores_per_executor=1*, *walltime='00:30'*, *memory_per_core=2000*, *memory_per_executor=None*, *jobname='sparkcluster'*, *template=None*, *extra_scheduler_options=''*, *config_dir=None*, *spark_home=None*, *master_log_dir=None*, *master_log_filename='spark_master.out'*, *scheduler=None*)

Bases: object

Generic SparkJob class

To implement other schedulers, you must simply extend this class and define some class variables:

- *_peek_command* (command to get stdout of current job)
- *_submit_command* (command to submit a job to the scheduler)
- *_job_regex* (regex to get the job ID from return string of submit command)

- *_kill_command* (scheduler command to kill a job)
- *_get_current_jobs* (scheduler command to return jobid, status, jobname one job per line)

See the LSFSparkJob class for an example.

`__init__`(*clusterid=None*, *jobid=None*, *ncores=4*, *cores_per_executor=1*, *walltime='00:30'*, *memory_per_core=2000*, *memory_per_executor=None*, *jobname='sparkcluster'*, *template=None*, *extra_scheduler_options="*, *config_dir=None*, *spark_home=None*, *master_log_dir=None*, *master_log_filename='spark_master.out'*, *scheduler=None*)

Creates a SparkJob

Parameters:

clusterid: int if a spark cluster is already running, initialize this SparkJob with its metadata

jobid: int same as *clusterid* but using directly the scheduler job ID

ncores: int number of cores to request

walltime: string walltime in *HH:MM* format as a string

memory_per_core: int memory to request per core from the scheduler in MB

memory_per_executor: int memory to give to each spark executor (i.e. the jvm part) in MB If using pyspark and python workers need a lot of memory, this should be less than *memory_per_core * ncores*.

jobname: string name for the job - only used for the scheduler

template: file path custom template to use for job submission

extra_scheduler_options: string A string with custom options for the scheduler

config_dir: directory path path to spark configuration directory

spark_home: path to spark directory; default is the *SPARK_HOME* environment variable, and if it is not set it defaults to *~/spark*

master_log_dir: path to directory; default is {spark_home}/logs

master_log_filename: Name of the file that the Spark master's output will be written to under {master_log_dir}; default is spark_master.out

scheduler: string specify manually which scheduler you want to use; usually the automatic determination will work fine so this should not be used

Example usage:

> from sparkhpc.sparkjob import sparkjob import findspark findspark.init() # this sets up the paths required to find spark libraries import pyspark
>
> sj = sparkjob(ncores=10)
>
> sj.wait_to_start()
>
> sc = pyspark.SparkContext(master=sj.master_url())
>
> sc.parallelize(. . .)

`classmethod current_clusters`()

Determine which Spark clusters are currently running or in the queue

`job_started`()

Check whether the job is running already or not

`master_ui`()

Get the UI address of the Spark master

master_url()

Get the URL of the Spark master

start_spark(*spark_conf=None*, *executor_memory=None*, *profiling=False*, *graphframes_package='graphframes:graphframes:0.3.0-spark2.0-s_2.11'*, *extra_conf=None*)

Launch a SparkContext

Parameters

spark_conf: path path to a spark configuration directory

executor_memory: string executor memory in java memory string format, e.g. '4G' If *None*, *memory_per_executor* is used.

profiling: boolean whether to turn on python profiling or not

graphframes_package: string which graphframes to load - if it isn't found, spark will attempt to download it

extra_conf: dict additional configuration options

stop()

Stop the current job

submit()

Write job file to current working directory and submit to the scheduler

wait_to_start(*timeout=60*)

Wait for the job to start or until timeout, whichever comes first

sparkhpc.sparkjob.**start_cluster**(*memory*, *cores_per_executor=1*, *timeout=30*, *spark_home=None*, *master_log_dir=None*, *master_log_filename='spark_master.out'*)

Start the spark cluster

This is the script used to launch spark on the compute resources assigned by the scheduler.

Parameters

memory: string memory specified using java memory format

timeout: int time in seconds to wait for the master to respond

spark_home: directory path path to base spark installation

master_log_dir: directory path path to directory where the spark master process writes its stdout/stderr to a file name spark_master.out

master_log_filename: string name of the file to write Spark master's output to.

Python Module Index

Index