SPC Documentation

Release 0.33

Wesley Brewer

Aug 05, 2020

Contents

The Scientific Platform for the Cloud (SPC) is a cloud platform for easily migrating and running scientific applications in the cloud. It is described in more detail in the following paper which is available from http://ieeexplore.ieee.org/document/7421906/:

> W Brewer, W Scott, and J Sanford, "An Integrated Cloud Platform for Rapid Interface Generation, Job Scheduling, Monitoring, Plotting, and Case Management of Scientific Applications", Proc. of the International Conference on Cloud Computing Research and Innovation, Singapore, IEEE Press, October 2015, pp. 156-165, doi:10.1109/ICCCRI.2015.24

This platform is ideally suited to run scientific applications that: (1) require an input deck full of integers, floats, strings, and booleans stored in a standardized file format such as INI, XML, JSON, YAML, or Namelist.input, (2) require some amount of processing time (i.e. not instantaneous – although it can handle such cases too), (3) require some plotting at the end of the simulation, (4) use MPI or MapReduce for parallelization (although handles serial cases well too). Moreover, it can handle other applications as well, but some amount of pre- or post-processing may be required. Such topics are described in more detail in the aforementioned paper.

CHAPTER 1

Getting started

The latest version of SPC can be downloaded from https://github.com/whbrewer/spc. This was developed using Python 2.7 for compatibility with other modules, and thus is not currently compatible with Python 3.

Currently, SPC cannot be installed as a typical application but simply runs from wherever the directory is located. To setup SPC, one needs to first generate a database, and also a Python configuration file (src/config.py) as follows:

```
> spc init
```

To start running the server, then type:

```
> spc run
```

This will start the web server listening on port 8580. Then, open your web browser to: http://localhost:8580/ (or http://127.0.0.1:8580, or http://0.0.0.0:8580).

“spc init” will also install a simple Python demo app called simply “dna”, which asks the user to input a string of DNA, and then will analyze its content.

CHAPTER 2

Installing Apps

Adding apps can be accomplished in two ways: (1) interactively via the user interface, or (2) via the command-line options (see section on package management).

2.1 A. Manually adding app through user interface

To install or modify apps interactively, one must be logged in as user admin (default password admin). Note, if user authentication is turned, off one will still need to login as admin, by clicking the hamburger icon , logging out, and then logging back in as admin. Before this can happen, enable authentication by setting "auth = True" in the src/config.py file. To install an app interactively, click the Apps button, then click the +Add button. Installation of an app requires five steps:

1. **Database entry setup** – The first step is to store some basic information about the app into the database:

    - **Name** – A unique name for the app, with no spaces or punctuation marks.

    Description – Description of the app. What does it do? Input format – There are three possible options for input format: (1) namelist, (2) INI format, (3) XML format. In the future, support may be added for JSON format as well.

    - **Command** – This is the command to run to start the simulation. This command is being run from the working directory of the case, which is user_data/user/app/case (e.g. user_data/joe/mendel/c82d3f). For example, this may look like:

    ```
    <rel_apps_dir>/mendel/mendel
    ```

    (or mendel.exe for a Windows machine). Here <rel_apps_dir> will be later replaced by the location where the SPC apps are installed (currently this would replaced by the string ../../../../src/spc_apps). For a Java application, this may look like:

    ```
    /usr/bin/java -jar <rel_apps_dir>/jmendel/dist/jmendel.jar
    ```

    NOTE: since v0.22, the app run command can no longer be specified through the web interface. It must instead be specified in the spc.json file. Therefore, after setting up an, app, one may generate an spc.json manifest file by going in to the app configuration page (Click on top right "hamburger" icon and click "Configure app", then click the "Export" button). This will output a spc.json file to the src/spc_apps/appname folder. Then, edit the command line in the spc.json file to be something like (replacing app with the name of your app):

```
"command": "<rel_apps_path>/app/app",
```

After setting up the database entry, to finish setting up the application will require several more steps, which can be controlled from the App Edit page, which can be accessed by clicking the cog wheel on the App Edit page

2. **Upload input file** – To accomplish this step, click the "Configure inputs" button, and following through the instructions. One must upload an input file that is consistent with the input format specified in step 1. So, for example, if namelist input format is specified, the upload file must be in namelist format. Also, the name of the input file should be the name of the app with the following extension:
   - INI format - appname.ini
   - XML format - appname.xml
   - JSON format - appname.json
   - YAML format - appname.yaml
   - TOML format - appname.toml
   - namelist.input - appname.in (e.g. mendel.in)
3. **Setup HTML template file** – one of the things that the upload input file format does is to create an HTML template file, `appname.tpl`, in the `views/apps` folder.
4. **Upload binary file** – To upload a binary file, click the "Configure Executable" button. Please note that the binary file must be compiled on the same operating system as the host machine running SPC. So, for example, if SPC is running on an Amazon EC2 instance running 32-bit Linux with a certain glibc version, it should be compiled on an equivalent machine running the same glibc version.
5. **Setup plots** – To configure plots, click the "Configure Plots" button, then click "Add Plot". Each plot must have at least one datasource. So, once a plot has been setup, one must click the "datasource" link connected with that plot, and then click the "Add Data Source" button. While there are options for label, plot type, and color, these are all stored together in JSON format under one field in the database called data_def, short for data definition.

Additional options. **Static assets** – It is also possible to include static assets such as JavaScript files, CSS files, etc. which currently cannot be uploaded through the web interface, but can be added later manually by copying the files to static/apps/appname. They can be referenced later in the views/apps/appname.tpl, for example refer to the file mendel.js as follows:

```
<script src="/static/apps/mendel/mendel.js"></script>
```

2.2 B. Adding app through SPC's package management system

Because it may take some time to setup an app, especially the plots, it is possible to save the configuration as a JSON manifest file called spc.json. Therefore, it is possible to create an SPC package of an app, which is simply a zipped archive containing the input file, binary, spc.json manifest file, and then make it available on the Internet (e.g. by hosting it on github.org or bitbucket.org). In this way, anyone else running SPC can easily install your app by running the command:

```
> spc install http://url/to/package.zip
```

or by running the command on a file:

```
> spc install /path/to/package.zip
```

For Mac OS X users, an example can be installed by running the following command:

```
> spc install https://github.com/whbrewer/fmendel-spc-osx/archive/master.zip
```

2.3 Pre-/Post-processing

The code for pre- and post-processing is in the processing.py file. This feature can be turned on by setting the pre-processing option in the database setup to the input filename. However, coding to handle the pre-processing step must be manually added to the processing.py file.

Pre-processing

The pre-processor is run just before starting the executable in the function execute() in main.py. This feature is called as:

```
if myapps[app].preprocess:
   run_params,_,_ = myapps[app].read_params(user,cid)
   processed_inputs = process.preprocess(...)
```

This feature can be used for things that need to be changed just before running.

Examples:

1. Your program writes its output to a different file than appname.out, e.g. out100.00 where 100 is an entry input by the user in the input form. Therefore you can add a couple lines in execute() such as:

   ```
   if myapps[app].preprocess == "terra.in":
      myapps[app].outfn = "out"+run_params['casenum']+".00"
   ```

2. Your program doesn't actually use an input file, but rather uses command line switches to control certain behavior. Since SPC requires an input file to interact with the user interface, one can use the pre-processing option to convert a file that looks like:

   ```
   [BASIC]
   g_popsize = 100
   o_polytype = NBH
   ...
   [MUTATIONS]
   f_del_prop = 0.9
   h_dominance = 0.5
   ...
   ```

to a file containing set of switches that the program reads:

```
-x2 -s2 -n100 -v5 -r10 -k1 -i4 -j0.5 -f0.9 -g100 -oNBH -h0.5 -c5 -u5
```

This was accomplished by adding the following code to the preprocess() function in process.py:

```
if fn == 'fpg.in': ...
```

This is not good practice, to embed code in the spc source code. In the future, code hooks should be implemented to look for pre-process specific code in the installed app folder.

3. Convert input key/value params to cmd-line style args:

```
for key, value in (params.iteritems()):
   option = '-' + key.split('_')[0] # extract 1st letter
   buf += option + value + ' '
sim_dir = os.path.join(base_dir,fn)
return _write_file(buf,sim_dir)
```

The pre-processing option may also be used if one needs to write e.g. a PBS run script pbs.script file for running parallel applications via MPI.

Post-processing

The post-processor is called when the user clicks on any plot. The post-processor may be used to convert raw output data into JSON form that is needed for a programs like the Flot JavaScript plotting program to plot the files correctly. This function is defined in the file process.py. Here is the doc string for the postprocess() function:

```
"""return data as an array...
   turn data that looks like this:
   100   0.98299944  200    1.00444448   300    0.95629907
   into something that looks like this:
   [[100, 0.98299944], [200, 1.00444448], [300, 0.95629907], ... ]"""
```

CHAPTER 3

User Authentication

User authentication can be enabled or disabled by setting the auth value in *config.py* to either `True` or `False`. There are two default accounts that are setup when the running “spc init”. They are: admin (password admin) and guest (password guest). The admin user can install, configure apps, run cases, and also can manage other users accounts (presently this is just limited to deleting other accounts, but may in the future include some options such as setting the default priority level, or disk quotas, etc.).

CHAPTER 4

Web Server

A number of different web servers can be used withing SPC. By default, the CherryPy web server will be setup. However, there are development servers, such as `wsgiref`, and also a number of multi-threaded servers, such as: cherrypy, bjoern, tornado, rocket, gae, etc. The web server can easily be changed by changing the server variable in config.py, e.g.:

```
server = 'rocket'
```

Of course, this assumes that the web server has been installed on the local machine (e.g. `sudo pip install rocket`).

NGINX & uWSGI

To use SPC in production, it is recommended to use NGINX and uWSGI. A configuration file for both NGINX and uWSGI are available in the spc/etc folder.

The following steps may be used to setup NGINX:

```
> sudo yum install nginx
> sudo cp spc/etc/nginx/conf.d/spc.conf /etc/nginx/conf.d/
> sudo chkconfig nginx --levels 2345 on
```

The following steps may be used to setup uWSGI:

```
> sudo pip install uwsgi
```

Now, edit `spc/src/spc/config.py` to set `server = 'uwsgi'`

After making the changes, you will need to restart SPC if it is running (if you are using upstart: `sudo initctl start spc`), and also will need to start up the NGINX server by running:

```
> sudo service nginx start
```

NOTE: if you ever change the SPC port number, will need to change in two places: (1) in the spc/src/spc/config.py file, and also in the NGINX config file in /etc/nginx/conf.d/config.py

CHAPTER 5

Job Scheduler

SPC currently uses a multi-processing scheduler. The scheduler uses Python's multiprocessing module and includes synchronization primitives such as Lock (mutex) and a BoundedSemaphore for ensuring that only a user-specified number of jobs can run concurrently. To change scheduler options, modify config.py. It is important to set the number of jobs that you will allow to run concurrently by setting the np parameter in config.py, e.g.:

```
np = 2
```

np is short for number of processors, but really represents how many processors can be scheduled concurrently. For example, if the value is 2, two jobs that require a single processor can be scheduled, or a single job that requires 2 processors can be scheduled.

CHAPTER 6

config.py options

All the config.py options are listed here:

- **auth** - (required) `True` means require username password authorization. `False` means disable authentication
- **mpirun** - (optional) path to MPI executable, e.g. `/usr/local/bin/run`
- **np** - (required) The number of jobs to schedule simultaneously
- **port** - (required) The port for the web server to listen on, e.g. `port = 8580`
- **remote_worker_url** - (optional) If you want to run the simulation on another SPC worker node at a different URL, specify it here.
- **server** - (required) Can be either 'uwsgi', 'wsgiref', 'cherrypy', 'rocket', 'gae', or any other servers supported by Bottle
- **submit_type** - (optional) if this is set to either `verify` or `noverify`, when the user clicks the green "Continue" button, the job will start directly, without echoing back the run parameters. This can be used in cases where additional run-time options are not needed, such as specifying the number of processors. For instantaneous applications with few parameters, it is recommended to use this setting. For simulations that require a wall-time or use multiple processes, this is not recommended. If this setting is not set, it will default to `verify`.
- **tab_title** - (optional) This is the title to use in the browser tab
- **time_zone** - (optional) Used to show job submit date/time in local timezone. This is needed in cases where Linux system shows time in UTC format. One of the supported time zones, e.g. `time_zone = "US/Eastern"`
- **worker** - (optional) can be `local` or `remote`

CHAPTER 7

MPI-based applications

MPI may be used for running parallel applications. First, one must install the MPI software (e.g. www.mpich.org)

In config.py, set the mpirun parameter, e.g.:

```
mpirun = '/usr/local/bin/mpiexec'
```

In config.py, the np parameter has to be set to a value of greater than 1. After clicking Start, then Continue, select the number of processors to use for this run. For example, setting the value to 2, will execute the command:

```
mpiexec -n 2 app.exe
```

CHAPTER 8

Docker

Docker may be used in conjunction with SPC to run remote jobs. To use Docker with SPC:

1. Install Docker:

```
#on Fedora/Centos, etc.
> sudo yum update -y; sudo yum install docker

# on Ubuntu
> sudo apt-get update; sudo apt-get install docker-ce
```

2. Install docker-py:

```
> sudo pip install docker-py
```

3. Startup docker:

```
> sudo service docker start
```

4. Pull a Docker images that is running an SPC worker:

```
> docker pull whbrewer/spc
```

Then, you can bring up the `/docker` view in SPC by clicking on the hamburger icon on the top right, and click the Docker option. From there, you can create a container from the image by clicking the icon under the docker images actions. On creation you may specify host and container port numbers, but since the container version of SPC is listening on port 8581, best to just leave them as the defaults given. Once you create the container, you will need to start it, so click the icon. The container should start running, and should be able to access from the browser using http://url:8581 (e.g. http://localhost:8581) . The container version of SPC is running on port 8581, so that it doesn't conflict with SPC running on the host.

One can schedule jobs from the host to run on the container by adding the following two options in the config.py such as:

```
submit_type = 'remote'
remote_worker_url = 'localhost:8581'
```

For this case, you will need to setup the container to as a *worker* node, which is explained next.

One can login to the Docker container and modify SPC to add new apps. Once this is done, the container must be saved using “docker commit”. One may also build their own Docker container by using an older version or starting from a Ubuntu base image. In this case, you will need to make a Dockerfile with contents such as:

```
FROM bf6b7e57fba6
WORKDIR /root/spc
ENTRYPOINT ["python", "spc", "runworker"]
```

Here, the FROM statement is the image that you are basing SPC on, possibly a previous SPC container, or an Ubuntu base image. The third argument “runworker” can be just “run” if you want to run a regular SPC installation, or “runworker” if you want to just use it as a worker node, i.e. no user interface. In this case, you can build the new image using a command such as:

```
docker build -t spc .
```

CHAPTER 9

Miscellaneous

How to pass a hidden "case_id" value used in the interface into the file?

Create a variable called "case_id" in the input file (e.g. case.ini). When you are uploading and parsing the input file and are asked to assign html entity types for each parameter, set the type to "hidden".

CHAPTER 10

Known Issues

- HTML input elements, which are disabled will cause problems. In essence, they will be interpreted as checkboxes that are not checked, so the values will be set to F, which may cause problems when the simulation program tries to read the value and is expecting say an integer value. Therefore, in lieu of using `disabled=true` on input tags, set `readOnly=true`.
- The restart option does not work for apps that use XML input files. This is a problem with the structure that is being output from SPC and needs to be fixed in the future.