Lewis & Clark BLT Documentation

Ben Glick, Jeremy McWilliams, others!

2018-02-19

Contents

Introduction

Lewis & Clark was able to acquire a high performance computing cluster during the summer of 2017.

Chapter 1

About the Cluster

1.1 Cluster Facts

The cluster has one multipurpose login node and three identical worker nodes. It has a total of 144 processing cores and 1500 gigabytes of memory. The login node also functions as the parallel filesystem for all of the workers, with 140TB of RAID-redundant disk space. This means that workers can access data stored in your home directory, which makes input and output staging extremely easy.

The cluster runs the CentOS Linux operating system, version 7.4. You can interact with it as you would with any command line based linux distribution. A full list of installed software packages and modules will be posted on this wiki.

1.2 Interacting with the Login Node

The only machine you should ever need to interact with is the login node, mayo. If you need help getting access to the machine, please see the Getting Connected section.

When you log in to mayo, you will receive a bunch of information, including a system summary and a message from the admins, which is copied below.

```
*   This machine is for transfering files on and off the BLT computational
*   infrastructure. Please DO NOT run any jobs on this machine. Please login
*   to the server "mayo.blt.lclark.edu" to use the cloud and run
*   jobs from that machine.
*
*   If we find jobs running on this machine the BLT Admins will kill them to ensure
*   other users will not be effected.
*
*   Users are given 25G of space for free and users can check their usage
*   using the command "quota -s". If you need more space for your project
*   please contact BLT Admins or IT support for details.
```

As this message suggests, please do not run any compute-intensive jobs from the login node. It is intended to be a place to set up workflows to be run, store data files, stage data into and out of the workers, and exist as a human interaction layer so that users don't need to deal with things like scheduler abstraction.

When you log in, you will find a number of hidden files in your home directory. (If you're curious, they can be listed with the command ls -a and will be the files that start with a dot (".")) Please do not remove these files as they store important information that the cluster needs access to. You will also find an empty

directory called perl5. If you have specific perl libraries you need, you can install them there. If you are not using perl, you can safely ignore or delete this directory.

Feel free to keep whatever data you need in your home directory. It is important to keep your directory organized in a reasonable way in order to ensure that you (and the worker nodes) will always be able to find the needed data efficiently. Also, feel free to look at what programs are installed in /local/cluster/bin, as these programs will always be able to run from any of the workers.

Once you feel comfortable interacting with the login node and are ready to start to submit compute jobs, please continue to the Submitting Jobs page.

Chapter 2

Getting Connected

2.1 Accounts

In order to gain access to the cluster, you first need an account. Contact the BLT Admins to request an account.

NOTE: Once you receive a temporary password, please reset it within 5 days of gaining access to the system.

2.2 Getting on the network

The BLT cluster is quite isolated from LC's public-facing infrastructure. In order to connect to it, you will need a copy of Cisco AnyConnect secure mobility client, which is available to LC students, faculty, and staff HERE.

If you are using Linux to connect to the cluster, the current version of Cisco AnyConnect will fail to install. Luckily, there is an open-source equivalent called OpenConnect, which installs as a menu option for debian and redhat based OSes. You will need to open your network settings and click the green plus button to add a new connection, and then select VPN when prompted. After that, put in `vpn.lclark.edu` for the gateway option and the same root CA certificate as you used when setting up LC secure. After you click save, it will ask for your LC id and password.

After you have installed and started AnyConnect:

1. Start a VPN session by typing `vpn.lclark.edu` in the text box and clicking "connect"
2. When prompted, put in your LC username and password for access Now, your computer is connected to the same virtual network as the cluster.

2.3 Logging In

NOTE: In order to log in, you will need an SSH client. If you are using a Mac or Linux machine, you already have one. If you are using Windows, you will need to install PuTTY or similar.

1. Open your SSH client

- On Mac Press the space bar and command key at the same time. then type "Terminal" and hit return
- On Linux Open a terminal window
- On Windows Open PuTTY

2. Log In!

- On Mac or Linux type `ssh <lclark username>@mayo.blt.lclark.edu` and type your password when prompted
- On Windows Open PuTTY, set "Host Name" to mayo.blt.lclark.edu and click "Open", and follow the prompt. Congratulations! You have logged in to the BLT cluster! See Using the Cluster for more information about what you can do.

Chapter 3

Submitting Jobs

3.1 Checking Usage

At any time, a user can check what the current availability of the cluster is by typing `SGE_Avail` on their command line. The output will look something like this:

#HOST	TOTRAM	FREERAM	TOTSLOTS	Q	QSLOTS	QFREESLOTS	QSTATUS	QTYPE
bacon	503.6	500.3	48	all.q	48	48	normal	BP
lettuce	503.6	500.2	48	all.q	48	48	normal	BP
tomato	503.6	500.2	48	all.q	48	48	normal	BP

Right now, according to this output, there are 3 hosts running: bacon, lettuce, and tomato. They each have 48 total slots and 48 free slots. They each have 500 GB of free RAM as well.

Additionally, users can check what the job queue looks like. Users can see what jobs are waiting to be run and what jobs are currently running. To do this, run the qstat command. If qstat comes back with no output, it means there are no jobs running at the moment. Here is some example output from the qstat command:

job-ID	prior	name	user	state	submit/start at	queue	slots
62	0.00000	runtime_test	glick	r	01/26/2018 18:59:00		
63	0.00000	runitme_test2	glick	qw	01/26/2018 18:59:02		
64	0.00000	runtime_test3	glick	qw	01/26/2018 18:59:04		

There are currently 3 jobs on the cluster, all submitted by the user "glick". They are jobs with ids 62,63, and 64. They each take up one slot (another name for a core). One is running, while the other two have state `qw`, which is short for "Queued and Waiting". This is usually an indication that either the cluster is busy or the scheduler has not yet scheduled the jobs.

3.2 A Note on Data

The home directories, /local/cluster/bin, and a few other things are mounted remotely to all of the worker nodes. This makes life easy. It means that if your script edits, reads, or otherwise depends on data from your home directory, you do not need to move the data, because the workers can access it directly. However, this also means that if your data edited by multiple jobs, there is no way to ensure that it will always be changed in the same order, so keep that in mind.

3.3 Jobs on the BLT Cluster

3.3.1 Batch Jobs

A Batch job is some set of UNIX command line commands which is executed on a single core of a worker node in serial (one after another). Batch jobs can be submitted by using the following command:

```
SGE_Batch -r "<some runtime id>" -c "<a UNIX command or commands>"
```

3.3.2 Parallel Jobs

Parallel jobs are just like Batch jobs, except that in a parallel job, multiple cores are reserved, rather than a single core. In order to reserve multiple cores, simply add the `-P` flag to the SGE_Batch command like so:

```
SGE_Batch -r "<runtime id>" -c "UNIX command" -P <number of processors>
```

Remember that SGE_Batch will not parallelize your code for you. If your code is not meant to run on multiple cores, then using any more than 1 processor core is a waste.

3.3.3 Parsl Workflows

Parsl is a python-based workflow management system that we can use to run jobs on the cluster without having to interact with the scheduler at all. They are run the same way that you would run any script on your local machine, and can orchestrate inter-process communication between almost any kind of application needed. In depth documentation about running parsl jobs is available at the Parsl Workflows page.

Chapter 4

Parsl Workflows

4.1 About Parsl

Parsl is a Python-based parallel scripting library that supports development and execution of asynchronous and parallel data-oriented workflows (dataflows). These workflows glue together existing executables (called Apps) and Python functions with control logic written in Python. Parsl brings implicit parallel execution to standard Python scripts. Parsl documentation is available (here)[http://parsl.readthedocs.io/en/latest/].

4.2 Using Parsl

Using parsl is a lot like using plain python. You define scripts that can be run the same way as you would normally run a python script: `python3 parsl_script.py`. The main difference with parsl scripts is that they don't execute on the machine they are run from. The parsl script will provision itself resources on the worker nodes of the cluster and manage where to run jobs without any human intervention. To accomplish this, instead of returning results directly, parsl returns futures. A future is a promise that a function will return a value, usually of a specific type. The future allows parsl to have an understanding of what will be returned from all of the apps it needs to run. This helps parsl to calculate what apps can be run at the same time.

4.3 Parsl Configuration

Parsl requires a configuration in order to understand what kind of an environment in which it is running. On this system, the configuration should be very similar to the example below:

```
config = {
    "sites": [{
        "site": "Local_IPP",
        "auth": {
            "channel": "local"
        },
        "execution": {
            "executor": "ipp",
            "provider": "sge",
            "script_dir": ".scripts",
            "scriptDir": ".scripts",
```

```
            "block": {
                "nodes": 1,
                "taskBlocks": 1,
                "walltime": "00:05:00",
                "initBlocks": 1,
                "minBlocks": YOUR_MIN_CORES_HERE,
                "maxBlocks": YOUR_MAX_CORES_HERE,
                "scriptDir": ".",
                "options": {
                    "partition": "debug"
                }
            }
        }
    }],
    "globals": {"lazyErrors": True},
    "controller": {"profile": "default"},
}
```

The parsl configuration is a python object, so the example can be pasted into your python script at the top, along with the following line of code:

```
dfk = DataFlowKernel(config=config)
```

4.4 Defining a Parsl Workflow

After that, you are ready to start defining your workflow. Here is an example workflow which calculates pi by generating random numbers:

```
@App('python', dfk)
def pi(total):
    # App functions have to import modules they will use.
    import random
    # Set the size of the box (edge length) in which we drop random points
    edge_length = 10000
    center = edge_length / 2
    c2 = center ** 2
    count = 0

    for i in range(total):
        # Drop a random point in the box.
        x, y = random.randint(1, edge_length), random.randint(1, edge_length)
        # Count points within the circle
        if (x - center)**2 + (y - center)**2 < c2:
            count += 1

    return (count * 4 / total)


@App('python', dfk)
def avg_n(inputs=[]):
    return sum(inputs) / len(inputs)

if __name__ == '__main__':
    # Call the workflow:
```

```
sims = [pi(10**6) for i in range(10)]
avg_pi = avg_n([task.result() for task in sims])

# Print the results
print("Average: {0:.12f}".format(avg_pi.result()))
```

You will notice a couple of things that are different from regular python. First, above each app is a decorator (`@App('python', dfk)`). This tells parsl that the app should be run on a worker machine, and not as a normal python function.

Second, the apps which take inputs have a special reserved parameter (`inputs`), which you can pass parsl futures into. Passing a future into an app will make the app wait for the result of the future before it runs the app.

Another important thing to remember is that all packages you use (anything you would use the `import` statement for) must be imported from within the app, otherwise it will not be accessible from the worker. You can see this in the example here with the `random` library.

4.5 Running a Parsl Workflow

After you have the proper configuration and workflow definition, you are ready to run your workflow! Make sure the workflow is in a python file on the login node (mayo) and run it like this: `python3 /path/to/your/workflow/file`

4.6 Video Tutorial

coming soon….

Appendix A

Job History

Date	Name	Department	Description	Research
2/9/18	Ben Kolligs	Physics	Python script running Monte Carlo simulations of a lattice model to calculate Renyi Entropy with the hope of better understanding the strong nuclear force.	Funded by the NSF and Murdock Charitable Trust, we are investigating the behavior of string breaking in the strong nuclear force when the temperature is increased.

Appendix B

Admin Reference