Lewis & Clark BLT Documentation

Ben Glick, Jeremy McWilliams, others!

2018-09-05

Contents

Introduction

Lewis & Clark was able to acquire a high performance computing cluster during the summer of 2017, as a result of Greta Binford's capital equipment request. The cluster is a smaller version of the cluster used at Oregon State University's Center for genome Research and Biocomputing.

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

2.2.1 For Windows and OSX Users

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.

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.

2.2.2 For Linux Users

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.

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.

2.4 Adding Files to Your Home Directory

If you want to add files to your home directory, you can use essentially any command line remote file transfer system you can think of, including rsync, scp, sftp, etc. We also have a graphical user interface on the web at (http://mayo.blt.lclark.edu/owncloud). You can log in to this service using your inital UNIX user account, but if you change your unix password, it must be manually changed separately in OwnCloud. Files that you drop to the owncloud folder “files” will go to the “files” directory in your home directory.

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….

Chapter 5

Setting up Groups

5.1 Senario for Groups

This section is for describing how to set up a group of users in which shared files might be needed for computation. Use cases include classroom/group assignments or research labs.

5.2 Procedures

Connect to mayo, and navigate to /users/lab

```
cd /users/lab
```

Create a directory in `lab`.

Appendix A

Job History

Date: 2/9/18
Name: Ben Kolligs
Department: Physics
Faculty Member: Anber
Description: Python script running Monte Carlo simulations of a lattice model to calculate Renyi Entropy
Research: Funded by the NSF and Murdock Charitable Trust, we are investigating the behavior of string bi

Date 2/20/18
Name: Sophia Horigan & Australia 2018 students
Department: Biology
Faculty Member: Binford
Description: Introduction to bioinformatics using TransDecoder, an program in the RNA-Seq pipeline that

Appendix B

Admin Reference

B.0.1 Software Installations and Changes

This page is intended to document all of the software installation that has been done so that we can re-do it if ever necessary. When something is installed on the cluster, you should document what it is, where it is installed, and how you installed it on this page. Note that small things like python packages do not need to be documented here.

B.0.2 General Notes

When you install something, please download the compressed (`.tar.gz`/`.zip`/`.bz2</code>`…) file to `/local/downloads/`. Please leave uncompressed source code there too. Do not remove either of those after installing the software.

B.0.3 SQLite3

We have (and must have) a slightly non-standard installation of SQLite. This just has to do with where everything is installed on our cluster's parallel filesystem. Because of this, we need to install sqlite3 from source instead of using `yum install sqlite3`. To do this, I essentially followed the tutorial here. All of the dependencies needed to install sqlite3 should be installed already. The exact commands used (which can be rerun verbatim if needed) are below. Make sure you install python *after* this, because it uses header files that are only included with the distribution of SQLite3.

```
cd /local/downloads/
wget https://www.sqlite.org/2018/sqlite-autoconf-3220000.tar.gz
tar xf ./sqlite-autoconf-3220000.tar.gz
cd ./sqlite-autoconf-3120200.tar.gz
./configure --prefix=/local/cluster --disable-static --enable-fts5 --enable-json1 CFLAGS="-g -O2 -DSQLI
make
make install
```

B.0.4 A Note On Python

Installing, configuring and managing python on a cluster like ours is frankly a mess. We have at least 5 different python interpreters installed, all of which have their own packages and package managers. If you change the configuration of any of them, please be very careful. Make sure you're using the right 'pip' tool

by either calling the fully-qualified path (e.g. '`/local/cluster/bin/pip`') or by saying `which pip` and ensuring that the selected one is in `/local/cluster/bin/`. Please do not install packages or make changes to the python in `/usr/bin/python`. This python is important for the `yum` package manager.

B.0.5 Python36

Because of the way they are bundled, Python3 must be installed *after* sqlite3 is installed. It is installed from source using the following commands, which were adapted from the same tutorial as we used to get SQLite3 (available here).

The exact commands I ran are below:

```
cd /local/downloads/
wget https://www.python.org/ftp/python/3.6.4/Python-3.6.4.tgz
tar xf ./Python-3.5.1.tgz
cd ./Python-3.5.1

LD_RUN_PATH=/local/cluster/lib/ LD_LIBRARY_PATH=/local/cluster/lib/ ./configure --prefix=/local/cluster

LD_RUN_PATH=/local/cluster/lib/ LD_LIBRARY_PATH=/local/cluster/lib/ make
LD_RUN_PATH=/local/cluster/lib/ LD_LIBRARY_PATH=/local/cluster/lib/ make test
LD_RUN_PATH=/local/cluster/lib/ LD_LIBRARY_PATH=/local/cluster/lib/ make install
```

Python 2 can be installed essentially identically, except with different version numbers on the things you download.

B.0.6 JupyterHub Notebook Server

Jupyterhub is the program we use to serve the multi-user notebook server located at mayo.blt.lclark.edu:8000. Installation of jupyterhub is easy. You can install it with 'sudo /local/cluster/bin/pip3 install jupyterhub'. To run jupyterhub, start a screen or tmux session, run the command `PATH=/local/cluster/bin/:$PATH; cd /local/cluster/jupyterhub-runtime/jupyterhub_run/ && jupyterhub -f jupyterhub_config.py`. Then detach the screen. Configuration is in the file `/local/cluster/jupyterhub-runtime/jupyterhub_config.py`

I needed to make an IPTables entry allowing traffic in and out on port 8000. If you need to change the port, remember to make a new iptables entry.

Jupyterhub example user password (for use case demonstrations): user:example, pass:processtruster76

B.0.7 Apache Httpd Web Server

We installed apache2 with the standard `sudo yum install httpd`. Apache2 is currently serving from its default server root at `/var/www/html`. PHP v5.4.16 was also installed, and has its configuration at `/etc/php.ini`

I needed to make an IPTables entry allowing traffic in and out on port 80. If you need to change the port, remember to make a new iptables entry.

B.0.8 OwnCloud

OwnCloud is essentially a google drive clone that will run on our cluster. We automatically create user owncloud accounts upon user creation and we also symlink their file dropbox to their home directory. For help with installation, follow this tutorial here.

B.0.9 Dropbox CLI

Follow page here. Our install location is somewhere else, in `/local/cluster/dropbox_dist`.

B.0.10 ProtTest

ProtTest is a bioinformatic tool for the selection of best-fit models of aminoacid replacement for the data at hand. ProtTest makes this selection by finding the model in the candidate list with the smallest Akaike Information Criterion (AIC), Bayesian Information Criterion (BIC) score or Decision Theory Criterion (DT). At the same time, ProtTest obtains model-averaged estimates of different parameters (including a model-averaged phylogenetic tree) and calculates their importance(Posada and Buckley 2004). ProtTest differs from its nucleotide analog jModeltest (Posada 2008) in that it does not include likelihood ratio tests, as not all models included in ProtTest are nested. It is written in java with MPJ, and it has a fairly strange installation. It's installed to `/local/cluster/prottest3`, and needs to be run from there. People should not put data files in that directory. It can be run something like this: `cd $PROTTEST_HOME java -jar prottest-3.4.2.jar -i examples/COX2_PF0016/alignment -all-matrices -all-distributions -threads 2`

B.0.11 MrBayes

MrBayesis a program used for Bayesian inference of phylogeny. It's installed normally. Instructions for building from source are here.

B.0.12 PAML

PAML needs to be installed on a per-user basis. It needs to be copied into the user's home directory and it's internal `bin` directory needs to be added to the user's PATH. It's compiled executables are in `/local/downloads/paml4.9g`.

B.0.13 MODELLER

MODELLER is A Program for Protein Structure Modeling. It's a special python interpreter which is installed normally in /local/cluster/bin/

B.0.14 RAxML

RAxML is a maximum likelihood phylogenetic bioinformatic tool. It's installed normally. However, there are two versions installed. There is an MPI version (suitable for tasks of size 48 slots or more) as well as a shared memory pool version (suitable for 48 slots or fewer). If you are using exactly 48 slots, flip a coin. The MPI version can be run with `raxmlHPC-MPI-AVX2` and the SMP version can be run with `raxmlHPC-PTHREADS-AVX2`

B.0.15 Intel OpenCL Runtime

The Intel OpenCL Runtime is required to run OpenCL programs on Intel CPUs. The latest packages are available here under OpenCL Runtime for Intel Core and Intel Xeon Processors. To install, run the install script and select `/local/cluster` as the target prefix. The files will be installed in `/local/cluster/opt/intel`. Be sure to symlink the libraries in `/local/cluster/opt/intel/opencl-${version_number}/lib64` to `/local/cluster/lib`:

```
find "$(realpath /local/cluster/opt/intel/opencl)/lib64/" -type f -name '*.so*' -print0 \
  | xargs -0 ln -s -t /local/cluster/lib
```

We are explicitly linking to the libraries in the version specific directory since the generic symlink requires `/etc/alternatives/opencl*` which won't be recreated on worker nodes.

Setting up OpenCL to run on worker nodes will require some additional work. The OpenCL runtime searches for platform specific ICDs in `/etc/OpenCL/vendors` by default. While the documentation states that this search path can be modified with an environment variable, `OPENCL_VENDOR_PATH`, it seems that not all ICD loader implementations support this. As a temporary workaround, we will recreate this directory structure in `/local/cluster/etc/opencl_icd_fix`:

```
fixdir='/local/cluster/etc/opencl_icd_fix'
for d in "$fixdir" "${fixdir}/vendors"; do
    if ! [ -d "$d" ]; then
        mkdir "$d"
    fi
done
find "$(realpath /local/cluster/opt/intel/opencl)/etc/" -type f -name '*.icd' -print0 \
  | xargs -0 ln -s -t "${fixdir}/vendors"
```

If `OPENCL_VENDOR_PATH` worked with the implementation shipped with Intel's runtime, we could just set it to our fix directory. Unfortunately, it does not. We need to create symlinks in each worker node:

```
ln -s /local/cluster/etc/opencl_icd_fix /etc/OpenCL
```

B.0.16 Hashcat

Hashcat is a suite of password cracking tools. It requires OpenCL supported devices along with the appropriate OpenCL runtimes and drivers. The latest release is available here. To install hashcat, first download the source tarball and untar it. Since hashcat lacks a configure script, we'll need to edit the Makefile manually. Find the definition of the `PREFIX` in the Makefile and define it as `/local/cluster`. Then run `make` and `make install`.

B.0.17 Set up procedure for classes/projects

Classes and projects often require shared materials (files/folders/etc.). We are able to do this with UNIX groups. To make a UNIX group for a class, first ensure that all students who need to be in the group have user accounts on the system.

Then, create the group: `groupadd <GROUP NAME>`

Create a shared folder in `/home/data`, and set its' owner and permissions accordingly: `mkdir /home/data/<GROUP NAME> chown <OWNER>:<GROUP NAME> /home/data/<GROUP NAME> chmod 755 /home/data/<GROUP NAME>` where `<OWNER>` is the username of the group owner (usually professor)

Now, add all the students to the group as follows (you can do this individually as follows or automate it as after that):

```
usermod -a -G <GROUP NAME> <STUDENT USERNAME>
```

To do this automatically, do the following:

First, create a file which contains a student's username on each line, like so:

```
jimmy
timmy
tommy
```

```
terry
jerry
```

Save this file as `users.txt`. Then, run the following script, changing the group name as necessary:

```
gn=<GROUP NAME>
while read p; do
  usermod -a -G $gn $p
done <users.txt
```