ClusterJob

An automated system for high-throughput reproducible computational research

Hatef Monajemi

Version 0.1, 2016-10-05.

Table of Contents

Getting Started

ClusterJob (CJ) is an automation system for high-throughput reproducible computations. Using CJ, one can conduct large-scale computational studies on remote clusters in a hassle-free and reproducible manner. CJ builds 'reproducible' computational packages that are easy to share with others. It is written mainly in perl and has simple commands that are easy to learn. The project started at Stanford University by Hatef Monajemi and his PhD advisor David L. Donoho with the goal of making large-scale scientific computing simpler and more efficient. Current implementation allows submission of MATLAB jobs. In future, we hope to include other heavily used programming languages such as R and Python.

CJ on Github

You may access the CJ Github repository at https://github.com/monajemi/clusterjob
This book is a work in progress. You may find the AsciiDoc source code for this book at https://github.com/monajemi/CJ-book.
A PDF copy of this documentation can be found at http://clusterjob.org/documentation/book.pdf.

Setting up CJ

Installing a CJ agent is very simple. Follow the two steps below to set one up!

Step 1: Setting up authentication key

CJ assumes that the secure shell (ssh) to cluster is handled in a password-free manner. There are various ways to achieve this. Some clusters use Kerberos and some others might use ssh-keygen. We will explain how you could setup a key using ssh-keygen in this document. ssh-keygen is a Unix utility that is used to generate, manage, and convert authentication keys for ssh authentication. If you already have a password-free connection to your cluster, you may skip this step.

1. Check SSH setup

   You can check to see if you already set up the public ssh key. Open a terminal and enter:

   ```
   $ ls -al ~/.ssh
   ```

   if you see any of the following, you probably have already setup the ssh-keygen

   - id_rsa.pub
   - id_dsa.pub
   - id_ecdsa.pub
   - id_ed25519.pub

2. Generate a SSH key on your machine

   If you have not setup ssh key, do not worry. It is very simple to set up. Open terminal, and enter,

```
$ ssh-keygen -t rsa -C "your_email@example.com"
# This will generate a SSH key. Just use the default setting if
# you are asked questions in the process of key generation
```

This will generate a key in '~/.ssh/id_rsa.pub'

3. Copy the key to remote server

   The last step is to copy the content of your `~/.ssh/id_rsa.pub` to `~/.ssh/authorized_keys` located on the remote server

4. Check connection

   You should check your authentication keys by trying to connect to server. In your terminal enter,

```
$ ssh username@cluster.stanford.edu
```

Step 2: Installing CJ

1. Clone ClusterJob from GitHub

   Clone ClusterJob to a directory where you would like to install it, say `~/CJ_install`

```
cd ~/
rm -rf monajemi-clusterjob-* CJ_install
curl -sL  https://github.com/monajemi/clusterjob/tarball/master | tar -zx -  && mv
monajemi-clusterjob-* CJ_install
```

   You may also use the following alternative command to clone CJ with the entire history

```
git clone https://github.com/monajemi/clusterjob.git ~/CJ_install
```

2. Install perl dependencies

   ClusterJob is written in perl and so depends on other perl modules. You can install CJ dependencies via `cpan` or `cpanm`. Copy and paste the following lines into your terminal:

```
sudo cpan -i Data::Dumper Data::UUID FindBin File::chdir File::Basename File::Spec
IO::Socket::INET Getopt::Declare  Term::ReadLine JSON::PP JSON::XS Digest::SHA
Time::Local Time::Piece Moo HTTP::Thin HTTP::Request::Common JSON URI
```

3. Provide configuration info

   You will need to update the contents of `~/CJ_install/cj_config` file to reflect your own

information.

This file contains the following information:

```
CJID             <YOUR CJ ID>
CJKEY            <YOUR CJ KEY>
SYNC_TYPE       auto
SYNC_INTERVAL   300
```

1. To use CJ remotely, you need to provide your unique CJID and CJKEY obtained from http://clusterjob.org.
2. If you plan to use CJ locally without syncing to CJ remote database, you must provide an arbitrary CJID.
3. If you do not include a CJKEY, your meta data will not sync to CJ remote database even if you have registered an acount on http://clusterjob.org.

You must keep your CJKEY private.

4. Provide cluster info

   You will need to update the contents of `/CJ_install/ssh_config` file to reflect your own server setup.

   The CJ convention for each remote machine is

   ```
   [MACHINE_ALIAS]
    Host    your_host
    User    your_username
    Bqs     batch_queuing_system
    Repo    CJ_remote_repo
    Matlib  Matlab libraries
   [MACHINE_ALIAS]
   ```

   

   You can add as many machines as you want to `ssh_config`.

5. Build an alias for CJ

   For easy use, you may want to add an alias for calling `src/CJ.pl` to your `~/.profile` or `~/.bashrc`:

   ```
   alias cj='perl ~/CJ_install/src/CJ.pl';
   ```

6. Initialize your CJ agent

   You may now initialize your CJ agent by

```
$ cj init
```

You may check if the agent is installed by

```
$ cj who
```

This should print out something like

agent: E9078FA4-8423-11E6-B9A8-DFE0D454C74A

You may install as many CJ agents as needed on a single machine or different machines.

CJ Basics

After you have successfully started a CJ agent, it is time to have some fun! In what follows, we will demonstrate how you can use CJ to make a reproducible computational package from your MATLAB code and run it on a remote cluster in a hassle-free way.

A simple Matlab example

We will now go over a simple example of running a Matlab code on a cluster to demonstrate the usage of ClusterJob.

Running Jobs

Suppose we would like to run the following simple matlab code. The code includes two `for` loops indexed by `i` and `j`, and for each such combination it write a line to the file `results.txt`.

simpleExample.m

```
% This is a test Matlab script for CJ
% Author: Hatef Monajemi June 28 2016

file = 'results.txt';

for i = 1:3
    for j = 1:5
                % write to a text file for testing reduce
                fid = fopen(file,'at');
                fprintf(fid, '%i,%i,%i\n', i,j,i+j);
                fclose(fid)
 end
end
```

This code runs on our personal machine without any error. We now wish to run this code on a cluster named **sherlock** using ClusterJob. To run this code serially, once we are in the folder containing `simpleExample.m`, we simply type the following command in a terminal.

```
$ cj run simpleExample.m sherlock -m "This is a simple test for run command"
```

This command should result in the following output:

```
CJmessage::runing [simpleExample.m] on [sherlock]
CJmessage::Sending from: /Users/hatef/github_projects/clusterjob/example/MATLAB
CJmessage::Creating reproducible script(s) reproduce_simpleExample.m
CJmessage::Sending package 07264a5d
CJmessage::Submitting job
CJmessage::1 job(s) submitted (10097640)
```

The `run` command uses only one processing core. We could however run our code in parallel by simply changing the command to `parrun`:

```
$ cj parrun simpleExample.m sherlock -m "This is a simple test for parrun command"
```

This time, you should see the following output informing you of submitting 15 jobs instead.

```
CJmessage::parruning [simpleExample.m] on [sherlock]
CJmessage::Sending from: /Users/hatef/github_projects/clusterjob/example/MATLAB
CJmessage::Creating reproducible script(s) reproduce_simpleExample.m
CJmessage::Sending package 0ed00c68
CJmessage::Submitting job(s)
CJmessage::15 job(s) submitted (10097772-10097786)
```

The parallelization of your code happens automatically with no further effort from you.

As you can see, each instance of the `run` command produces a reproducible package with a distinct PID (for Package IDentifier). A PID is a SHA1 code, which is essentially a 40 digits long hexadecimal number. Though PIDs are 40 digits long, when using CJ commands you can provide only a short version of it, which contains the initial 8 characters only.

A PID is valid as long as it is a hexadecimal number of length 8 or more

The full PID can be retrieved using `$ cj info short_pid` command.

Checking Status

You can check the status of your jobs using `state` command.

```
$ cj state 07264a5d

pid 07264a5d33bab71c1463f651b1ff920f6d32bb1c
remote_account: monajemi@sherlock.stanford.edu
job_id: 10109624
state: COMPLETED
```

Or for a parallel case,

```
$ cj state 0ed00c68

pid 0ed00c6851c504af7d8064a954aba44cf1da40f2
remote_account: monajemi@sherlock.stanford.edu
1       10097772        COMPLETED
2       10097773        COMPLETED
3       10097774        COMPLETED
4       10097775        COMPLETED
5       10097776        COMPLETED
6       10097777        COMPLETED
7       10097778        COMPLETED
8       10097779        COMPLETED
9       10097780        COMPLETED
10      10097781        COMPLETED
11      10097782        COMPLETED
12      10097783        COMPLETED
13      10097784        COMPLETED
14      10097785        COMPLETED
15      10097786        COMPLETED
```

Getting Results

You can pull your computational package entirely or partially using `get` command. this command will pull the package associated with a particular PID into a temporary directory on your local machine:

```
$ cj get 07264a5d

    CJmessage::Getting results from 'sherlock'
    CJmessage::Please see your last results in
    /Users/hatef/CJ_get_tmp/07264a5d33bab71c1463f651b1ff920f6d32bb1c
```

If we now look at the contents of this package, we would see the following. You may want to take a moment to see what is inside each file.

```
$ ls -lt /Users/hatef/CJ_get_tmp/07264a5d33bab71c1463f651b1ff920f6d32bb1c
total 96
-rwxr-xr-x  1 hatef  staff  2297 Oct  6 16:19 CJ_CONFIRMATION.TXT
-rw-r--r--  1 hatef  staff  2865 Oct  6 16:18 CJrandState.mat
-rw-r--r--  1 hatef  staff    90 Oct  6 16:18 results.txt
drwxr-xr-x  5 hatef  staff   170 Oct  6 16:18 logs
drwxr-xr-x  3 hatef  staff   102 Oct  6 16:18 scripts
-rw-r--r--  1 hatef  staff    29 Oct  6 16:17 qsub.info
-rw-r--r--  1 hatef  staff  1606 Oct  6 16:17 bashMain.sh
-rw-r--r--  1 hatef  staff   901 Oct  6 16:17 master.sh
-rw-r--r--  1 hatef  staff   623 Oct  6 16:17 reproduce_simpleExample.m
-rw-r--r--  1 hatef  staff   308 Oct  6 16:17 simpleExample.m
```

`CJ_CONFIRMATION.TXT`
: contains confirmation that this package was run by ClusterJob.

`CJrandState.mat`
: contains the information about the random seed that generated the results so that other people can reproduce your results.

`results.txt`
: is the output of your code

`reproduce_simpleExample.m`
: reproduces these results upon execution.

For a parallel package, if we use `get`, we will see the following output:

```
$ cj get 0ed00c6851c504
     CJmessage::Getting results from 'sherlock'
     CJmessage::Run REDUCE before GET for reducing parrun packages
     CJmessage::Please see your last results in
     /Users/hatef/CJ_get_tmp/0ed00c6851c504af7d8064a954aba44cf1da40f2
```

For parallel packages, one typically needs to use `reduce` before `get`

Clearly, we are asked to use `reduce` before `get` for a parallel package. We will discuss `reduce` command in the next section. If we now look at the folder containing this parallel package, we see:

```
$ ls -lt /Users/hatef/CJ_get_tmp/0ed00c6851c504af7d8064a954aba44cf1da40f2
total 80
-rwxr-xr-x  1 hatef  staff   2429 Oct  7 16:06 CJ_CONFIRMATION.TXT
drwxr-xr-x  9 hatef  staff    306 Oct  6 22:28 13
drwxr-xr-x  9 hatef  staff    306 Oct  6 22:26 15
drwxr-xr-x  9 hatef  staff    306 Oct  6 22:26 14
drwxr-xr-x  9 hatef  staff    306 Oct  6 22:25 11
drwxr-xr-x  9 hatef  staff    306 Oct  6 22:25 10
drwxr-xr-x  9 hatef  staff    306 Oct  6 22:25 6
drwxr-xr-x  9 hatef  staff    306 Oct  6 22:25 7
drwxr-xr-x  9 hatef  staff    306 Oct  6 22:25 8
drwxr-xr-x  9 hatef  staff    306 Oct  6 22:25 3
drwxr-xr-x  9 hatef  staff    306 Oct  6 22:25 5
drwxr-xr-x  9 hatef  staff    306 Oct  6 22:25 9
drwxr-xr-x  9 hatef  staff    306 Oct  6 22:25 4
drwxr-xr-x  9 hatef  staff    306 Oct  6 22:25 12
drwxr-xr-x  9 hatef  staff    306 Oct  6 22:25 2
drwxr-xr-x  9 hatef  staff    306 Oct  6 22:24 1
-rw-r--r--  1 hatef  staff    435 Oct  6 22:23 qsub.info
-rw-r--r--  1 hatef  staff  11591 Oct  6 22:23 master.sh
-rw-r--r--  1 hatef  staff    308 Oct  6 22:23 simpleExample.m
```

We now see 15 directories for 15 jobs submitted. If we further look into one of these directories:

```
$ ls -lt /Users/hatef/CJ_get_tmp/0ed00c6851c504af7d8064a954aba44cf1da40f2/1
total 40
-rw-r--r--  1 hatef  staff  2865 Oct  6 22:24 CJrandState.mat
-rw-r--r--  1 hatef  staff     6 Oct  6 22:24 results.txt
drwxr-xr-x  5 hatef  staff   170 Oct  6 22:23 logs
drwxr-xr-x  3 hatef  staff   102 Oct  6 22:23 scripts
-rw-r--r--  1 hatef  staff  1953 Oct  6 22:23 bashMain.sh
-rw-r--r--  1 hatef  staff   515 Oct  6 22:23 reproduce_simpleExample.m
-rw-r--r--  1 hatef  staff   196 Oct  6 22:23 simpleExample.m
```

Reducing Results

The `reduce` command is designed for reducing the results of a parallel run into a single file. This is because the `parrun` command executes the `for` loops in the main script in parallel by generating independent sub-directories and submitting a separate job for each index combination.

If we look at the `results.txt` file in sub-directories 1 and 2 for instance, we see:

```
$ cat /Users/hatef/CJ_get_tmp/0ed00c6851c504af7d8064a954aba44cf1da40f2/1/results.txt
1,1,2
$ cat /Users/hatef/CJ_get_tmp/0ed00c6851c504af7d8064a954aba44cf1da40f2/2/results.txt
1,2,3
```

We certainly want these results to be reduced to one single `results.txt` file. This is done via the `reduce` command.

```
$ cj reduce results.txt 0ed00c68

CJmessage::Checking progress of runs...
CJmessage::Reducing results...
CJmessage::Do you want to submit the reduce script to the queue via srun?(recommneded
for big jobs) Y/N?
n

 SubPackage 1 Collected (6.67%)
 SubPackage 2 Collected (13.33%)
 SubPackage 3 Collected (20.00%)
 SubPackage 4 Collected (26.67%)
 SubPackage 5 Collected (33.33%)
 SubPackage 6 Collected (40.00%)
 SubPackage 7 Collected (46.67%)
 SubPackage 8 Collected (53.33%)
 SubPackage 9 Collected (60.00%)
 SubPackage 10 Collected (66.67%)
 SubPackage 11 Collected (73.33%)
 SubPackage 12 Collected (80.00%)
 SubPackage 13 Collected (86.67%)
 SubPackage 14 Collected (93.33%)
 SubPackage 15 Collected (100.00%)
CJmessage::Reducing results done! Please use "CJ get 0ed00c68 " to get your results.
```

This will produce the new file `results.txt` in the main directory, which contains:

```
$ cat /Users/hatef/CJ_get_tmp/0ed00c6851c504af7d8064a954aba44cf1da40f2/results.txt
1,1,2
1,2,3
1,3,4
1,4,5
1,5,6
2,1,3
2,2,4
2,3,5
2,4,6
2,5,7
3,1,4
3,2,5
3,3,6
3,4,7
3,5,8
```

The file to reduce can be any acceptable MATLAB output, for instance `output.mat`.

CJ commands

To see a full list of options

```
cj -help
```

In what follows, we review important commands.

RUN

```
cj run <script> <machine> -dep <DEPENDENCY_FOLDER> -m <MESSAGE> -mem <MEM_REQUESTED>
```

DEPLOY

```
cj deploy <script> <machine> -dep <DEPENDENCY_FOLDER> -m <MESSAGE> -mem
<MEM_REQUESTED>
```

PARRUN

parellel run for embarssingly parallel problems.

```
cj parrun <script> <machine> -dep <DEPENDENCY_FOLDER> -m <MESSAGE> -mem
<MEM_REQUESTED>
```

RERUN

To rerun a previously failed package.

```
cj rerun <PACKAGE> <FOLDER_NUMBER>
```

STATE

To see the state of the last job submitted through CJ,

```
cj state
```

To see the state of a particular package,

```
cj state <PACKAGE>
```

To see the state of a particular folder in a parrun package,

```
cj state <PACKAGE> <FOLDER_NUMBER>
```

LOG

To see the last N instances of CJ run,

```
cj log <N>
```

To see detailed log of a particular package,

```
cj log <PACKAGE>
```

To see all the packages

```
cj log all
```

INFO

To see the information of the last call to CJ,

```
cj info
```

To see the information of a particular package,

```
cj info <PACKAGE>
```

REDUCE

To reduce the results of the last parrun call,

```
cj reduce <RESULTS_FILENAME>
```

To reduce the results of a particular parrun package,

```
cj reduce <RESULTS_FILENAME> <PACKAGE>
```

GET

To get the results of the last call back,

```
cj get
```

To get the results of a particular package call back,

```
cj get <PACKAGE>
```

CLEAN

To remove the last package and its associated jobs,

```
cj clean
```

To remove a particular package and its associated jobs,

```
cj clean <PACKAGE>
```

SHOW

To show the program run by CJ at the last call,

```
cj show
```

To show the program run by CJ for a particular package,

```
cj show <PACKAGE>
```