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 a system for automatizing reproducibility and hassle-free submission of computational jobs to remote cluster from your local machine. CJ is written in perl language and produces 'reporoducible' computational packages for academic publications. The project started at Stanford University by Hatef Monajemi and his PhD advisor David L. Donoho to make the large-scale scientific computing tasks more efficient. Current implementation allows submission of MATLAB jobs. In the future versions, we hope to include other heavily used programming languages such as Python, and R.

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

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`

```
$ 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 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`.

4. 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.

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.

Running jobs

We will now go over a simple example to demostrate Clusterjob usage.

Matlab simple example

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) reproducible_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) reproducible_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.

CJ commands

CJ can currently perform the following tasks:

`run`,`deploy`,`parrun`, `rerun`, `state`,`history`, `show`,`info`, `reduce`,`gather`,`get`, `clean`

To see a full list of options

```
cj -help
```

In what follows, we go over these tasks one by one.

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

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>
```

HISTORY

To see the last N lines of CJ history,

```
cj -h[istory] <N>
```

To see the history of a particular package,

```
cj -h[istory] <PACKAGE>
```

To see all the history

```
cj -h[istory] 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>
```