CSC BioWeek 2018: Using Taito cluster for high throughput data analysis

7. 2. 2018

Running Jobs in CSC Servers

Exercise 1: Running a simple batch job in Taito

We will run a small alignment using BWA:

https://research.csc.fi/-/bwa

Log into **taito.csc.fi**

```
cd $WRKDIR
mkdir bwa_ex
cd bwa_ex
```

Copy a set of test files to your current folder:

```
cp /wrk/trng87/data/ex1/* .
```

Many of the bioinformatics tools in Taito are loaded with command: `module load biokit`. There are, however, several tools that have their own set up commands, or which don't need a set up command. You can check the software specific information from the bioscience program list:

http://research.csc.fi/bioscience_programs

Among the files you copied, is a template batch job script called `bwa_run.sh` Open it with a text editor. You can use for example *nano or emacs* editors:

```
nano bwa_run.sh
emacs bwa_run.sh
```

Edit batch job script so that:

- It reserves 4 cores in one node, 8 GB memory and 30 min time
- loads the biokit module as described on the program web page

Submit the job with command:

```
sbatch bwa_run.sh
```

After submission you can follow the status of your job with commands:

```
squeue -u trng*XX*
squeue -l -u trng*XX*
```

Exercise 2: Running an array job

(http://research.csc.fi/taito-array-jobs)

In this exercise we will run quality analysis on some fastq files using fastqc software using *array batch job* approach.

Log into **Taito.csc.fi**

First copy and unpack the input data set:

```
cd $WRKDIR
mkdir array_ex
cd array_ex
cp /wrk/trng87/data/ex2/* .
```

Now you should have 6 new files in your directory.

There is also a template batch job file `array_run.sh` Next open a text editor and edit the batch job script so that it

- loads the biokit module
-
- reserves 1 core, 1 GB memory/core and 20 min time
- runs an array job of 6 sub jobs

And then submit the job with command `sbatch`.

Use *squeue* to check what the job looks like.

Use `sacct` and `seff` to look at used resources

The provided template works only with files named in this very specific way. In real-world applications this is typically not the case.

Look at the lecture hand-outs for details, and change the batch job script so that it uses a list of file names instead.

First make a list of the input files:

```
ls *.gz > list.txt
```

In the script you can use for example:

```
name=$(sed -n ${SLURM_ARRAY_TASK_ID}p list.txt)
fastqc ${name}
```

After submission you can follow the status of your job with commands:

```
squeue -u trngXX
```

```
squeue -l -u trngXX
```

Exersice 3: Array job using sbatch_commandlist

Do the same task as Exersice 2, but using `sbatch_commandlist`

As an optional extra task you can write a script that first writes the commandlist file

Exercise 4: Looking at used resources

After the jobs in Exercises 2 -3 have finished, you can check what resources it used with

```
seff <jobid>
```

(If your jobs have not finished, you can use jobids 18314555 and 18315642 instead.)

Were the resource requests reasonable?

Another tool to look at resource usage is `sacct`

To see details of a single job you can use

```
sacct -l -j <jobid>
```

sacct lists a rather long list of values, many of which are of interest mainly to sysadmins. To get a more manageable output you can use the –o option to define which fields you want to see.

```
sacct -o jobid,jobname,maxrss,maxvmsize,state,elapsed -j <jobid>
```

Here we have elected to show jobid (JobID), jobname (JobName), maximum used memory (MaxRSS), maximum used virtual memory (MaxVMSize), state of the job (State) and elapsed time (Elapsed). These can be helpful when we decide on the resource allocation parameters for our next similar job.

Compare results from `sacct` and `seff`. Which datafield from `sacct` you need to replicate the info from `seff`?

Why might `sacct` be preferable for array jobs?

For details on the different available fields and other options see

```
man sacct
```

More information about running batch jobs in Taito:

http://research.csc.fi/taito-batch-jobs

As an optional extra task you re-run one of the jobs in exercises 2-3 and observe it with top.

Extra exercise: Installing your own application

We did not cover this on the lectures, but you can try this if you want. You can find more information in the **CSC Computing Environment User Guide**:

https://research.csc.fi/csc-guide-user-specific-directories-at-the-servers-of-csc

In this example we install MCL Markov cluster algorithm program to users own $USERAPPL directory in Taito.

Move to your $USERAPPL directory and create there a new directory called *mcl*

```
cd $USERAPPL
mkdir mcl
```

Go to the just created **mcl** directory and download the installation package with wget command

```
cd mcl
```

```
wget http://www.micans.org/mcl/src/mcl-latest.tar.gz
```

In this case the installation package is a tar-archive file that has been compressed with gzip program. You can unpack this file with commands

```
gunzip mcl-latest.tar.gz
tar xvf mcl-latest.tar
```

After unpacking, the **ls** command shows that a new directory called *mcl-14-137* has been created to your *mcl* directory. This directory contains the actual installation files and documentation of the software. Create a new empty directory called *version-14-137* to the mcl directory.

```
ls
mkdir version-14-137
```

After this go to the *mcl-14-137* directory and study its' content

```
Cd mcl-14-137
ls -l
```

Installation packages contain often a short installation instructions. Typically this instruction file is called as *INSTALL* or *README*. In this case you should read the *INSTALL* file to find out how the installation should be done.

```
less INSTALL
```

Many open source software tools are installed using following three steps:

1. Building up the so called *Makefile* with a **./configur**e command.
2. Running **make** command that compiles the source code according to the instructions in the *Makefile*
3. Installing the compiled executables with command **make install**

Normally the installation packages assume that the user has permissions to install the software to the locations where the standard linux commands and programs normally get installed. However, at CSC this is not the case. You can install software only to your own disk areas. Often you can use option **--prefix=*/path/*** to tell to the configure command, where to the program should be installed. In this case we wish to install the software to the *version-14-137* directory in you $USERAPPL area. Thus you must use following ./configure command:

```
./configure --prefix=$USERAPPL/mcl/version-14-137
```

The configure command checks that all the compilers and libraries, that the software needs, are available. It is not uncommon, that *./configure* reports about missing libraries or incorrect compilation options. In those cases you can check if the missing library or program can be taken in use with the module system. CSC environment has several compiler and program versions available. In some cases you may for example need to use certain C-compiler or python version in order to install the software. If you still fail with the installation, ask help from the HelpDesk of CSC.

In the case of *mcl,* the *./configure* script runs without error messages when you use GNU-compilers. The GNU compilers are set up with command:

```
module switch intel/13.1.0 gcc
```

Next need to compile and install the software with commands:

```
make
```

```
make install
```

If *make* and *make install* commands don't give any error messages, you have successfully installed your software. Typically the executables, i.e. the compiled programs that can be launched, are stored to a sub directory called *bin*. In this case the bin directory is created to subdirectory *$USERAPPL/mcl/version-14-137.*

Running command

```
ls $USERAPPL/mcl/version-14-137/bin
```

now shows the programs you have installed:

clm clmformat mcl mclcm mclpipeline mcx mcxarray mcxassemble mcxdump mcxi mcxload mcxmap mcxrand mcxsubs

The name of the directory that contains the executables may vary between different software. In any case, to be able to use the programs you must tell the location of your own executables to the command shell. This can be done by adding the directory path of you executables to the **$PATH** environment variable. In this case we add path "**${USERAPPL}/mcl/version-14-137/bin**" to the $PATH variable. This is done with command:

```
export PATH=${PATH}\:${USERAPPL}/mcl/version-14-137/bin
```

Note that the first PATH word in the command above is without the dollar sign. Now you can launch the program you have installed. For example

```
mcl -h
```

Remember that also in the future, when you log in to CSC, the PATH variable must be set up before you can use **mcl** command. Also in the batch job files you need to run the correct *export **PATH*** command above before executing the program you have installed yourself.

If the software you have installed works correctly, you can remove the installation package and temporary directories that were used during the compilation. In this case we could remove the *mcl-latest.tar* file and the directory *mcl-14-137/*

```
cd $USERAPPL/mcl
rm mcl-latest.tar
rm -rf mcl-14-137/
```