Building an HPC blast pipeline

Bonnie Hurwitz

Abstract

This tutorial shows you how to create a pipeline on the University of Arizona HPC. Specifically, we will be creating a pipeline to run blast, using scripts you are writing or have already written from ABE 487 weeks 10 and 11. The pipeline will do the following:

1. split each of the files from the night/day transcriptome into files with 10,000 sequences each
2. run blastp against uniprot_sprot.fasta
3. parse the blast results

Citation: Bonnie Hurwitz Building an HPC blast pipeline. **protocols.io**
dx.doi.org/10.17504/protocols.io.d5a82d
Published: 10 Nov 2015

Guidelines

Before starting this tutorial, you need to complete the scripts from the computational homework for week 10 and 11.

Protocol

Step 1.
Login to the hpc (at the University of Arizona) with your user name

cmd COMMAND
```
ssh username@login.hpc.arizona.edu
```

NOTES
Bonnie Hurwitz 10 Nov 2015
Note that this script runs on the head node. This script is not compute intensive, and we need to create the smaller files first, before we can distribute the data on the cluster as smaller blast jobs.

Step 2.
Create a directory for the pipeline

cmd COMMAND
```
mkdir hpc-blast
cd hpc-blast
```

Step 3.
Go to the hpc-blast directory and create sub directories to hold std-err, std-out, data, blastdb, blast-results and scripts.

cmd COMMAND
```
mkdir std-err std-out data blastdb blast-results scripts
```

Step 4.

Go to the blastdb directory and create the blast database.

cmd COMMAND

```
cd blastdb
module load irods
iget /iplant/home/shared/imicrobe/projects/43/CAM_PROJ_MarineMicrobes.pep.fa
module load blast
makeblastdb -in CAM_PROJ_MarineMicrobes.pep.fa -dbtype prot
cd ..
```

NOTES

Bonnie Hurwitz 10 Nov 2015
go to the blastdb directory
load irods
get the peptides from the Gordon Betty Moore Foundation Marine Microbes in iPlant.
load blast
create the blast database to compare against
go up a directory

Step 5.

Copy the input files over from iPlant. We will be using the day/night transcriptomes we discussed last week.

cmd COMMAND

```
cd data
icd /iplant/home/shared/imicrobe/class/night_day
iget SRR006596_day.fasta
iget SRR006597_night.fasta
cd ..
```

NOTES

Bonnie Hurwitz 10 Nov 2015
go to the data directory
change directories in irods to the day/night transcriptome data directory
get the day transcriptome fasta file
get the night transcriptome fasta file
go up a directory

Step 6.

You will use two of the Perl scripts from the homework in the hpc pipeline. Put these into the scripts directory.

cmd COMMAND

```
cd scripts
cp /my-path/week10/01-fasta-splitter.pl 01-fasta-splitter.pl
cp /my-path/week11/01-bio-searchio.pl 01-bio-searchio.pl
cd ..
```

NOTES

Bonnie Hurwitz 10 Nov 2015
go to the scripts directory
copy the fasta-splitter and blast-search perl programs from your homework to the scripts dir.
note you will need to change the path to point to the location of the scripts in your home directory.

Step 7.

config.sh: Now we need to create the config.sh script. This script will contain all of the environmental variables that the user will be able to modify for running blast. The idea of this script is to give the user one place to make any changes they need to run their particular blast job. The

user should not have to go in and change any of the scripts in the 'scripts' directory. Instead, we will capture all of these variable here.
* note the the config.sh script is in the hpc-blast directory (in the first part of the pipeline), where it is easy for the user to modify it.

cmd COMMAND

```
vi config.sh

#enter all of the text below
-----------------------------

#!/bin/bash

# directories we need
export CWD=$PWD
export SCRIPT_DIR="$CWD/scripts"
export FASTA_DIR="$CWD/data"
export STDERR_DIR="$CWD/std-err"
export STDOUT_DIR="$CWD/std-out"
export SPLIT_FA_DIR="$FASTA_DIR/faSplit"
export SPLIT_READ_CT="25000"
export BLAST_OUT_DIR="$CWD/blast-results"

# BLAST parameters
export EVAL="1e-10"
export BLAST="blastx"
export BLASTDB="$CWD/blastdb/CAM_PROJ_MarineMicrobes.pep.fa"
```

NOTES

Bonnie Hurwitz 10 Nov 2015
set the parameters for the location of the directories

set the blast parameters that the user can change (e.g. e-value, blast program, and database to compare against). Note that we created the blast database in a previous step.

Step 8.
blast-pipeline.sh: Now we will create the "driver script", the script that is used to run each part of the pipeline.

cmd COMMAND

```
vi blast-pipeline.sh

#enter all of the text below
-----------------------------

#!/bin/sh

# Step1: get the env variables and create a list of files
source ./config.sh

# Step 2: get all of the fasta files for input
cd "$FASTA_DIR"
export FILES_LIST="$FASTA_DIR/files-list"
pwd
ls *.fasta | sed "s/^\.\///" > $FILES_LIST

# Step 3: split the fasta files into smaller chunks
$SCRIPT_DIR/run-fasta-splitter.sh
```

```
# Step 4: run blast on each of the file chunks
while read FILE; do
    export FILE="$FILE"
    cd $SPLIT_FA_DIR
    export SPLIT_FILES_LIST="$SPLIT_FA_DIR/split.$FILE"
    ls $FILE.* > $SPLIT_FILES_LIST
    NUM_SPLIT_FILES=$(wc -l $SPLIT_FILES_LIST | cut -d ' ' -f 1)

    ## run blast on each of the smaller chunks against a blast database
    JOB_ID1=$(qsub -v SCRIPT_DIR,SPLIT_FA_DIR,BLAST,EVAL,BLASTDB,BLAST_OUT_DIR,FILE -
N blast -e "$STDERR_DIR" -o "$STDOUT_DIR" -J 1-$NUM_SPLIT_FILES $SCRIPT_DIR/run-blast.sh)

    ## parse the blast results for each of the chunked files
    JOB_ID2=$(qsub -v SCRIPT_DIR,BLAST_OUT_DIR,FILE -W depend=afterany:$JOB_ID1 -
e "$STDERR_DIR" -o "$STDOUT_DIR" -J 1-$NUM_SPLIT_FILES $SCRIPT_DIR/run-parse-blast.sh)

done < $FILES_LIST
```

NOTES

Bonnie Hurwitz 09 Nov 2015

Note that steps 1-3 are run on the head node of the cluster, these steps are not compute intensive. These steps set up the environmental variables and split up the fasta files into smaller chunks to run on the cluster. Step 4 runs blast and parses the blast results on the cluster once the blast analysis has finished. Note that these are being sent to compute nodes using the "qsub" command, and that the environmental variables that are useful for each step are being sent along as well. Also note that the second step is "held" until the first step completes. You can set this to run, only in the case that the prior step completes without errors "afterok" or no matter what "afterany". The first requires that none of the jobs produces an error. Also note that we are using a job array "-J 1-NUM_SPLIT_FILES", where the number of split files is equal to the total split file for that file. Also, note that the qsub commands run shell scripts that execute the actual work.

ANNOTATIONS

Nicholas Lytal 16 Nov 2015

If you get the error "qsub: Bad GID for job execution", you should double-check your group name. Enter "va" on the command line to see which group you are in. For Steps 10 and 11, make sure the first $PBS line is set to THAT group.

Step 9.

run-fasta-splitter.sh. Create a shell script that runs the fasta splitter perl code.

cmd COMMAND

```
cd scripts

vi run-fasta-splitter.sh

#enter the text below

#!/bin/bash

source /usr/share/Modules/init/bash

if [[ ! -d "$SPLIT_FA_DIR" ]]; then
    echo Cannot find faSplit \"$SPLIT_FA_DIR\"
    mkdir -p $SPLIT_FA_DIR
fi

if [[ ! -e "$FILES_LIST" ]]; then
    echo Cannot find files list \"$FILES_LIST\"
```

```
    exit 1
fi

cd $FASTA_DIR
while read FILE; do
   echo $FILE
   $SCRIPT_DIR/01-fasta-splitter.pl -n $SPLIT_READ_CT -o $SPLIT_FA_DIR $FILE
done < $FILES_LIST
```

NOTES

Bonnie Hurwitz 10 Nov 2015

Note that this script runs on the head node. This script is not compute intensive, and we need to create the smaller files first, before we can distribute the data on the cluster as smaller blast jobs.

Step 10.

run-blast.sh: Create a shell script that runs blast on a compute node.

cmd COMMAND

```
vi run-blast.sh

# enter the text below

#!/bin/bash

#PBS -W group_list=bhurwitz
#PBS -q windfall
#PBS -l jobtype=cluster_only
#PBS -l select=1:ncpus=2:mem=4gb
#PBS -l place=pack:shared
#PBS -l walltime=24:00:00
#PBS -l cput=24:00:00
#PBS -M youruser@email.arizona.edu
#PBS -m bea

# BLAST parameters
NUM_THREADS=2    # make sure this is requested in the above "select"
DESC=10
ALN=10
MAX_HSPS=10

module load blast
cd "$SPLIT_FA_DIR"
FASTA="$SPLIT_FA_DIR/$FILE.${PBS_ARRAY_INDEX}"

# run blast on each file chunk
$BLAST -query $FASTA -db $BLASTDB -evalue $EVAL -
out $BLAST_OUT_DIR/$FILE.${PBS_ARRAY_INDEX}.blastout -max_hsps $MAX_HSPS -
num_descriptions $DESC -num_alignments $ALN -num_threads $NUM_THREADS
```

Step 11.

run-parse-blast.sh: Create a shell script to parse the blast output that runs on a compute node on the cluster.

cmd COMMAND

```
vi run-parse-blast.sh

# enter the text below

#!/bin/bash

#PBS -W group_list=bhurwitz
```

```
#PBS -q windfall
#PBS -l jobtype=cluster_only
#PBS -l select=1:ncpus=2:mem=4gb
#PBS -l place=pack:shared
#PBS -l walltime=24:00:00
#PBS -l cput=24:00:00
#PBS -M youruser@email.arizona.edu
#PBS -m bea

module load blast
cd $BLAST_OUT_DIR
BLOUT="$BLAST_OUT_DIR/$FILE.${PBS_ARRAY_INDEX}.blastout"
echo File \"$FILE.${PBS_ARRAY_INDEX}.blastout\"
$SCRIPT_DIR/04-blast-search.pl $BLOUT > $BLOUT.parsed
```

NOTES

Bonnie Hurwitz 10 Nov 2015

Like the previous script the PBS commands at the beginning of the script indicate the resource needed to run the job. Also note that we need to load blast and perl, to access blast commands and the version of perl with bioperl installed. Also note that we are using the variable ${PBS_ARRAY_INDEX}. This variable stores information about which blast job we are running from 1-NUM_SPLIT_FILES that is specified in the driver script with a "-J"

Step 12.

04-blast-search.pl: modify the blast search script.

Update the e-value to be less stringent.

Modify the Perl script 04-blast-search.pl to include additional information about the hit.

cmd COMMAND

```
# make a few modifications in the 01-bio-searchio.pl script
# change the e-value in the script
my $min = 1e-10;

# update the print statement
# add additional information about the hit.

say join "\t",
                    $result->query_name,
                    $hit->name,
                    $hit->description,
                    $hsp->percent_identity,
                    $hsp->length('total'),
                    $hsp->evalue;
```

Step 13.

Make all of the scripts executable and then run the pipeline.

If you don't get any errors, the pipeline will take <1 day to run.

cmd COMMAND

```
chmod 755 *sh *pl
cd ..
chmod *sh
./blast-pipeline.sh
```

make sure your scripts are executable for before running

Step 14.

Check to see if the pipeline is running

cmd COMMAND

```
qstat -u username
```

for me this command is: qstat -u bhurwitz

Step 15.

Once the pipeline is finished running (and you don't see the jobs listed under qstat), check for std-errors and std-out.

Note you can check while the pipeline is running too.

```
cmd COMMAND
cd std-err
ls -1
# you should see zero length files
cd ..
cd std-out
ls -l
# you should see some description about the amount of compute time used per job
```

Step 16.

Now you try.

See if you can update the pipeline to add one final step that puts all of the parsed blast output into a single file for each of the original fasta files.

Note that you can comment out the parts of the blast-pipeline.sh that you have already run in the steps above, to test your code. Otherwise, these will run again.

Also, before you run this final step, make sure the blast parsing is complete.

Can this run on the head node, or should we run this on a compute node?

The complete pipeline can be found in github here.

Step 17.

Commit your code to your own github abe487 repository using the directory:

hpc-blast