Processing GBS Data using Westgrid

Introduction

This guide will cover how to process data using Westgrid system resources. The first step in this process it to sign up for a Westgrid account of your own. To do this, follow the steps on the Westgrid website. You should run all experiments/computations account.

The logon information for Ravi's account (the main account) is:

- Username: ravi0850
- Password: r0850+NajRB
- Compute Canada Account: eie-552-01

A Syntax Note

In this document, all code where you, the user, will need to input something (such as file name) is demarked with "<>", i.e.,:

```
<Please put something here>
```

In cases where user input is optional, for example the email field in BT_ST_Writer, this input is demarked with "[]", i.e.,:

```
[this code is optional]
```

Overview of Westgrid Resources

Westgrid is a network of computer systems that serves Western Canada. We are currently using the Jasper system on Westgrid. Details and specifications about Jasper can be found here.

Note that Jasper is currently scheduled to shut down on October 31st, 2017. New servers (Cedar, and another) are being built to replace Jasper, and when these come online it would probably be a good idea to switch these systems.

You can request for software to be installed on the systems (for instance, VCFTools) by emailing support@westgrid.ca. The Support service for Westgrid is top notch, and they are generally willing and able to answer any questions you might have.

Note that there is a requested storage limit of only 100 GB of data at a time for any user. For this reason, it is recommended that you only work on one sequencing project at a time, in order to respect this storage limit. Additionally storage space can be requested by contacting Westgrid support.

Logging on to a Westgrid system

###Using a Mac or Linux Westgrid is accessed using SSH protocol. SSH protocol is native in Unix-based systems (i.e., MacOS, Linux), and you can log on to a Westgrid system as follows when using these systems:

1. Open a terminal
2. Type `ssh -X <username>@<system>.westgrid.ca`

In this case, the `<system>` field will likely be jasper, until cedar or another appropriate system is found. The `-X` argument is not necessary, but is recommended as it opens up X11 forwarding which enables GUI windows to be used (e.g., Filezilla or gedit).

Using Windows

Unfortunately, SSH protocol is not natively supported in Windows. Fortunately, the program PuTTy makes connecting to Westgrid (or any other system using SSH protocol) very easy. A PuTTy installer is located on the Jade drive in the Resources folder. Alternatively, you can download PuTTy here. to connect to Westgrid system. It is also recommended that you download and install Xming-Mesa, which will allow you to use X11 forwarding.

When connecting via PuTTy the hostname is `<username>@<system>.westgrid.ca` and the connection port is `22`

Overview of Data Directory Structure

The pipeline for Westgrid has been designed with the following organizational structure in mind. Many of these directories are created the first time teh BT_ST_Writer.py script is run.

- Main Directory
  - data : created by user, contains the fastq files for each sample
  - pbs_scripts : created by user, directory where pbs scripts are generated
  - ref_genome : created by user, directory should contain reference genome fasta file and the trimmomatic reference fasta file

- regions : contains text files with the regions for the reference genome in a addition to a file with the list of regions. Created by Region_Splitter.py
- results : this directory, and all sub-directories are created by BT_ST_Writer.py
  - bam_file_lists : Contains the bam file lists
  - bowtie_alignments : Contains the alignment files that are generated by the BT_ST_Writer.py script
    - all_mapped_reads : Alignments with all reads, regardless of mapping quality
    - multi_map_reads : Alignments with only reads that map to multiple regions
    - unique_map_reads : Alignments with only reads that map to unique regions
  - mpileup_results : contains the results from the VC_Writer.py output (mpileup and bcftools call output)
  - trimmed_seqs : Contains the trimmed sequences generated by trimmomatic

Data Processing Pipeline

#####NOTE: All scripts were written using Python 3.5; depending on the system, you may have to use python3 instead of python when running the .py scripts

Overview

1. Pre-process the data on our local Bionic server. This includes splitting the chip files into individual fastq files for each sample and creating bowtie and samtools indexes for the reference genome.
2. Transfer sample fastq files and reference genome to the Westgrid server
3. Generate and submit PBS files for the bowtie and samtools (BT_ST) pipeline for each sample using BT_ST_Writer.py on Westgrid
4. Break up the reference genome into several smaller regions using Region_Splitter.py (can be done using either Westgrid or)
5. Generate and submit PBS files for the variant calling (VC; mpileup and bcftools call) step for each ref. genome regions using VC_Writer.py on Westgrid
6. Transfer the results back to Bionic
7. Run VCFTools filtering and generate a tab file on Bionic

Step 1: Pre-process Data (Bionic)

1. Create Barcode.tab file, associating each sample name with it's barcode sequence
   - generated using two spreadsheets given from biologists
     - one should contain the sample name on each well position when they did their sequencing
       - a well is just a slot in the machine. Aka plate position
   - the other should contain the barcode assigned to each well

        - usually used the same barcode file, so this file can be reused
        - BC-Proton-Pstl.xlsx
    - combine the two sheets to find the barcode associated with each sample id
    - all barcodes should be the same length
        - barcodes are usually suffixed with GAT. This gives a buffer so some can be added or removed to enforce same size
            - ensure that barcode file is in UTF8 format
            - sample barcode file: (sampleID, barcode)

1206	CTAAGGTAACGA
1207	TAAGGAGAACGA
WI	AAGAGGATTCGA

2. Split out chip fastQ files into seperate ones for each sample (barcode)
    - chip fastQ files contain giant lists of sequences, but we want to parse through and look at each sample individually
        - each chip represents a run. We split them out individually to avoid using a huge file, but maybe they could be joined first
    - use fastx_barcode_splitter to split out each barcode into it's own file for each chip
    - `$ cat ./Chip1.fastq | fastx_barcode_splitter.pl --bcfile barcode_file.tab --bol --prefix Chip1/ --suffix ".fq"`
        - prefix is used to specify which directory to save the output to. This direcotry will have to be created before the command is run
        - must be run once for each Chipi.fq file
    - if you have fastq.gz files, you can use gunzip instead of cat to uncompress in place
        - `$gunzip -c ./Chip1.fastq.gz | fastx_barcode_splitter.pl --bcfile barcode_file.tab --bol --prefix Chip1/ --suffix ".fq"`

3. Merge sample fastQ files from each chip, into combined sample files

    - go into each new ./Chipi directory, find .fq files with matching names, and concatenate them into a new merged directory

    - can use Craig's simple Merge script:

```
ls $1 | while read FILE; do
        cat $1/"$FILE" $2/"$FILE" $3/"$FILE" >> $4/"$FILE"
  done
```

        - usage: `Merge.sh ./Chip1 ./Chip2 ./Chip3 ./Merged`

- reads through all files in first folder, finds matches in others, and saves results in last folder

Step 2: Transfer data to Westgrid

1. Open filezilla and connect to the Westgrid
   - Connection port is 22; hostname is `<system>.westgrid.ca`
2. Create a directory for the project on Westgrid (can be done with Filezilla, right click on server area)
3. Create ref_genome, data, and pbs_script sub-directories in the project directory
4. Transfer the sample fastq files to the data directory using Filezilla's drag and drop interface
5. Transfer the reference genome fasta files and index files (.bt, .bt2, and .fai extensions) into the ref_genome directory
6. Transfer the needed adapter sequence fasta file to the ref_genome directory
   - Illumina data: TruSeq3-SE.fa
   - Ion Torrent data: IonTorrent.fa
7. Transfer the BT_ST_Writer.py, VC_Writer.py, and submit.sh scripts into the pbs_script directory

Step 3: Create BT_ST PBS Scripts with BT_ST_Writer.py

1. Navigate to the pbs_scripts directory

2. Run the BT_ST_Writer.py script, e.g.:

   ```
   python BT_ST_Writer.py --data ../data --ref_genome ../ref_genome --results ../results --mem_required 4gb --email john.hayes@usask.ca
   ```

   Note: Email field is optional.

   If using relative paths, they must be relative to the directory where the pbs file will be submitted to data, reference genome, and results must be relative to the directory where the PBS files will be submitted (generally the pbs_script directory, but ultimately this is the users choice)

   Second Note: The script has a number of abbreviations for the arguments as well, you can use `python BT_ST_Writer.py --help` to see a full list of options

   Final Note: The mem_required option was added, as certain genomes (*cough* wheat *cough*) require large amounts of memory; usually 4gb is good. Run a test case for one sample on Bionic first if you are unsure how much memory is required. (Memory resources can be managed with `htop`)

3. Submit an individual pbs file as a test, before submitting all the pbs files. Individual files can be submitted using the command:

   ```
   qsub &lt;pbs_file&gt;
   ```

4. Once you have verified that the pbs scripts work, submit the pbs files using the shell script `./submit.sh`

   ```
   ##Submit.sh
   /#! bin/bash/
   for f in *.pbs
   do
     qsub $f
   done
   ```

   Note: `submit.sh` will submit all pbs files in the directory; please ensure that only the pbs files you want to submit are present

5. Wait for files to run; depending on how busy the system is, this can take anywhere between a few hours to a day (usually not more than 24 hours)

6. Clean-up: After submitting a PBS file, two extra files will be generated, one with the extension pbs.e and one with extension pbs.o. The pbs.e file contains all error output and program output while the pbs.o file contains all terminal output (such as that generated by echo). These files, in addition to the pbs files, should be transferred to a sub-directory in order to expedite Step 5. To do this easily use:

   ```
   mkdir <subdirectory>
   mv *.pbs* <subdirectory>
   ```

Step 4: Breaking the reference file into regions

There are two ways that the reference file can be broken into regions:

1. Generate a list of regions from the .fai file using Region_Splitter.py
2. Break up the reference fasta file into several smaller fasta files

Currently, only option 1 is fully supported and tested, though option 2 is nominally supported in VC_Writer (see step 5)

This step is currently fairly tricky; but the key is this: For region files, they must be text file which is arranged in bed format. The bed format is tab seperated, three column table which is a three column format. An example would be (note: column headers are used, they are only included here for clarity/explanatory purposes):

Region	Region Start Position	Region End Position
Chr1	0	(Region lengtht - 1)**

** Note: Bed files are 0 indexed (i.e., the first position = 0), so the end position is the length - 1 (to account for zero index)

The region file may contain only one region (e.g., a full chromosome) or multiple regions (e.g., a large number of small scaffolds)

In addition to generating this list, you must also generate a txt file with a list of the region files, with one region file per line. The path to these region files can be either absolute or relative to the **PBS submission location**.

The Region_Splitter.py file uses Python's re module (regular expressions) to parse through the .fai file. This file has a module which will need to be hacked by you, the user, in order to appropriately separate your regions (I recommend that you read through the .fai file to find the best way to break up the regions)

Step 5: Create VC PBS Scripts with VC_Writer.py

PBS files for the variant calling (VC) step can be created with VC_Writer.py The script runs in one of two modes, which are called using the -t flag:

- `-t region` : the reference genome has been broken into regions with separater region files
- `-t fasta` : the reference genome has been broken into several smaller fasta files**

  ** Note that each of these fasta files must be indexed using samtools faidx

VC_Writer takes input via flag arguments (i.e., --arg value, -arg value), and the full list of input options can been generated by running:

```
python VC_Writer.py --help
```

An example using the region option is:

```
python VC_Writer.py -t region -l ../regions/region_list.txt -r ../results -f ../ref_genome/reference_genome.fa -b ../results/bam_file_lists/bamfile.txt [--file_ext .txt] [-e john.hayes@usask.ca]
```

Note that the arguments in "[]" are optional. The --file_ext refers to the file extension of the region files or the fasta files

Once the PBS files have been generated, submit one PBS file for testing. On confirmation that it works, use the `./submit.sh` script to submit the PBS files. Depending on server load, this process can several hours to one day (usually not more than 24 h).

Step 6: Transfer Data Back to Bionic

On the homestretch!

Once you have all the data analyzed, transfer it back to the Bionic server. You can usually get away with just transferring the contents of the Results directory. It is recommended to transfer this data to an appropriate directory in the WorkingDirectory (i.e., on the SSD), as this will speed up the following processes.

Step 7: Combine the pileup files and filter using VCFTools

Navigate to the directory containing the mpileup results. You should have a number of .vcf.gz files, which we would like to combine into a single file. To do this, use the BCFTools concat feature. To concatenate all the .vcf.gz files in a directory use:

```
bcftools concat -O z -o <outfilename.vcf.gz> *.vcf.gz
```

Here, the `-O z` argument lets bcftools know you want the output in a compresssed vcf format. You could also use `-O v` if you wanted an uncompressed vcf file.

Once the files are combined, we would like to filter the variants by quality to help eliminate false positives. We do this using VCFTools. To filter:

```
vcftools --gzvcf <combined vcffile> --minDP 6 --maf 0.1 --max-missing 0.8 --recode --out <file_name>
```

** Note that the out file name should not include an extension (vcftools adds one automatically)

For filter paramters:

- `--minDP` is the read depth, i.e., how many reads were found at that position, and is usually set to 6; setting too low can lead to false reads, too high can lead to too few variants surviving
- `--maf` is minimum allele frequency, or how often that allele was found in all the samples; The value is a decimal percent (i.e., 0.1 = 10%), and can vary from 0 to 1; lower values results in less filtering (i.e., more SNPs are reported) while high values will increase the filtering (resulting in fewer SNPs)
- `--max-missing` is the number of alleles that were unmappable for that SNP. In other words, if 8 of 10 samples had a SNP at a given position, while the other to had a blank (./.) read at this position, the missing percentage would be 0.2 (20%). Max-missing can vary between 0 - 1, with lower values resulting in greater filtering.

Once you have filtered the file you will have an uncompressed vcf file, which can be converted to a tab-separated format for further analysis. To do this, we will pipe the output from cat (a.k.a. the file read) to the vcf-to-tab function. This can be done with the following command:

```
cat <vcf file> | vcf-to-tab > <output_file.tab>
```