Illumina pipeline based on CASAVA 1.8

Table of contents

Section 1: The Pipeline

The Illumina pipeline is a completely automated system from when the data comes off the sequencers to when a BAM is created. This section covers different major steps in the process.

1.1 Overview of pipeline

The pipeline starts after sequencers finish sequencing and RTA (on-instrument software component) completes writing all the files to the cluster.

1) The cron daemon on the cluster looks for new flowcells twice an hour. On finding a new flowcell, it instantiates the pipeline to start the analysis.

2) The pipeline prepares the flowcell for analysis. It uploads analysis start date to LIMS, contacts LIMS to obtain the necessary information to analyze the flowcell and write a flowcell definition XML file. It also writes files required for demultiplexing the flowcell.

3) The pipeline executes CASAVA 1.8's bclToFastq conversion tool to generate fastq files from the bcl files written by the sequencers.

4) The pipeline creates final sequence files by consolidating the CASAVA generated fastq files and removing the reads that fail purity filtering.

5) The pipeline uploads the sequence level results (e.g. lane yield, phasing and pre-phasing information) to LIMS and performs sequence analysis.

6) In sequence analysis step, number of unique reads, reads with adapter sequences, distribution of N bases are calculated and appropriate results are uploaded to LIMS.

7) The pipeline starts the alignment using BWA.

8) After BWA completes writing a SAM file, the pipeline applies several Picard and custom programs to generate a final BAM. These involve fixing mate information, sorting a bam in coordinate sorted order, marking duplicates and calculating various alignment and other metrics.

9) The pipeline calculates capture stats if a chip design is provided in the flowcell definition XML (FCDefinition.xml).

10) The pipeline performs final result upload to LIMS (alignment percentage, alignment error, result directory, bam file path etc) to LIMS.

11) It performs cleanup by deleting intermediate files within the directory of the sequencing event and zips the sequence files.

12) At this point, primary analysis is complete and any additional steps can be added (e.g to generate SNP/Indel etc by adding suitable commands in the post run code).

13) Two days after completion of alignment, the result directories are obtained from LIMS and added to the list for archival by another cron job.

1.2 Detailed description

This section explains in more detail each of the steps from the previous section.

1) Detecting that a new flowcell is available for analysis:

   Code path: ipipeV2/auto_start/startAnalysis.rb

   Each Illumina HiSeq running RTA version 1.12 writes a file "RTAComplete.txt" in the flowcell directory after the RTA has completed transferring all the files. The cron job runs /stornext/snfs5/next-gen/Illumina/ipipeV2/auto_start/startAnalysis.rb. It loops through all flowcell directories and finds new directories having "RTAComplete.txt" file. It instantiates the pipeline for each of these flowcells and adds their names to a file "done_list.txt" to prevent them from being selected for analysis again.

   For GAIIx, the marker file RTAComplete.txt is absent. In this case, startAnalysis script looks for three files Basecalling_Netcopy_complete.txt, Basecalling_Netcopy_complete_READ1.txt, and Basecalling_Netcopy_complete_READ2.txt, which are written at the end of read 1 and read 2 respectively. For GAIIx flowcells that don't have these files, the analysis must be started manually for now.

2) Preparing a flowcell for analysis:

   Code path: ipipeV2/bin/PreProcessor.rb

   "PreProcessor.rb" prepares the flowcell for analysis. It performs the following actions:

   a) Upload analysis start date to LIMS.

   b) Contact LIMS to obtain the list of barcodes for the flowcell and other information such as reference paths, chip design, sample name and library name and writes all this information in a file ./Data/Intensities/BaseCalls/FCDefinition.xml. [More details to follow later]

   c) Using the information in FCDefinition.xml, the pipeline writes a SampleSheet.csv in "Basecalls" directory of the flowcell. The first line of SampleSheet.csv MUST be a header describing various fields in the file.

   d) Using the information in FCDefinition.xml, the pipeline writes a barcode_definition file which contains a local copy of index tags and their corresponding sequences. This is specifically designed to handle barcodes of different lengths per flowcell. [More details to follow later]

3) Creating directory structure and converting bcl to fastq files:

Code path: ipipeV2/bin/BclToFastQConvertor.rb

a) This script invokes CASAVA's bcl to fastq convertor script to create a directory structure for the analysis and run "make" on it to start bcl to fastq conversion. The parameters are configured to allow for one mismatch in barcode index, ignore missing bcl file(s) and ignore missing stats file(s).

b) This script also writes a shell script "barcodes.sh" in the "BaseCalls" directory. "barcodes.sh" contains commands to build final sequence files for each lane barcode in the flowcell. This script is executed upon successful completion of the "make" tool in the previous step, through job dependencies in MOAB.

4) Building final sequence files for alignment:

   Code path: ipipeV2/bin/LaneAnalyzer.rb

   a) This script is invoked for each lane barcode (sequencing event) per flowcell. It consumes CASAVA generated fastq files and writes final sequence files (two for a paired-end run, one for a fragment run) after applying purity filtering. In other words, the fastq reads produced by CASAVA that have not passed purity filtering are not written to final sequence files.

   b) It also produces BWAConfigParams.txt which contains the configuration parameters for the alignment section.

5) Post Sequence actions:

   Code path: ipipeV2/wrappers/PostSequenceCommands.rb

   a) It uploads the sequence level results (lane yield, phasing, prephasing) to LIMS.

   b) It invokes "SequenceAnalyzer.jar"

   c) It invokes "Aligner.rb" to start the alignment.

6) Sequence-level analysis:

   Code path: ipipeV2/java/SequenceAnalyzer.jar

   It performs sequence level analysis to calculate the percentage of unique reads and distribution of adaptor reads. The results are reported in text and XML format, with two distribution charts namely DistributionOfN.png and AdaptorReadDistribution.png.

   Resource requirements: It is submitted to the cluster with 1 core, 8G memory reservation. It writes many files to temp directory /space1/tmp (approximately 3000 or so). The final output file(s) are small.

7) Alignment:

Code path:	ipipeV2/bin/Aligner.rb
	ipipeV2/lib/AlignerHelper.rb
	ipipeV2/java/MateInfoFixer.jar
	ipipeV2/java/BAMAnalyzer.jar
	ipipeV2/blackbox_wrappers/CaptureStats.rb
	[Picard] MarkDuplicates.jar

Configuration file : BWAConfigParams.txt located in the sequence event directory.

a) Aligner reads the configuration file, finds the sequence files in the sequence event directory and unzips them if required.

b) It applies BWA aln command to produce suffix array index (.sai) files, one sai file per sequence file.

c) It applies BWA sampe (for paired end runs) or samse (for fragment runs) to produce a SAM file.

d) It executes a custom Java application "MateInfoFixer.jar" to fix mate information and fix CIGAR and mapping quality for unmapped reads. The output is a coordinate sorted bam file.

e) It uses Picard's MarkDuplicates.jar to mark the duplicates on the BAM.

f) BAMAnalyzer.jar calculates alignment and other statistics.

g) If the configuration file specified a capture chip design, CaptureStats.rb is executed to calculate capture stats. Capture results are uploaded to LIMS.

Important changes:

(1) The sequence files produced by CASAVA 1.8 have sanger base quality format (PHRED+33) instead of Illumina format (PHRED+64) as seen in sequence files of CASAVA 1.7. For backward compatibility with previous CASAVA versions, Aligner.rb looks for a parameter "BASE_QUAL_FORMAT" in the configuration file. If this parameter is missing, it **assumes** PHRED+64 and invokes BWA accordingly.

(2) Reduction in I/O operations and reduction in run time: To achieve these goals, I developed a custom application "MateInfoFixer.jar" based on Picard APIs. Through a single pass over the SAM file, it fixes mate information, fixes CIGAR for unmapped reads and applies coordinate sorting to produce a BAM file. As a result, I/O operations and execution time to convert a SAM to a coordinate sorted BAM are reduced. [More details to follow later]

(3) File name changes:

Final BAM files are named as per the convention: Flowcell-Lane-Barcode_marked.bam.

e.g. 64J0NAAXX-1-ID04_marked.bam

Sequence files are named using the convention: Flowcell-Lane-Barcode_Read_sequence.txt,

e.g.
64J0NAAXX-1-ID04_1_sequence.txt (Sequence file for read 1)
64J0NAAXX-1-ID04_2_sequence.txt (Sequence file for read 2)

Resource requirements:

(1) BWA aln command locks down 1 node and uses 8 cores on a node. For paired end runs, 2 nodes are locked concurrently.

(2) BWA sampe/samse commands lock down one node.

(3) All the subsequent commands consume one node. They use upto 22G memory. The node reservation is done for 28G/8 cores (or maximum number of cores).

8) Post alignment step:

Code path: ipipeV2/wrappers/PostAlignmentProcess.rb

a) It uploads alignment results to LIMS.

b) It emails analysis results.

c) It zips the purity filtered sequence files.

d) It deletes intermediate bam, sam, sai and other files.

e) Anything else that needs to run (e.g. SNP calling etc) can be added here.

2 Pipeline Architecture

This section describes various components of the pipeline in more detail.

2.1 LIMS Integration

The pipeline obtains the "input" for the analysis from LIMS and then publishes the results to LIMS. In this section, the interaction between the pipeline and LIMS is discussed.

1) At the start, the pipeline contacts LIMS to obtain information about the analysis parameters e.g. reference path, chip design, sample name, library name etc.

2) On receiving the information from LIMS, the pipeline writes a configuration file named FCDefinition.xml. This file lies in the "BaseCalls" directory of the flowcell.

3) There is no other interaction with LIMS to obtain analysis parameters. All the subsequent interactions are to publish the results.

4) On completing sequence file generation, the pipeline uploads the sequence generation results. For read 1, the following parameters are uploaded:

   PERCENT_PHASING
   PERCENT_PREPHASING
   PERCENT_PF_READS
   FIRST_CYCLE_INT_PF
   PERCENT_INTENSITY_AFTER_20_CYCLES_PF
   PIPELINE_VERSION casava1.8
   LANE_YIELD_MBASES
   RAW_READS
   PF_READS
   PERCENT_PERFECT_INDEX
   PERCENT_1MISMATCH_INDEX
   PERCENT_Q30_BASES MEAN_QUAL_SCORE

   For read 2, the following parameters are uploaded:

   PERCENT_PHASING
   PERCENT_PREPHASING
   PERCENT_PF_READS
   FIRST_CYCLE_INT_PF
   PERCENT_INTENSITY_AFTER_20_CYCLES_PF
   PIPELINE_VERSION casava1.8

5) On completing sequence analysis (calculating unique percentage), the pipeline uploads uniqueness results. The parameter name is UNIQUE_PERCENT.

6) On completing capture stats calculation, the pipeline uploads capture results to LIMS.

7) The final upload is analysis results (alignment percentage and alignment error). At this time, the pipeline also uploads the directory path where the BAM / sequence files live, and the BAM path. The analysis is considered complete after this upload step. Following parameters are uploaded for read 1:

   PERCENT_ALIGN_PF
   PERCENT_ERROR_RATE_PF
   PIPELINE_VERSION
   RESULTS_PATH
   REFERENCE_PATH
   BAM_PATH

   For read 2, the following parameters are uploaded:

   PERCENT_ALIGN_PF
   PERCENT_ERROR_RATE_PF
   PIPELINE_VERSION

LIMS team has provided several perl scripts as LIMS APIs. These live under ipipeV2/lims_api directory:

a) getFlowcellInfo.pl – retrieves the names of all the lane barcodes used in the flowcell.

b) getAnalysisPreData.pl – Given a lane barcode, returns the number of cycles, reference path, chip design, sample name and library names.

c) setIlluminaCaptureResults.pl – Sets the capture stats results for a given lane barcode.

d) setIlluminaLaneStatus.pl – Sets the results for three states namely, unique percentage calculation (UNIQUE_PERCENT_FINISHED), sequence generation metrics (SEQUENCE_FINISHED) and alignment metrics (ALIGNMENT_FINISHED).

e) setFlowcellAnalysisStartDate.pl – Sets the analysis start date for the flowcell. The date is not supplied to this script. LIMS uses current timestamp when this script is called and sets the analysis start date accordingly.

The pipeline uses the following wrappers around the above listed perl scripts:

a) lims_api/FlowcellPlanDownloader.jar – This utility is the pipeline replacement for getFlowcellInfo.pl and getAnalysisPreData.pl. It directly hits jsp pages on LIMS to retrieve the information necessary to analyze a flowcell and writes FCDefinition.xml in the specified directory. In the pipeline, ipipeV2/bin/PreProcessor.rb uses this jar file.

b) lims_api/AnalysisResultUploader.jar – This tool has been developed to upload various analysis results (capture results, analysis results, sequence level results and uniqueness results) to LIMS.

It is intended as a replacement for setIlluminaLaneStatus.pl and setIlluminaCaptureResults.pl. However, this is NOT currently integrated in the pipeline. This should be a future addition.

c) wrappers/ResultUploader.rb – This pipeline script uses setIlluminaLaneStatus.pl to upload the sequence generation metrics and alignment metrics. (Eventually this script should be modified to use lims_api/AnalysisResultUploader.jar).

d) blackbox_wrappers/CaptureStats.rb – This pipeline script calculates capture stats and directly uploads the results to LIMS. It internally uses setIlluminaCaptureResults.pl. (However, this can also be modified to use lims_api/AnalysisResultUploader.jar).

Motivation behind developing alternative set of programs to interact with LIMS:

Each perl script developed by LIMS team is hardcoded with the IP address and the port number of the LIMS server. When there are several collaborators, in order to make the pipeline work against multiple LIMS systems (e.g. HGSC LIMS and WGL LIMS), we would need to create a duplicate copy of all the code, replace the server IP / port number in the LIMS scripts and have the pipeline use those scripts. This is clearly not a scalable approach if the number of different LIMS systems increase.

To work around this issue, each Java program uses a configuration file located at ipipeV2/lims_api/limsClient/LimsInfo.config to determine the LIMS server IP address and port number. Thus, these tools can work with several different LIMS systems **as long as the names of the LIMS JSP pages remain unchanged**.

Typical content of LimsInfo.config file

```
HGSC=http://lims-1.hgsc.bcm.tmc.edu/ngenlims
WGL=http://gen2.hgsc.bcm.tmc.edu:8683/ngenlims_wgl
```

(At present, we have two instances of LIMS, hence we have two entries in this file). On modifying any entry in this file, please run ipipeV2/lims_api/GenerateJars.sh to regenerate the Java executables.

Q: How does the pipeline know which instance of LIMS to use ?

The pipeline looks at the config_parameters.yml file (section Pipeline parameters). It obtains the name of the LIMS system from the section “lims/databaseName”. This value must be correctly set to enable the pipeline to use the correct LIMS instance.

2.2 Detecting when a flowcell is available for analysis

Since the pipeline automatically schedules the analysis jobs for flowcells, it needs to understand when a flowcell is ready for analysis. This section describes this process in more detail. Currently, we have two types of sequencers namely, HiSeq sequencers with RTA version 1.12 and GAIIx sequencers with RTA version 1.9.

The algorithm for Hiseq sequencers running RTA 1.12:

1) When sequencing completes and RTA finishes writing all the files over to the flowcell directory on the cluster, the RTA writes a file RTAComplete.txt.

2) At specified time intervals, the cron job navigates through all the sequencer directories and builds the list of new child directories (i.e. new flowcells for which analysis is not yet started).

3) If the pipeline finds a new flowcell, it checks for the presence of RTAComplete.txt file. If this file is not found, this flowcell is skipped.

4) However, if the pipeline finds the RTAComplete.txt file, it writes another empty file ".rsync_finished" in the flowcell directory.

5) In the next iteration of the cron job, the pipeline finds this flowcell with file ".rsync_finished". This time, it appends the name of the flowcell to a file named "done_list.txt", to indicate that the analysis has already been started for the flowcell. Moreover, it also starts the analysis by executing the command

   ```
   ruby ipipeV2/bin/PreProcessor.rb fcname=FCName action=all
   ```

   where FCName is the complete flowcell directory name (run directory name)

Motivation behind using two different marker files (RTAComplete.txt and .rsync_finished):

Initially, the Hiseq sequencers were writing to intermediate dump volumes. The directories from dump volumes were periodically copied over to the cluster. In such cases, the assumption that marker file RTAComplete.txt is the last file copied to the cluster did not hold true. Hence, it was necessary to have another file (".rsync_finished") to let the pipeline know that RTA has completed.

With the current setup, sequencers copy data directly to the cluster, so the file RTAComplete.txt is indeed the last file written to the cluster. However, under certain circumstances, the sequencing team might have to abort a run prematurely, e.g. aborting read 2 of a paired end run for some fluidics issue on the sequencer. In these cases, the file RTAComplete.txt will not be written to the cluster. However, by adding the file ".rsync_finished", the pipeline will still be able to analyze the partial run and produce the data.

The algorithm for GAIIx sequencers running RTA version 1.9

1) For RTA version 1.9, there isn't a clean way that indicates RTA finished copying the data.

2) Almost all the flowcells that are sequenced are paired-end. Hence, the algorithm to detect whether a flowcell is available ***assumes*** that every flowcell is paired-end.

3) The cron job finds the new flowcell directories to analyze just like the previous case.

4) Instead of looking for RTAComplete.txt file, it looks for the presence of the following three files:

   a) Basecalling_Netcopy_complete_READ1.txt – written after read1 data is copied

b) Basecalling_Netcopy_complete_READ2.txt – written after read2 data is copied
c) Basecalling_Netcopy_complete.txt – written after read1 data is copied and updated again after read2 data is copied

5) If the pipeline finds these three files **and** the most recent modification time of the marker file Basecalling_Netcopy_complete.txt is more than 1 hour, it adds the name of the flowcell to "done_list.txt" discussed earlier and starts the analysis on the flowcell by executing the command

   ```
   ruby ipipeV2/bin/PreProcessor.rb fcname=FCName action=all
   ```

It is clear that there is not one file that is written at the end to indicate end of copy for GAIIx (RTA 1.9). Hence, the pipeline assumes that every flowcell has to be paired-end. Therefore, it waits for the three files described earlier. As per Illumina tech support, presence of Basecalling_Netcopy_complete.txt does not guarantee that all the data is copied over, but merely implies that data transfer is nearly complete. Hence, the pipeline adds one hour delay to give time for any remaining files to be copied over.

Just like HiSeqs, if for some reason, the sequencing team runs a fragment flowcell, or aborts read 2, a file ".rsync_finished" can be added to the flowcell directory. This will enable the pipeline to process this flowcell.

2.3 Configuration files in the pipeline

This section discusses various configuration files used by the pipeline during the course of analysis.

2.3.1 Flowcell definition file

After a flowcell has been picked up for analysis, the pipeline contacts LIMS and gets the "input" data for analysis. It writes all this "input" to a file named "FCDefinition.xml" in the BaseCalls directory of the flowcell. This section presents a typical flowcell definition XML file.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<FCInfo Name="64J0NAAXX" NumCycles="101+7" Type="paired">
<LaneBarcodeList>
<LaneBarcode Name="1-ID04"/>
<LaneBarcode Name="2-ID04"/>
<LaneBarcode Name="3-ID06"/>
<LaneBarcode Name="4-ID06"/>
<LaneBarcode Name="5"/>
<LaneBarcode Name="6-ID12"/>
<LaneBarcode Name="7"/>
<LaneBarcode Name="8"/>
</LaneBarcodeList>
<LaneBarcodeInfo>
<LaneBarcode ChipDesign="/users/p-wgl/vcrome2.1_hg19" ID="1-ID04" Library="WGLTEST.NA12878-5-
1_2E" ReferencePath="/stornext/snfswgl/next-gen/Illumina/bwa_ref
erences/h/hg19/original/hg19.fa" Sample="TG.NA12878-5"/>
```

```
<LaneBarcode ChipDesign=”/users/p-wgl/vcrome2.1_hg19” ID=”2-ID04” Library=”WGLTEST.NA12878-5-
1_2E” ReferencePath=”/stornext/snfswgl/next-gen/Illumina/bwa_ref
erences/h/hg19/original/hg19.fa” Sample=”TG.NA12878-5”/>
<LaneBarcode ChipDesign=”/users/p-wgl/vcrome2.1_hg19” ID=”3-ID06” Library=”WGLTEST.NA12891-5-
1_2E” ReferencePath=”/stornext/snfswgl/next-gen/Illumina/bwa_ref
erences/h/hg19/original/hg19.fa” Sample=”TG.NA12891-5”/>
<LaneBarcode ChipDesign=”/users/p-wgl/vcrome2.1_hg19” ID=”4-ID06” Library=”WGLTEST.NA12891-5-
1_2E” ReferencePath=”/stornext/snfswgl/next-gen/Illumina/bwa_ref
erences/h/hg19/original/hg19.fa” Sample=”TG.NA12891-5”/>
<LaneBarcode ChipDesign=”/users/p-wgl/vcrome2.1_hg19” ID=”5” Library=”IWX_WGLTEST.HS-1011-5-
1_1pA” ReferencePath=”/stornext/snfswgl/next-gen/Illumina/bwa_ref
erences/h/hg19/original/hg19.fa” Sample=”WGL.HS-1011-5”/>
<LaneBarcode ChipDesign=”/users/p-wgl/vcrome2.1_hg19” ID=”6-ID12” Library=”WGLTEST.HS-1011-5-
1_2E” ReferencePath=”/stornext/snfswgl/next-gen/Illumina/bwa_ref
erences/h/hg19/original/hg19.fa” Sample=”WGL.HS-1011-5”/>
<LaneBarcode ChipDesign=”/users/p-wgl/vcrome2.1_hg19” ID=”7” Library=”IWX_WGLTEST.NA12892-5-
1_1pA” ReferencePath=”/stornext/snfswgl/next-gen/Illumina/bwa_ref
erences/h/hg19/original/hg19.fa” Sample=”TG.NA12892-5”/>
<LaneBarcode ChipDesign=”None” ID=”8” Library=”IQC_PhiX_000pA”
ReferencePath=”/stornext/snfswgl/next-gen/Illumina/bwa_references/p/phix/original/PhiX_plus_SN
P.fa”/>
</LaneBarcodeInfo>
</FCInfo>
```

Note:

1) LaneBarcodeList contains the list of all the barcode names used in this flowcell.
2) Each XML element LaneBarcode contains the relevant information to analyze one sequencing event of that flowcell.
3) The field number of cycles (NumCycles) is not used for analysis to protect the pipeline against user errors. The pipeline uses as many cycles available in the file system to perform the analysis.
4) If read 2 for a paired-end flowcell was aborted due to some error, the flowcell can be analyzed in the “fragment” mode by editing this file and replacing the text “paired” with “fragment”, and then by re-generating the result directories.

2.3.2 Configuration file BWAConfigParams.txt

The pipeline writes this file in the sequencing event directory of every sequencing event. It is used by the alignment section of the pipeline to feed the parameters to BWA.

A typical configuration file looks like:

```
REFERENCE_PATH=/stornext/snfswgl/next-gen/Illumina/bwa_references/h/hg19/original/hg19.fa
LIBRARY_NAME=WGLTEST.NA12878-5-1_2E
SAMPLE_NAME=TG.NA12878-5
FILTER_PHIX=false
CHIP_DESIGN=/users/p-wgl/vcrome2.1_hg19
RG_PU_FIELD=USI-EAS034_20110907_64J0NAAXX-1-ID04
```

```
FC_BARCODE=64J0NAAXX-1-ID04
BASE_QUAL_FORMAT=PHRED+33
SCHEDULER_QUEUE=wgl
```

Things to know:

1) Reference path is the full path to the reference fasta file.
2) Library and sample names – self explanatory ☺
3) FILTER_PHIX: this value is ALWAYS false nowadays. Till February 2011, we were aligning all sequence events with a hybrid reference that contained phix reference in it. The flag FILTER_PHIX=true instructed a pipeline program to remove the reads aligning to phix. This concept was introduced to account for 2% phix spike in the reads, and was aborted from February 2011. But, the parameter still lives on for legacy reasons.
4) RG_PU_FIELD: The name of PU (Platform unit) field in RG tag of the BAM header.
5) FC_BARCODE: Unique identifier for the sequencing event.
6) BASE_QUAL_FORMAT: With CASAVA 1.8, sequence files have qualities encoded in PHRED+33 format. With CASAVA 1.7, it was PHRED+64. If this field is not specified, Aligner assumes the default format PHRED+64.
7) SCHEDULER_QUEUE: the name of the queue to use on the cluster.

2.3.3 Pipeline configuration parameters

To produce sequence files and perform alignment on them, the pipeline needs to know location of various dependencies, properties of scheduling queues etc. These are described in another config file named "config_params.yml". Please note that unlike previous two configuration files, this is a global configuration file, whose contents does not change for different flowcells.

This file lives under ipipeV2/config directory in the pipeline. Typical contents look like:

```
picard:
  path: "/stornext/snfs5/next-gen/Illumina/ipipeV2/picard/current"
  stringency: "VALIDATION_STRINGENCY=LENIENT"
  tempDir: "TMP_DIR=/space1/tmp"
  maxRecordsInRAM: "MAX_RECORDS_IN_RAM=3000000"
  maxHeapSize: "-Xmx22G"
bwa:
  path: "/stornext/snfs5/next-gen/niravs_scratch/code/bwa_test/bwa_0_5_9/bwa-0.5.9/bwa"
scheduler:
  queue:
    normal:
      maxCores: 8
      maxMemory: 28000
    high:
      maxCores: 8
      maxMemory: 28000
    hptest:
```

```
    maxCores: 16
    maxMemory: 28000
casava:
  bclToFastqPath: “/stornext/snfs5/next-gen/Illumina/GAPipeline/CASAVA1_8/CASAVA1_8-
Install/bin/configureBclToFastq.pl”
captureStats:
  codeDirectory: “/stornext/snfs5/next-gen/software/hgsc/capture_stats”
  captureCode: “CaptureStatsBAM5”
sequencers:
  rootDir: “/stornext/snfs0/next-gen/Illumina/Instruments”
lims:
  databaseName: “HGSC”
```

1) The block “Picard” represents the configuration parameters passed on to all Picard programs and programs developed using Picard APIs. It also contains the location of Picard installation for the pipeline.

2) The block “bwa” contains the bwa installation directory.

3) The block scheduler describes each available queue. (Note that HGSC config file will not have information about wgl queue and vice versa. It also contains the maximum number of cores and the maximum memory that could be requested by various pipeline tools. These are used by Aligner.rb to determine the maximum resources for various steps in the alignment process.

4) The block CASAVA contains the installation directory of CASAVA 1.8.

5) The block “captureStats” contains the directory and class name of the program to run capture stats.

6) **The block “sequencers” represents the root directory that the pipeline searches to look for new flowcells. Currently, it is set to /stornext/snfs0/next-gen/Illumina/Instruments. The value of this parameter must be suitably changed if the location where sequencers write the data is changed. Current pipeline version supports only one root directory to search for the flowcells.**

7) The “lims” block describes the LIMS database to use. More details are in LIMS integration section.

2.4 Result directory structure

The directory structure has substantial changes between earlier CASAVA versions and CASAVA 1.8. This section describes those changes.

1) Within flowcell's directory, the pipeline creates a directory named "Results" that becomes the parent directory for all CASAVA 1.8 output.
2) Under the "Results" directory, CASAVA creates a directory named "Basecall_Stats_FCName". Many CASAVA generated QC metrics are reported there. The file "Demultiplex_Stats.htm" is of particularly interest because it contains information such as the yield of the run. Several values from this file are parsed and uploaded to LIMS.

3) Under the "Results/Project_Flowcellname" directory, directories equivalent to CASAVA 1.7's GERALD directories are created. Each directory contains the sequence files for one sequencing event (lane barcode). These directories are named "Sample_Flowcell-Lane-Barcode". For example, if flowcell 64J0NAAXX had the following lane barcodes:

   1-ID04, 2-ID04, 3-ID06, 4-ID06, 5, 6-ID12, 7 and 8,

   the directories that are created would be named:

   Sample_64J0NAAXX-1-ID04 – contains everything related to 1-ID04
   Sample_64J0NAAXX-2-ID04
   Sample_64J0NAAXX-3-ID06
   Sample_64J0NAAXX-4-ID06
   Sample_64J0NAAXX-5 - Sequencing event on lane 5 without barcodes
   Sample_64J0NAAXX-6-ID12
   Sample_64J0NAAXX-7
   Sample_64J0NAAXX-8

Now we discuss the contents of the directory of a single sequencing event. (e.g. Sample_64J0NAAXX-1-ID04). Typical contents would be something like:

Directory casava_fastq	Intermediate unfiltered sequence files produced by CASAVA 1.8 are contained in this directory.
64J0NAAXX-1-ID04_1_sequence.txt.bz2 64J0NAAXX-1-ID04_2_sequence.txt.bz2	Final purity filtered files produced by the pipeline. They get zipped after final bam is created. **Please note that these files have quality format Phred+33**.
ReadsPassingPurityFilter.metrics	Contains the number of reads that passed purity filtering. This number represents the percentage of reads in final sequence files against the number of raw reads in sequence files generated by CASAVA
BWAConfigParams.txt	Configuration file used for alignment
64J0NAAXX-1-ID04_uniqueness.txt 64J0NAAXX-1-ID04_uniqueness.xml	Output of SequenceAnalyzer in text and XML formats.
DistributionOfN.png AdaptorReadDistribution.png	Plots generated by SequenceAnalyzer

BWA_Map_Stats.txt BAMAnalysisInfo.xml	Output of BAMAnalyzer
BaseQualPerPosition.png FR_InsertSizeDist.png FR_InsertSizeDist.csv AvgQualScoreDist.csv	Plots and distribution charts generated by BAMAnalyzer
capture_stats	Directory containing all the files generated by capture stats program
*.o	Files containing STDOUT from various programs
*.e	Files containing STDERR from various programs
64J0NAAXX-1-ID04_marked.bam	Final bam file

2.5 Pipeline log files, result files and error handling

The pipeline creates various log files and result files. These are described in greater detail in this section.

2.5.1 Logging in the pipeline

The pipeline writes the STDOUT and STDERR of all operations in various directories in the flowcell. This section describes the location and the content of some important log files. Each log file is named with extension ".o" or ".e". "*.o" represents STDOUT and "*.e" represents STDERR.

1) FC/Data/Intensities/BaseCalls directory: contains two log files namely "fcPlanDownloader.o" and "fcPlanDownloader.e". These files contain the STDOUT and STDERR log while interacting with LIMS to obtain the flowcell plan. Specifically, the pipeline logs every request made to LIMS to obtain flowcell information, response from LIMS and the round-trip response time.

2) FC/Results directory: It contains two sets of log files. The files named *_BclToFastQ.o and *_BclToFastq.e contain the STDOUT and STDERR of the make command to generate CASAVA fastq files. Another set of log files FC-LaneBarcodes.o and FC-LaneBarcodes.e contain the commands and MOAB job numbers to produce final sequence files and the post sequence commands.

3) Sequencing event directory : This directory contains all the necessary information for one sequencing event. Its location is (FC/Result/Project_FC/Sample_FCBarcode). For example, the path for the Flowcell barcode C0AN2ACXX-6 would be the following:

   111123_SN896_0150_BC0AN2ACXX/Results/Project_111123_SN896_0150_BC0AN2ACXX/Sample_C0AN2ACXX-6

   This directory contains several log files that contain the STDOUT and STDERR information for every command applied in the pipeline. A file named *_processBam.o is of specific interest because this file contains all the information about the execution environment (MOAB node name, PBS Job ID, free disk space on /space1/tmp at the start of BAM finishing step, working directory and execution time of each command in the BAM finishing steps (i.e. MateInfoFixer / FixMateInformation, MarkDuplicates and BAMAnalyzer). Additionally, each of these commands also produces its own set of log files which are appropriately named. These files contain more logging information about the specific command.

2.5.2 Files containing various metrics / results

In this section, we discuss various result files generated by the pipeline at various stages.

1) Demultiplex_Stats.htm:
   Location: FC/Results/Basecall_Stats_*
   Description: This file is produced by CASAVA software after the reads are demultiplexed. It contains information such as lane yield, mean quality score and some other parameters uploaded to LIMS at "SEQUENCE_FINISHED" stage.
   Parser Code: ipipeV2/wrappers/ResultUploader.rb

2) BustardSummary.xml:
   Location: FC/Results/Basecall_Stats_*
   Description: This file is also produced by CASAVA after demultiplexing the reads. It contains information such as phasing and pre-phasing. Relevant information from this file is parsed and uploaded to LIMS at "SEQUENCE_FINISHED" stage.
   Parser Code: ipipeV2/wrappers/ResultUploader.rb

3) FC-Barcode_uniqueness.txt / FC-Barcode_uniqueness.xml:
   Location: FC/Results/Project_*/Sample_FC-Barcode
   Description: These file contain the uniqueness results (i.e. output of Sequenceanalyzer.jar in text and XML formats. The uniqueness percentage is parsed and uploaded to LIMS after SequenceAnalyzer.jar finished.
   Parser Code: ipipeV2/wrappers/SequenceAnalyzerWrapper.rb

4) ReadsPassingPurityFilter.metrics:
   Location: FC/Results/Project_*/Sample_FC-Barcode
   Description: This file contains the number and percentage of reads that passed the purity filtering and were added to final sequence files.
   The information from this file is not currently uploaded to LIMS.

5) BWA_Map_Stats.txt and BAMAnalysisInfo.xml:
   Location: FC/Results/Project_*/Sample_FC-Barcode
   Description: These files contain the output of BAMAnalyzer in text and XML formats.
   Parser Code: ipipeV2/wrappers/ResultUploader.rb

In addition, within the Sample_FC-Barcode directory, there are several files that contain the plot and the distribution of various metrics. Those are described below:

AdaptorReadDistribution.png	Distribution of adaptor reads in the sequences Produced by SequenceAnalyzer.jar
DistributionOfN.png	Distrubution of N bases in the sequences Produced by SequenceAnalyzer.jar
BaseQualPerPosition.png	Distribution of base quality score vs base position Produced by BAMAnalyzer.jar
AvgQualScoreDist.csv	Distribution of base quality score vs base position in CSV format, Produced by BAMAnalyzer.jar
*_ InsertSizeDist.png and _InsertSizeDist.csv	Distribution of insert size for a specific orientation of read pairs in image and CSV formats. There can be upto three orientations: a)FR (FR_InsertSize.*) b) RF (RF_InsertSize.*) c) Tandem (Tandem_InsertSize.*) For meaning of these orientations, please refer to the README file for BAMAnalyzer.
metrics.foo	File produced by Picard's MarkDuplicate.jar. It is not read by any other program in the pipeline.

2.5.3 Communicating errors in the pipeline

This section elaborates the functionality of providing information about errors in the pipeline. Errors can occur in the pipeline due to various reasons. Some of the most frequent reasons are:

1) Unable to get analysis information from LIMS (i.e. failure while building FCDefinition.xml).

2) Problems during the alignment phase. Most of these errors are because of bad sectors / failures of temporary drive /space1/tmp on the nodes.

3) Errors in the pipeline code while preparing a flowcell for analysis.

4) Failures in CASAVA (Illumina's software).

Primary way of communicating errors to the users is via emails. The pipeline sends out error emails on encountering unrecoverable situations. It uses a helper class in the ipipeV2/lib directory named "ErrorHandler.rb". This class encapsulates most of the information about error messages. It uses the list of email addresses listed under "EMAIL_ERRORS" in file ipipeV2/config/email_recepients.txt.

During the alignment phase and beyond, if any of the programs fail, they exit with non-zero status. The pipeline monitors the exit code of these programs, and sends out emails whenever it finds that a program failed (i.e. returned with non-zero error code). More detail about the error can be found in the associated STDOUT / STDERR files of the program that aborted.

In the current implementation, the pipeline cannot send email messages when Illumina's "make" program aborts. If subsequent jobs remain pending for several hours, this might be a symptom to check if Illumina's software crashed.

2.6 Reduction in execution time in the BAM finishing step

By exploiting some characteristics of BWA version 0.5.9, it was possible to reduce the execution time of the BAM finishing step. This is described in detail in this section.

The pipeline has been optimized to work with the current version of BWA (0.5.9). This version of BWA produces an unsorted SAM file, where both the reads of each pair are placed next to each other.

This property of the SAM file has been used to reduce execution time and I/O operations. To convert a SAM to a final BAM, originally we performed the following operations:

1) Sort the SAM to produce a BAM file that is "coordinate" sorted, and fix flags for the mate. This was originally accomplished by Picard's FixMateInformation.jar. FixMateInformation.jar sorts the SAM based on queryname order, fixes flags of the mates and then re-sorts the BAM in coordinate order.

2) Mark duplicates on the BAM file. This is done by Picard's MarkDuplicates.jar.

3) Fix CIGAR and mapping quality for unmapped reads. Whenever a read aligns over a boundary of two consecutive chromosomes, BWA marks the read as unmapped, but does not remove the CIGAR or mapping quality. As a result, several downstream tools can throw validation errors. FixCIGAR.jar corrects this behavior.

The drawback of this initial approach is that three passes and two sort operations over a BAM are required to fix the mate information and fix CIGAR. Instead, with the optimized version, the following happens:

1) The custom application MateInfoFixer.jar performs SAM to coordinate sorted BAM conversion through one pass of the SAM file and one sort operation. Moreover, while fixing mate information, it also fixes the CIGAR for unmapped reads. This application exploits the BWA property that mates of a read pair are located next to each other in the SAM file, thus, eliminating the need to sort the SAM based on querynames.

2) The pipeline now executes Picard's MarkDuplicates.jar and BAMAnalyzer as in the previous case.

3) Finally, it executes BAMAnalyzer.jar to produce alignment and other metrics.

Since this optimization is based on the SAM file format of BWA version 0.5.9, it must be validated with future versions of BWA. If the current assumption (mates of a read are consecutive reads in the SAM file), does not remain true for future versions, the pipeline should be modified as follows:

a) Remove MateInfoFixer.jar.

b) Run Picard's FixMateInformation.jar after BWA SAM file is created.

c) Run Picard's MarkDuplicates.jar next.

d) Run FixCIGAR.

e) Finally run BAMAnalyzer.

Note: This optimization has only been applied to paired-end runs because there's no concept of mate for fragment runs. For fragment runs, the pipeline currently executes the sequence of following commands:

a) Picard's SortSam.jar to convert a SAM file to a coordinate sorted BAM file.

b) Picard's MarkDuplicates.jar to mark duplicates.

c) FixCIGAR.jar to fix CIGAR for unmapped reads.

d) BAMAnalyzer.jar to calculate various alignment and insert size statistics.

As a result, nothing needs to be changed with BWA version upgrades for the fragment runs.

2.7 Tools in ipipeV2/java directory

Most of the Java tools used in the pipeline live under ipipeV2/java. The only exceptions are capture stats code and java programs to interact with LIMS.

1) AttachmentMailer.jar: emails files as attachments to intended recipients. It requires a config file ipipeV2/java/tools/EmailHost.config. This config file contains the SMTP server name of the BCM server. To use this program with a different email server system, please change the value in this file and rebuild the program. Typical usage :

   ```
   java –jar /PathToipipeV2/ipipeV2/java/AttachmentMailer.jar sender=sol-pipe@bcm.edu
   dest=myemail@bcm.edu attach=FileToAttach
   ```

2) BAMAnalyzer.jar : Calculates alignment, insert size and pair-wise metrics on a single BAM or a set of BAMs.

3) BAMHeaderFixer.jar: Can be used to fix several header fields in the BAM file such as sample name, library name and other fields in the RG tag of the bam file.

4) CIGARFixer.jar: When a read aligns between two adjacent chromosomes, BWA sets this read as unmapped but keeps CIGAR string and mapping quality unchanged. As a result, Picard tools throw validation errors. Thus, this tool resets mapping quality to zero and resets CIGAR to empty string.

5) MateInfoFixer.jar: This is a custom tool to replicate the functionality provided by Picard's FixMateInformation.jar. This tool has been tested to work with SAM files produced by BWA version 0.5.9. It processes a SAM file to generate a coordinate sorted BAM file, where flags of mates are correctly fixed. The main difference between this tool and Picard's FixMateInformation is the reduction in I/O operations and execution time. Picard's FixMateInformation requires 2 passes and 2 sort operations to create a coordinate sorted BAM. However, MateInfoFixer performs the same task using 1 pass through the BAM file and 1 sort operation.

6) BarcodeEditDist.jar: can be used to determine the edit-distance between a set of barcodes. Edit distance is defined as the number of substitutions, additions or deletions to be applied to one barcode to convert it to the other barcode. It requires an input file having two comma separated fields namely, barcode string name and the actual barcode sequence. For example, typical content of this file could be something like:

```
ID01,ATCACG
ID02,CGATGT
ID03,TTAGGC
ID04,TGACCA
ID05,ACAGTG
ID06,GCCAAT
ID07,CAGATC
```

7) SequenceAnalyzer.jar: reads unzipped fastq sequence files and calculates the percentage of unique reads (using K-mer approach, where K-mer size is 30), calculates the percentage of adaptor reads and finds the reads with large number of "N"s in them. The output is a text / xml file containing the results. In addition, png files containing distribution charts and csv files containing the distribution data are generated.

8) FastqTrimmer.jar: Due to sequencing or other errors, the fastq sequence files may contain numerous "N" characters (undetermined bases), or have poor quality. In such cases, alignment would be very poor. The alignment can be improved by truncating regions from the sequence files and remapping those sequence files. FastqTrimmer.jar can be used to truncate the bad regions from sequence files.

9) PEToFragConvertor.jar: **Strictly experimental tool**. This tool reads a BAM file and creates a new BAM file with read 2 (second reads in the pair) removed. It also removes all the corresponding paired flags from the first reads and removes all duplicate flags from the bam. (This tool is not being used in daily pipeline activities. It was developed to help Christian Buhay with some additional analysis).

10) FastqDecontaminator.jar: Another experimental tool. It filters the reads from the sequence files and writes the "passed" reads into another file. A read whose index (barcode) tag completely matches the given sequence tag, or a read whose index tag varies in at most one position from

the given sequence tag is considered “passed”. This is not of much use in the CASAVA 1.8 pipeline. However, it is still kept as part of the pipeline tool for any future experimental use.

11) mail.jar: Supporting java library file for the AttachmentMailer.jar. Do not delete this file, because the AttachmentMailer.jar will not work without it.

Question: What is the fastest way to build all the jar files in the java directory ?

Please run the ruby script buildall.rb in the java directory. It rebuilds all the jar utilities by using supporting bash scripts named “Generate*Jar.sh”.

SequenceAnalyzer and BAMAnalyzer are the two most important tools. They are described in more detail in the remainder of this section.

2.7.1 Metrics generated by SequenceAnalyzer

1) Uniqueness percentage: SequenceAnalyzer.jar calculates the number of unique reads in a sequencing event based on a k-mer approach, with k-mer size 30.

2) Screening for adaptor sequences: It also generates a distribution plot “AdaptorReadDistribution.png” that shows the distribution of the reads having adaptor sequence. Sequence “GATCGGAA” is used as the adaptor sequence. If some part of a read matches exactly with the adaptor sequence, this read is considered to have an adaptor. In practice, it was observed that usually < 0.5% of the reads had adaptors.

3) Screening for bad reads: If a read has 15% or more occurrence of N bases, it is reported as a bad read.

2.7.2 Metrics generated by BAMAnalyzer

BAManalyzer generates several metrics related to alignment, yield, insert size distribution and information about various pairs. The meaning of these metrics is described in this section.

Alignment metrics : These are reported for each read type. There are 3 read types namely, fragment, read1 (first read in the pair) or read2 (second read in the pair).

1) Total reads : Number of reads of a given type.

2) Unmapped reads : Number of reads of a given type that don’t align.

3) Mapped reads : Number of reads of a given type that align.

4) Percentage mapped reads : Percentage of reads that align. It is calculated over total reads of the given type.

5) Percent mismatch : This value is reported as the alignment error percentage in LIMS. It is calculated as Percentage of mismatches (from CIGAR/MD tag) over total mapped bases.

Mathematically, Percent_Mismatch = totalMismatches / totalMappedBases * 100

6) Exact match reads : The number of reads that align without any variation from the reference. i.e., no mismatch in CIGAR or MD tags.

7) Percentage of exact match reads : Percentage of reads that map without any mismatch over the total number of mapped reads. (i.e. totalExactMatches / mappedReads * 100)

8) Duplicate reads : Number of duplicate reads as reported by Picard’s MarkDuplicate.jar.

9) Percentage duplicate reads : Percentage of duplicate reads over mapped reads. The formula is Percentage_Dup = (totalDuplicateReads / mappedReads * 100).

Yield metrics : These are also reported for each read type.

1) Total bases : The total number of bases per read type.

2) Valid bases : The total number of valid (i.e. non “N”) bases. It is calculated as TotalBases – SumOfNBases.

3) Effective bases : Total number of bases from non-duplicate reads after eliminating “N” bases. It is calculated as TotalBasesFromNonDuplicateReads – SumOfNBasesInNonDuplicateReads.

Insert size metrics : These are reported per each insert orientation. There are three types of insert size orientations. They are :

(5' --F-->   <--R-- 5') ---- FR orientation
(<--R-- 5'   5' --F-->) ---- RF orientation
(5' --F-->   5' --F-->) ---- Tandem orientation
(<--R-- 5'   <--R-- 5') ---- Tandem orientation

1) Total pairs : Total read pairs that have the given insert size orientation, map to the same chromosome and are not duplicates.

2) Percentage of pairs : represents the percentage of read pairs with a given insert size orientation, that map to the same chromosome and are non-duplicates. The percentage is calculated over all read pairs that map to the same chromosome. Mathematically,

   Percentage_pairs = sameChrMappedPairsForGivenInsertSizeOrientation / SameChrMappedPairs * 100

3) Mode insert size : Mode value of the insert size for the given orientation.

4) Median insert size : Median value of the insert size for the given orientation.

Pair metrics: are reported for pairs of reads.

1) Mapped pairs : Total number of pairs where both reads map, although not to the same chromosome.

2) Percentage of mapped pairs : is the ratio of the mapped pairs over the total number of pairs in the BAM file. i.e. mappedPairs / totalPairs * 100

3) Same chromosome pairs : Total number of pairs where both the reads map to the same chromosome.

4) Percentage of same chromosome pairs : is the ratio of the number of same chromosome pairs to the total number of pairs expressed as a percentage. i.e., sameChrPairs / totalPairs * 100.

5) Unmapped pairs : Total number of pairs where both the reads are unmapped.

6) Percentage of unmapped pairs : is the percentage of unmapped pairs over total pairs. Mathematically, it is totalUnmappedPairs / totalPairs * 100.

7) Read one mapped (Mapped first read) : Number of read pairs where only the first read in the pair aligns and the second read in the pair does not align.

8) Percenage of Read one mapped : is the percentage of number of read one mapped pairs over the total number of pairs. Mathematically, it is read1Mapped / totalPairs * 100.0.

9) Read two mapped (Mapped second read) : Number of read pairs where only the second read in the pair aligns and the first read in the pair does not align.

10) Percentage of Read two mapped : is the percentage of number of read two mapped pairs over the total number of pairs. Mathematically, it is read2Mapped / totalPairs * 100.0.

2.8 Tools in bulkcommands directory

Quite frequently, requests (tickets) come in to remap several sequencing events with another reference, or re-run capture stats on one or several sequencing events. There are two scripts in this directory that can help to automate bulk-submission of the jobs on the cluster.

1) ipipeV2/bulkcommands/bulkremap.rb: This tool can submit the jobs to remap several sequencing events. The only restriction is that all those sequencing events must be mapped with the same reference. The input to this script is a filename containing the list of result directories, one directory per line and the new reference path.

   Usage:
   ruby bulkremap.rb ResultPathFile NewReferencePath ChipDesign

   where ResultPathFile is one result directory per line that needs to be remapped with new reference.

ChipDesign is an optional parameter. If supplied, capture stats are calculated. Please provide complete path to the chip design.

It also updates BWAConfigParams.txt in those result directories before actually starting remap operation.

2) ipipeV2/bulkcommands/bulkcapstats.rb: This tool can submit jobs to re-run capture stats on one or several sequencing events. It accepts a filename containing the result directories (one entry per line) and the new chip design.

   Usage:

   ```
   ruby bulkcapstats.rb ResultPathFile newChipDesign
   ```

   (Don't add /users/p-illumina to the parameter newChipDesign. This script will automatically take care of it.).

2.9 Hooks in the pipeline to add new functionality

The pipeline has two main places where additional code can be added to extend the functionality of the pipeline.

1) ipipeV2/wrappers/PostSequenceCommands.rb: This script is run after final purity filtered fastq files are created. Currently, it is setup to start sequence analysis and the alignment. However, additional functionality can be introduced here. The new code must perform its own job submission to the cluster.

2) ipipeV2/wrappers/PostAlignmentProcess.rb: This is run at the end of the mapping pipeline. Current behavior is to upload the results to LIMS, email the analysis results, clean intermediate files in the result directory, zip sequence files and start SNP calling. Additional functionality can be added here. The new functions must perform their own job submission.

2.10 Barcode handling and sample swap recovery

This section describes in detail how configuration files are prepared for demultiplexing the reads based on their barcodes, and if a sample swap occurs, what could be the potential solution to recover from it.

2.10.1 Barcode handling

This section discusses the details about handling barcodes. We have two sets of barcodes namely, Illumina provided barcodes (with labels ID01 through ID27) and the other set labeled (IDMB1 through IDMB96). The IDMB barcodes are 9 bases in length and ID barcodes are 6 bases in length. The definition of barcode names and their base sequence is contained in a global configuration file ipipeV2/config/barcode_label.txt.

The pipeline performs the following steps to prepare the flowcell for demultiplexing:

1) Contact LIMS and build the flowcell plan.

2) Build a local copy of barcode definition file. This file is named “barcode_definition.txt”. Its location is the “BaseCalls” directory of the flowcell. This file is the subset of the master barcode_label.txt in the config directory, where, the barcode names and the sequence tags which are actually used in the flowcell are described. Thus, this file can be different for different flowcells, depending on the barcodes used in the flowcell. (But the master copy remains unchanged unless new barcodes have to be added to the system).

3) Using the flowcell plan and the barcode_definition.txt file, the pipeline writes a CSV file named “SampleSheet.csv”. This is written to the “BaseCalls” directory of the flowcell.

4) Now the flowcell has been prepared for demultiplexing, the pipeline invokes CASAVA’s demultiplexer to create result directories and start demultiplexing the reads.

5) The Illumina pipeline does not follow the CASAVA 1.8 approach of organizing all the reads belonging to the same sample in one result directory. On the other hand, while building SampleSheet.csv, it “fakes” the sample names in it by providing LIMS flowcell barcode as the sample name. Thus, we get one result directory per lane barcode irrespective whether these sequence events belong to the same sample or not. More details to follow later.

Mixing barcodes of different lengths in the same flowcell:

It is not recommended to mix different sets (barcodes of different lengths) in the same flowcell, because Illumina’s software requires all the barcodes to have the same length.

I have implemented a work-around to handle barcodes of different lengths in the pipeline. Based on my observation, when barcodes of different lengths are used in the same flowcell (same lane or different lanes does not matter), Illumina’s RTA **changes** the barcode sequences of shorter barcodes by appending extra bases.

For example, If barcode ID01 (sequence ATCACG) and IDMB1 (GACTTCGTA) are used in the same flowcell, the actual sequence for ID01 would be changed to ATCACGCTC. Illumina’s software added 3 additional bases to it. From my observation, I found that suffix sequence to be “CTC”. (Accurate as of August 2011 / RTA version 1.12 – can change unpredictably in future).

Thus, as a workaround, whenever pipeline encounters shorter barcodes and longer barcodes in the same flowcell, it appends “CTC” to shorter barcodes and writes those sequences in the local barcode_definition.txt file. This is the main reason for creating a local copy of barcode definitions. For example, on encountering barcodes ID01 and IDMB1, the local copy of barcode definition file would contain the following entries:

ID01,ATCACG**CTC**
IDMB1,GACTTCGTA

The actual sequences written to the SampleSheet.csv are the modified sequences from this local barcode definition file.

This approach has been working based on the assumption that sequencers append “CTC” to shorter barcodes. If this assumption changes, appropriate modifications should be made to ipipeV2/lib/BarcodeDefinitionBuilder.rb file. The code in ipipeV2/bin/PreProcessor.rb calls BarcodeDefinitionBuilder.rb.

2.10.2 Recovering from sample swap(s)

There are several reasons that can cause samples to be swapped. How that happens is beyond the scope of this document. This section describes a protocol to follow to recover from swapped sample(s).

1) After a swap has been identified, find the correct analysis information for the swapped lanes. E.g. if FOOFOOAXX-1 is swapped with some other sample, find out the correct sample name, library name, reference path and chip design for FOOFOAXX-1.

2) If the new reference is the same as the original reference used for alignment, that is the best possible case. In this case, only sample name and library names in the BAM header need to be changed. Use the pipeline tool ipipeV2/java/BAMHeaderFixer.jar to fix this.

3) If the new reference path is different from the original reference, update the BWAConfigParams.txt file for FOOFOOAXX-1, and remap the data.

Motivation behind using synthetic (aka fake) sample names in SampleSheet.csv

While demultiplexing the reads, the pipeline provides synthetic sample names to CASAVA. Instead of using actual sample names, the pipeline uses flowcell barcode name in SampleSheet.csv (which is used to demultiplex the reads). The primary purpose of using synthetic sample names is to minimize inadvertent data losses after a sample swap is detected.

CASAVA 1.8 claims to keep all the reads belonging to the same sample in the same result directory, which is used for analysis. However, if one or more samples were swapped, the reads from different sequencing events can be mixed up in a given sequencing event level directory. Unfortunately, with CASAVA 1.8.0, there is no clear way to segregate the reads. In this case, the only possible alternatives would be to either re-analyze the whole flowcell starting from bcl to fastq conversion, or re-sequence the flowcell, if the bcl files were already cleaned.

Thus, the method of using synthetic sample names ensures that the reads of a given sequencing event do not mix with any other sequencing event. As a result, the data can be recovered by simply remapping with the correct reference. The downside of this approach is that manual merges are required downstream to merge several samples together in a single BAM file.

2.11 Backing up the data

Currently this is implemented and executed as part of CASAVA 1.7 pipeline. The current path is of the pipeline code is /stornext/snfs5/next-gen/Illumina/ipipe/archive. It should be trivial to copy this directory to CASAVA 1.8 pipeline and run it from there. However, Derrick Dugas (ddugas@bcm.edu) should be informed if any changes are made to the code path.

This directory contains a shell script autoarchive.sh which is invoked by the cron job at 8:30 AM each morning. This shell script executes ArchiveListBuilder.rb.

ArchiveListBuilder.rb executes a LIMS query to obtain all the directory paths uploaded two days (172800 seconds) earlier than the current timestamp. It parses the directory paths from the LIMS response string and appends them to a file named "archive_request_list.txt".

At 9:30 AM each day, Derrick's cron job executes. This job locks down archive_request_list.txt file and starts the archive operation. When these directories complete archiving, the corresponding entries are removed from the archive_request_list.txt and appended to "archive_illumina_done_list.txt".

When result paths were not uploaded to LIMS on a given day (i.e. today – 2 days), the pipeline sends out an error email to subscribed watchers. It is safe to ignore this email.

While this approach works, it has a flaw. If on a given day, the number of directories to be sent to archive is very high, or archive operation fails for any reason, the "archive_request_file.txt" remains locked until archive completes. As a result, when the pipeline cron job attempts to append a new set of directories to the archive_request_list.txt, it will fail with the error "permission denied". This error is captured and sent via email. The current workaround is to manually run the LIMS query for the given date, obtain the list of result directories and copy them to another file, and inform Derrick that he needs to use a different file for archiving in addition to the archive_request_list.txt.

To avoid the manual handshake, another script ArchiveListBuilder_wip.rb has been developed. Whenever "archive_request_list.txt" is locked, it writes the new set of directories in "archive_request_list_toadd.txt". Moreover, the next time it runs, append all entries from "archive_request_list_toadd.txt" to "arhive_request_list.txt". Hence, this would eliminate manual handshake with Derrick. ***However, I have not been able to sufficiently test this script yet, hence it is not yet deployed in the cron job. So, manual handshakes are still required.***

Please note that only result directories are sent to archive list. Also files with extension ".o" and ".e" (the STDOUT and STDERR logs of programs) are not archived.

2.12 Adding new sequencers to the pipeline

This section describes the steps to add new sequencers to the pipeline.

1) Get the name of the sequencer from the sequencing team. For example, let's assume that new name is 7001133.

2) Create a directory for the sequencer in the root directory (The root directory information is the value of key “rootDir” under the “sequencers” section of config.yml.). e.g. the root directory would be /stornext/snfs0/next-gen/Illumina/Instruments/

   Within this directory, create another directory 7001133. So the directory structure looks like:

   /stornext/snfs0/next-gen/Illumina/Instruments/7001133

3) Create an empty file “done_list.txt” in the newly created sequencer directory (7001133).

4) Provide the path of newly created directory to sys-admin team so that they can configure the sequencer to write to this directory. E.g. provide them the path /stornext/snfs0/next-gen/Illumina/Instruments/7001133.

5) Now the pipeline will search this directory for any new flowcells whenever they become available.

2.13 Preparing a new reference for BWA

This section elaborates the steps to follow to prepare a new reference genome for analysis with BWA.

1) Obtain the reference fasta file for the genome. Place it in the suitable directory under /stornext/snfs0/hgsc-refs/Illumina/bwa_references/. Current convention is to create a letter directory corresponding to the first letter of the species and then another directory for the species in which reference genome is placed. For example, for the sheep reference (Ovis aries), create the following directory structure

   /stornext/snfs0/hgsc-refs/Illumina/bwa_references/o/Ovis_aries/original

   This is just convention. BWA does not require this directory structure.

2) Untar and unzip the fasta file and if this generates several fasta files, concatenate them into one single file. While concatenating, it would be useful for downstream analyses (post-alignment tools) if the chromosomes are in increasing order in the reference. E.g. the order of chromosomes should be chr1, chr2, chr3 ….

3) Logon interactively to MOAB.

4) Run the command

   /PathToBWA/bwa index –a algorithm_name reference_file_name

   reference_file_name is the name of the reference fasta file, e.g. sheep.fa

The value for algorithm name must be either “is” or “bwtsw” depending on the size of the reference fasta file.

If the fasts file is less than 2G in size (say a few hundred Mbytes), use “is” as the algorithm name. However, if the reference file is larger than 2G, use “bwtsw” as the algorithm name. The details of this command can be found at http://bio-bwa.sourceforge.net/bwa.shtml

5) Copy the complete directory path with fasta file and pass it to Adi Jakkamsetti (jakkamse@bcm.edu), Yi Han (yhan@bcm.edu) and David Sexton (dsexton@bcm.edu) to add to LIMS. e.g. the path to pass for the sheep reference genome in the above example would be

/stornext/snfs0/hgsc-refs/Illumina/bwa_references/o/Ovis_aries/original/sheep.fa

Section 3 Troubleshooting

This section describes several common problems encountered in the pipeline and their potential solutions.

3.1 My data has high yield but poor alignment results. What could be the reason / solution ?

1) Check the result files and the distribution plots produced by BAMAnalyzer.jar / SequenceAnalyzer.jar. Specifically, check the distribution plots BaseQualPerPosition.png and DistributionOfN.png.

2) Low alignment can occur if the sequence files contain a large number of “N” bases (undetermined bases). In such cases, BWA will not be able to align those reads. This information would be captured in DistributionOfN.png file. For example, for a sequencing event D04V1ACXX-1-ID06, alignment percentage of read 2 was 0. Distribution of “N” reached 100% in the middle of read 2 and the base quality did not recover. In this case, the sequencer had jumped 20 cycles which introduced so many “N” bases in the reads.

3) The corresponding distribution charts are shown below.

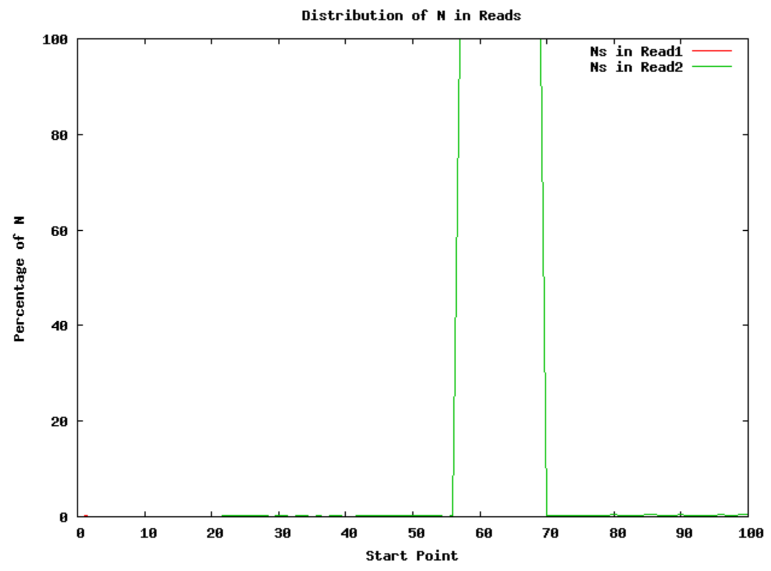


Fig. 1. Distribution of Ns in the reads

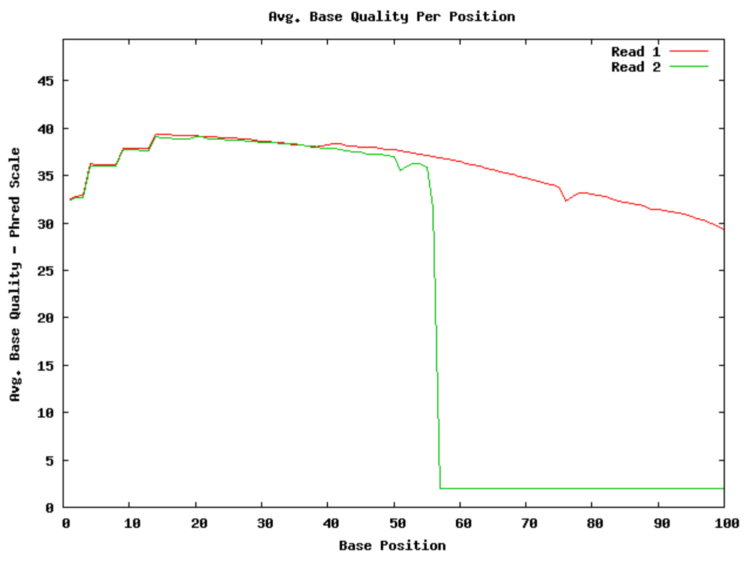


Fig. 2. Distribution of Average Base quality against position in the read

4) In such a situation, one work-around is to trim the sequence file for read 2. Use the pipeline tool ipipeV2/java/FastqTrimmer.jar. Specify the position where to start the trim and the number of bases to trim (remove). If the number of bases exceeds the read length, this tool automatically adjusts the number of bases to trim.

5) Now create a new directory named "trimmed" within the sequencing event directory. (The location / name of the directory can be any arbitrary value). Copy (or move) relevant sequence files to the new directory. Also, move BWAConfigParams.txt to the new directory.

6) Remap the data using ipipeV2/bin/Aligner.rb.

Note: It is possible that the base quality distribution may be good and number of N bases may be low and yet alignment result might be poor. Please work with sequencing team to ensure that this is not the case of a flowcell / sample swap and the reference applied for alignment is correct.

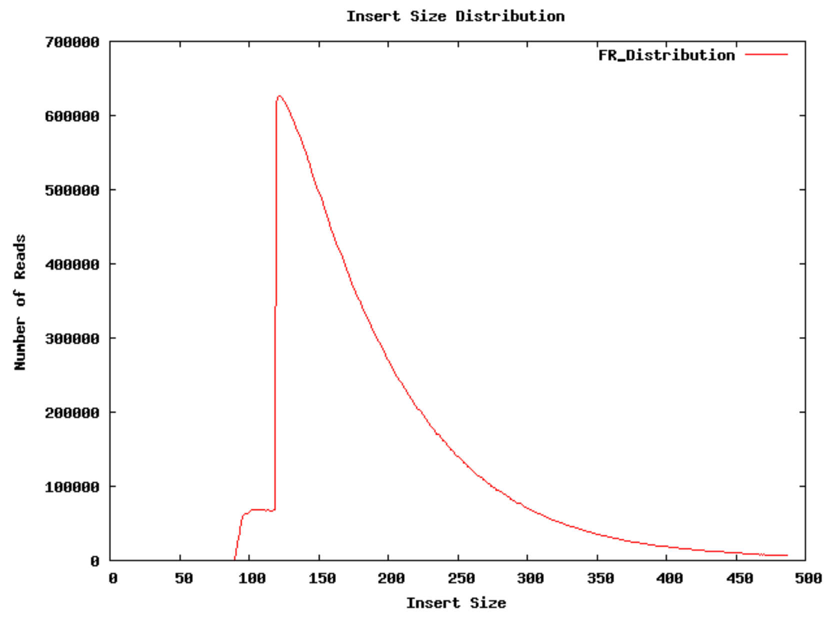
3.2 BWA alignment sampe command is taking days. Why ?

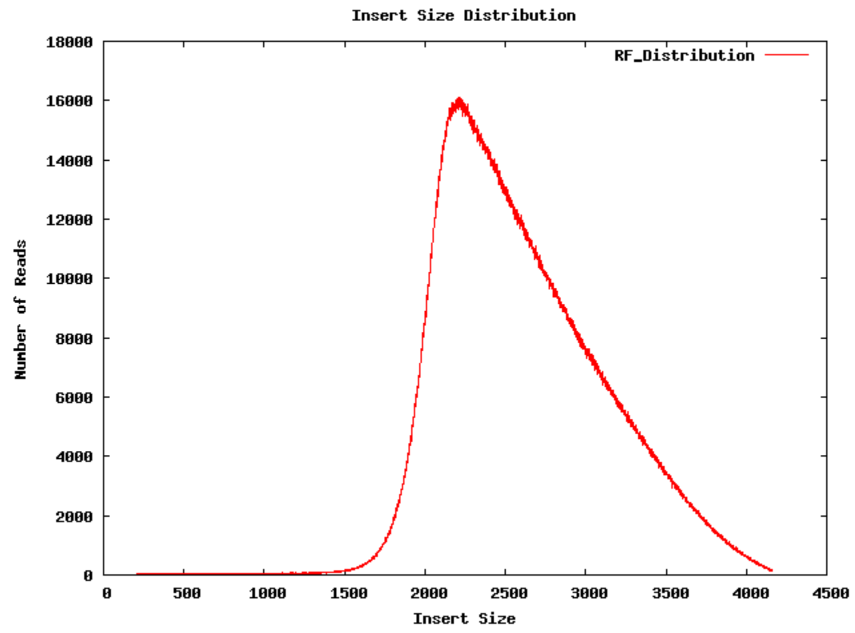
In some cases, BWA sampe command takes many days to complete. There can be several reasons for that.

One possible reason is wrong reference path. Since BWA is an aggressive aligner, it will attempt to align as many reads as possible. When sequence reads are not very close to the reference sequence, BWA has to perform more computation to find the alignment, which contributes to the slowness.

However, we have observed that even when reference path is correct, sampe can take days in some cases. Example of such a case is lane barcode D0GJRACXX-7. The directory path is /stornext/snfs0/next-gen/Illumina/Instruments/700733/111123_SN733_0201_AD0GJRACXX/Results/Project_111123_SN733_0201_AD0GJRACXX/Sample_D0GJRACXX-7

In this specific sequencing event, due to some problem in library preparation, reads of this lane had two different orientations with 10 times difference in mean insert size. See the distribution charts below.





While aligning, BWA uses some estimate on insert size to place the reads. However, if the reads cannot be placed based on their estimated insert size, it has to run smith-waterman algorithm on all those reads. This increases the computation time.

3.3 Read 2 in the flowcell had problems and it was aborted. Now it must be analyzed as a fragment. How can that be done ?

If the flowcell was configured as a paired end flowcell and now read 2 is aborted, it may not be automatically picked up for analysis, depending on RTA behaved during flowcell termination. Let's first consider the case where this flowcell is not automatically selected for analysis. In such a case, follow the steps outlined below:

1) Navigate to the flowcell directory and run the command:

   ```
   ruby /FullPath/ipipeV2/bin/PreProcessor.rb fcname=Flowcell_Dir_Name action= build_fc_defn
   ```

   The pipeline will now contact LIMS, get analysis information and writes the configuration file ./Data/Intensities/BaseCalls/FCDefinition.xml within the flowcell.

2) Edit this file using vim or any other text editor and change the value of "Type" in the second line from "paired" to "fragment". Please make sure that the word fragment is enclosed within quotes.

3) Navigate back to the main flowcell directory (in step 1) and run the following command:

   ```
   ruby /FullPath/ipipeV2/bin/PreProcessor.rb fcname=Flowcell_Dir_Name action=upload_start_date
   action=build_sample_sheet action=build_barcode_defn action=run_next_step
   ```

   This command will perform all pre-processing actions except building FCDefinition.xml, which already has been created in the previous step.

4) Navigate to the parent directory (i.e. directory of the sequencer). Append the name of this flowcell (complete directory name) to the "done_list.txt" file.

5) The flowcell will analyze as a fragment.

Let's consider the second case that appropriate files were written by the RTA and the pipeline automatically selected this flowcell for analysis. Please follow the steps listed below:

1) Kill any running / pending jobs of this flowcell.

2) Delete the "Result" directory and all its sub-directories from the flowcell directory.

3) Navigate to the BaseCalls directory, and edit FCDefinition.xml to have Type="fragment" instead of Type="paired".

4) Navigate back to the main flowcell directory and run the following command:

   ```
   ruby /FullPath/ipipeV2/bin/PreProcessor.rb fcname=Flowcell_Dir_Name action=upload_start_date
   action=build_sample_sheet action=build_barcode_defn action=run_next_step
   ```

3.4 Something went wrong with demultiplexing. We need to run demultiplexing with custom value for qseq mask. How can this be done ?

Due to some sequencing errors it is possible that the pipeline may not be able to correctly demultiplex the barcodes. In this case, the whole flowcell must be analyzed again while specifying a value for qseq mask (or use_bases_mask). Yi or someone from the sequencing team will let you know if this needs to be done. Follow the subsequent steps to demultiplex again:

1) Delete the Result directory and kill any pending / running jobs for this flowcell.

2) Run ipipeV2/bin/PreProcessor.rb with the following parameters:

   ```
   ruby /FullPath/ipipeV2/bin/PreProcessor.rb fcname=Flowcell_Dir_Name action=upload_start_date action= build_fc_defn action=build_sample_sheet action=build_barcode_defn
   ```

   Please note that this command is run without specifying the action=run_next_step.

3) After this command completes, run ipipeV2/bin/BclToFastQConvertor.rb with the following parameters:

   ```
   ruby /FullPath/ipipeV2/bin/BclToFastQConvertor.rb fcname=Flowcell_Dir_Name
   use_bases_mask=some_value
   ```

   To get the correct value for use_bases_mask, refer to CASAVA 1.8 manual. Yi Han (yhan@bcm.edu) can also help with that. Please note that the syntax of this parameter has changed between CASAVA 1.7 and 1.8, so please ensure that the value conforms to CASAVA 1.8 syntax.

This flowcell will now undergo demultiplexing using custom parameters.

3.5 LIMS provided incorrect sample / library names for many sequencing events. How can this error be fixed in BAM headers ?

Due to user or other errors, it is possible that LIMS can provide incorrect library names or sample names. Once the correct names are identified, BAM headers must be changed. For each sequencing event, run the following command:

```
java –jar /PathToipipeV2/ipipeV2/java/BAMHeaderFixer.jar I=BamToFix S=New_Sample L=New_Library
```

If many BAMs are affected, please go to bulkcommands directory, and create a bulk fix script similar to bulkremap and run the above listed command over a list of directories.

3.6 Several analysis result values did not get uploaded to LIMS. How can this be fixed ?

Go the sequencing event directory for the lane / barcode for which data is missing from LIMS. If alignment results (such as alignment percentage, percent alignment error) are missing run the following command :

```
ruby /PathToipipeV2/wrappers/ResultUploader.rb ANALYSIS_FINISHED
```

If sequence level metrics such as yield, phasing, prephasing are missing, run the following command :

```
ruby /PathToipipeV2/wrappers/ResultUploader.rb SEQUENCE_FINISHED
```

If uniqueness percentage is missing, open the file *_sequenceAnalysis.o. It contains the upload command used to upload uniqueness percentage to LIMS. Copy paste this command and execute it at the command line.

3.7 After demultiplexing, the yield of a sequencing event was very low. What happened ?

1) Navigate to the sequencing event directory.

2) Check the BAMAnalysisInfo.xml (or BWA_Map_Stats.txt). This is the output of BAMAnalyzer.jar. Check the number of Total Reads.

3) If the value for "Total Reads" is low, it is certain that less reads were used to build the sequence files.

4) Check the file "ReadsPassingPurityFilter.metrics" and find out the number of reads that passed the purify filtering. Only the reads that pass purity filtering are added to the final sequence files. This percentage should be 80% to 85% and higher. If this value is low, it implies that most reads did not pass purity filtering, so they did not get used to make sequence files.

5) If the number of reads that passed purity filtering is high, then please check the file "Demultiplex_Stats.htm" in the directory Flowcell_Dir/Results/Basecall_Stats_*. You might have to FTP or email this file to your desktop.

   To email this file as an attachment, use the following command:

   ```
   java –jar /PathToipipeV2/ipipeV2/java/AttachmentMailer.jar sender=sol-pipe@bcm.edu
   dest=myemail@bcm.edu attach=Demultiplex_Stats.htm
   ```

   The name of the sender can be anything. However, to prevent this email from going to spam folder, please use a name different from your email address.

6) For the sequencing event under investigation, please check the values for "% Perfect index reads" and "% One mismatch reads (index)". The sum of these two values should be close to 100. If this value is close to 100, it means that most of the index reads were correctly identified and placed in the correct directories. However, if this value is lower than 100, (say 30), it means that indexes matched for only 30% of the reads. Hence, only 30% of the reads were placed in the sequencing directory. Remaining 70% of the reads ended up in directory Flowcell/Results/Undetermined_indices.

3.8 The pipeline sent out an error email that MarkDuplicates.jar (or any of the Java programs such as FixMateInformation.jar, MateInfoFixer.jar, BAMAnalyzer.jar or SequenceAnalyzer.jar failed). What should I do ?

If any of the post BWA programs fail (exit with non-zero code), the pipeline will send an appropriate email message describing the sequencing event name, and the program that failed.

1) Please navigate to the sequencing directory. The error information will be included in two files namely *_processBam.o and the actual log file of the program that failed. In most cases observed so far, these failures are primarily due to problems with /space1/tmp. The corresponding log file (Program_name.e) will have some information (exception message) describing the error. Typically, exception messages might contain text like "Input/Output error" or "IO Exception" or something similar.

2) If the failure was due to bad local disk, obtain the name of the node on which the failed job was running. The node name can be obtained from the file *_processBam.o.

3) Restart the alignment by running the command

   ```
   ruby /PathToipipeV2/bin/Aligner.rb
   ```

4) Pass on the name of the node to the sys-admin team so that it can be removed from the cluster.

3.9 CASAVA software aborted while demultiplexing the reads. What should be done ?

There is no clear protocol to follow if CASAVA breaks down. With version 1.8, we use CASAVA only for bcl to fastq conversion in the Illumina pipeline. Hence, the scope of CASAVA failures affecting the pipeline is reduced. One potential protocol to follow is outlined below:

1) Try to find out the cause of failure. Navigate to the directory Flowcell_Dir_Name/Results. Look at files *_BclToFastQ.o and *._BclToFastQ.e. These files contain the STDOUT and STDERR of CASAVA bcl to fastq conversion.

2) The last few lines of these files will probably show some error. If the error message is self-explanatory, it is good. Otherwise, please send an email to Illumina tech-support techsupport@illumina.com and/or ask Yi Han for help yhan@bcm.edu.

3) If the STDOUT or STDERR in step 1) provide information about a specific bcl file or cycle that might bad, then it is possible to remove those particular cycles from the analysis. This can be done by following the steps outlined below:

   a) Navigate to the Results directory within the flowcell directory.

   b) Make a copy of the file named "Makefile".

   c) Edit the original "Makefile" and perform required changes.

   d) Logon interactively to MOAB and run make with required number of cores. e.g. "make –j8".

   e) If this make job completes without any problem, browse to the BaseCalls directory.

   f) Run the bash script barcodes.sh.

   g) The rest of the analysis should continue from sequence building.

4) For help with modifying Makefile or config.xml file(s), please contact Yi Han yhan@bcm.edu.

3.10 MOAB system had a catastrophic failure. All running jobs and information about which jobs were running is wiped out. How can we resurrect the pipeline ?

This is a rare occurrence, but whenever it happens, it takes considerable effort to bring all the jobs back. The key thing to determine which flowcells / sequencing events were running during the time of crash.

LIMS team can help to determine the flowcells / sequencing events for which analysis start date was uploaded to LIMS, but analysis end date was not uploaded. This information gives an insight to what was currently running.

If some flowcells were in the initial stages of the analysis, i.e. bcl to fastq conversion or sequence building, then it is relatively easy on the user to restart the analysis from scratch. To do this, delete the Results directory and run the following command to start the analysis all over again :

```
ruby /PathToIpipeV2/ipipeV2/bin/PreProcessor.rb fcname=FC_Name action=all
```

If some lanes are in the final stages of the alignment, such as “processBam” stage, it would be more efficient to restart from the alignment phase by using the command

```
ruby /PathToIpipeV2/ipipeV2/bin/Aligner.rb
```

However, if there are too many aborted sequencing events or it is difficult to determine exactly where every sequencing event was, it would be easier to find the names of flowcells that were analyzing from LIMS, delete “Results” directory in each of them and restart the analysis from the beginning.