Contents

1 Introduction
  1.1 Recipes .................................................. 3

2 Installation .................................................... 5
  2.1 Build from source .......................................... 5

3 The human genome reference ................................. 7
  3.1 BWA reference for WGS/WES data .......................... 8
  3.2 STAR reference for RNA-seq data .......................... 8

4 Hadoop setup .................................................. 9
  4.1 Single node ............................................... 9
  4.2 Multi node ............................................... 11
  4.3 Hadoop configuration ....................................... 11
  4.4 Intel’s Hadoop Adapter for Lustre .......................... 12

5 Uploading the references ..................................... 15
  5.1 HDFS ..................................................... 15
  5.2 Amazon S3 ............................................... 16
  5.3 GPFS & Lustre ............................................. 16

6 Halvade Preprocessing ....................................... 17
  6.1 Performance .............................................. 17
  6.2 Synopsis ................................................ 17
  6.3 Required options .......................................... 18
  6.4 Optional options .......................................... 18

7 Example datasets ............................................. 19
  7.1 Whole genome sequencing sample .......................... 19
  7.2 RNA-seq sample .......................................... 19

8 Run Halvade ................................................... 21
  8.1 Configuration ............................................ 21
  8.2 Run ...................................................... 22
  8.3 Amazon AWS .............................................. 22

9 Halvade Options .............................................. 25
  9.1 Required options .......................................... 25
9.2 Optional options ................................................................. 26

10 Troubleshooting ................................................................. 29
10.1 Find the logs to solve Halvade errors ........................................... 29
10.2 Halvade having a reference file error while downloading/loading .......... 29
10.3 Halvade with BAM input seems stuck at the MarkDuplicates step .......... 29
Halvade implements the best-practices pipelines from Broad Institute using the MapReduce framework. Halvade supports pipelines for WGS and Exome sequencing data as well as RNA-seq data. The best-practices pipeline is parallelized using MapReduce where the Map phase performs the read mapping and the Reduce phase takes care of the variant calling with the necessary preprocessing steps. By using other existing, reliable tools like BWA, STAR, samtools, picard and GATK we ensure that the variants are correctly called. Additionally these tools get frequent updates and Halvade allows simple replacement of the binary file to update the tool.

Contents:
Halvade is a Hadoop MapReduce implementation of the best practices pipeline from Broad Institute for whole genome and exome sequencing (DNA) as well as RNA-seq. Halvade will produce a VCF output file which contains the single nucleotide variants (SNVs) and additionally insertions and deletions (indels) in certain pipelines. This program requires Hadoop on either a local cluster with one or more nodes or an Amazon EMR cluster to run. As Hadoop is typically run on a linux cluster, this documentation only provides information for a linux setup. The GATK used in Halvade only works with Java v1.7, so this version of Java should be installed on every node.

Note: Halvade is available under the GNU license and provides a binary with all opensource tools. However, since the GATK has its own license, which is available online here, the GATK binary is not provided in the bin.tar.gz file and needs to be added individually.

Halvade depends on existing tools to run the pipeline, these tools require additional data besides the raw sequenced reads. This data consists of the human genome reference FASTA file, some additional files created using the reference and the dbSNP database. The index file used in the BWA or STAR aligners are created with the tools itself. The FASTA index and dictionary files used by the GATK are created by samtools and Picard. The naming convention for these files and additional information to run Halvade is provided in this Halvade documentation.

1.1 Recipes

Two recipes have been created to run Halvade on WGS and RNA-seq data. These show all commands needed to run Halvade with the example data. The recipe for the DNA-seq pipeline can be found here. The RNA-seq data is typically less big and therefore we chose to provide a recipe to run the Halvade rna pipeline on a single node Hadoop environment, which can be found here.
CHAPTER 2

Installation

Every Halvade release is available at github. Download and extract the latest release, currently v1.2.0:

```
1. wget https://github.com/biointec/halvade/releases/download/v1.2.0/Halvade_v1.2.0.tar.gz
2. tar -xvf Halvade_v1.2.0.tar.gz
```

The files that are provided in this package are the following:

```
example.config
runHalvade.py
halvade_bootstrap.sh
HalvadeWithLibs.jar
HalvadeUploaderWithLibs.jar
bin.tar.gz
```

2.1 Build from source

Halvade can also be built from the source files. To do this, first you need to clone the github repository and build the package with ant as follows:

```
1. git clone https://github.com/biointec/halvade.git
2. cd halvade/halvade/
3. ant
4. cd ../halvade_upload_tool/
5. ant
6. cd ../
```

This will build the two jar files in the respective dist subdirectories. The scripts and example configuration files can be found in the scripts directory, move the jar files to the script directory so the scripts have access to the jar file:

```
1. cp halvade/dist/HalvadeWithLibs.jar scripts/
2. cp halvade_upload_tool/dist/HalvadeUploaderWithLibs.jar scripts/
```
The next step is to download the human genome reference files and prepare them to use with Halvade.
CHAPTER 3

The human genome reference

Halvade uses the genome reference FASTA file (ucsc.hg19.fasta), found in the GATK resource bundle, to build the index files for both BWA and STAR. The FASTA file comes with an index and a dictionary file. Additionally a full dbSNP file (version 138) is used when recalibrating the base scores for the reads. These files are all found in the GATK resource bundle which is available here. This FTP site has a limited number of parallel downloads and might not load at these times. Here is how you download the files using the terminal in the current directory, it is advised to make a new directory for all reference files:

```bash
mkdir halvade_refs/
cd halvade_refs/
wget ftp://gsapubftp-anonymous@ftp.broadinstitute.org/bundle/hg19/ucsc.hg19.fasta.gz
wget ftp://gsapubftp-anonymous@ftp.broadinstitute.org/bundle/hg19/ucsc.hg19.fasta.fai.gz
wget ftp://gsapubftp-anonymous@ftp.broadinstitute.org/bundle/hg19/ucsc.hg19.dict.gz
mkdir dbsnp
cd dbsnp
wget ftp://gsapubftp-anonymous@ftp.broadinstitute.org/bundle/hg19/dbsnp_138.hg19.vcf.gz
wget ftp://gsapubftp-anonymous@ftp.broadinstitute.org/bundle/hg19/dbsnp_138.hg19.vcf.idx.gz
cd ../
```

Next we need to unzip all these files so they can be used in Halvade:

```bash
mkdir halvade_refs/
cd halvade_refs/
gunzip ucsc.hg19.fasta.gz

```

The index files, the reference and the dbSNP file need to be uploaded to the HDFS server if a cluster with more than one node is used to run Halvade. Setting up Halvade is described in the following parts of the documentation.
3.1 BWA reference for WGS/WES data

The BWA aligner is used for the whole genome and exome sequencing pipelines. A BWT index of the reference FASTA file needs to be created to run BWA, which needs to be accessible by Halvade so BWA can be started correctly. The BWA binary is available in the bin.tar.gz archive, which is provided in every Halvade release.

```
1. tar -xvf bin.tar.gz
2. ./bin/bwa index ucsc.hg19.fasta
```

This process will create 5 files with the provided name as a prefix, this naming convention is important as Halvade finds this index by the FASTA prefix ucsc.hg19.

The reference folder should contain these files:
- /home/user/halvade/ref/ucsc.hg19.fasta
- /home/user/halvade/ref/ucsc.hg19.fasta.fai
- /home/user/halvade/ref/ucsc.hg19.dict
- /home/user/halvade/ref/ucsc.hg19.fasta.amb
- /home/user/halvade/ref/ucsc.hg19.fasta.ann
- /home/user/halvade/ref/ucsc.hg19.fasta.bwt
- /home/user/halvade/ref/ucsc.hg19.fasta.pac
- /home/user/halvade/ref/ucsc.hg19.fasta.sa
- /home/user/halvade/ref/dbsnp/dbsnp_138.hg19.vcf
- /home/user/halvade/ref/dbsnp/dbsnp_138.hg19.vcf.idx

3.2 STAR reference for RNA-seq data

**Note:** The process to build the STAR index requires a minimum of 32 GBytes of RAM, make sure there is sufficient RAM memory.

The RNA-seq pipeline uses the STAR aligner to perform the read alignment step. Similarly to BWA, the STAR aligner requires an index of the reference FASTA file. Again, this can be created by using the STAR binary which is provided in the bin.tar.gz archive which is available in every Halvade release.

```
1. tar -xvf bin.tar.gz
2. mkdir ./STAR_ref/
3. ./bin/STAR --genomeDir ./STAR_ref/ --genomeFastaFiles ucsc.hg19.fasta --runMode genomeGenerate --runThreadN 4
```

The shown command to build the STAR genome index uses 4 threads, this should be updated to reflect the number of cores available. After the STAR genome index has been created, the provided output folder will contain all files needed by STAR and in turn by Halvade.
Halvade runs on the Hadoop MapReduce framework, if Hadoop MapReduce version 2.0 or newer is already installed on your cluster, you can continue to the Hadoop configuration section to make sure the advised configuration is enabled. Halvade uses GATK, which requires a specific eversion of Java, currently version 1.7. To make sure GATK works as expected the correct version of Java needs to be installed on every node in the cluster and set as the default Java instance, in Ubuntu use these commands:

```
sudo apt-get install openjdk-7-jre
sudo update-alternatives --config java
```

### 4.1 Single node

To run Hadoop on a single node, it is advised to install Hadoop in psuedo-distributed mode. The following instructions are based on this tutorial and can be used for additional information. Hadoop requires ssh and rsync to run, to install these on your system, run these commands (on Ubuntu):

```
sudo apt-get install ssh rsync
```

Download and unzip the Hadoop distribution (here 2.7.2):

```
wget http://www.eu.apache.org/dist/hadoop/common/hadoop-2.7.2/hadoop-2.7.2.tar.gz
tar -xvf hadoop-2.7.2.tar.gz
```

To configure the Hadoop installation to run in psuedo-distributed mode edit these files as follows, creating the file or replacing the line if necessary:

**etc/hadoop/hadoop-env.sh**:

```
export JAVA_HOME=/your/java/bin/directory
```

**etc/hadoop/core-site.xml**:
Additionally we need to make sure that that the node can make a passwordless connection to localhost with ssh, check if `ssh localhost` works without a password. If this isn’t the case run the following commands:

1. `ssh-keygen -t dsa -P '' -f ~/.ssh/id_dsa`
2. `cat ~/.ssh/id_dsa.pub >> ~/.ssh/authorized_keys`
3. `chmod 0600 ~/.ssh/authorized_keys`

Now we need to format the NameNode and start the HDFS and Yarn services, do this as follows:

1. `bin/hdfs namenode -format`
2. `sbin/start-dfs.sh`
3. `sbin/start-yarn.sh`
4. `bin/hdfs dfs -mkdir /user`
5. `bin/hdfs dfs -mkdir /user/<username>`

Now Hadoop can be run from the `bin/hadoop` command and for ease of use this directory can be added to the `PATH` variable by adding this line to your `.bashrc` file:

```
export PATH=$PATH:/hadoop/install/dir/bin
```

After the `Hadoop configuration` has been updated to run Halvade optimally on your node, the services will need to be restarted. To restart the pseudo-distributed Hadoop environment run these commands:
4.2 Multi node

For the Hadoop installation on a multi node cluster, we refer to the manual given by Cloudera to install CDH 5 or later and configure the Hadoop cluster. You can find this detailed description online here.

4.3 Hadoop configuration

After Hadoop is installed, the configuration needs to be updated to run Halvade in an optimal environment. In Halvade, each task processes a portion of the input data. However, the execution time can vary to a certain degree. For this the task timeout needs to be set high enough, in mapred-site.xml change this property to 30 minutes:

```xml
<property>
  <name>mapreduce.task.timeout</name>
  <value>1800000</value>
</property>
```

The Yarn scheduler needs to know how many cores and how much memory is available on the nodes, this is set in yarn-site.xml. This is very important for the number of tasks that will be started on the cluster. In this example, nodes with 128 GBytes of memory and 24 cores are used. Because some of the tools used benefit from the hyperthreading capabilities of a CPU, the vcores is set to 48 if hyperthreading is available:

```xml
<property>
  <name>yarn.nodemanager.resource.memory-mb</name>
  <value>131072</value>
</property>
<property>
  <name>yarn.nodemanager.resource.cpu-vcores</name>
  <value>48</value>
</property>
<property>
  <name>yarn.scheduler.maximum-allocation-mb</name>
  <value>131072</value>
</property>
<property>
  <name>yarn.scheduler.minimum-allocation-mb</name>
  <value>512</value>
</property>
<property>
  <name>yarn.scheduler.maximum-allocation-vcores</name>
  <value>48</value>
</property>
<property>
  <name>yarn.scheduler.minimum-allocation-vcores</name>
  <value>1</value>
</property>
```

After this, the configuration needs to be pushed to all nodes and certain running services restarted. On a single node cluster with Hadoop in pseudo-distributed mode run:
On a multi node cluster the services running on different nodes need to be restarted after distributing the configuration files, these following commands assume a CDH 5 installation according to the guide shown before:

```bash
scp *-site.xml myuser@myCDHnode-<n>.mycompany.com:/etc/hadoop/conf.my_cluster/
```

On the ResourceManager run:

```bash
sudo service hadoop-yarn-resourcemanager restart
```

On each NodeManager run:

```bash
sudo service hadoop-yarn-nodemanager restart
```

On the JobHistory server run:

```bash
sudo service hadoop-mapreduce-historyserver restart
```

For the RNA-seq pipeline, the memory check needs to be disabled because Halvade uses multiple instances of the STAR aligner when aligning the reads. The genome index files are first loaded into shared memory so every instance can access this instead of loading the reference itself. However, due to the way Hadoop checks physical memory, which includes the shared memory, this check should be disabled. To do this, add these properties to the `yarn-site.xml` file.

```xml
<property>
  <name>yarn.nodemanager.vmem-check-enabled</name>
  <value>false</value>
</property>
<property>
  <name>yarn.nodemanager.pmem-check-enabled</name>
  <value>false</value>
</property>
```

### 4.4 Intel’s Hadoop Adapter for Lustre

When using Lustre as the filesystem instead of HDFS, using Intel’s adapter for Lustre will increase the performance of Halvade. To enable the Adapter for Lustre you need to change some configurations in your Hadoop installation. In `core-site.xml` you need to point to the location of Lustre and set the Lustre FileSystem class, if Lustre is mounted on `/mnt/lustre/`, add these to the file:

```xml
<property>
  <name>fs.defaultFS</name>
  <value>lustre:///</value>
</property>
<property>
  <name>fs.lustre.impl</name>
  <value>org.apache.hadoop.fs.LustreFileSystem</value>
</property>
<property>
  <name>fs.AbstractFileSystem.lustre.impl</name>
</property>
```

(continues on next page)
Additionally, you need to set the Shuffle class in mapred-site.xml:

```xml
<property>
  <name>mapreduce.job.map.output.collector.class</name>
  <value>org.apache.hadoop.mapred.SharedFsPlugins$MapOutputBuffer</value>
</property>
<property>
  <name>mapreduce.job.reduce.shuffle.consumer.plugin.class</name>
  <value>org.apache.hadoop.mapred.SharedFsPlugins$Shuffle</value>
</property>
```

After adding these settings to the configuration, the files need to be pushed to all nodes again and all services restarted, see above. Additionally the jar containing Intel’s Adapter for Lustre should be available on all nodes and added to the classpath of Hadoop. To do this you can find the directories that are currently in your hadoop classpath and add the jar to one of these on every node. To find the directories, run this command:

```
hadoop classpath
```
Uploading the references

The reference data needs to be available to all nodes in the cluster, which is why they should be available on the distributed filesystem. When running Halvade, the references will be copied to local scratch on every node when they need to be accessed to increase the performance of subsequent accessing of the file.

**Note:** The reference files shouldn’t be uploaded to the distributed filesystem if a single node Hadoop environment is used. The tool would download them to local scratch to use. Instead we put the files on local scratch and add some additional files so that Halvade can find the correct references. Additionally, the `refdir` option should be set that points to the directory with all reference files when running Halvade. There are four files that are used to find the corresponding reference files and directories, these should be added to correspond with the reference names:

```
touch ucsc.hg19.bwa_ref
touch ucsc.hg19.gatk_ref
touch STAR_ref/.star_ref
touch dbsnp/.dbsnp
```

### 5.1 HDFS

The reference files need to be copied to the HDFS so that Halvade can distribute them to every node to be used locally. Here we will create a directory on HDFS where all the files will be collected, execute the following commands to do this:

```
hdfs dfs -mkdir -p /user/ddecap/halvade/ref/dbsnp
hdfs dfs -put ucsc.hg19.* /user/ddecap/halvade/ref/
hdfs dfs -put dbsnp/dbsnp_138.hg19.* /user/ddecap/halvade/ref/dbsnp/
```

# for the RNA pipeline copy the STAR ref:
```
hdfs dfs -put STAR_ref/ /user/ddecap/halvade/ref/
```
5.2 Amazon S3

To copy the files to Amazon AWS with the terminal the AWS Command Line Interface needs to be installed using this documentation. If the bucket you want to use is called halv_bucket, execute the following commands:

```bash
aws s3 cp ./ s3://halv_bucket/user/ddecap/halvade/ref/ --include "ucsc.hg19.*"
aws s3 cp dbsnp/ s3://halv_bucket/user/ddecap/halvade/ref/dbsnp/ --include "dbsnp_138.hg19.*"
aws s3 cp STAR_ref/ s3://halv_bucket/user/ddecap/halvade/ref/ --recursive
```

5.3 GPFS & Lustre

Typically GPFS or Lustre are mounted on the directory on every node, the reference files simply need to be copied to that directory. If /mnt/dfs is the mounted distributed filesystem, execute the following commands:

```bash
mkdir -p /mnt/dfs/halvade/ref/dbsnp
cp ucsc.hg19.* /mnt/dfs/halvade/ref/
cp -r dbsnp/dbsnp_138.hg19.* /mnt/dfs/halvade/ref/dbsnp/

# for the RNA pipeline copy the STAR ref:
cp -r STAR_ref/ /mnt/dfs/halvade/ref/
```
The Halvade Uploader will preprocess the FASTQ files, this will interleave the paired-end reads and split the files in pieces of 60MB by default (this can be changed with the size option). The Halvade Uploader will automatically upload these preprocessed files to the given output directory on either local scratch, GPFS, HDFS, Amazon S3 or any other distributed file system. Note that the output of this step is the input of the Halvade command.

Note: This step is not required if the input file is an aligned BAM file.

### 6.1 Performance

For better performance it is advised to increase the Java heap memory for the Hadoop command, e.g. for 32GB:

```
export HADOOP_HEAPSIZE=32768
```

### 6.2 Synopsis

```
1. hadoop jar HalvadeUploaderWithLibs.jar -l /dir/to/input.manifest -O /halvade/out/ -t 8
2. hadoop jar HalvadeUploaderWithLibs.jar -l /dir/to/reads1.fastq -2 /dir/to/reads2.fastq -O /halvade/out/ -t 8
3. hadoop jar HalvadeUploaderWithLibs.jar -l /dir/to/input.manifest -O s3://bucketname/halvade/out/ -profile /dir/to/credentials.txt -t 8
```

The manifest file if used, contains per line a pair of files (reads1 and reads2) separated by a tab:

```
/path/to/file1_reads1.fq.gz  /path/to/file1_reads2.fq.gz
/path/to/file2_reads1.fq.gz  /path/to/file2_reads2.fq.gz
```
6.3 Required options

-1 STR Manifest/Input file. This string gives the absolute path of the manifest file or the first input FASTQ file. This manifest file contains a line per file pair, separated by a tab: /dir/to/fastq1.fastq /dir/to/fastq2.fastq. If this is equal to ‘-‘ then the fastq reads are read from standard input.

-O STR Output directory. This string gives the directory where the output files will be put.

6.4 Optional options

-2 STR Input file 2. This gives the second pair of paired-end reads in a FASTQ file.

--dfs Input on a DFS. This enables reading data from a distributed filesystem like HDFS and Amazon S3.

-i Interleaved. This is used when one FASTQ input file is given, the input file is assumed to have both pairs of paired-end reads and the reads are interleaved.

--lz4 Lz4 compression. This enables lz4 compression, this is faster than gzip but will require more disk space. The lz4 compression library needs to be enabled in the Hadoop distribution for this to work.

-p, --profile STR AWS profile. Gives the path of the credentials file used to access S3. This should have been configured when installing the Amazon EMR Command Line Interface. By default this is ~/.aws/credentials.

-s, --size INT Size. This sets the maximum file size (in bytes) of each interleaved file [60MB].

--snappy Snappy compression. This enables snappy compression, this is faster than gzip but will require more disk space. Snappy requires less disk space than lz4 and is comparable in compression speed. The snappy compression library needs to be enabled in the Hadoop distribution for this to work.

--sse Server side encryption. Turns on Server side encryption (SSE) when transferring the data to the Amazon S3 storage.

-t INT Threads. This sets the number of threads used to preprocess the input data. Performance will be limited if the heap memory isn’t sufficient.
Example datasets

The input data for these pipelines typically consist of either 2 FASTQ files for paired-end reads or a BAM file containing already aligned reads.

### 7.1 Whole genome sequencing sample

The whole genome sequencing sample is the NA12878 dataset, this dataset is typically used in similar benchmarks and papers. This dataset consists of 1.5 billion paired-end reads of 100 basepairs in length. This translates into a 50x coverage. Execute the following commands to download and preprocess the data:

```bash
1 wget ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR194/ERR194147/ERR194147_1.fastq.gz
2 wget ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR194/ERR194147/ERR194147_2.fastq.gz
3 export HADOOP_HEAPSIZE=32768
4 hadoop jar HalvadeUploaderWithLibs.jar -1 ERR194147_1.fastq.gz -2 ERR194147_2.fastq.gz
5 -O /user/ddecap/halvade/wgsin/ -t 16
```

### 7.2 RNA-seq sample

The RNA-seq example dataset is found in the encode project under the SK-MEL-5 experiment. The ENCSR201WVA dataset provides both paired FASTQ files and aligned BAM files. In this example we will download a single replicate of the ENCBS524EJL bio sample available in paired FASTQ files. To download and preprocess the FASTQ files run these commands in the terminal:

```bash
1 wget https://www.encodeproject.org/files/ENCFF005NLJ/@@download/ENCFF005NLJ.fastq.gz
2 wget https://www.encodeproject.org/files/ENCFF635CQM/@@download/ENCFF635CQM.fastq.gz
3 export HADOOP_HEAPSIZE=32768
4 hadoop jar HalvadeUploaderWithLibs.jar -l ENCF005NLJ.fastq.gz -2 ENCF635CQM.fastq.gz
5 -O /user/ddecap/halvade/rnain/ -t 16
```
Run Halvade

To run Halvade the GATK binary needs to be added to the bin.tar.gz file by executing the following commands:

```
1. tar -xvf bin.tar.gz
2. rm bin.tar.gz
3. cp GenomeAnalysisTK.jar bin/
4. tar -cvzf bin.tar.gz bin/*
```

Similar to the reference files, this file needs to be uploaded to the distributed filesystem if a cluster with more than one node is used. Run this command when using HDFS as distributed storage:

```
hdfs dfs -put bin.tar.gz /user/ddecap/halvade/
```

### 8.1 Configuration

Halvade is started using a python script `runHalvade.py`, this script reads the configuration from a file given in the first argument. This file contains all options you intend to give to the Halvade command. The configuration file uses a `=` character when a value needs to be provided to the option and the value should be quoted if it is a string. To add an options without arguments add a new line with just the option name. Commented lines, starting with `#`, are ignored by the script. The `example.config` file contains the most basic example, in which the necessary options are provided. The file looks like this:

```
N=5
M=64
C=16
B="/user/ddecap/halvade/bin.tar.gz"
D="/user/ddecap/halvade/ref/dbsnp/dbsnp_138.hg19.vcf"
R="/user/ddecap/halvade/ref/ucsc.hg19"
I="/user/ddecap/halvade/in/"
O="/user/ddecap/halvade/out/"
```

Note that the input folder is the output of the preprocessing step with HalvadeUploader.
To run the RNA-seq pipeline two additional options need to be provided:

```bash
star="/user/ddecap/halvade/ref/STAR_ref/"
```

If the nodes in the cluster have hyperthreading enabled, add the `smt` option to improve performance. To run the pipeline with a `bam` file as input, add the `bam` option.

**Note:** When running the Halvade job on a single node, it is not required to upload the reference files to the distributed filesystem. However, the input data should still be preprocessed with the Halvade Uploader tool and put on the distributed filesystem. Running Halvade in this setup, some additional files should be present to allow Halvade to find the references, these should have been added in a previous step. To show Halvade where to find the reference files, add the directory where the required files can be found like this:

```bash
refdir="/user/ddecap/halvade/ref/"
```

This folder is expected to be on local scratch or a mounted distributed filesystem so this doesn’t require a prefix.

### 8.2 Run

When all desired configuration for Halvade have been added to the `config` file, simply run the following command to start Halvade:

```bash
python runHalvade.py
```

This will start Halvade, which in turn will start the necessary Hadoop jobs. The script will return the ID of the process (PID) which is used in the filenames to store the standard out and error logs, `halvadePID.stdout` and `halvadePID.stderr`. The output of Halvade will be a single VCF file which can be found in the subdirectory `merge` of the provided output directory.

### 8.3 Amazon AWS

To run Halvade on an Amazon EMR cluster, the AWS Command Line Interface needs to be installed, installation instructions can be found here. To run Halvade on Amazon EMR, some additional configurations need to be added so the `runHalvade.py` script knows Halvade should be started on Amazon EMR. As the Halvade jar isn’t available on every node yet, this needs to be uploaded to Amazon S3 first. Similarly, the `bootstrap` script, which creates the halvade/ directory on the mounted SSD’s for intermediate data, needs to be uploaded as well.

```bash
aws s3 cp HalvadeWithLibs.jar s3://halv_bucket/user/ddecap/halvade/ref/
aws s3 cp halvade_bootstrap.sh s3://halv_bucket/user/ddecap/halvade/ref/
```

To use Halvade on Amazon EMR an AMI version of 3.1.0 or newer should be used. Add the following EMR configuration to run Halvade on Amazon EMR:

```bash
emr_jar="s3://halv_bucket/user/ddecap/halvade/HalvadeWithLibs.jar"
emr_script="s3://halv_bucket/user/ddecap/halvade/halvade_bootstrap.sh"
emr_type="c3.8xlarge"
emr_ami_v="3.1.0"
tmp="/mnt/halvade/
emr_s3logging="s3://halv_bucket/user/ddecap/halvade/logs/"
```
The `tmp` option is updated to point to the local SSD’s on the Amazon EMR nodes, which are mounted in the `/mnt/` folder. The `emr_s3logging` argument is used to save all Hadoop master and task logs for debugging purposes.

Additionally to run the script the default EMR need to be created in order to work, run this command:
CHAPTER 9

Halvade Options

Any directory given in the command line option needs to be accessible by all nodes. This can be either on HDFS, GPFS, Amazon S3 or any other distributed filesystem. When using one node this can also be local scratch. If no prefix is used, HDFS will be used by default. However, the default file system can be changed with the `fs.defaultFS` configuration of Hadoop. When this is changed the directories can simply be given without any prefix, else a prefix `file://` needs to be given for local scratch and mounted GPFS directories. For data stored on S3 when using the Amazon EMR service, the directories need to contain the bucket name as a prefix, e.g. `S3://bucketname/`. A script `runHalvade.py` is provided to gather all options in a simple config file which then calls Halvade with all provided options.

9.1 Required options

- **-B STR**  Binary location. This string gives the location where `bin.tar.gz` is located.
- **-D STR**  DBSNP file location. This string gives the absolute filename of the DBSNP file, this file needs to be compatible with the reference FASTA file provided by the –R option.
- **-I STR**  Input directory. The string points to the directory containing the preprocessed input (output of HalvadeUploader) or a BAM file on the used file system.
- **-O STR**  Output directory. This string points to the directory which will contain the output VCF file of Halvade.
- **-R STR**  Reference Location. This string gives the prefix (with .fasta extension) of the absolute filename of the reference in FASTA format. The corresponding index files, built with BWA, need to be in this directory having the same prefix as the reference FASTA file. The STAR genome index can be located in a different folder.
- **-M, --mem <INT>**  Memory size. This gives the total memory each node in the cluster has. The memory size is given in GB.
-N, --nodes INT  Node count. This gives the total number of nodes in the local cluster or the number of nodes you want to request when using Amazon EMR. Amazon AWS has a limit of 20 nodes unless the nodes are reserved for an extended period of time.

-C, --vcores INT  Vcores count. This gives the number of cores that can be used per node on the cluster (to enable simultaneous multithreading use the --smt option).

9.2 Optional options

-A, --justalign  Just align. This option is used to only align the data. The aligned reads are written to the output folder set with the –O option.

--aln INT  Select Aligner. Sets the aligner used in Halvade. Possible values are 0 (bwa aln+sampe), 1 (bwa mem)[default], 2 (bowtie2), 3 (cushaw2). Note that these tools need to be present in the bin.tar.gz file.

--bam  Bam input. This option enables reading aligned BAM input, using this will avoid realigning. If a realignment is required, the data needs to be transformed to FASTQ files, shuffled and preprocessed for Halvade.

--bed STR  Bed region. This option uses a BED file to split the genome in genomic regions that will be processed by one reduce task. This is used when feature count is enabled and the bed region give the known gene boundaries to avoid counting double.

--CA <STR=STR>  Custom arguments. This options allows the tools run with Halvade to be run with additional arguments. The arguments are given in this form: tool-name=extra arguments. All options must be correct for the tool in question, multiple arguments can be added by giving a quoted string and separating the arguments with a space. Possible toolnames are bwa_aln, bwa_mem, bwa_sampe, star, elprep, samtools_view, bedtools_bdsnp, bedtools_exome, picard_buildbamin, picard_addorreplacereadgroup, picard_markduplicates, picard_cleansam, gatk_realigntargetcreator, gatk_indelrealigner, gatk_base recalibrator, gatk_printreads, gatk_combinevariants, gatk_variantcaller, gatk_variantannotator, gatk_variantfiltration, gatk_splitncigarreads.

--combine  Combine VCF. With this option Halvade will combine VCF files in the input directory and not perform variant calling if the revelant files are found. This is done by default after the variant calling.

--count  Count reads. This counts the reads per Halvade region, this is only used for debugging purposes.

--drop  Drop. Halvade will drop all paired-end reads where the pairs are aligned to different chromosomes.

--dryrun  Dry run. This will initialize Halvade, which calculates the task sizes and region sizes of the chromosomes, but Halvade will not execute the Hadoop jobs.

--fbed STR  Filter on bed. This option will enable the reads to be filtered on the given bed file before performing the GATK steps. This is typically used in an exome dataset where only reads in a known bed file are expected.

--filter_dbsnp  Filter dbsnp. This flag turns on filtering of the dbSNP file before using it in the GATK. This can improve performance in some cases but typically the overhead of converting is too big.
--gff STR  GFF file. This sets the GFF file that will be used by Featurecounts to count the number of reads per exon.

--id STR  Read Group ID. This string sets the Read Group ID which will be used when adding Read Group information to the intermediate results. [GROUP1]

--illumina  Convert Illumina scores. This Option forces Halvade to convert every basepair quality to the Illumina format.

-J STR  Java. This string sets the location of the Java binary, this file should be present on every node in the cluster. If this is not set Halvade with use the default Java. This can be used if the default Java is 1.6 and GATK requires version 1.7.

--keep  Keep intermediate files. This option enables all intermediate files to be kept in the temporary folder set by --tmp. This allows the user to check the data after processing.

--lb STR  Read Group Library. This string sets the Read Group Library which will be used when adding Read Group information to the intermediate results. [LIB1]

--mapmem INT  Map Memory. This sets the memory available for the containers assigned for the map tasks.

--merge_bam  Merge BAM output. With this option set, Halvade will not perform variant calling but only read alignment. All alignments will be merged into 1 output BAM file.

--mpn INT  Maps per node. This overrides the number of map tasks that are run simultaneously on each node. Only use this when the number of map containers per node does not make sense for your cluster.

--elprep  elPrep. Use elPrep in the preprocessing steps, by default Picard is used which is a slower but requires less memory. ElPrep provides a more efficient execution of the preprocessing algorithms.

--pl STR  Read Group Platform. This string sets the Read Group Platform which will be used when adding Read Group information to the intermediate results. [ILLUMINA]

--pu STR  Read Group Platform Unit. This string sets the Read Group Platform Unit which will be used when adding Read Group information to the intermediate results. [UNIT1]

--redistribute  Redistribute Cores. This is an optimization to better utilize the CPU cores at the end of the map phase, to improve load balancing. Only use when the cores per container is less than 4.

--redmem INT  Reduce Memory. This sets the memory available for the containers assigned for the reduce tasks.

--remove_dups  Remove Duplicates. This will remove the found PCR duplicates in the corresponding step.

--report_all  Report all output. This option will give all VCF output records in the merged output file. By default the VCF record with the highest score will be kept if multiple records are found at the same location.

--rna  RNA pipeline. This options enables Halvade to run the RNA-seq pipeline instead of the default DNA pipeline. This option requires an additional argument S which points to the STAR genome directory.

9.2. Optional options
--rpn INT  Reduces per node. This overrides the number of reduce tasks that are run simultaneously on each node. Only use this when the number of reduce containers per node does not make sense for your cluster.

-S, --star STR  Star genome. This gives the directory of the STAR genome reference.

--sec INT  stand_call_conf. The value of this option will be used for the stand_call_conf when calling the GATK Variant Caller.

--sec INT  stand_emit_conf. The value of this option will be used for the stand_emit_conf when calling the GATK Variant Caller.

--single  Single-end reads. This option sets the input to be single-ended reads. By default, Halvade reads in paired-end interleaved FASTQ files.

--sm STR  Read Group Sample Name. This string sets the Read Group Sample Name which will be used when adding Read Group information to the intermediate results. [SAMPLE1]

--smt  Simultaneous multithreading. This option enables Halvade to use simultaneous multithreading on each node.

--stargtf STR  GFF for STAR. This option point to the GFF/GTF file to be used when rebuilding the STAR genome, this can improve accuracy when finding splice sites.

--tmp STR  Temporary directory. This string gives the location where intermediate files will be stored. This should be on a local disk for every node for optimal performance.

-U, --unifiedgenotyper  UnifiedGenotyper. With this option Halvade will use the UnifiedGenotyper tool from GATK instead of the HaplotypeCaller tool, which is used by default. The UnifiedGenotyper is faster but less accurate.

--update_rg  Update read group. This forces the readgroup to be updated to the one provided by the options, even if the input is read from a BAM file with a read group present.

-v INT  Verbosity. This sets the verbosity level for debugging, default is [2].
10.1 Find the logs to solve Halvade errors

If Halvade doesn’t finish due to an error, the error itself is printed in the output of the Hadoop command. However, more information can be found in the individual task stderr logs of the MapReduce job. The location of these log files is set in the MapReduce settings. Typically these are stored at $yarn.log.dir/userlogs or if the YARN_LOG_DIR environment is set under $YARN_LOG_DIR/userlogs. It’s highly likely that all reduce tasks give a similar result, so look at the stderr log of any reduce task. This log will show where Halvade is running into problems. If it isn’t clear from the log, try to run the last command, with the error, manually. The exact command should be printed in the log as an array of strings, run this command with the shown option.

10.2 Halvade having a reference file error while downloading/loading

It is possible that Halvade isn’t downloading the files correctly after a Halvade job has been terminated before it has finished. The reference files are downloaded to a single node by using a locking file system. A value of one is written to the locked file if the reference file has been downloaded and zero otherwise. If this occurs it is best to delete the tmp folder or these locked files (filez of only a few bytes in size) so that Halvade can go through the downloading process correctly.

10.3 Halvade with BAM input seems stuck at the MarkDuplicates step

If the stderr log of a reduce task shows it started the MarkDuplicates file but didn’t finish in a considerable time. Then it is highly likely it is finding incorrect aligned reads and giving error messages that slow the process to the point where it seems stuck. If this is the case, look at the header file of the BAM file and the fasta dictionary file, if the contig order is different then this is the source of the problem. Halvade assigns the contig by index in the sequence dictionary when reading the bam files and this causes wrong matching between steps. To fix this, the sequence dictionary of the bam files needs to be reordered the same as the dict file. This can be done with the ReorderSam tool by Picard like this:
Use the new file as input and Halvade should run without any problems now.