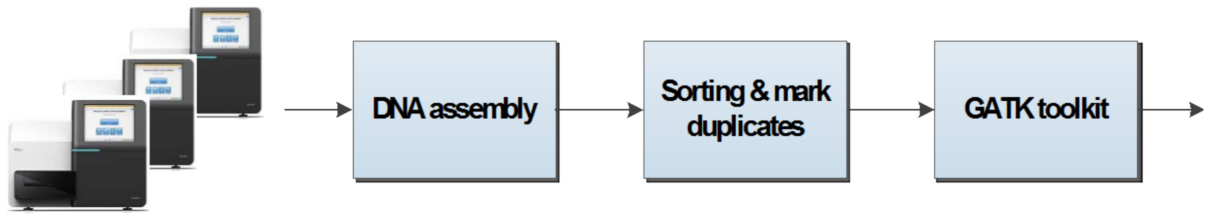
Supercomputing for Big Data ET4310 (2016)

Assignment 3

Hamid Mushtaq and Zaid Al-Ars

Background

Fast progress in next generation sequencing of DNA has resulted in the availability of large DNA data sets ready for analysis. However, DNA analysis has become the bottleneck in using these data sets, as it requires powerful and scalable tools to perform the needed analysis. A typical analysis pipeline consists of a number of steps, not all of which can readily scale on a distributed computing infrastructure. In this assignment, you will create a framework that implements an in-memory distributed version of the GATK DNA analysis pipeline using Apache Spark.



The figure shown above shows a typical DNA analysis and variant calling pipeline. The input data set to the pipeline is generated by sequencing a DNA sample in a sequencing machine and acquiring the DNA sequencing data. This is done in a massively parallel fashion with millions of short DNA pieces (called *short reads*) being sequenced at the same time. These reads are stored in large files of sizes in the range of hundreds of giga bytes. The DNA is usually over-sampled, resulting in generating multiple short reads for each segment of the DNA, typically with a coverage ranging from 30x to 80x, depending on the experiment. One standard file format used today to store these reads is called the FASTQ file format. Once the input data becomes available, the first step in the DNA analysis pipeline is to align the sequenced reads to a human reference genome and reconstruct the sequenced genome from the short read sequences, a process that is called *DNA assembly*. A well-known program used to perform this step is the Burrows Wheel Aligner, typically using the BWA-MEM algorithm. The results of this assembly step are stored in a file with the SAM file format.

After assembly, the short reads in the SAM file are sorted and duplicated reads are marked to avoid using them in downstream analysis. This reduces the bias effect resulting from the way the DNA sample is prepared and sequenced. A widely-used tool to perform this step is the Picard tool. The output of this tool is a compressed file format called BAM. Subsequently, a number of steps are performed to refine the read data and finally to identify the DNA mutations (or so-called variant calling). *GATK toolkit* is a commonly-used toolkit for this type of analysis that contains the following tools: Indel realignment, Base recalibration and the Haplotype caller. The output of this tool set is a VCF (variant call format) file with the variations in the sequenced DNA. These widely-used tools described here make part of the Broad best practices pipeline, widely used for variant calling both in research and in the clinic. You will use the GATK pipeline for demonstrating the distribution capabilities of Spark. If interested, you can find more info over GATK here https://www.broadinstitute.org/gatk/

System requirements and Installation

The requirements for software packages are the same as were for the previous labs except that for part 3 of the assignment, you need to make sure that your spark's hadoop version matches the hadoop version that would be installed on the Kova machine.

While part 1 can be done on any computer with at least 3 GB of RAM, for part 2 and 3, you will need access to the Kova (kova-01.ewi.tudelft.nl) machine. Make sure though that you do not run your program with more than 32 GB, since using too much memory would interfere with the programs of other users.

Important points

You must keep the following points in mind for this assignment.

- The division of marks is the following
    - Code functionality (40%)
    - Code (Readability, structure, performance, etc.) (20%)
    - Comments (25%)
    - Report (15%)
- Each statement in your code must be accompanied by a comment, especially on each RDD transformation or action. This is so that you could demonstrate that you thoroughly understand your code, even if you took a bit of help from somewhere. Also, for each RDD, you must specify the type of its elements in your comments. For example

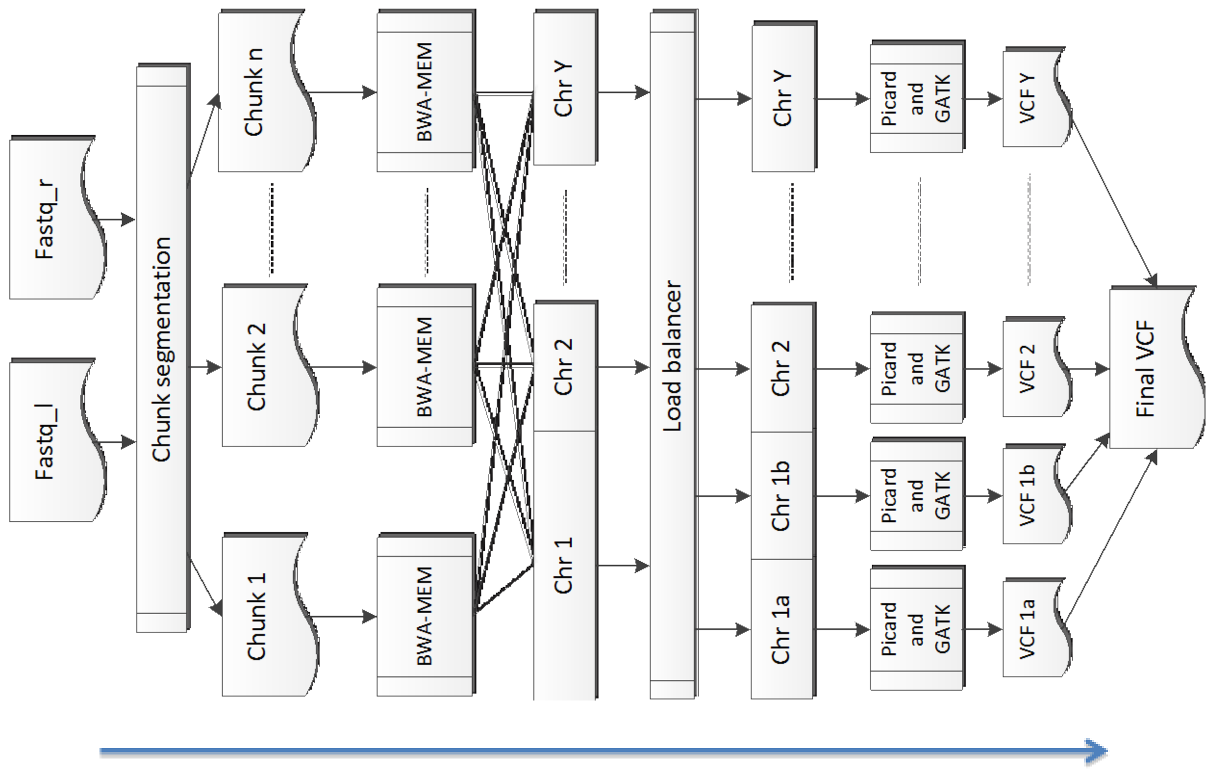
  ```
  // This is an RDD that describes … The contents of the RDD describes ((chromosome-number, pos), sam-record))
  SomeRDD = …
  ```
- In your report, you don't need to go too much into details, as through your comments, you should already be able to explain your code quite well. Secondly, the report should be no longer than 6 pages. You must use the following template for the report.

  https://www.dropbox.com/s/h9sqvr9afj0iigl/Report.docx?dl=0

  Make sure also that you follow the points given in the template.
- You must submit a folder (in zipped form) containing the code and your report. That zip file should be named as **SBD_L3_YourName_YourStudentNumber**. You must use the template provided for the exercise. You can modify the run script and add more source files if required though.
- All three parts of the assignment should be provided in separate folders, namely **part1**, **part2** and **part3**.
- To allow fair usage of the server, never use more than 32 GB on the Kova machine.

The assignment

This lab assignment will replace the regular GATK pipeline with a distributed Spark version.



Time progression of the pipeline

The assignment tackles this challenge by using the Apache Spark big data infrastructure, as shown in the figure above. In a typical DNA sequencing experiment, two FASTQ files are generated, that represent the 2 ends of a pair of DNA sequences. These 2 input FASTQ files are divided and merged into interleaved chunks using the chunks segmentation tool, written in Scala and using Spark, that is also able to upload these files into HDFS, thereby making them accessible to all nodes in a distributed cluster. Each chunk is then processed by a separate BWA-MEM task that performs DNA alignment of the short reads against a reference genome. The output of this step represents a list of read alignments (so-called SAM records) that would normally be stored in a SAM file. These records are then read into <key, value> pairs in the memory and regrouped into sub-chromosomal regions by a load balancer according to the number of reads per chromosome. This ensures a better distribution of the subsequent tasks and a better utilization of the computational cluster.

Subsequently, these records are sorted using the position of the aligned reads on each chromosome region (which replaces the Picard sort step in the pipeline). Then the rest of the GATK tool set is performed in parallel by executing it on each chromosomal region separately, resulting in various VCF files. Finally, the content of those VCF files are merged into one with all variants identified in the analysis.

Part 1: Chunking the input Fastq files (30 points)

The first step is to divide the two provided FASTQ input files into many smaller chunks that are used as input to the parallelized GATK Spark pipeline. The two input files are placed at **/data/spark/fastq** on the Kova machine. If you do not want to use Kova for this assignment, you can also download these two files from https://www.dropbox.com/s/f2i5g1yyckcgod3/fastq.tar.gz?dl=0 to the computer where you want to run this program.

You have to write a program in Scala using Spark, that creates interleaved chunks from data of these two files. Note that each DNA short read consists of 4 lines in the FASTQ files. You have to interleave such that the first read of fastq2.fq should come immediately after the first read of fastq1.fq, the second of fastq2.fq after the second of fastq1.fq and so on. Therefore, the interleaved content would look as follows.

Read1_of_fastq1.fq
Read1_of_fastq2.fq
Read2_of_fastq1.fq
Read2_of_fastq2.fq
Read3_of_fastq1.fq
……………………………..
ReadN_of_fastq1.fq
ReadN_of_fastq2.fq

You have to use the following template for this part. https://www.dropbox.com/sh/77h78rsyzuda1bs/AAA7uOR41ZMoNnwzlCqOef3Aa?dl=0
The run bash script for this code takes three arguments. The first being the number of chunks (number of parallel tasks used in the program must be equal to the number of chunks) to make (we recommend making 8 chunks), the second one being the input folder where the fastq files are placed, and the third one being the output folder where the output chunks would be placed. The output chunks must be compressed using gzip by your program. Therefore, there extension would be **.fq.gz**.

You are not allowed to modify the FASTQ files by inserting delimiters. This means, you must be able to separate reads without using any delimiters. Moreover, any form of sequential processing is not allowed. Furthermore, all the chunks must be written in parallel using either of **foreachPartitions**, **mapPartitions** and **mapPartitionsWithIndex** functions.

Part 2: DNA Sequence Analysis (50 points)

In this step, you are going to implement the DNA Sequence Analysis program itself. Start with the template code given at https://www.dropbox.com/sh/lisr444l6i99vgl/AAA9OFsm_JHBppcXhAU-22t8a?dl=0. The python script to run this program doesn't take any arguments, as a config

(**config.xml**) file is used instead to set the properties of the program. The details of each configuration in that file is given below.

- **refFolder**: The reference files used by the GATK pipeline. We have already put all the reference files in **/data/spark/ref** folder of the Kova machine, so you don't need to change that. In a cluster (That is going to be used in part 3), these reference files also need to be copied on each node, but since copying these files would take too much disk space, you can also use the same folder for Part 3 of this assignment.
- **toolsFolder**: All the tools used by the GATK pipeline. We have already placed all the tools in **/data/spark/tools** folder of the Kova machine, so you also don't need to change that. For execution on a cluster (Part 3), each tool must be downloaded to the temporary folder of the node where it is being used.
- **tmpFolder**: All the intermediate files produced by the program should be placed here. Your program must delete these files when no longer required. For execution on a cluster (Part 3), there would be such folder on each node.
- **inputFolder**: In case of local mode execution (This part of the assignment), this represents the local folder containing input chunk files used by the parallelized GATK pipeline (The files that you produced with the program of part 1). In case of cluster execution mode (Part 3 of the assignment), this folder would be located in HDFS. In that case, the files would have to be downloaded in the **tmpFolder** first and read from there.
- **outpuFolder**: Folder in which the vcf files (including the final combined vcf file) would be produced. In case of cluster mode, this folder would be located in HDFS. Please use the name **output.vcf** for the final output file.
- **numInstances**: Number of parallel tasks (where each task could be multithreaded) to execute. In cluster mode, this is equal to the number of executors (or containers), which are the different parallel processes that Spark executes. For fair usage on the Kova machine, do not set this value to more than 4.
- **numThreads**: Number of threads used by bwa and GATK programs. For fair usage on the Kova machine, do not set this value to more than 4.

Note that the Memory string and reference file names are hard coded as constants at the top of the code. Write the program using the following steps.

- The name of the input chunk files for BWA should be read from the **inputFolder** by having the program search for all the files in that folder. Note that you are not allowed to hard code the input file names in the program. For cluster mode (Part 3), this means that these files would be searched from an HDFS folder.
- The BWA step should output data in the form **<chromosome number, SAM record>**. Since, there are 24 number of chromosomes in total, and you would be using maximum 4 tasks, you have to assign a particular region to each chromosome, such that you have in total as many chromosome regions as **numInstances**. Also, each chromosome region should have almost the same number of mapped reads (SAM records). This you would have to do by using load balancing. After load balancing, you must have key value pairs of the form **<chromosome region, SAM record>.**

- For each chromosome region, perform variant calling. First you would have to group the SAM records for each region. Inside the variant calling, you also have to sort the SAM records by position. For this, you would first have to write a method for comparing the SAM records. In that method, first you compare the SAM records by their chromosome numbers. If the chromosome numbers are the same, you have to compare them using their starting positions (returned by **getAlignmentStart** method of the **SAMRecord** class). The data returned by variant calling should be of the type **<Integer, <Integer, String>>**, where the key is the chromosome number, and the key of the value is the starting position of the read (SAM record). The String here represents a line of the output *.vcf file.
- From the data returned by variant calling, a combined *.vcf file should be produced, such that the lines are sorted according to chromosome numbers, and for each chromosome, further sorted by positions of the reads (SAM records).
- Check the accuracy and correctness of the output *.vcf file by using the **comparison.py** script given in the **compare** folder. First argument should be **ref.vcf** (also provided in the **compare** folder) while the second argument should be the *.vcf file produced by your program. The difference showed by **comparison.py** should be less than 35.
- The **config** object must be broadcasted to all nodes using **sc.broadcast**.
- You also have to implement code to perform logging of the execution. Put your logs in a folder **outputFolder/log**. For bwa, make a folder **outputFolder/log/bwa** while for variant calling, create a folder **outputFolder/log/vc**. There would be a separate log files for each task. For example, if you name a bwa task by its chunk name, you could have a log file **outputFolder/log/bwa/chunk1_log.txt**. In these logs, you must print the commands used for executing the different parts, along with the timestamps. There is no hard and fast rule on how your log should look like though. For example, a part of log of bwa could look like something as follows.

```
[17:31:42]	gunzip -c /tmp/spark/chunk9.fq.gz. Size of input file = 175451507 bytes
[17:31:52]	bwa mem started: /tmp/spark/bwa mem /tmp/spark/human_g1k_v37_decoy.fasta -p -t 4 /tmp/spark/chunk9.fq
[17:35:56]	bwa mem completed for chunk9.fq. Number of key value pairs = 26839
```

  Similarly, a variant calling log could look something like this.

```
[15:30:55]	java -Xmx4864m -jar /tmp/spark/GenomeAnalysisTK.jar -T RealignerTargetCreator -nt 4 -R /tmp/spark/human_g1k_v37_decoy.fasta -I /tmp/spark/7_2.bam -o /tmp/spark/7_2.intervals -L
/tmp/spark/7_2.bed
[15:31:09]	java -Xmx4864m -jar /tmp/spark/GenomeAnalysisTK.jar -T IndelRealigner -R /tmp/spark/human_g1k_v37_decoy.fasta -I /tmp/spark/7_2.bam -targetIntervals /tmp/spark/7_2.intervals -o
/tmp/spark/7_2-2.bam -L /tmp/spark/7_2.bed
[15:32:01]	java -Xmx4864m -jar /tmp/spark/GenomeAnalysisTK.jar -T BaseRecalibrator -nct 4 -R /tmp/spark/human_g1k_v37_decoy.fasta -I /tmp/spark/7_2-2.bam -o /tmp/spark/7_2.table -L
/tmp/spark/7_2.bed --disable_auto_index_creation_and_locking_when_reading_rods -knownSites /tmp/spark/dbsnp_138.b37.vcf
[15:33:22]	java -Xmx4864m -jar /tmp/spark/GenomeAnalysisTK.jar -T PrintReads -R /tmp/spark/human_g1k_v37_decoy.fasta -I /tmp/spark/7_2-2.bam -o /tmp/spark/7_2-3.bam -BQSR
/tmp/spark/7_2.table -L /tmp/spark/7_2.bed
[15:35:33]	java -Xmx4864m -jar /tmp/spark/GenomeAnalysisTK.jar -T HaplotypeCaller -nct 4 -R /tmp/spark/human_g1k_v37_decoy.fasta -I /tmp/spark/7_2-3.bam --genotyping_mode DISCOVERY -o
/tmp/spark/7_2.vcf -stand_call_conf 30 -stand_emit_conf 10 -L /tmp/spark/7_2.bed --no_cmdline_in_header --disable_auto_index_creation_and_locking_when_reading_rods
[15:36:32]	Output written to vcf file
```

  For cluster mode (Part 3), these logs would obviously be on the HDFS.

Part 3: Porting the solution to a cluster (20 points)

In this part, you have to modify the program in part 2, so that it runs with a Hadoop cluster (using the command flag "--master yarn-cluster" with spark-submit) and HDFS. The meaning of the configurations in **config.xml** for this part are already explained in the text for Part 2 of this assignment. So have a look at that, before implementing your code for this part.

For this part, you will use the pseudo-distributed mode of Hadoop that would be installed on the Kova machine. Besides the code, you would also have to modify your **run.py** script to be able to run the program in cluster mode. Give both **--driver-memory** and **--executor-memory** a suitable value. Moreover, **--num-executors** should be equal to **numIntances**. Note that the input and output folders in this case would be in the HDFS. So, you must write code to read and write files to and from HDFS. Write these functions in a separate file, either in Java or Scala.