Genome Analysis with MapReduce

Merina Maharjan

June 15, 2011

Abstract

Genome sequencing technology has been improved intensely, but the number of bases generated by modern sequencing techniques has also been growing at an exponential rate. There are next generation sequencing technologies working with large data sets of genome data, such as the 1000 Genomes Project. In the report we discuss analysis tools for next generation DNA sequencing data using a structured programming framework called the Genome Analysis Toolkit (GATK). This framework is used to build genome analysis tools easily and efficiently by using the functional programming paradigm called MapReduce. MapReduce is a distributed programming framework which is used for processing and extracting knowledge from large data sets. The report also describes Hadoop, which is a software scheme for cloud computing that provides MapReduce functionality for computing clusters and Hadoop Distributed File System for storing large data. The report illustrates the use of MapReduce in Hadoop by running the simple WordCount class and an application called Hadoop-BAM. As a part of the results of the report, we describe the execution of a simple custom built example with GATK.

Contents

1 Introduction

Cloud computing offers new approaches for scientific computing and is already used in several ways. It is particularly interesting for the field of bioinformatics where cloud computing has the ability to store large data sets in the cloud. There are large data sets in bioinformatics which are measured in terabytes. Cloud computing offers answers for many of the constraints encountered when dealing with extremely large data sets [7].

In bioinformatics, many applications of cloud computing are being developed using Hadoop and MapReduce [15]. For example, the Cloudburst software [12], is based on a parallel read-mapping algorithm optimized for mapping next-generation sequencing data to the human genome and other reference genomes. Hadoop is a software layer for cloud computing used to distribute application data, and to parallelize and manage application execution across the computers. It is also used for detecting machine failures and then recovering application data. *Hadoop is widely used in different areas with significant amount of data such as finance, technology, telecom, media and entertainment, government, research institutions, bioinformatics, and other markets with significant data* [4]. This report gives a brief introduction to Hadoop, Hadoop Distributed File System (HDFS), and MapReduce.

The report also focuses on the MapReduce framework in bioinformatics with the example of a framework called Genome Analysis Toolkit (GATK). GATK is a framework for analysing and accessing next generation sequencing data. GATK breaks up terabases (amount of DNA sequence data) into smaller and more manageable kilobases sized pieces, called shards. Bases are the nitrogen-containing components of the DNA molecule. There are four types of bases; adenine (A), thymine (T), guanine (G), and cytosine (C). The shards for example contain all the relevant information about single nucleotide polymorphisms (SNPs). Here, SNPs are variations occurring in DNA sequences when there is a difference in single nucleotide in the genome between members of a biological species or paired chromosomes in an individual [18]. The GATK is based on the MapReduce framework developed by Google [8]. The architecture of MapReduce, HDFS, and GATK is outlined in the report.

The report is organized as follows; Section 2 gives an introduction to Hadoop, MapReduce, HDFS, and GATK. Section 3 shortly discusses the work flow of MapReduce, HDFS, and GATK. Section 4 explains the set-up required for the installation of Hadoop and GATK. Section 5 discusses the results of the report with the example programs of Hadoop and GATK. Finally, Section 6 concludes the report.

2 Background

2.1 Hadoop

Hadoop is a flexible infrastructure for large scale computing and data processing on a network of commodity hardware. It supports large data distributed applications and allows applications to work with thousands of nodes.

Hadoop is an Apache Software Foundation project. Apache Hadoop[1] is an open source software platform for distributed processing of large data sets across clusters of computers using a simple programming model. This distributed computing platform is written in Java and originally created by Doug Cutting [5].

Hadoop includes sub-projects such as:

Hadoop MapReduce is used for processing and extracting knowledge from large data sets on compute clusters.

HDFS is a scalable and distributed file system used for file storage. It supports a configurable degree of replication for reliable storage and provides high throughput access to application data. HDFS is inspired by the Google File System (GFS).

HBase is a distributed database that supports storage of large tables and runs on top of HDFS.

Pig and Hive are used for data analysis. Pig is a high level language running on top of MapReduce. It is an execution framework for parallel computing. Hive is running on top of Hadoop and provides database functionality.

Hadoop consists of a package called Hadoop Common which supports the above mentioned sub-projects. Hadoop Common includes file system and serialization libraries. This package contains the jar files, scripts and documentation necessary to run Hadoop.

Hadoop provides a reliable shared storage and analysis system. The storage is provided by HDFS, and the analysis by Hadoop MapReduce [16]. So, HDFS and Hadoop MapReduce are the most commonly used sub-projects of Hadoop.

2.2 Hadoop Distributed File System

HDFS is the file system component of Hadoop. It is a distributed, scalable and reliable primary storage system. HDFS is written in Java and was inspired by the Google File System (GFS) paper published by Google in the year 2004 [10]. GFS is a scalable distributed file system for large distributed data intensive applications. According to the GFS paper, GFS *was designed and implemented to meet the rapidly growing demands of Google's data processing needs.*

HDFS stores file system metadata and application data separately. As in other distributed file system, like GFS, HDFS stores metadata on a Namenode server and application data on Datanode servers [11]. The Namenode, which is a dedicated server, is responsible for maintaining the HDFS directory tree and it is a centralized service in the cluster operated on a single node. The Datanode is responsible for storing HDFS blocks on behalf of local or remote clients.

One of the main advantages of using HDFS is the data awareness between the jobtracker and tasktrackers in Hadoop MapReduce. The jobtracker schedules MapReduce jobs to tasktrackers with awareness of the data location. This helps to reduce the amount of traffic in the network and also prevents the transfer of unnecessary

[1] http://hadoop.apache.org/

data. HDFS also provides global access to files in a cluster. Files in HDFS are divided into large blocks, typically of size 64MB, and each block is stored as a separate file in the local file system [13].

2.3 MapReduce

MapReduce is a programming paradigm for handling large data in a distributed computing environment. According to Dean and Ghemawat [8], *MapReduce is a programming model and an associated implementation for processing and generating large data sets.* It is an application framework that allows programmers to write functions to process their data. This technology was created at Google in the year 2004 to manage large computing infrastructures in a scalable way.

MapReduce breaks the processing functions into two parts; the map function and the reduce function. The programmer can specify these functions. The map function processes a key and value pair to output a set of intermediate key and value pairs, and the reduce function combines all intermediate values associated with the same intermediate key. The key here is the offset in the file and the value is the contents of the line read. The functions provided by the programmer have associated types:

Map map(input_key1, input_value1)

list(output_key2, output_value2)

Reduce reduce(output_key2, list(output_value2))

list(output_value2)

MapReduce schemes transform lists of input data elements into lists of output data elements. The first task of MapReduce scheme is mapping. In this task, a list of data elements is provided, one at a time to a map function, which transforms each element individually to an output data element. The second task is called reducing, where values are added together. The reduce function gets an iterator of input values from an input list, then it combines the values together and returns a single output value. Let us take the example of counting the number of occurrence of each word in a document. The map function outputs each word with an associated count of occurrence. The reduce function sums together all the counts passed for a particular word. In Section 5.1, we shall elaborate more on word count example regarding Hadoop MapReduce.

In Hadoop, a centralized JobTracker is responsible for splitting the input data into pieces for processing by independent Map and Reduce tasks. These tasks are scheduled on a cluster of nodes for execution. On each node there is a TaskTracker that runs MapReduce tasks and periodically contacts the JobTracker to report the task completions and request new tasks.

In GATK, MapReduce programming model is used to develop analysis tools so that it will be easy to parallelize and distribute processing, and easy to extract information from genome data sets. With GATK, there is only thread-wise parallelism, and all tasks are run on the same physical machine. The GATK is structured into traversals and walkers. The traversals provides a bunch of data to the analysis walker, and the

walker provides the map and reduce methods that consumes the data. Section 3.3 and 5.3 gives a description of the traversal and the walker.

2.4 Genome Analysis Toolkit

Next generation DNA sequencing projects are revolutionizing new software tools in DNA sequencing technology field to analyse the massive data sets generated by Next generation sequencing (NGS) [14]. The first genome project based on the NGS platform is the 1000 Genomes Project, which is cataloguing human genetic variation [1]. In our report, we discuss a structured programming framework called Genome Analysis Toolkit (GATK) which is used for developing efficient tools for analysing large genome data sets. GATK is developed by the Genome Sequencing and Analysis Group from Broad Institute of MIT and Harvard University[2]. This framework helps developers and researchers to develop efficient and robust analysis tools for next generation DNA sequencing.

GATK has been used in human medical sequencing projects, for example the Cancer Genome Atlas [9]. It helps in analysing the NGS data and solving the data management problem by separating data access patterns from analysis algorithms. One feature of GATK is that it breaks up terabases of sequence data into shards, which contain reference data and information about SNPs. Another feature is that it provides several options for users to parallelize tasks. *"With interval processing, users can split tasks by genomic locations and farm out each interval to a* GATK *instance on a distributed computing system, like the Sun Grid Engine or load sharing facility,"* the authors write in the Genome Research paper [9].

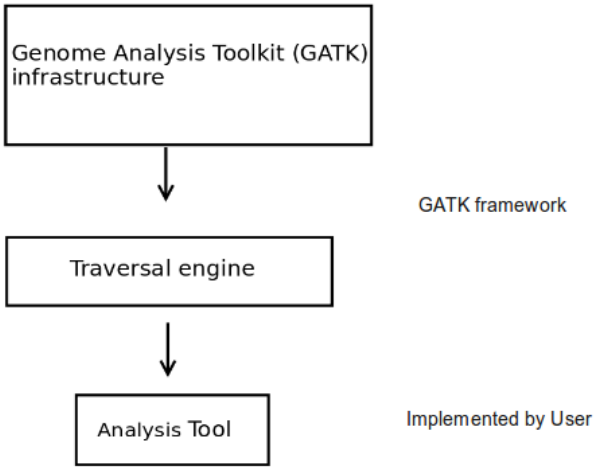


Figure 1: The Genome Analysis Toolkit(GATK) structure

Other features are the following; it lets users combine several BAM (Binary Alignment Map) files, and combine multiple sequencing runs and other input files into a single analysis. The GATK also provides various approaches for the parallelization of

[2]http://www.broadinstitute.org/

tasks. It supports automatic shared memory parallelization and manages multiple instances of the traversal engine and the walker on a single machine.

Figure 1 shows that, the GATK provides an infrastructure and Traversal engine to the users and the users can implement their own specific analysis tools on top of this framework. The GATK framework is designed in a way that can support the most common paradigms of analysis algorithms. GATK separates data access patterns from analysis algorithms, and provides users with a set of data traversals and data walkers which together can provide a programmatic access to analytical tools. The data traversal provides a series of units of data to the walker, and the walker uses each data and generates an output for them. The paper on *"The Genome Analysis Toolkit"* [9], lists the types of traversal that are available in GATK. They are: TraverseLoci, TraverseReads, TraverseDuplicates, and TraverseLocusWindows. This report discusses TraverseLoci and TraverseReads in detail in Section 3.3.

The analysis tool is separated from the common data management GATK infrastructure so that it is easy for the users to write their own specific tools, to analyse the data and then map them across the data. The toolkit is available as open source software on GATK's website [2].

3 Architecture

3.1 HDFS Data Flow

The Hadoop Distributed File System consists of a namenode, a filesystem image and several datanodes. Whenever the clients request data blocks, a request is sent to the namenode. The namenode looks for the blocks in the metadata. Metadata here is the File System (FS) image that has all the distributed data. The namenode finds the replica information for the file from the metadata and sends the list of datanode addresses and block information to the client. In the example from Figure 2, if the

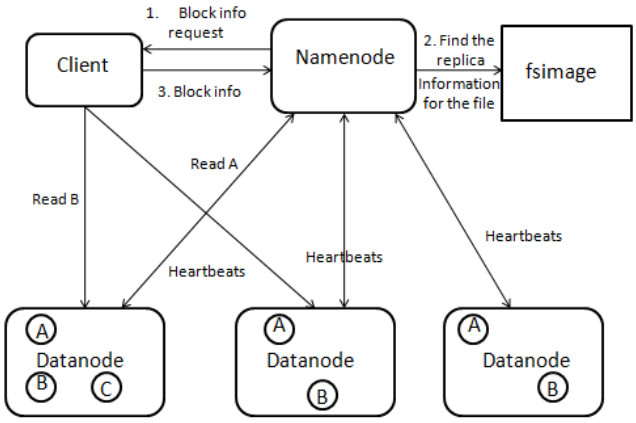


Figure 2: Hadoop Distributed File System Data Flow

client requests the data block named B, the namenode finds the information from the metadata and sends the information to the client that block B is in all three datanodes

that the namenode provided. But here the client can read the data from only one datanode and it does not have to read from other datanodes. If the client sends the request for blocks A, B, and C that belong to the same file, then the client can read A from the first datanode, B from the second datanode, and C from the third datanode simultaneously. This makes the read speed fast as the client can read from three datanodes in parallel. HDFS can also replicate the blocks of the file. This feature is used for fault tolerance which means that the distributed file system can automatically recover the datanodes from crashes because they have block replication. The number of copies of blocks replication can be specified while creating the file otherwise it takes 3 as default.

The namenode is the master node of the HDFS, but it is used only for finding the information and passing it to the client. The main work is done in datanodes. This system has the advantage of not overloading the namenode. Here, heartbeat messages are used by the namenode to check that datanodes are still alive.

3.2 MapReduce Data Flow

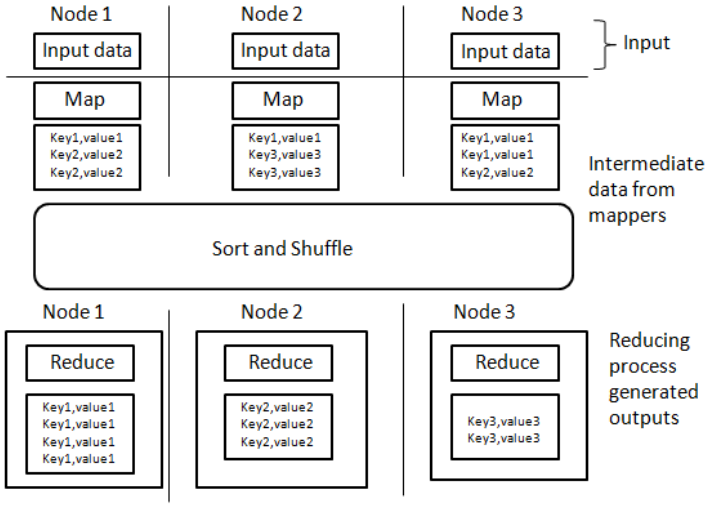


Figure 3: MapReduce Data Flow

MapReduce inputs comes from input files loaded onto a processing cluster in a distributed file system. These files are evenly distributed across all nodes as shown in Figure 3. All nodes in the cluster have mapping task running on them. These mapping tasks are equivalent, so any mapper can process any input data. The map function processes the data and presents its output as key and value pairs. After all map functions have completed, MapReduce allocates the values according to the keys to the reducers. The reducer tasks are spread across the same nodes in the cluster as the mappers. Each reducer first shuffles and sorts all the keys, and then runs the reduce function over them. The reduce function finally outputs key and value pairs. This output of reduce can be stored in HDFS for the reliability and can be used by another MapReduce job as separate program.

There is no explicit communication between mappers in different nodes and also between reducers in different nodes. All the data are scattered in the distributed file system and MapReduce tasks are run closer to the data. The sorting of the keys after the map phase helps to control the flow of data. Duplicate keys always arrive to one reducer and any problems that can be mapped to MapReduce programming paradigm can be easily computed in the cluster. However, all problems may not necessarily be suitable for the MapReduce programming paradigm.

3.3 GATK Architecture

GATK is structured into traversals and walkers. The traversals provide a sequence of associated sets of data to the walkers, and the walker provide the map and reduce functions that consumes the data and gives an output for each set to be reduced. The MapReduce framework is used in GATK so that it can split computations into two steps.

1. First, the large problem is subdivided into many discrete independent pieces and passed through the map function.

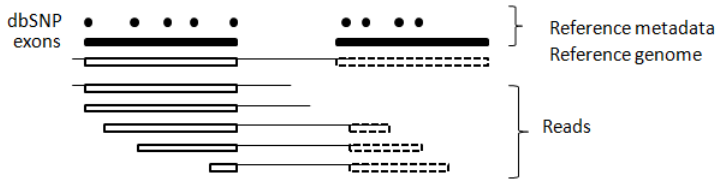
2. Second, the reduce function gives the final result by combining the map results.

Figure 4 shows the methods for accessing data for several analysing tools such as for counting reads, reporting average coverage of sequencer reads over the genome, building base quality histograms, and calling SNPs [9]. The GATK takes a reference genome as an input, which is in FASTA format. In FASTA format, a sequence is represented as a series of lines and are typically of length 80 or 120 characters [17]. The reference metadata are associated with the positions on the reference genome. In addition to the reference genome and reference metadata, there are short subsequence of genome data called reads. Reads are in Sequence Alignment Map (SAM) format. The binary version of the SAM format, which is called as Binary Alignment Map (BAM) format is compressed and indexed, and is used by the GATK. BAM format is used for the performance purpose in GATK, because of its smaller size and ability to be indexed for analyse.

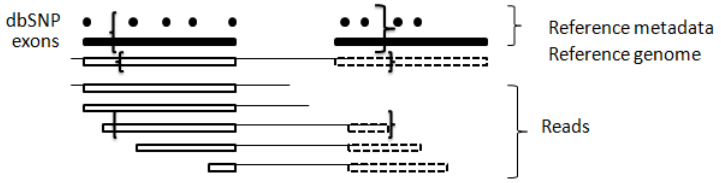
As mentioned in Section 2.4, GATK consists of read-based or locus-based traversals. The *read-based traversals* provide a sequencer read and its associated reference data during each iteration of the traversal. The TraverseReads is one of the types of read-based traversals applicable in GATK. In TraverseReads, each reads and associated reference data are passed to analysis walker only once. The method for accessing data in TraverseReads is by reading each sequence read.

The *locus-based traversals* provide the reference base, associated reference ordered data, and the pileup of read bases at the given locus[3]. One of the locus-based traversals that is applicable in GATK is called TraverseLoci. In TraverseLoci, each single base locus with its reads, referenced data, and reference base is passed to the analysis walker. The method for accessing data in TraverseLoci is by reading every read that is covering a single base position in the genome.

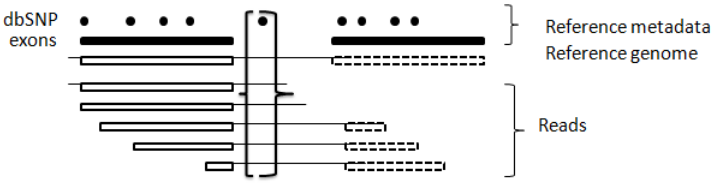
[3]Locus is the specific location of a DNA sequence on a chromosome.



(a) MapReduce over the genome



(b) MapReduce by read



(c) MapReduce by loci

Figure 4: MapReduce framework used over the genome by read-based and loci-based traversal in GATK [9].

In the input BAM file, the iterations are repeated respectively for each read or each reference base.

4 Implementation

4.1 Hadoop

Hadoop is supported by the GNU/Linux platform [5] and requires Java 1.6 or above installed on the system. In order to use Hadoop scripts that can manage remote Hadoop daemons, ssh must be installed.

There are three supported modes to operate Hadoop clusters,described as follows:

Local (Standalone) Mode:

Hadoop is by default configured to run in a non-distributed mode. This mode is run as a single Java process and is mainly useful for debugging.

Pseudo-Distributed Mode:

In pseudo-distributed mode, Hadoop is run as several Java processes. We have used pseudo-distributed mode to run the example programs of Section 5.1

and 5.2 on top of Hadoop. We have to change three configure xml files: conf/core-site.xml, conf/hdfs-site.xml and conf/mapred-site.xml.

The configuration used for the examples reads as follows:

conf/core-site.xml:

```
<configuration>
  <property>
    <name>hadoop.tmp.dir</name>
    <value>/tmp/${user.name}</value>
    <description>The name of the default file system.
    A URI whose scheme and authority determine the
    FileSystem implementation.
    </description>
    <final>true</final>
  </property>
</configuration>
```

conf/hdfs-site.xml:

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
    <description>Default block replication.
    The actual number of replications can be specified
    when the file is created. The default is used if
    replication is not specified in create time. The
    default value of dfs.replication is 3. However,
    we have only one node available, so we set
    dfs.replication to 1.
    </description>
  </property>
</configuration>
```

conf/mapred-site.xml:

```
<configuration>
  <property>
    <name>mapred.job.tracker</name>
    <value>localhost:9001</value>
    <description>The host and port that the MapReduce job
    tracker runs at. If "local", then jobs are run in-process
    as a single map and reduce task.
    </description>
  </property>
</configuration>
```

Fully-Distributed Mode:

In fully-distributed mode, Hadoop is run on a cluster of possibly many computers. In the cluster, one machine runs the Namenode and Jobtracker and the remaining machines are used as Datanodes and Tasktrackers.

4.2 GATK

The Genome Analysis Toolkit is supported by Linux and other POSIX compatible platforms. Since it is a Java based tool, in order to run GATK, it requires the Java version "1.6.0-16" JRE installed in the system. The system requires Java 1.6 JDK and Ant 1.7.1 (or greater) to develop the walker.

After installing all prerequisites, the GATK source can be downloaded from the ftp server of the Broad Institute provided on their website[4]. The version of GATK used in our experiments is *GenomeAnalysisTK-1.0.4418*.

5 Results

5.1 Hadoop Word Count Example

To briefly describe how to get started with Hadoop MapReduce after the installation, we now discuss the standard example called *WordCount* [6]. This example is about counting the number of occurrence of the words in a file using Hadoop MapReduce. The application works with a local-standalone, pseudo-distributed or fully-distributed Hadoop installation. WordCount.java is found in the hadoop-0.20.2 release in the folder "wordcount_classes".

In WordCount.java, the main() method is the driver for the MapReduce job, which configures how we want the job to be run. For this, we use the class *JobConf*, which allows to set parameters for the job. Its function takes two parameters that are input and output paths. These paths are specified in the JobConf object along with the input and output types of the keys and values. JobConf class allows to set parameters for the job.

```
public static void main(String[] args) throws Exception {
       JobConf conf = new JobConf(WordCount.class);
       conf.setJobName("wordcount");

       //the keys are String
     conf.setOutputKeyClass(Text.class);
     //the values are int
       conf.setOutputValueClass(IntWritable.class);

         //configure mapper, combiner and reducer classes
       conf.setMapperClass(Map.class);
       conf.setCombinerClass(Reduce.class);
       conf.setReducerClass(Reduce.class);
```

[4]http://www.broadinstitute.org/gsa/wiki/index.php/Downloading_the_GATK

```
//set the types for input and output
  conf.setInputFormat(TextInputFormat.class);
  conf.setOutputFormat(TextOutputFormat.class);

//specify the input and output paths
  FileInputFormat.setInputPaths(conf, new Path(args[0]));
  FileOutputFormat.setOutputPath(conf, new Path(args[1]));

//run the job
JobClient.runJob(conf);
}
```

In map function, we just split up the line into words using *StringTokenizer*, separate the word and the count, and store the count against the word. The code for the Map class is as follows:

```
public static class Map extends MapReduceBase implements Mapper
<LongWritable,Text, Text, IntWritable> {
      private final static IntWritable one = new IntWritable(1);
     private Text word = new Text();

      public void map(LongWritable key, Text value, OutputCollector
      <Text,IntWritable>output,Reporter reporter)throws IOException
      {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens())
        {
          word.set(tokenizer.nextToken());
          output.collect(word, one);
        }
      }
```

The Map class extends the MapReduceBase and implements the interface Mapper. The Mapper interface has four arguments: input key, input value, output key, and output value. The input key is an integer offset of the current line from the beginning of the file, the input value is the text of the line read which contains a row of data, output key is the word and output value is a number of count against a word.

Hadoop uses its own data types such as, Text for String, IntWritable for int and LongWritable for long data type. The map function uses OutputCollector object to receive the values to emit to the next phase of execution, and also uses Reporter object to provide the information of the current task.

```
public static class Reduce extends MapReduceBase implements Reducer
<Text, IntWritable,Text, IntWritable> {
      public void reduce(Text key, Iterator<IntWritable> values,
      OutputCollector<Text, IntWritable>output,Reporter reporter)
      throws IOException {
        int sum = 0;
```

```
  while (values.hasNext()) {
  sum += values.next().get();
  }
output.collect(key, new IntWritable(sum));
}
```

After the execution of a mapper, Hadoop collects the data from all the map outputs and forwards them to a number of reducer according to the key. For each key, all the values are returned in a single iterator. Then there is a Reduce class, which also extends the MapReduceBase and implements the interface Reducer. Each Reducer has words (data type Text) as an input key and an iterator of all the counts (data type IntWritable) for the word. The output key is again the word and the output value is the counts with data types Text and IntWritable respectively.

This example also uses the Combiner, which is configured in main method as "conf.setCombinerClass(Reduce.class)". If the document contains the word "good" 4 times, the pair ("good",1) is emitted 4 times, all of these are then forwarded to the reducer. By using a Combiner, these can be put into a single ("good",4) pair to be forwarded to reducer. This helps in reducing the total bandwidth required for the shuffle process, and speeding up the job.

Now to run the WordCount in hadoop, compile WordCount.java and create a jar. So, to execute the program we compile the folder, and set the path to its jar file. Commands to run the example are as follows:

```
Start the hadoop daemons:
$bin/start-all.sh

Make the directory in File system:
$bin/hadoop fs -mkdir new

Copy the input files into the distributed file system:
$bin/hadoop fs -copyFromLocal testt.txt new

Run the examples provided in hadoop-0.20.2:
$bin/hadoop jar hadoop-*-examples.jar wordcount new newout

Copy the output files from the distributed file system to
the local file system and examine them:
$bin/hadoop fs -text newout/part-r-00000

Stop the hadoop daemons:
$bin/stop-all.sh
```

The input file testt.txt contains:

```
happy good good happy world good good world world
```

The output of the WordCount from the testt.txt file is: we have words good, happy and world counted as 4, 2, and 3 respectively.

5.2 Hadoop-BAM

Hadoop-BAM[5] is a library which manages BAM (Binary Alignment Map) and BGZF compressed files using the Hadoop MapReduce structure. This application includes the program for indexing both BAM and BGZF files, so that these files become splittable by Hadoop, and also for sorting the BAM files.

Hadoop-BAM requires version 0.20.2 of Hadoop[6] installed in the system, as currently only this version is supported by the application. Hadoop-BAM uses version 1.27 of Picard[7] to read and write the BAM files.

To use the functionality of Hadoop-BAM, Picard's 'sam-1.27.jar' or above and Hadoop's 'hadoop-0.20.2-core.jar' are required in the CLASSPATH environment variable. So, the first step is to add the jar files to the CLASSPATH.

```
$gedit ~/.bashrc
export CLASSPATH=/path/hadoop-0.20.2-core.jar:/path/sam-1.27.jar
$bash
$echo $CLASSPATH
```

Then to build the Hadoop-BAM, we have to use Apache Ant (version 1.6 or greater) with the following command:

```
$ant jar
```

This will create the 'Hadoop-BAM.jar' file. This jar file can then be used also to call the non-distributed utility programs that are included in Hadoop-BAM. We need Hadoop-BAM.jar as well in CLASSPATH.

```
$gedit ~/.bashrc
export CLASSPATH=/path/Hadoop-BAM.jar
$bash
$echo $CLASSPATH
```

To use a BAM file with BAMInputFormat class, we have to run SplittingBAMIndexer. This will create a .splitting-bai file used by BAMInputFormat.

```
$java fi.tkk.ics.hadoop.bam.SplittingBAMIndexer 1024 exampleBAM.bam
```

Here, 1024 is the granularity used for SplittingBAMIndexer, which results in a 3-megabyte index for a 50-gigabyte BAM file. The granularity specifies the maximum amount of alignments that may be transferred from one Hadoop mapper to another. If Hadoop places a split point in the index file between two offsets, then it is rounded to one of them.

Hadoop-BAM has two utilities; BAMReader and BAMSort. These utilities are meant for usage within Hadoop. Before using the utilities, we have to set the environment variable HADOOP_CLASSPATH in hadoop_end.sh as:

```
export HADOOP\_CLASSPATH=/path/Hadoop-BAM-1.0/Hadoop-BAM.jar:
                         /path/Hadoop-BAM-1.0/sam-1.38.jar
```

[5] http://sourceforge.net/projects/Hadoop-BAM/
[6] http://hadoop.apache.org/
[7] http://picard.sourceforge.net/

In this report we have used BAMSort utility as an example. BAMSort is an application that uses Hadoop MapReduce to sort the inputs of a BAM file. The example BAM file is used as an input.

```
Start the hadoop daemons:
$bin/start-all.sh

Copy the input bam and .splitting-bai files into the distributed
filesystem:
$bin/hadoop fs -put /path/exampleBAM.bam exampleBAM.bam
$bin/hadoop fs -put /path/exampleBAM.bam.splitting-bai
exampleBAM.bam.splitting-bai
```

Run the utility (BAMSort):

```
$bin/hadoop jar Hadoop-BAM.jar fi.tkk.ics.hadoop.bam.util.hadoop.
BAMSort outputfile exampleBAM.bam
```

The output of this example is in Appendix A.

5.3 GATK Walker Example

From Section 2.4, we know GATK is designed in such a way that developers can easily write their own analysis tools. For example, to compute the average read length and to access every read and walk through them there is TraverseReads. To calculate the average read depth across the genome and to access information reference base and read bases at every base in the genome there is TraverseLoci. The GATK infrastructure also provides the code for indexing, retrieving, and parsing NGS data. GATK implements a MapReduce programming framework that allows analysis tasks to be performed in parallel.

The important idea behind the Genome Analysis Toolkit is the Walker class. The developers can write their own walker implementing the following three operations [3]:

- Filter, which reduces the size of the dataset by applying a predicate.
- Map, which applies a function to each individual element in a dataset by mapping it to a new element.
- Reduce, which recursively combines the elements of a list.

In Section 3.3, we discussed the GATK traversals and in this Section we describe the GATK walkers. There are two types of walkers provided by GATK; *LocusWalker* and *ReadWalker*. LocusWalker provides all reads, reference bases, and reference ordered data that recursively overlap a single base in the reference. ReadWalker provides only one read, reference base, and reference ordered data at a time that recursively overlaps to the read. Basically, a LocusWalker walks over the data set one location at a time and a ReadWalker walks over the data set one read at a time. The GATK engine will produce a result by first filtering the dataset, then running a map operation, and finally reducing the map operation to a single result.

There are some examples provided by the Genome Sequencing and Analysis Group (GSA) of the Broad institute which can be found in the release GenomeAnalysisTK-1.0.4418. For example, CountLociWalker.java and GATKPaperGenotyper.java. CountLociWalker counts the number of Loci walked over a single run of the GATK. GATKPaperGenotyper is a simple bayesian genotyper that outputs a text based call format. In this report, we have a simple example of a genome sequence analysis application built in GATK infrastructure by extending the LocusWalker. The genome sequence analysis application counts the occurrence of the reference genome sequence and matches the location where the sequences are located in the input BAM file.

We have used the example of an external walkers provided on the website of GATK [3] and have modified it according to our requirements. The main class of GATK is in org.broadinstitude.sting.gatk.CommandLineGATK. In order to use the GATK infrastructure and traversal engine, GenomeAnalysisTK.jar is required to be in the CLASSPATH environment variable.

```
package org.broadinstitute.sting.gatk.examples;

import org.broadinstitute.sting.gatk.walkers.LocusWalker;
import org.broadinstitute.sting.gatk.contexts.AlignmentContext;
import org.broadinstitute.sting.gatk.contexts.ReferenceContext;
import org.broadinstitute.sting.gatk.refdata.RefMetaDataTracker;
import org.broadinstitute.sting.utils.GenomeLoc;
import org.broadinstitute.sting.commandline.Output;

import java.util.List;
import java.util.ArrayList;
import java.io.PrintStream;

public class GenomeAnalysisWalker extends LocusWalker<Integer,Long>
 {
@Argument(fullName="sequence",shortName="s",doc="Genome Data",
required=false)
    public String sequence = null;

    @Output
        PrintStream out;
    boolean matchInProgress = false;
        byte previousBase = 'N'; // N is not valid
        GenomeLoc location = null;
        List<GenomeLoc> locations = new ArrayList<GenomeLoc>();

    @Override
        public Integer map(RefMetaDataTracker tracker,
        ReferenceContext ref, AlignmentContext context) {
        out.printf("Location %s; your reference base is %c and
        you have %d reads%n",context.getLocation(),ref.getBase(),
        context.getBasePileup().size() );
        int retVal = 0;
```

```
if (sequence == null) {
sequence = "ATGC";
}

byte[] tokens = sequence.getBytes();

if (ref.getBase() == tokens[0]) {
    matchInProgress = true;
    location = context.getLocation();
} else if (ref.getBase() == tokens[1]) {
    if (previousRead != tokens[0]) {
matchInProgress = false;
    }
} else if (ref.getBase() == tokens[2]) {
    if (previousRead != tokens[1]) {
matchInProgress = false;
    }
} else {
    if (previousRead == tokens[2] && matchInProgress) {
locations.add(location);
  retVal = 1;
    } else {
matchInProgress = false;
    }
}
previousRead = ref.getBase();
return retVal;
    }
```

The function map runs once per single base locus and takes three arguments: tracker, ref, and context. Tracker is the access for the reference metadata. The reference base that lines up with the locus are passed in ref and the context is a data structure consisting of the reads which overlaps the locus and the base from the reference.

We create a boolean flag *matchInProgress*, which keeps track of the base match with the given sequence. Initially, the flag is set false, if the base is matched with the sequence the flag is true. The variable *previousBase* compares the previous base and the current base match with the sequence. This variable is initially assigned as invalid base, for example 'N'. The previousBase is changed to the valid base once it gets the base from ref.getBase() method. The locus of a first base of sequence is stored in *GenomeLoc location* variable. The location variable is initially null, and changes its value when the location of the base the first base is found. After the complete sequence match, the location of that base is stored in the array named *locations*.

The user can give their own input sequence of bases to the application; such as TGAC and ATGC. If the user does not give any input sequence then the application by default checks for ATGC sequence. The sequence given are splitted first and each single base is stored in an array called *tokens*. For example we count the occurrence of the genome sequence ATGC. So, first check for the base A which is stored in token[0] and when found, the location of A is stored in GenomeLoc location variable. The

previousBase checks the match of remaining bases TGC of the given sequence. Once, the sequence is matched, the location of A is stored in the array *locations*. Finally, the count of occurrence of the given sequence and the locations where the sequence occurred is sent to the reducer.

```
    @Override
       public Long reduceInit() {
           return 0L;
       }
    @Override
        public Long reduce(Integer value, Long sum) {
            return sum + value;
        }
    @Override
        public void onTraversalDone(Long result) {
        out.println("The locations where" + sequence +
         "were found are:");
        for (GenomeLoc location: locations) {
            out.println(location);
        }
    out.println("Total count of" + sequence +
                "sequence found: " + result);
        }
}
```

Once the mapper is executed it provides an initial value for the reduce function. At first the base case for the inductive step is 0 that indicates the walker has seen 0 loci. Then, the reduce function combines the result of the previous map with the accumulator. The reduce function has two parameters: value which is the result of map and sum which is an accumulator for the reduce. This function returns the total number of loci processed so far.

The final result is retrieved by the traversal which prints the total count of the occurrence of a given sequence and the sequence loci where the matches were found.

Commands to run the application are as follows:

```
Set CLASSPATH environment variable:
$gedit ~/.bashrc
export CLASSPATH=/path/GenomeAnalysisTK.jar
$bash
$echo $CLASSPATH

Compile the class file:
$ant dist

Run the example BAM and FASTA file to the Walker:
$java -jar dist/GenomeAnalysisTK.jar -I ../exampleBAM.bam
-R ../exampleFASTA.fasta -s ATGC -T GenomeAnalysisWalker
```

Here -I is for BAM input file, -R is for FASTA reference file, -s is the short name for the user input sequence and -T is the type of analysis to run.

The output of this example is in Appendix B.

6 Conclusions

In the report we have illustrated the Genome Analysis Toolkit (GATK) framework designed by Genome Sequencing and Analysis Group of Broad Institute from Harvard and MIT. GATK uses the MapReduce programming philosophy to process the genome data. GATK's MapReduce separates data access patterns from the analysis algorithms. This programming framework enables developers and researchers to develop analysis tool easily for next generation sequencing projects by providing structured data traversals and data walkers. The data traversals provide a collection of data presentation schemes to walker developers, and the walkers provide the map and reduce methods that consume the data.

We also discussed the background on Hadoop, HDFS, and the MapReduce framework. The architecture of these topics were described briefly. The example of WordCount and BAMSort were taken to illustrate Hadoop, Hadoop MapReduce, and HDFS. Then we built a GenomeAnalysis Walker using GATK's MapReduce programming framework to demonstrate the use of the GATK framework. This application was developed to find the total count of the occurrences of the given sequence and the locations of that sequence in a sample BAM file.

A Hadoop-BAM Output

```
11/04/07 16:15:00 INFO input.FileInputFormat: Total input paths to
process : 1
11/04/07 16:15:00 INFO util.NativeCodeLoader: Loaded the native-hadoop
library
11/04/07 16:15:00 INFO zlib.ZlibFactory: Successfully loaded &
initialized native-zlib library
11/04/07 16:15:00 INFO compress.CodecPool: Got brand-new compressor
11/04/07 16:15:00 INFO input.FileInputFormat: Total input paths to
process : 1
11/04/07 16:15:01 INFO mapred.JobClient:Running job:job_201104071613_0001
11/04/07 16:15:02 INFO mapred.JobClient:map 0% reduce 0%
11/04/07 16:15:11 INFO mapred.JobClient:map 100% reduce 0%
11/04/07 16:15:23 INFO mapred.JobClient:map 100% reduce 100%
11/04/07 16:15:25 INFO mapred.JobClient:Job complete:job_201104071613_0001
11/04/07 16:15:25 INFO mapred.JobClient:Counters: 17
11/04/07 16:15:25 INFO mapred.JobClient:Job Counters
11/04/07 16:15:25 INFO mapred.JobClient:Launched reduce tasks=1
11/04/07 16:15:25 INFO mapred.JobClient:Launched map tasks=1
11/04/07 16:15:25 INFO mapred.JobClient:Data-local map tasks=1
11/04/07 16:15:25 INFO mapred.JobClient:FileSystemCounters
11/04/07 16:15:25 INFO mapred.JobClient:FILE_BYTES_READ=7852
11/04/07 16:15:25 INFO mapred.JobClient:HDFS_BYTES_READ=18114
11/04/07 16:15:25 INFO mapred.JobClient:FILE_BYTES_WRITTEN=15736
11/04/07 16:15:25 INFO mapred.JobClient:HDFS_BYTES_WRITTEN=3168
11/04/07 16:15:25 INFO mapred.JobClient:Map-Reduce Framework
11/04/07 16:15:25 INFO mapred.JobClient:Reduce input groups=33
11/04/07 16:15:25 INFO mapred.JobClient:Combine output records=0
11/04/07 16:15:25 INFO mapred.JobClient:Map input records=33
11/04/07 16:15:25 INFO mapred.JobClient:Reduce shuffle bytes=7852
11/04/07 16:15:25 INFO mapred.JobClient:Reduce output records=33
11/04/07 16:15:25 INFO mapred.JobClient:Spilled Records=66
11/04/07 16:15:25 INFO mapred.JobClient:Map output bytes=7747
11/04/07 16:15:25 INFO mapred.JobClient:Combine input records=0
11/04/07 16:15:25 INFO mapred.JobClient:Map output records=33
11/04/07 16:15:25 INFO mapred.JobClient:Reduce input records=33
```

B GATK Output

```
Location chr1:97310; your reference base is T and you have 1 reads
Location chr1:97311; your reference base is A and you have 1 reads
Location chr1:97312; your reference base is A and you have 1 reads
Location chr1:97313; your reference base is T and you have 1 reads
Location chr1:97314; your reference base is T and you have 1 reads
Location chr1:97315; your reference base is T and you have 1 reads
Location chr1:97316; your reference base is C and you have 1 reads
Location chr1:97317; your reference base is A and you have 1 reads
Location chr1:97318; your reference base is T and you have 1 reads
Location chr1:97319; your reference base is G and you have 1 reads
Location chr1:97320; your reference base is C and you have 1 reads
Location chr1:97321; your reference base is A and you have 1 reads
Location chr1:97322; your reference base is A and you have 1 reads
Location chr1:97323; your reference base is T and you have 1 reads
Location chr1:97324; your reference base is C and you have 1 reads
Location chr1:97325; your reference base is T and you have 1 reads
Location chr1:97326; your reference base is T and you have 1 reads
Location chr1:97327; your reference base is C and you have 1 reads
Location chr1:97328; your reference base is A and you have 1 reads
Location chr1:97329; your reference base is T and you have 1 reads
Location chr1:97330; your reference base is G and you have 1 reads
Location chr1:97331; your reference base is T and you have 1 reads
Location chr1:97332; your reference base is T and you have 1 reads
Location chr1:97333; your reference base is A and you have 1 reads
Location chr1:97334; your reference base is T and you have 1 reads
Location chr1:97335; your reference base is G and you have 1 reads
Location chr1:97336; your reference base is G and you have 1 reads
Location chr1:97337; your reference base is G and you have 1 reads
Location chr1:97338; your reference base is G and you have 1 reads
Location chr1:97339; your reference base is A and you have 1 reads
Location chr1:97340; your reference base is T and you have 1 reads
The locations where ATGC were found are:
chr1:57575
chr1:94674
chr1:97317
Total count of ATGC sequence found: 3
INFO  10:35:09,891 TraversalEngine - done 2.03e+03
1.1 s        8.8 m     100.0%         1.1 s     0.0 s
INFO  10:35:09,898 TraversalEngine - Total runtime 1.07 secs,
0.02 min, 0.00 hours
INFO  10:35:09,900 TraversalEngine - 0 reads were filtered out
during traversal out of 33 total (0.00%)
INFO  10:35:09,906 GATKRunReport - Aggregating data for run report
```

References

[1] 1000 Genomes Project. http://www.1000genomes.org/.

[2] Broad Institute. http://www.broadinstitute.org/gsa/wiki/index.php/The_Genome_Analysis_Toolkit.

[3] GATK Walkers. http://www.broadinstitute.org/gsa/wiki/index.php/Your_first_walker.

[4] Hadoop. http://www.cloudera.com/what-is-hadoop/.

[5] Hadoop Overview. http://wiki.apache.org/hadoop/ProjectDescription.

[6] MapReduce. http://hadoop.apache.org/common/docs/r0.17.0/mapred_tutorial.html.

[7] Alex Bateman and Matt Wood. Cloud computing. *Bioinformatics*, 25(12):1475, 2009.

[8] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified Data Processing on Large Clusters. *Commun. ACM*, 51(1):107–113, 2008.

[9] Matthew Hanna Eric Banks et al Aaron Mckenna. The Genome Analysis Toolkit: A MapReduce framework for analyzing next-generation DNA sequencing data. *CSH Press*, (1), 2010.

[10] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, pages 29–43, New York, NY, USA, 2003. ACM.

[11] Hairong Kuang, Sanjay Radai, and Robert Chansler Konstantin Shvachko. The Hadoop Distributed File System. (1):1–10, 2010.

[12] Michael C. Schatz. Cloudburst: Highly Sensitive Read Mapping with MapReduce. *Bioinformatics*, 25(11):1363–1369, 2009.

[13] Jeffrey Shafer, Scott Rixner, and Alan L.Cox. The Hadoop Distributed Filesystem: Balancing Portability and Performance. *IEEE*, (1):122–132, 2010.

[14] Jay Shendure and Hanlee Ji. Next generation DNA sequencing. *Nat Biotechnol*, 26:1135–1145, 2008.

[15] Ronald Taylor. An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics. *BMC Bioinformatics*, 11(12), 2010.

[16] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc, Gravenstein Highway North, Sebastopol, first edition, June 2009.

[17] Wikipedia. FASTA format— Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/wiki/Fasta_format, 2011. [Online; accessed 2-May-2011].

[18] Wikipedia. Single-nucleotide polymorphism— Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/wiki/Single-nucleotide_polymorphism, 2011. [Online; accessed 25-May-2011].