Project 3: Hadoop Blast
Cloud Computing
Spring 2017

Professor Judy Qiu

Goal

By this point you should have gone over the sections concerning Hadoop Setup and a few Hadoop programs. Now you are going to blend these applications by implementing a parallel version of BLAST (Basic Local Alignment Search Tool: `http://blast.ncbi.nlm.nih.gov/Blast.cgi`) using the programming interfaces of the Hadoop MapReduce framework. Note that this application is written in ”Map-Only” fashion, which means no reduce code is necessary.

Deliverables

You are required to turn in the following items in a zip file (username_HadoopBlast.zip)

- The source code of Hadoop Blast you implemented.
- Technical report (username_HadoopBlast_report.docx) that answers the following questions.
  - What is Hadoop Distributed Cache and how is it used in this program?
  - Write the two lines that put and get values from Distributed cache. Also include the method and class information.
  - In previous projects we used Hadoop's TextInputFormat to feed in the file splits line by line to map tasks. In this program, however, we want to feed in a whole file to a single map task. What is the technique used to achieve this? Also, briefly explain what are the key and value pairs you receive as input to a map task and what methods are responsible for producing these pairs?
  - Do you think this particular implementation will work if the input files are larger than the default HDFS block size? Briefly explain why. [Hint: you can test what will happen by concatenating the same input file multiple times to create a larger input file in the resources/blast_input folder]
  - If you wanted to extend this program such that all output files will be concatenated into a single file, what key and value pairs would you need to emit from the map task? Also, how would you use these in the reduce that you would need to add?
- The 4 output FASTA files: celllines_1.fa to celllines_4.fa.

Evaluation

The point total for this project is 3, where the distribution is as follows:

- Completeness of your code and output (1 points)
- Correctness of written report (2 points)

Introduction

Hadoop-Blast is an advanced Hadoop program which helps BLAST, a bioinformatics application, to utilize the computing capability of Hadoop. This exercise shows the details of its implementation, and provides an example of how to handle similar approaches in other applications.

BLAST is one of the most widely used bioinformatics applications written in C++. The version we are using is v2.2.23, which houses new features and better performance. The database used in the following settings is a subset of a full 8.5GB(nr)database; its full name is Non-redundant protein sequence database. Optionally, for more details on how to run the BLAST binary, please see Big Data for Science tutorial page for Blast Installation [NOT required for the assignment].

In this project, we have provided a sketch code which contains just one java class for you to implement:

- RunnerMap.java: The pleasingly-parallel/map-only Map class which takes the prepackaged Blast (v2.2.23) Binary Program and optimized database from Hadoop's Distributed Cache, then executes BLAST binary as java external process with the assigned FASTA file. These are passed as key-value pairs of **(filename, filepath on HDFS)** handled by a provided customized Hadoop MapReduce InputFormat DataFileInputFormat.java.

The detail dataflow can be seen in Figure 1. You will implement the RunnerMap.java, which copies the distributed cache and assigned FASTA file to local, then run the BLAST binary with correct parameters.

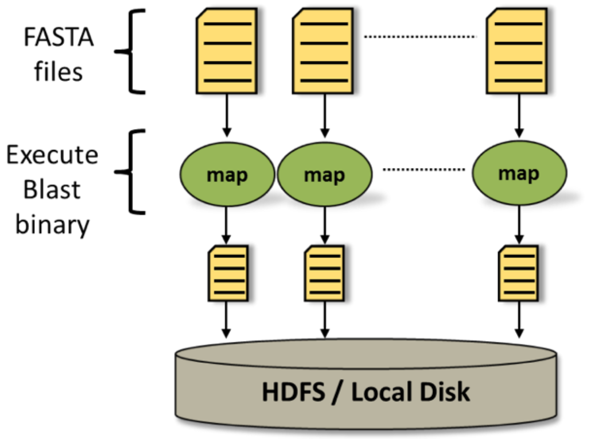

Figure 1: Hadoop Blast dataflow

Normally, for any Hadoop MapReduce program, input data is uploaded and stored in the Hadoop Distributed File System (HDFS) before computation in order to generate **(key, value)** pairs to the mapper. Initially, the BLAST input data is a set of FASTA files located in the local file system. Then it will be uploaded to the HDFS and distributed across the compute nodes. Hadoop framework reads the application records from HDFS with the InputFormat interface and generates **(key, value)** pair input streams; here, we use a provided customized Hadoop MapReduce InputFormat DataFileInputFormat.java to generate key-value pairs of **(filename, filepath on HDFS)**. For this Hadoop Blast program, the map function initially sets up the distributed cache and generates the two absolute location filepaths for Blast binary and Blast Database. Afterwards it copies the assigned FASTA file to local disk by looking up the file from HDFS and generating an absolute filepath. Once this is accomplished and file dependencies are stored in the local disk, we call an external java process and execute the Blast binary with the correct parameters. Finally, the output FASTA file of Blast binary will be uploaded back to HDFS.

Sketch for Hadoop Blast

You need to complete one file in the provided pacakge inside "cgl/hadoop/apps/runner": RunnerMap.java. Code snapshots are shown below.

```java
/*file: RunnerMap.java*/
package cgl.hadoop.apps.runner;

import java.io.File;
import java.io.IOException;
import java.net.URI;
import java.net.URISyntaxException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.filecache.DistributedCache;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

/**
 * @author Thilina Gunarathne (tgunarat@cs.indiana.edu)
 *
 * @editor Stephen, TAK-LON WU (taklwu@indiana.edu)
 */

public class RunnerMap extends Mapper<String, String, IntWritable, Text> {

  private String localDB = "";
  private String localBlastProgram = "";


  @Override
  public void setup(Context context) throws IOException{
    Configuration conf = context.getConfiguration();
    Path[] local = DistributedCache.getLocalCacheArchives(conf);

    /* Write your code here
           get two absolute filepath for localDB and localBlastBinary
        */

  }


  public void map(String key, String value, Context context) throws IOException,
    InterruptedException {

    long startTime = System.currentTimeMillis();
    String endTime = "";

    Configuration conf = context.getConfiguration();
    String programDir = conf.get(DataAnalysis.PROGRAM_DIR);
    String execName = conf.get(DataAnalysis.EXECUTABLE);
    String cmdArgs = conf.get(DataAnalysis.PARAMETERS);
    String outputDir = conf.get(DataAnalysis.OUTPUT_DIR);
    String workingDir = conf.get(DataAnalysis.WORKING_DIR);

      String localInputFile = null;
      String outFile = null;
      String stdOutFile = null;
      String stdErrFile = null;

    System.out.println("the map key : " + key);
    System.out.println("the value path : " + value.toString());
    System.out.println("Local DB : " + this.localDB);

        /*
          Write your code to get localInputFile, outFile,
          stdOutFile and stdErrFile
        */


    // Prepare the arguments to the executable
    String execCommand = cmdArgs.replaceAll("#_INPUTFILE_#", localInputFile);
    if (cmdArgs.indexOf("#_OUTPUTFILE_#") > -1) {
      execCommand = execCommand.replaceAll("#_OUTPUTFILE_#", outFile);
    }else{
      outFile = stdOutFile;
    }

    endTime = Double.toString(((System.currentTimeMillis() - startTime) / 1000.0));
    System.out.println("Before running the executable Finished in " + endTime + " seconds");

    execCommand = this.localBlastProgram + File.separator + execName
    + " " + execCommand + " -db " + this.localDB;
    //Create the external process

    startTime = System.currentTimeMillis();

    Process p = Runtime.getRuntime().exec(execCommand);

    OutputHandler inputStream = new OutputHandler(p.getInputStream(), "INPUT", stdOutFile);
    OutputHandler errorStream = new OutputHandler(p.getErrorStream(), "ERROR", stdErrFile);

    // start the stream threads.
    inputStream.start();
    errorStream.start();

    p.waitFor();
    //end time of this procress
    endTime = Double.toString(((System.currentTimeMillis() - startTime) / 1000.0));
    System.out.println("Program Finished in " + endTime + " seconds");

    //Upload the results to HDFS
    startTime = System.currentTimeMillis();

    Path outputDirPath = new Path(outputDir);
    Path outputFileName = new Path(outputDirPath,fileNameOnly);
    fs.copyFromLocalFile(new Path(outFile),outputFileName);

    endTime = Double.toString(((System.currentTimeMillis() - startTime) / 1000.0));
    System.out.println("Upload Result Finished in " + endTime + " seconds");

  }
}
```

In addition, if you need to understand the dataflow and main program, please look into the DataAnalysis.java.

```java
/*file: DataAnalysis.java*/
package cgl.hadoop.apps.runner;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.filecache.DistributedCache;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.SequenceFileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

import cgl.hadoop.apps.DataFileInputFormat;
import java.net.URI;

```

```java
public class DataAnalysis extends Configured implements Tool {

  public static String WORKING_DIR = "working_dir";
  public static String OUTPUT_DIR = "out_dir";
  public static String EXECUTABLE = "exec_name";
  public static String PROGRAM_DIR = "exec_dir";
  public static String PARAMETERS = "params";
  public static String DB_NAME = "nr";
  public static String DB_ARCHIVE = "BlastDB.tar.gz";

/**
 * Launch the MapReduce computation.
 * This method first, remove any previous working directories and create a new one
 * Then the data (file names) is copied to this new directory and launch the
 * MapReduce (map-only though) computation.
 * @param numMapTasks - Number of map tasks.
 * @param numReduceTasks - Number of reduce tasks =0.
 * @param programDir - The directory where the Cap3 program is.
 * @param execName - Name of the executable.
 * @param dataDir - Directory where the data is located.
 * @param outputDir - Output directory to place the output.
 * @param cmdArgs - These are the command line arguments to the Cap3 program.
 * @throws Exception - Throws any exception occurs in this program.
 */
  void launch(int numReduceTasks, String programDir,
      String execName, String workingDir, String databaseArchive, String databaseName,
      String dataDir, String outputDir, String cmdArgs) throws Exception {

    Configuration conf = new Configuration();
    Job job = new Job(conf, execName);

    // First get the file system handler, delete any previous files, add the
    // files and write the data to it, then pass its name as a parameter to
    // job
    Path hdMainDir = new Path(outputDir);
    FileSystem fs = FileSystem.get(conf);
    fs.delete(hdMainDir, true);

    Path hdOutDir = new Path(hdMainDir, "out");

    // Starting the data analysis.
    Configuration jc = job.getConfiguration();

    jc.set(WORKING_DIR, workingDir);
    jc.set(EXECUTABLE, execName);
    jc.set(PROGRAM_DIR, programDir); // this the name of the executable archive
    jc.set(DB_ARCHIVE, databaseArchive);
    jc.set(DB_NAME, databaseName);
    jc.set(PARAMETERS, cmdArgs);
    jc.set(OUTPUT_DIR, outputDir);


    long startTime = System.currentTimeMillis();
    DistributedCache.addCacheArchive(new URI(programDir), jc);
    System.out.println("Add Distributed Cache in "
        + (System.currentTimeMillis() - startTime) / 1000.0
        + " seconds");


    FileInputFormat.setInputPaths(job, dataDir);
    FileOutputFormat.setOutputPath(job, hdOutDir);

    job.setJarByClass(DataAnalysis.class);
    job.setMapperClass(RunnerMap.class);
    job.setOutputKeyClass(IntWritable.class);
    job.setOutputValueClass(Text.class);

    job.setInputFormatClass(DataFileInputFormat.class);
    job.setOutputFormatClass(SequenceFileOutputFormat.class);
    job.setNumReduceTasks(numReduceTasks);

    startTime = System.currentTimeMillis();

    int exitStatus = job.waitForCompletion(true) ? 0 : 1;
    System.out.println("Job Finished in "
        + (System.currentTimeMillis() - startTime) / 1000.0
        + " seconds");
    //clean the cache


    System.exit(exitStatus);
  }

  public int run(String[] args) throws Exception {
    if (args.length < 8) {
      System.err.println("Usage: DataAnalysis <Executable and Database Archive on HDFS>
       <Executable> <Working_Dir> <Database dir under archive> <Database name>
        <HDFS_Input_dir> <HDFS_Output_dir> <Cmd_args>");
      ToolRunner.printGenericCommandUsage(System.err);
      return -1;
    }
    String programDir = args[0];
    String execName = args[1];
    String workingDir = args[2];
    String databaseArchive = args[3];
    String databaseName = args[4];
    String inputDir = args[5];
    String outputDir = args[6];
    //"#_INPUTFILE_# -p 95 -o 49 -t 100"
    String cmdArgs = args[7] ;

    int numReduceTasks = 0;// We don't need reduce here.

    launch(numReduceTasks, programDir, execName, workingDir ,
      databaseArchive, databaseName,inputDir, outputDir, cmdArgs);
    return 0;
  }

  public static void main(String[] argv) throws Exception {
    int res = ToolRunner.run(new Configuration(), new DataAnalysis(), argv);
    System.exit(res);
  }
}
```

Edit

The sketch code is stored within the provided VirtualBox image. Use linux text editor vi/vim to add your code.

```
$ cd /root/MoocHomeworks/HadoopBlast/
$ vim src/cgl/hadoop/apps/runner/RunnerMap.java
```

Compile and run your code

Use the same one-click script compileAndExecHadoopBlast.sh as in prior homework. Standard error messages such as compile errors, execution errors, etc. will be redirected on the screen. Follow the same debugging format.

```
$ cd /root/MoocHomeworks/HadoopBlast/
$ ./compileAndExecHadoopBlast.sh
```

View the result

The result is generated at /root/MoocHomeworks/HadoopBlast/output/HDFS_blast_output . There should be 4 output FASTA files with .fa extension

```
$ cd /root/MoocHomeworks/HadoopBlast/output/HDFS_blast_output
$ ls
```