Hadoop\_BLAST\_(Project 3)

**Authors**:

Hemanth Sharma (UID#2000048054 ) – Initial Draft

Nilima Sahoo (UID# 0003644966) – Review & Editing

**Introduction to BLAST**

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.

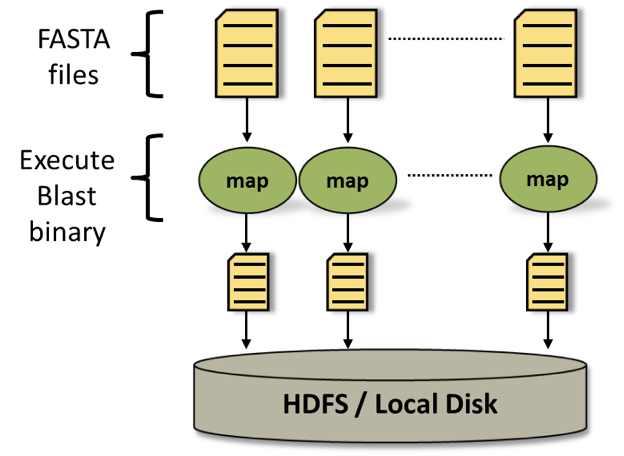


Figure 1: Hadoop Blast dataflow

Normally, for any Hadoop MapReduce program, input data is uploaded and stored in the Hadoop Dis- tributed 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.

BLAST or Basic Local Alignment Search Tool matches a given gene sequence against a database of known sequences.

NCBI Basic Local Alignment Search Tool (BLAST) finds regions of local similarity between sequences. The program compares nucleotide or protein sequences to sequence databases and calculates the statistical significance of matches. BLAST can be used to infer functional and evolutionary relationships between sequences as well as help identify members of gene families. – (from BLAST README).

1. **HADOOP DISTRIBUTED CACHE**

Hadoop Distributed Cache is a framework which copies necessary files onto the slave node before any tasks for the job are executed on that node. It is efficient because the files are only copied once per job and the ability to cache archives which are un-archived on the slave nodes.

In our case the FASTA files are uploaded to HDFS from local, the nodes download the files from HDFS and perform map operation, here the mapper is provided with filenames and path of file names as key and value pair and once the operation is complete the output files are uploaded back to HDFS.

1. **Lines that put and get values from Distributed cache.**
2. Path[] local = DistributedCache.getLocalCacheArchives(conf)

The classDistributedCache and the method is getLocalCacheArchives

**Class and Method information and usage: taken from the Hadoop.apache.org website**

**Class Information:**

DistributedCache.addCacheArchive(new URI(programdirectory, job);

Class : DistributedCache

Method: addCacheArchive

**addCacheArchive :**

public static void addCacheArchive(URI uri, Configuration conf)

Add a archives to be localized to the conf. Intended to be used by user code.

Parameters:

uri - The uri of the cache to be localized

conf - Configuration to add the cache to

**getLocalCacheArchives :**

public static Path[] getLocalCacheArchives(Configuration conf)

throws IOException

Return the path array of the localized caches. Intended to be used by user code.

Parameters: conf - Configuration that contains the localized archives

Returns: A path array of localized caches

Throws: IOException

1. **Technique used:**

We used FileRecordReader.java to feed whole file into single map task instead of feeding line by line like in the previous projects.

The Key and Value pairs are

Key: Name of the files: Generated by method getCurrentKey () in FileRecordReader.java

Value: Path of the file: Generated by the method getCurrentValue () in FileRecordReader.java

1. **When files are concatenated into Single file :**

The implementation worked when all the files are concatenated: Attaching the concatenated file output in the zipfile as lab.txt

1. **Extend this program such that all output files will be concatenated**

We can assign a key (concatenated) by adding filenames and path using a delimiter. When it reaches the reduce program we can get file name and path by splitting the key (concatenated) using the delimiter. Finally we can get a single output files by concatenating or merging the output using ***getmerge***

1. **Added the output files and code in the Hadoop\_Blast.zip**