Report for Advanced Project (Advisor: Prof. Michael Schatz, CSHL)

Objective:

Following were the objective of this project:

- to globally sort the alignments produced by *bwa/bowtie2* program
- to extract these alignments from HDFS to local file system
- by developing above two functionalities, contributing towards development of *Jnomics*, an open source cloud scale DNA sequence analysis tool

Global Sorting:

- In *Jnomics*, the output generated by the alignment programs like, *bwa and bowtie2*, is not sorted globally. What is means that during shuffling phase of Hadoop, the alignments are sorted locally and thus all the alignments present in a single file are good
- However, if we take into account all the files created on HDFS by the above mentioned programs(*bwa and bowtie2*), then these alignments are not globally sorted
- To globally sort the output, following approach has been used:
    - the output generated by *bwa/bowtie2* is in SAM format and key for sorting is composed of RNAME and RPOS. So, the same SAM file acts as an input to the global sort program. Each record of a SAM file in Jnomics is represented by a class, *SAMRecordWritable*
    - we divide the genome into fixed sized bins based on a user defined bin size. So, lets say the genome length is 10000 and the user defined bin size is 1000, so in total 10 bins are created:

        bin 0: 0-999
        bin 1: 1000-1999 ...
        bin 9: 9000-9999
    - and then alignments are assigned a bin depending on RPOS. So, if RPOS is 887, then it falls under bin 0. If RPOS is 9946, then it falls under bin 9
    - using the map-reduce framework, mapper emits out a key-value pair of *SamtoolsKey* and *SAMRecordWritable*. *SamtoolsKey* is a custom class created to handle the key related things and is initialized with the appropriate bin and reference name inside map() of Mapper. *SAMRecordWritable* is a *Jnomics* based class and denotes a SAM record and remain unchanged
    - using combiner and partitioner, the output of mapper is processed and is then sent to reducer
    - the number of reducers have been configured to a default value of 100 else a user defined value is taken. The reason for creating multiple reducers is each key generated by partitioner can be assigned to a single reducer. So, the load gets distributed which also helps us in time efficiency
    - the way reducer works is that once it receives the input, it fetches the reference name from the key and checks if it is an unmapped chromosome i.e if it matches "*". If yes, then it writes down all those records in a file named *"~~UNMAPPED"*. If not, then it writes down all those records in a file named as: "<referenceName>-<alignmentStart>"
    - in this process, reducer also writes down the SAM header in a separate file named as *"header.sam"*. This file is written only once, irrespective of the number of reducers and is used during next phase of global sorting, as mentioned in the next section
    - once this set of mapper and reducer processes is done, the output is generated on HDFS
    - the program which implements this functionality in *Jnomics* is, *globalSort*

Extract globally sorted alignments

- once the above process is finished successfully, the next thing is to merge all the sorted alignements together using a separate program into 1 single file on local file system to get the globally sorted file
- this is done using another program in Jnomics, *globalSortExt*, which should be run only after *globalSort*
- this program taken the header file and the alignment files created by *globalSort* on HDFS as input. From the header file, it figures out the order of reference names and thus reads the alignments present on HDFS in the same order and thus merge them all together into a single file on the local file system
- finally, it merges the unmapped chromosomes in this local file, to get the final output of globally sorted alignments

What else was tried

- **Compression**
  I tried using Gzip compression on map output as well on the final output and I was able to achieve a compression ratio of 89%. The program also ran faster than the regular program. The other option was to use LZO compression which is faster but not that efficient during compression