Referential DNA Data Compression using Hadoop Map Reduce Framework

Raju Bhukya and Sumit Deshmuk
Department of Computer Science and Engineering, National Institute of Technology, India

Abstract: *The indispensable knowledge of Deoxyribonucleic Acid (DNA) sequences and sharply reducing cost of the DNA sequencing techniques has attracted numerous researchers in the field of Genetics. These sequences are getting available at an exponential rate leading to the bulging size of molecular biology databases making large disk arrays and compute clusters inevitable for analysis.In this paper, we proposed referential DNA data compression using hadoop MapReduce Framework to process humongous amount of genetic data in distributed environment on high performance compute clusters. Our method has successfully achieved a better balance between compression ratio and the amount of time required for DNA data compression as compared to other Referential DNA Data Compression methods.*

Keywords: *Compression, map reduce sequences, dna sequences.*

Received August 12, 2017; accepted April 17, 2018
https://doi.org/10.34028/iajit/17/2/8

1. Introduction

The inception of the Deoxyribonucleic Acid (DNA) sequencing by Sanger sequencing and maxam-gilbert sequencing in mid 1970s has pulled a lot attention of researchers from numerous applied fields such as medical diagnosis, biotechnology, forensic biology, virology and biological systematics like European Bioinformatics Institute (EBI), National Center for Biotechnology Information (NCBI), National Institutes of Health (NIH) Genetic Sequence Database (GENBANK) and many more are making genome sequences publicly available to attract more researchers in the field of Bioinformatics. In 2005, with the genome analyzer, a single sequencing run could produce roughly one gigabase of data [5] with cost of 3 billion dollars in early days. By 2014, the rate increased by 1000 times to a 1.8 terabases of data in a single sequencing run. In contrast, the HiSeq Enterprise Translation Management System (XTM) Ten, released in 2014, can sequence over 45 human genomes in a single day for approximately $1000 each. Demand of DNA sequences and reducing cost of DNA sequencing techniques is more, a very large amount of genomic data is produced in the molecular biology databases.

Human DNA consists of about 3 billion bases, and more than 99 percent of those bases are common in all people. This fact has made researchers to exploit similarity between sequences of same species and focus on reference based sequence compression technique in which one sequence is selected as reference and all the other sequences of the same species are encoded with respect to the reference. It has been found that referential DNA sequence compression has achieved better results compared to other compression techniques such as Bit Encoding Techniques, Dictionary based compression and Statistical Compression. DNA sequences for future analysis efficiently in terms of space and time. It has been found that most of the DNA-compression techniques focus mostly on

Compression ratio rather than compression and decompression time or speed during the compression process. Those techniques have not capitalized on the emergence of various parallel data processing techniques in distributed environment. In this paper we have proposed Referential DNA data compression using map reduce framework which has achieved a better balance between compression ratio and compression time. Our method has achieved similar compression ratio but has reduced the compression time by considerable margin using distributed processing.

In this paper we have mentioned four methods for referential DNA data compression which we have followed from Referential Compression of Highly Similar Sequences (FRESCO) [11] method with capability of running them on high performance compute clusters using map reduce framework. The terms and sequence representations are mentioned in it. We have created and tested our own mapreduce algorithms for referential first order compression, Second order compression, reference rewriting and reference selection to make them suitable to use in distributed environment for big genomic data.

This paper is organized as follows. In section 2 we discussed about related work. Map Reduce methods are proposed in section 3, experimental results are

presented in section 4 and conclusion and future scopeis discussed in sections 5 and 6

2. Related Work

Bit encoding technique uses two bits to represent each character: 00 for 'A', 01 for 'C', 10 for 'G' and 11 for 'T'. It achieves a fixed compression ratio of 4:1. Dictionary Based Algorithms create dictionary of frequently occurring substrings. It makes use of Lempel-Ziv compression algorithms [8] like LZ77, LZ78. Based on frequency of repeats and their distribution, this method achieves compression ratio of up to 6:1. In Statistical compression algorithms, the input is scanned and probability distribution of the symbols and repeats is determined from previous occurrences and matches in the subsets of the input.

High probability of predicting next symbol or repeat, results in high compression ratios. Wang and Zhang [12] used widely known statistical DNA compression algorithm which makes use of both statistical properties and repetition within the sequences. This technique achieves compression ratio of 4:1 to 8:1 depending on the probability distribution.

Referential compression algorithms recently have gained popularity in genome compression. Similarity between reference sequence and input sequence, in the form sequence matches, is encoded with various representations. The reference selected in the process must be preserved and should not be changed. Genome Resequencing (GRS) [6], Relative Lempel-Ziv (RLZ) [6], Relative Lempel-Ziv Optimization (RLZOpt) [9], Genome Resequencing Encoding (GReEn) [2], Thousands Genomes Compressor (TGC) [1], Genome Differential Compressor 2 (GDC2) [4], and FRESCO [3] are some of the known referential genome compression models. Arafat [1] multilayer model-based approach for text compression.

RLZ [7] can represent individual human genomes in around 0.1 bits per base supporting rapid access. RLZopt [8] technique shows 50% improvement in compression ratio of RLZ method. GReEn [3] has faster running time and compression gains of over 100 folds for some sequences. GDC2 [3] presents an LZ77-style compression scheme for relative compression of multiple genomes of the same species. GDC2 [4] uses effective search structure with two level compression algorithm. It applies Ziv-Lampel factoring of all sequences from the collection. It has achieved a very impressive compression ratio and compression speed of nearly 200MB/s. A Framework for Referential Sequence Compression (FRESCO) uses Single Nucleotide Polymorphisms (SNPs) which follows exact match preceding it, for more compact representation. FRESCO enhances compression ratio by applying second order compression. GDC and FRESCO also allow selection and modification of a reference sequence for achieving better compression. The intention of improving speed of compression algorithm, we process multiple DNA sequences in parallel manner over a big data compute clusters.

3. Overview of Mapreduce Framework

MapReduce supports parallel computations on vast amounts of data in-parallel on large clusters of commodity hardware in reliable, Fault-tolerant manner. It groups together all intermediate values associated with the same intermediate key and passes them to the Reduce function. The Reduce function, accepts an intermediate key and a set of values for that key. It merges together these values to form a possibly smaller set of values. Typically just zero or one output value is produced per Reduce invocation. The intermediate values are supplied to the user's reduce function via an iterator. Combiner can be viewed as mini-reducers in the map phase. Partitioner comes into the picture when we are working on more than one reducer. So, the partitioner decides which reducer is responsible for a particular key. Both the input and output of the job is stored in a file-system.

4. Referential DNA Data Compression Using Mapreduce Framework

4.1. Referential Compression: First Order Compression

In the following part, we have prepared an algorithm for first order referential dna data compression using hadoop mapreduce framework. In first order compression, we select one sequence as reference and encode the rest with respect to reference in distributed environment. Referentially Matched Entry (RME) [10] is a triplet <start, length, mismatch>, where start is a number indicating the start of match within the reference, length denotes the match length and mismatch denotes the first character after the match. It is used to encode input DNA sequence.

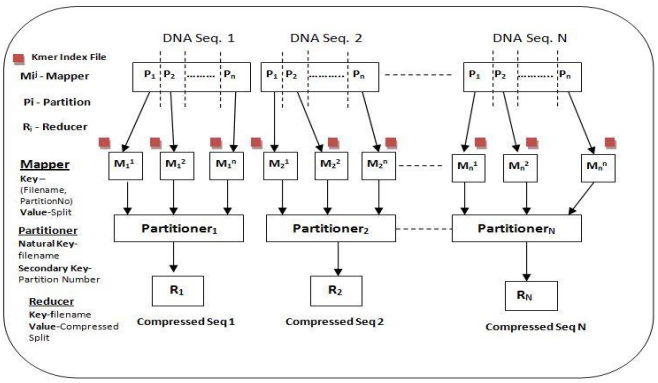

Figure 1. Referential compression: first level compression.

4.2. Proposed Mapreduce Method

As the size of human DNA sequence is very large, we can split the input DNA sequence into multiple

partitions and compress each split in parallel in the distributed environment. The result of each split will be gathered in single file for the given input sequence. At the same time for processing enormous amount of data, we can compress multiple DNA sequences of same species simultaneously in distributed environment using big data processing clusters.

In Figure 1 Pi is ith partition of given input sequence, Mij is the Mapper processing jth split of ith file, Partitioner i is ith partitioner and Ri is ith Reducer.In the above Map Reduce Job, the number of partitioners is equal to number of reducers. The input dataset to the Map Reduce job is the collection of all to be compressed input files. The proposed Map Reduce method requires one MapReduce job to complete the first order compression.

- *Example*1: Reference Sequence: ACCTGTCACTGGATT ACTAATTCCAATTTA, length=30
- Input Sequence: ACCTATCACTGGGTTACTATTTCCA GTTT, length = 29 Split Size: 10
  - *Mapper* 1: key= (sequence1,0), input value= ACCTGTCACT, After Compression= <0,4,A><5,5,->
  - *Mapper* 2: key= (sequence1,1),input value= GATTACTAA AfterCompression: <10,2,G> <13,6,T>
  - *Mapper* 3: key= (sequence1,2),input value= TTCCAATTT After Compression: <20,5,G><26,3,->
- *Reducer*: key=Sequece1,Value={[<0,4,A><5,5,->],[<10,2,G><13,6,T>],[<20,5,G><26,3,->]}Output= <0,4,A><5,7,G><13,6,T><20,5,G><26,3,->

As Reducer checks last RME of ith compressed sequence and first RME of (i+1)th compressed sequence, here reducer will check last RME <5,5,-> of 0th partition and first RME <10,2,G> of 1st partition. These RME's are merged as start1+length1= start 2 and mischar is '-'for <5,5,->.Similarly for subsequent partitions the above condition is tested.

Algorithm 1: First Order Compression Map Task

Input: Keyin=<FileName,PartitionNo>, Valuein= SplitSequence
Output: Keyin=<FileName,PartitionNo>, Valueout=CompressedSequence
1.Class RefComMapper
2. method map(<FileName,PartitionNumber> ,SplitSequence)
3. Reference = DistributedCache.getCacheFile()
4. Compressed_Seq= Referential_Compression (SplitSequence,Reference)
5. Emit(<FileName,PartitionNumber>, CompressedSequence)
6. method map close

Algorithm 2: First Order Compression Reduce Task

Input: Keyin=<FileName,PartitionNo>,Valuein= SplitSequence Output: Keyin=<FileName>,Valueout= CompressedSequence
1.ClassRefComReducer
2. methodReduce(<FileName,PartitionNumber>, CompressedSequence)
3. Iterator it = CompressedSequence.getIterator();
4. prev = null;
5. while it.hasnext() do
6. cur = it.next();
7. start1 = prev.getLastRMEStart();
8. start2 = cur.getCurRMEStart();
9. matchLen1 = prev.getMatchLen();
10 if(start1+matchLen1 == start2) then
11 Merge Last RME of prev and First RME of cur
12 prev = cur;
13 Emit(FileName, modifiedRME)
14 end while
15 method close

To find the location of the match in reference sequence, we created K-mer based Hash table on reference sequence. It hashes every K length string in reference and attaches a list of locations to that k-mer in the Hash table where it can be found in reference sequence. We stored the reference in the form of Hash table in distributed cache before map reduce job starts. So I/O overhead to load this reference in every slave node from Distributed Cache is less as compared to loading it from HDFS.

4.3. Second Order Compression

DNA sequences of similar species are highly similar. The similarity between the sequences is in the form long matches interrupted by SNP's and Insertion and Deletion (INDEL's) mostly [17]. Due to the high similarity between the sequences, the SNP's or INDEL's and their positions are same in multiple sequences. So the result of the referential compression shows that compressed sequences have some similarity in terms of referentially matched entry and their order. This fact helps to compress the DNA sequences to further levels. Second order compression is also a referential compression method in which some compressed sequences are selected as reference and encode other sequences with respect to reference sequences (Figure 2).

- *Example* 1.

rc1=[<0,10,A><12,17,C><32,15,A><48,24,G> <74,20,T><94,30,C><130,15,T><146,31,G>]
rc2=[<0,11,C><12,17,C><32,15,A><48,24,G><73,20, T><94,30,C> <130,15,T><147,31,G>]

If we make rc1 as reference and compress rc2 with respect to rc1, rc2 will be compressed to SOrc2 =[<0,11,C>{1,3,0}<73,20,T>{5,2,0}<147,31,G>].

Where {si, ml, fi} denotes second order referentially matched entry with si as start index of RME in reference sequence, ml as number of matched RME's

in same first order compressed sequence and fi is the file index of the matched reference file represented as the integer. We have mapped reference filenames to integer index starting from 0. The reason for mapping integer value to filename is to reduce the size second order referentially matched entry.

RME Number is the RME index in reference from the start and Reference File Index is nothing but compressed reference file name mapped to integer value. This integer valued File Index not only reduces size of referentially match entry but also reduces the size of Hash Map created. The Hash Map is serialized and stored on Hadoop Distributed File System (HDFS). At the time of second order compression, each slave node performing Map and Reduce tasks load this serialized Hash Map from HDFS and deserialize it to Hash Map object.

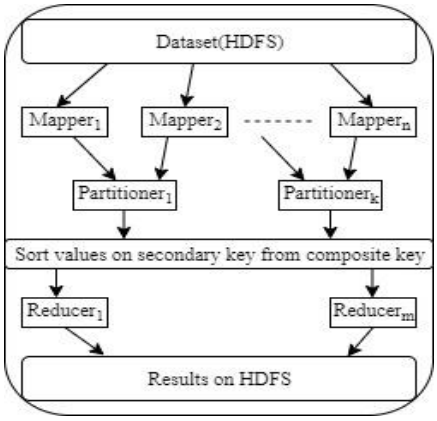

Figure 2. Second order compression using map reduce framework.

4.3.1. Second Order Compression using Map Reduce Framework

Proposed algorithm uses one map reduce job to perform second order compression. The input dataset used is the collection of referentially compressed files obtained from first order compression. We used nline input format to partition the data in input dataset.

- *Map Phase*: in map phase, each mapper reads multiple lines from compressed sequences which are the compressed result of multiple splits in the first order compression. Mapper finds the similarity of input sequence with reference sequences and replaces consecutive matched RME's across sequences by new entry{Start RME Number, NoOf Matched RME's, RefFileIndex}. If a particular RME has been found that second order compression improves the compression ratios by considerable margin.

Algorithm 3: Second Order Compression Map Task

Input: Keyin=(FileName, LineOffset), Valuein= (NCompressedLines) Output:Keyout= (FileName, LineOffset), Valueout = (SOCompressedLines)

```
1. Class SOCompMapper
2.  Method Map(<FileName,LineOffset>,
     NCompressedLines)
3.  Reference = LoadHashMap(RefSequences);
4.  for rmei in N CompressedLines do
5.   if(rmei € Reference) then
6.                match_length = 1;
7.     Get RME_Index in Reference File and
      FileIndex of Reference File
8.     Set Start-index to RME_Index
9.     while next rmei in input and Reference matches
        do
10.      Increment Match_length and RME_Index
11.    end while
12.    if (match_length>1) then
13.     Append {start_index,match_length,FileIndex}
       to SOComp;
14.    else
15.      Append rmeito SOComp;
16.    end if
17.  else
18.     Append rmeito SOComp;
19.  end if
20  end for
21. Emit ((<FileName,LineOffset>,SoCompSequence);
22. Method Map Close
```

- *Partitioner*: partitioner helps for partitioning the intermediate keys generated by mappers. We have used filename from output key of mapper to send it to a particular reducer. The second order compressed results of same input file are sent to one reducer.
- *Reduce Phase*: each Reducer gets results of multiple mappers grouped by file name of first order compressed files in sorted order of their line offset. Reducer only merges the result of mappers in single file in sorted order as shown in Figure 3.

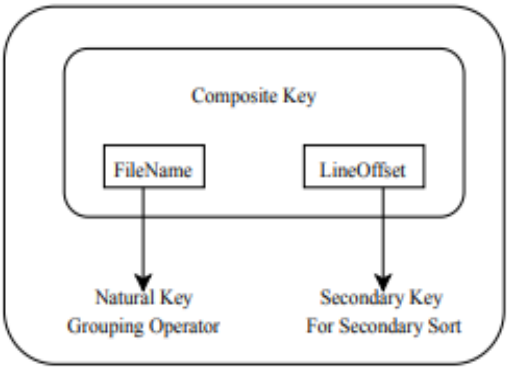

Figure 3. Partitioning and sorting on composite key.

Algorithm 4: Second Order Compression Reduce Task

Input: Keyin=FileName , LineOffset),Valuein= (SOCompressedLines) Output: Keyout= (FileName, LineOffset),Valueout= (SOCompressedSeq)

```
1. Class SOCompReducer
2   MethodReduce(<FileName, LineOffset>,
     SOCompressedLines)
3.    Iterator it = SoCompSequence.getIterator();
4.    while it.hasNext() do
5.       SOCompressedSeq.append(it.next())
6.    end while
7.  Method Close
```

To get better compression ratios, we should obtain RME's with longer matches from reference and less number of RME's as a result of first order compression.

4.4. Reference Rewriting

The best reference selected in reference selection algorithm may produce same SNP's in multiple input sequences with respect to reference. So modifying reference by Replacement, Insertion or deletion of reference base character will result in less number of RME's with longer matches. The necessary condition for rewriting reference base is that the two RME's must be consecutive in the compressed sequence with respect to their location in reference sequence. If <s1, l1, c1> and <s2, l2, c2> are consecutive RME's then reference base rewriting criteria are given as follows:

INSERT Candidate: If s1+l1=s2 and rewrite location, l=s1+l1, Update Candidate: If s1+l1+1=s2 and rewrite location, l=s1+l1, Delete Candidate: If s1+l1+2=s2 and rewrite location, l=s1+l1.

- *MapReduce Method*: The input to our Map Reduce job is the set of referential compressed sequences after first order compression. The input sequence is the collection of RME's. We have used NLine Input Format to give N lines of input file to different mapper nodes.
- *Map Phase*: In the map phase, we analyze two consecutive RME's and check whether they satisfy criteria for Insert/Update/Delete base character rewrite.

If they satisfy the criteria we emit first RME's <start,length> as the key and <base character, INS/UPD/DEL> as value.

- *Reduce Phase*: At each reducer, from how many compressed sequences the key<start,length> is emitted, is calculated. If frequency of counted base character for INS/UPD/DEL is greater than the threshold, we emit (start+length) as key and rewrite character as value. The rewrite results are accumulated in file, which are further used to modify reference file (Figure 4).

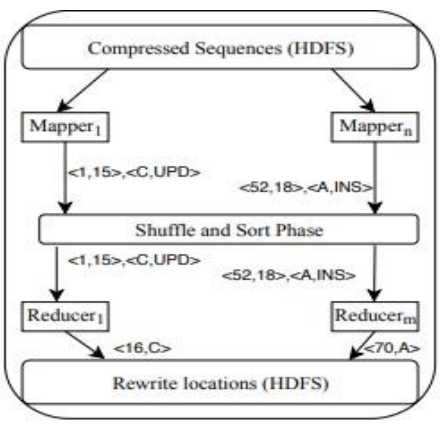

Figure 4. Flowchart of reference rewriting mapreduce algorithm.

Algorithm 5: Reference Rewriting Map Task

```
Input: Keyin=<FileName,LineOffset>Valuein=
<CompressedNLines> Output: Keyout=
<start,matchLength>Valueout= <MisChar,Criteria>
1.Class RefRewMapper
2.  Method Map(<FileName ,lineOffset>, CompSeq)
3.    for all RME r(i) and r(i+1) ∈CompSeqcsdo
4.     start1 = r(i).getStart(), start2 = r(i+1).getStart();
5.     matchLength = r(i).getMLen(),
       mischar= r(i).getMischar();
6      if(start1+matchLength == start2) then
7.      Emit(<start,matchLength>,<mischar,INS>)
8.     else if(start1+matchLength+1 == start2) then
9.       Emit(<start,matchLength>,<mischar,UPD>)
10.    else if(start1+matchLength+2 == start2) then
11.      Emit(<start,matchLength>,<mischar,DEL>)
12.    end for
13.  close method Map
```

Algorithm 6: *Reference Rewriting Reduce Task*

```
Input: Keyin=<start,matchLength>, Valuein= <MisChar
,Criteria> Output: Keyout=<RewriteLocation> ,Valueout=
<MisChar>
1. classRefRewReducer
2.  method Reduce(<start,mLen>,<Mischars,Criteria>)
3.    sum ← 0, Asum ← 0, Csum ← 0, Gsum ← 0,
      Tsum ← 0, loc = start+mLen
4.    Identify whether criteria is INS/UPD/DEL
5.    Find Mischar(A/C/T/G) and their counts in
      valuein array
6.      if Asum> threshold then
7.              Emit(loc, 'A')
8.      else if Csum> threshold then
9.              Emit(loc, 'C')
10.     else if Gsum> threshold then
11.             Emit(loc, 'G')
12.     else if Tsum> threshold then
13.             Emit(loc, 'T')
14.     end if
15.   close method Reduce
```

The reference rewriting using map reduce framework is implemented and tested in distributed environment. The results after Reference rewriting have shown improvement in referential compression. Also the time required for reference rewriting in distributed environ has improved compared to FRESCO method.

4.5. Reference Rewriting

To achieve better referential compression, selection of reference sequence plays a very important role. Best reference sequence selected will always result in less number of RME's in compressed sequence, for majority of the input sequences, compared to other candidate reference sequences. To select the best reference sequence we have followed the heuristic in (Wandelt *et al.*, [11]). If rc1 and rc2 are two referentially compressed sequences after first order compression, rsim is given as follows rsim (rc1, rc2)= | rc1 U rc2| - | rc1 ∩ rc2.

As per the method mentioned in FRESCO, we require to find the candidate sequence which results in lowest rsim value. Each input sequence si is divided

into 'P' equal blocks and depending on K value, only every Kth block is considered for compression

- *Reference Selection using MapReduce Paradigm*:

Algorithm 7: *Reference Selection Map Task*

```
Input:    Keyin = FileName,Valuein =SequencePart
Output: Keyout= 1 ,Valueout= Rsim[]
1.ClassRefSelectionMapper
2.   method Map(<Filename>,<SequencePart>)
3.        Rsim[n] = {0}
4.        RMESet[] = Invoke First Order Compression
          Job for SequencePart
5.        for every RMEi in RMESetdo
6.          if RMEi ∈HashedRME then
7.            get indexes of filenames containing RMEi
8.            Increment value Rsim[index] for all file
              indexes
9.          end if
10.       end for
11.       Emit(1,Rsim[])
12.  close method Map
```

Algorithm 8: Reference Selection Reduce Task

```
Input : Keyin= 1 , Valuein= collection of Rsim[]
Output:Keyout = Null, Valueout = BestCandidateRefName
1.ClassRefSelReducer
2. method Reduce(1,{Rsim1[],Rsim2[],…,Rsimm[])
3.       min = 0, index = -1
4.       Add ith index of every Rsimk[] in FRsim[]
5.       for every val i in FRsim[] do
6.          if vali< min then
7.                min =Vali
8.                index = i
9.          end if
10.      end for
11.      Emit(null,FileName[i])
12.  close method Reduce
```

5. Experimental Results

We performed experiments on (1+8 nodes) cluster machine having front node with 2 X Intel® xeon® E5-2640 (2.5 GHz / 6-core/15MB/95w) processor, 64 GB RAM, 3 X 600GB HDD and remaining 8 nodes with 1 X Intel® xeon® E5-2640 (2.5 GHz/6-core/15MB/95w) processor, 16GB RAM, 2 X 300GB HDD. We considered two datasets for experiments: Human Genome and Arabidopsis Thaliana as shown in Table 2. Human DNA sequences are taken from phase 1 of 1000 Genome project and AT sequences are taken from 1001 Genome Project. The input given to MapReduce Job is in the form of FASTA files. GDC has also given documentation and scripts to convert VCF files of individual chromosomes to FASTA files. We compared GDC, FRESCO and proposed method on MapReduce framework in terms of Compression Ratio and Compression Time for human chromosomes (H.chr) as shown in Table 1.

Table 1. 1+8 Node cluster machine, results for 50 input samples.

Sample Name.	Compression Ratio			Compression Time(in sec)		
	GDC	FRESCO	Proposed	GDC	FRESCO	Proposed
H.chr-15	704	648	481	230	36.4	24.1
H. chr-17	626	549	367	560	27.4	22.9
H.chr-18	572	487	318	189	28.2	21.6
H.chr-19	554	472	428	430.4	22.3	17.5
H. chr-20	603	526	403	197.7	17	19.2
H.chr-21	672	525	394	53.4	17	17.6
H.chr-22	830	752	579	81.2	17.3	17.4
H.chr-01	652	565	420	520	87	42.3
AT genome	141	122	104	160	42	21

It is observed that we have achieved comparable compression ratio with GDC and FRESCO but our method is superior in terms of compression time in most of the cases. The compression time improves very well if file size is greater than default HDFS block size. In our case, HDFS default block size was 128MB. For Analysis, we selected first 50 samples of each some Human chromosome and 18 samples of Adenine and Thymine (AT) sequences shown in Table 2.

Table 2. Input dataset for human chromosomes.

Sample	File Size	Dataset Size
Human chr.1	249 MB	12.45 GB
Human chr.15	101.3 MB	4.83 GB
Human chr.17	78.2 MB	3.82 GB
Human chr.18	75.5 MB	3.68 GB
Human chr.19	57.1 MB	2.85 GB
Human chr.20	60.9 MB	3.1 GB
Human chr.21	46.6 MB	2.36 GB
Human chr.22	49.5 MB	2.4 GB
AT Genome	119 MB	1.89 GB

The Table 3 shows size of input dataset of human chromosome samples and 18 Arabidopsis Thaliana samples before compression and after compression. So for Human chromosome 1, each file is divided into two splits leading to more parallelism and less book-keeping information for name node. The variation in Compression Ratio and Compression Time can be seen in the following Figures 5, 6, and 7.

Table 3. Compression results in size and compression speed.

Sample Name	Input Dataset Size	Size after compression	Compression Speed
Human chr-15	4.83 GB	10.06 MB	201 MB/s
Human chr-17	3.82 GB	10.61 MB	166 MB/s
Human chr-18	3.68 GB	11.87 MB	170.4 MB/s
Human chr-19	2.85 GB	6.7 MB	162 MB/s
Human chr-20	3.1 GB	7.69 MB	162.4 MB/s
Human chr-21	2.36 GB	6.05 MB	134.1 MB/s
Human chr-22	2.4 GB	4.21 MB	137.9 MB/s
Human chr-01	12.45 GB	29.64 MB	294.5 MB/s
AT genome	1.89 GB	18.17MB	90.6 MB/s

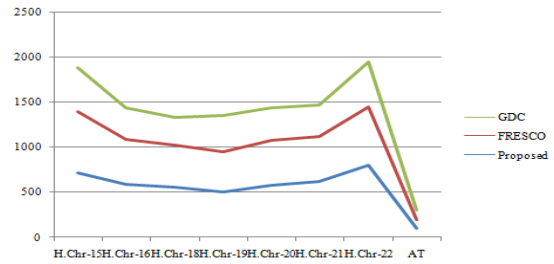

Figure 5. Chromosome wise analysis of compression ratio.

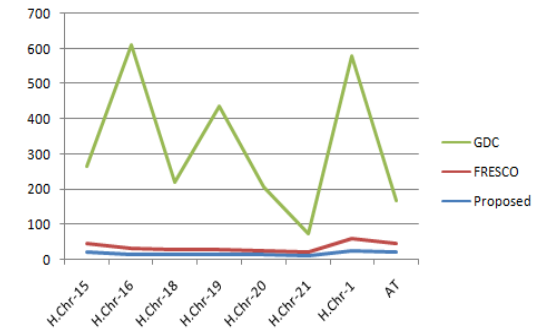

Figure 6. Chromosome wise analysis of compression time.

More the file size, more compression speed will be achieved. Sequences like Human Chromosome 1 gives best result in terms of Compression Speed and time.

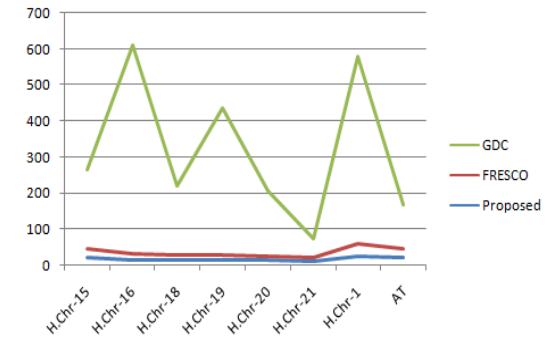

Figure 7. Chromosome wise analysis of compression speed.

We performed Second Order Compression on the results obtained after first order compression. The compression factor further increases based on wise selection of reference compressed sequences obtained after First order compression. We randomly selected five samples from compressed sequences, and prepared Hash Index to find the match between input first ordered compressed sequences and reference compressed sequence. Adding more compressed sequences in reference improved compression ratio further. For compressed Human chromosome of HG00097 sample from 1000 genome project, we compressed it further to 191 KB from 214 KB. Results for some of the compressed samples after second order compression can be analysed in the following diagram. Figure 8 shows the chromosome wise variation in the compression speed.

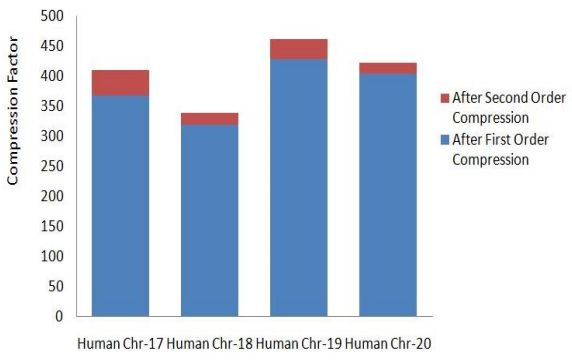

Figure 8. Chromosome wise analysis of compression speed.

6. Conclusions and Future Work

As per the experiments results it is found that proposed method gives efficient results in distributed environment in terms of compression ratio and time required for compression. The proposed method is comparable or near similar to other referential compression methods like GDC and FRESCO, but the time required for compression has been improved considerably when multiple DNA samples are considered for compression using MapReduce.

In future, referential genome compression can be implemented on Distributed Parallel Processing Frameworks like Apache Spark and finding maximal common sub-graphs. Apache Spark has shown better results in some cases over MapReduce Framework.

References

[1] Arafat A., "Multilayer Model for Arabic Text Compression," *The International Arab Journal of Information Technology*, vol. 8, no. 2, pp.188-196, 2011.

[2] Deorowicz S. and Grabowski S., "Robust Relative Compression of Genomes with Random Access," *Bioinformatics*, vol. 27, no. 21, pp. 2979-2986 2011.

[3] Deorowicz S., Danek S., and Grabowski S., "Genome Compression: A Novel Approach for Large Collections," *Bioinformatics*, vol. 29, no. 20, pp. 2572-2578, 2013.

[4] Deorowicz S., Danek A., and Niemiec M., "GDC 2: Compression of Large Collections of Genomes," Technical Report, 2015.

[5] El-Metwally S., Ouda O., and Helmy M., *An Introduction to Next Generation Sequencing Technologies and Challenges in Sequence* Assembly, Springer Science and Business, 2014.

[6] Kuruppu S., Puglisi S., and Zobel J., "Relative Lempel-Ziv Compression of Genomes for Large-Scale Storage and Retrieval," *in Proceedings of the 17th International Conference on String Processing and Information Retrieval*, Los Cabos, pp. 201-206, 2010.

[7] Kuruppu S., Puglisi S., and Zobel J., "Optimized Relative LempelZiv Compression of Genomes," *in Proceedings of the 34th Australasian Computer Science Conference*, Australia, pp. 91-98, 2011.

[8] Kuruppu S., Smith B., Conway T., and Zobel J., "Iterative Dictionary Construction for Compression of Large DNA Data Sets," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 9, no. 1, pp. 137-149, 2012.

[9] Pinho A., Pratas D., and Garcia S., "GReEn: A Tool for Efficient Compression of Genome Resequencing Data," *Nucleic Acids Research*, vol. 40, no. 4, pp. 1-16, 2011.

[10] Wandelt S. and Leser U., *Adaptive Efficient Compression of Genomes*, Algorithms for Molecular Biology, 2012.
[11] Wandelt S. and Leser U., "FRESCO: Referential Compression of Highly Similar Sequences," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 10, no. 5, pp. 1275-1288, 2013.
[12] Wang C. and Zhang D., "A Novel Compression Tool for Efficient Storage of Genome Resequencing Data," *Nucleic Acids Research*, vol. 39, no. 7, pp .1-6, 2011.

Raju Bhukya has received his P.hD in Computer Science and Engineering (CSE) from National Institute of Technology (NIT) Warangal in the year 2014. He is currently working as an Assistant Professor in the Department of CSE NIT Warangal, Telangana, India. He is currently working in the areas of Bio-Informatics and Data Mining.

Sumit Deshmuk is M.Tech student of CSE department at NIT Warangal. He has interest in analyzing information contained in genome sequences using distributed computing and Big Data Analytics framework like Hadoop Map-Reduce and Apache Spark to reduce cost and time involved in DNA Sequence Analysis.