International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 18(16s) 2026 pp. 1435-1457
https://doi.org/10.70917/ijcisim-2026-4676
Article

ClustChain: A Blockchain-Based Framework for Efficient Genomic Data Storage Using Attribute-Based Clustering

Anju Arya[1], Amita Malik[2]

[1,2]Deenbandhu Chhotu Ram University of Science and Technology, Murthal, Sonepat Haryana, India
[1]anjuarya12@ymail.com
[2]amitamalik.cse@dcrustm.org

Abstract: Genomic data has become an essential component of modern healthcare, driving innovations leading to more precise personalized care. While the exponential growth of genomic data opens significant opportunities for personalized healthcare, it has introduced several challenges related to genomic data. Among these, data storage and data retrieval remain the major concerns due to the large size of genomic data, while issues related to data integrity, data privacy and data security complicate the situation further. To address these challenges, blockchain technology has been explored as a viable solution to provide tamperproof, secure, and trustworthy data management solution. This paper proposes an on-chain storage model for genomic data management on blockchain platform. The proposed solution aims to reduce storage requirement through an attribute-based clustering mechanism in combination with a data compression method applied to genomic data. At the same time, it focuses on reducing stream data retrieval time through a query-aware data organisation in a data stream. Multichain platform and python 3 have been used to perform the experiment. The results indicate an average reduction in storage requirement of 18% for 10k reads, 20% for 100k reads, and 22% for 1000k reads in a BAM file. However, the reduction can be even more depending upon the cluster to which the BAM reads are mapped. Additionally, there is an improvement in stream load time due to reduced storage requirements on data streams and supports both simple and complex queries. It offers a memory-efficient data management solution for genomic data.
Keywords: Genomic data, Streams, Disk storage, Clustering, Memory, Multichain

1. Introduction

Genomic data plays a transformative role in healthcare by providing detailed insights into an individual's genetic makeup and the functioning of the body at molecular level. This information facilitates more effective healthcare interventions, thereby supporting personalized care for the patients. Genomic data analysis supports various applications such as personalized medicine, disease risk prediction, targeted therapies, pharmacogenomics, disease progression, precision diagnostic, and public health and epidemiology at a broader level [1] [2] [3]. Consequently, the benefits of genomic data spans from an individualized care to population-level healthcare planning and decision-making.

The increasing integration of genomic information into healthcare systems has prompted medical institutions and other stakeholders to reconsider their data management strategies. Among various data management challenges, data storage and data retrieval remain the major concerns due to the large size of genomic data, while issues related to data integrity, data privacy and data security complicate the situation further. To address these challenges, blockchain technology has been explored as a viable solution to provide tamperproof, secure, and trustworthy data management solution. The inherent characteristics of blockchain technology such as decentralization, immutability, traceability, and accountability, makes it suitable for handling genomic data [4].

Most of the existing genomic data management solutions primarily focus on secure data storage, data privacy, and data sharing, while limited attention paid to storage scalability, optimized data retrieval, and efficient query mechanism. To address these concerns, this paper proposes an on-chain storage model for genomic data. The proposed solution aims to reduce storage requirement through an attribute-based clustering mechanism in combination with a data compression method applied to genomic data. At the same time, it focuses on reducing stream data retrieval time through a query-aware data organisation in a data stream. Additionally, the proposed mechanism generates chromosome-wise summary of genomic data stored on blockchain for every user. This information helps to avoid redundant entries, in case of multiple file uploads from the same user. It also supports both simple and complex queries

The remainder of this paper is organized as follows. Section 2 reviews the related literature, followed by the proposed methodology in Section 3. Section 4 presents the system architecture, followed by implementation details in section 5. Section 6 presents the experimental results and discussion. Section 7 concludes the paper and outlines future research directions

2. Related work

Various blockchain and non-blockchain based approaches are proposed by researchers to address the storage challenges associated with genomic data. Among non-blockchain based approaches, compression methods have been explored to be effective to reduce storage requirements of genomic data. Several compressors exist for SAM files [5], but majority of the methods support some kind of lossy compression. An attempt in [6] to introduced a privacy preserving standard SECRAM that utilizes reference-based compression and general compression approach, achieving18% storage improvement for BAM files. Several blockchain-based approaches have been proposed for healthcare data management [7] [8] [9] [10] [11] [12][13]. A number of studies focus on managing genomic data access logs on-chain while the genomic data is stored off-chain. The paper [14] presents a baseline and enhanced method for efficient logging & querying of genomic data using a hierarchical timestamp structure. The authors in [15] presented a two-level indexing scheme for faster data retrieval of genomic log data and a binary search technique for timestamp based range queries. The authors in paper [16] utilized multichain data streams to store genomic access logs in the key filed of the stream. The data field stores data in hex format resulting in slower data retrieval and increased query time. The authors in [17] presented another on-chain storage solution for data access logs in combination with bucketization and batch loading concept to handle complex queries. The authors in [18] presented forensic DNA database framework that store cryptographic signatures on blockchain while maintaining DNA profiles in external databases.

Some solutions have focussed on small scale genomic data on blockchain. The authors in [19] presented a time & space efficient solution for storing pharmacogenomic data (gene-variant-drug) using an indexing based multi-mapping approach. While the authors in [20] presented an array-based storage solution with multiple indexing methods for gene-drug interaction data.

Limited studies have focussed on large-scale genomic data storage on blockchain. The authors in [21] proposed a blockchain applied FASTQ and FASTA Lossless Compression (BAQALC) approach that allows for the efficient transmission and storage of the DNA sequences. But the proposed work focuses on efficient sharing and storage, not on querying. However, the work presented in [22] involves an on-chain storage and query solution to manage large genomic datasets on blockchain using a reference-based data compression and genomic location-based binning method to speed-up query process.

Since genomic files contain multiple reads aligned to same genomic positions, there are chances of high similarity among multiple reads. Clustering concept can be utilised for grouping similar data based on features or thresholds. Existing studies have applied clustering to genomic data compression [23], , gene expression analysis [24], [25] and microbial genomic sequence analysis [26]. These studies indicate the need for more effort on clustering-based methods to improve genomic data storage demand.

In literature, majority of the blockchain-based genomic storage solutions use off-chain storage model, while only data file hash, cryptographic links, or analysis results are stored on the blockchain (i.e. on-chain). Although this approach reduces overhead on blockchain, it relies on external storage systems for ensuring data availability and integrity. Whereas, on-chain storage system offers enhanced confidence in terms of data immutability, authenticity, and integrity. However, when it comes to on-chain storage solution, the large size of genomic data files presents a significant challenge due to higher storage demand and scalability concerns. Only a few approaches like [22] have attempted to store genomic data in BAM format on blockchain system. Therefore, this paper aims to develop a clustering-inspired on-chain storage solution for genomic data management.

```
@PG     ID:bwa.B-707A9E06       PN:bwa  VN:0.7.15-r1140 CL:/data/NYGC/Software/bwa/bwa-0.7.15/bwa mem -Y -K 100000000 -t 16 -R @RG\tID:HG00114_AGCTCGCT-GCAGAATC_HFKW
@PG     ID:GATK PrintReads-7676B0A4     VN:3.5-0-g36282e4       CL:readGroup=null platform=null number=-1 sample_file=[] sample_name=[] simplify=false no_pg_tag=fals
@PG     ID:samtools     PN:samtools     PP:GATK PrintReads-7676B0A4     VN:1.16.1       CL:samtools view -ho chr1[100].sam HG00114.loci.bam chr1:56994629-56995006
A00404:41:HFH25DSXX:3:1108:15284:30279  83      chr1    56994629        60      150M    =       56994418        -361    ACTCCCCAATAAAGCTTGCTCTTTTTAGCTCAGGGCCAGGGAAAC
A00404:40:HFGYNDSXX:4:1419:15989:27242  1187    chr1    56994634        60      150M    =       56994859        375     CCAATAAAGCTTGCTCTTTTTAGCTCAGGGCCAGGGAAACACAGG
A00404:40:HFGYNDSXX:4:1419:15998:27226  163     chr1    56994634        60      150M    =       56994859        375     CCAATAAAGCTTGCTCTTTTTAGCTCAGGGCCAGGGAAACACAGG
A00404:40:HFGYNDSXX:4:1241:23411:19993  99      chr1    56994643        60      150M    =       56994836        343     CTTGCTCTTTTTAGCTCAGGGCCAGGGAAACACAGGCTGTCACTT
A00404:41:HFH25DSXX:2:1206:30282:14074  83      chr1    56994645        60      150M    =       56994333        -462    TGCTCTTTTTAGCTCAGGGCCAGGGAAACACAGGCTGTCACTTGG
A00404:41:HFH25DSXX:2:2206:27959:10300  1107    chr1    56994645        60      150M    =       56994333        -462    TGCTCTTTTTAGCTCAGGGCCAGGGAAACACAGGCTGTCACTTGG
A00404:41:HFH25DSXX:4:1202:28013:22451  163     chr1    56994647        60      150M    =       56995089        592     CTCTTTTTAGCTCAGGGCCAGGGAAACACAGGCTGTCACTTGGTC
A00404:41:HFH25DSXX:4:2202:28709:33082  1187    chr1    56994647        60      3M1D147M        =       56995089        592     CTCTTTTAGCTCAGGGCCAGGGAAACACAGGCTGTCA
A00297:35:HFKWWDSXX:3:2628:1823:25128   163     chr1    56994651        60      150M    =       56994825        324     TTTTAGCTCAGGGCCAGGGAAACACAGGCTGTCACTTGGTCAGAT
A00297:35:HFKWWDSXX:1:1512:7627:11600   99      chr1    56994659        60      150M    =       56995098        589     CAGGGCCAGGGAAACACAGGCTGTCACTTGGTCAGATGGAATCTT
A00297:35:HFKWWDSXX:1:1512:7744:11052   1123    chr1    56994659        60      150M    =       56995098        589     CAGGGCCAGGGAAACACAGGCTGTCACTTGGTCAGATGGAATCTT
A00404:41:HFH25DSXX:3:1213:13458:16344  147     chr1    56994661        60      150M    =       56994332        -479    GGGCCAGGGAAACACAGGCTGTCACTTGGTCAGATGGAATCTTAG
A00404:41:HFH25DSXX:1:1355:20663:27853  163     chr1    56994665        60      150M    =       56994903        388     CAGGGAAACACAGGCTGTCACTTGGTCAGATGGAATCTTAGGTAA
A00404:40:HFGYNDSXX:3:1363:25355:24799  163     chr1    56994666        60      150M    =       56994996        480     AGGGAAACACAGGCTGTCACTTGGTCAGATGGAATCTTAGGTAAA
A00404:41:HFH25DSXX:1:1542:7771:29293   83      chr1    56994678        60      150M    =       56994446        -382    GCTGTCACTTGGTCAGATGGAATCTTAGGTAAAGCAATTTCCTCT
A00297:35:HFKWWDSXX:4:2237:26386:10332  147     chr1    56994683        60      150M    =       56994396        -437    CACTTGGTCAGATGGAATCTTAGGTAAAGCAATTTCCTCTTCCGA
A00404:41:HFH25DSXX:3:1357:27416:34569  163     chr1    56994689        60      150M    =       56994868        329     GTCAGATGGAATCTTAGGTAAAGCAATTTCCTCTTCCGATCCAAG
A00404:40:HFGYNDSXX:4:2137:18945:21684  83      chr1    56994692        60      150M    =       56994411        -431    AGATGGAATCTTAGGTAAAGCAATTTCCTCTTCCGATCCAAGCCT
A00404:40:HFGYNDSXX:2:1423:22299:24236  147     chr1    56994698        60      150M    =       56994330        -518    AATCTTATGTAAAGCAATTTCCTCTTCGGATCCTAGCCTGAATAT
A00297:35:HFKWWDSXX:1:1375:7979:19100   83      chr1    56994706        60      150M    =       56994443        -413    GTAAAGCAATTTCCTCTTCCGATCCAAGCCTGAATATCCAGTGGA
A00297:35:HFKWWDSXX:2:2242:25192:13432  163     chr1    56994707        60      150M    =       56995175        618     TAAAGCAATTTCCTCTTCCGATCCAAGCCTGAATATCCAGTGGAA
A00404:40:HFGYNDSXX:3:2545:5385:21809   83      chr1    56994707        60      150M    =       56994564        -293    TAAAGCAATTTCCTCTTCCGATCCAAGCCTGAATATCCAGTGGAA
A00404:41:HFH25DSXX:4:2124:23990:29512  99      chr1    56994708        60      150M    =       56995395        837     AAAGCAATTTCCTCTTCCGATCCAAGCCTGAATATCCAGTGGAAA
```

Fig-1: Snippet of BAM file

3. Genomic Dataset

The sequenced genomic data used in this paper is stored in SAM/BAM format [27][28] and the data items are in the form of reads. The information available in SAM/BAM files can be categorized into 2 sections namely, header section and alignment section. The header section stores metadata related to genomic data available in BAM file, while actual genomic data is stored in alignment section. Figure 1 shows the snippet from a BAM file. The first 3 lines represents header data and the remaining lines represents alignment data. It is not possible to show both sections completely in a single figure, since the header section covers several lines followed by genomic reads of alignment section, Therefore, for a comprehensive understanding of information available in these sections, figure 2 shows few lines representing different categories of metadata in header section and a sample read is shown in alignment section. The third section 'SAM fields' is not part of BAM file but it is added in figure 2 to provide a brief understanding of BAM attributes, also called fields, and the way to read their corresponding value when a genomic read is parsed. Note that all values in a genomic read are separated by spaces.

The human DNA comprises of 23 kinds of chromosomes, each varying in length based on the number of base pairs or nucleotides stored in them. To handle this genomic data on blockchain platform, chromosome-wise data management can be opted for better performance in terms of data storage and retrieval.

Header Section

@HD VN:1.5 GO:none SO:coordinate
@SQ SN:chr1 LN:248956432 M5:6aef897c3d6ff0c78aff06ac189178dd
UR:/gpfs/internal/sweng/production/Resources/GRCh38_1000genomes/GRCh38_full_analysis_set_plus_decoy_hla.fa
.....
@RG ID:HG00114_AGCTCGCT-GCAGAATC_HFGYNDSXX_L001 PL:illumina PM:Unknown LB:HG00114
DS:GRCh38 SM:HG00114 CN:NYGenome PU:HFGYNDSXX.1.AGCTCGCT
.....
@PG ID:GATK PrintReads-637DDBE0 VN:3.5-0-g36282e4 CL:readGroup=null platform=null number=-1 sample_file=[]
sample_name=[] simplify=false no_pg_tag=false
.....

Alignment Section

A00297:35:HFKXWDSXX:4:2270:30825:14043 163 chr1 56996080 60 150M = 56996444 521
AGTTAGTGTAAACTGTGGGCTTTAAATGATAACCATTACCTTCATGTATCA CTTTTCGCTTTGTGAACAAGTA TACA
ATCATTTTTCAAATGAATCTCCCAAATCATTTTAGTTTCTATTG ATATGAAGAATCAAATGCAGGCCTATTCA
??+????????????5??????????
??????????????????5???? MC:Z:150M MQ:i:60 AS:i:150 XS:i:0 MD:Z:150 NM:i:0 RG:Z:HG00114_AGCTCGCT-
GCAGAATC_HFKWWDSXX_L004 PG:Z:MarkDuplicates
.....

SAM/BAM fields

QNAME: A00297:35:HFKXWDSXX:4:2270:30825:14043
FLAG: 163
RNAME: chr1
POS: 56996080
MAPQ: 60
CIGAR: 150M
RNEXT: =
PNEXT: 56996444
TLEN: 521
SEQ:
AGTTAGTGTAAACTGTGGGCTTTAAATGATAACCATTACCTTCATGTATCACTTTTCGCTTTGTGAACAAGTATACAATCATTTTTCA
AATGAATCTCCCAAATCATTTTAGTTTCTATTGATATGAAGAATCAAATGCAGGCCTATTCA
QUAL:
??+????????????5??????????
??????????????????5????
TAGS: MC:Z:150M MQ:i:60 AS:i:150 XS:i:0 MD:Z:150 NM:i:0 RG:Z:HG00114_AGCTCGCT-
GCAGAATC_HFKWWDSXX_L004 PG:Z:MarkDuplicates

Fig-2: Sample of various information in SAM/BAM file

4. Proposed methodology

To address the storage concern of genomic big data, the proposed method applies an attribute-based clustering approach in combination with a compression method to reduce the overall storage demand. There are 2 levels of processing performed on BAM reads to reduce the overall storage requirement.

Level 1: Modify SEQ field

The SEQ field contains sequencing data. The length of SEQ field depends on the sequencing machine, typically around 100–150 bases per read. The size of this field is reduced by either applying reference-based compression against a reference human genome [22] or using Zlib [29] to compress the long string.

Level 2: Cluster BAM Reads

To further reduce the overall size of the genomic data, reads are grouped genomic position-wise (ie. POS value). All the reads having same POS value are grouped together and then clustered based on similarity levels.

Multichain streams, an append-only structures, are utilised in the proposed approach to store chromosome-wise data in the form of key-value pairs. The contributions of the proposed approach to resolve the high storage requirement issue of genomic data is summarised below.

- **Clustered BAM data:** The processed BAM file obtained from raw genomic data contains multiple reads aligned to same genomic location or having same POS field value for a chromosome due to high coverage depth. It has been observed that the reads having same POS value for a particular chromosome, may have

identical or different values for various BAM fields. The BAM reads are clustered into 3 different groups based on their similarity level to reduce the overall storage requirement on blockchain.

- **Maintaining genomic coverage information:** The genomic coverage information is maintained about the genomic locations covered in the sequencing data stored on-chain. It would help in avoiding redundant storage of same data repeatedly on blockchain when another genomic data file of the same individual is uploaded on blockchain in future.
- **Reduced Stream Load Time:** Multiple streams of fixed length are created to store genomic data for different chromosomes of varying lengths. However, the stream load time depends upon the number of reads stored in a stream but still it would not be too high and somewhat uniform since every stream has a fixed range of genomic locations for which reads are stored in it.

4.1 Clustering Approach

Clustering involves grouping of similar data items based on their values or category This concept is suitable for genomic data obtained from NGS technology due to the inherent similarity between multiple data items (called reads) aligned to same genomic locations. It aims to optimize storage requirement of genomic data on blockchain by organizing genomic sequences or reads into clusters.

Most of the BAM fields, also called attributes, are important for analysis depending upon the purpose. In the proposed framework, QNAME and optional TAG fields are treated as non-prime attributes because they are not directly required for routine variant-centric analyses. Furthermore, several alignment-related TAGs, such as MD and NM, contain information that can be derived from the alignment data and reference genome when needed. Therefore, optional tag fields need not be included among the prime attributes from a primary storage perspective, as they primarily provide supplementary alignment or metadata information. Tags required for specific clinical or research workflows may be preserved separately, whereas the remaining tags may be regenerated, or excluded to achieve improved storage efficiency.

In order to apply clustering concept, all the BAM attributes are analysed to identify prime attributes which can be used to group genomic reads in a cluster. Prime attributes are those BAM attributes which contain significant information and can't be ignored. Among the 12 BAM fields, first 11 fields are mandatory whose value must be available in a genomic read. Also, first field, QNAME is a unique identifier for a genomic read. This means the information primarily needed for genomic analysis is in the remaining 10 fields, namely, FLAG, RNAME, POS, MAPQ, CIGAR, RNEXT, PNEXT, TLEN, SEQ, and QUAL. These fields can be referred as prime BAM attributes for clustering process. The prime BAM attributes have the higher possibility of having same values for a group of reads with same POS value and RNAME value (ie. Chromosome number).

To optimise genomic data storage, the clustering approach analyse the information stored in prime attributes. For a particular chromosome, the BAM reads having same POS value can be clustered into three groups depending up on the level of similarity between prime BAM fields, as mentioned below.

Cluster 1: This cluster comprises of BAM reads having all prime attributes same.

Cluster 2: This cluster comprises of BAM reads having all prime attributes same, except PNEXT, TLEN and QUAL.

Cluster 3: This cluster comprises of BAM reads having sequencing information different or most of the prime attributes being different.

The next section describes the system architecture and the data storage layout on the blockchain layer to implement the proposed mechanism in a query-aware manner.

4.1 System Description

This section covers the system architecture in general and a framework for data stream design on blockchain layer in accordance with the proposed methodology for optimizing storage requirement for genomic data and handling the supported queries. In this work, multichain platform is used for implementation. There are 2 kinds of nodes in the blockchain network. One is primary nodes responsible for blockchain creation and management. The other nodes are secondary, also called user nodes, which subscribe to data streams for their activities.

System Architecture

This work utilizes a blockchain-enabled genomic data management framework, as shown in figure 3, consisting of five layers as discussed below:

a) **Data Acquisition Layer**

In data acquisition layer, stakeholders or users upload their genomic data file (BAM file), obtained from hospitals, testing labs, research labs, or genomic data repositories. Each file gets a file identifier to manage genomic data for different users and same users in case of multiple file uploads. Multichain streams, a key-value datastore, are used to store genomic data. The uploaded genomic file is then transferred to the data pre-processing stage to transform the genomic data into a data stream-supported format.

b) **Data Pre-processing Layer**

This layer is responsible for reading genomic data file uploaded by the user and transforming it into a format supported by data streams. At this stage, the genomic reads are extracted from BAM files uploaded by the user. The information obtained from header section is stored on metadata stream whereas the genomic reads extracted from the alignment section undergo compression and data clustering process before being stored on the data streams. The output of this layer is a set of optimized genomic data clusters that are subsequently stored on blockchain layer.

c) **Blockchain Layer**

This layer follows an on-chain storage design. That means, all the genomic data will be stored on blockchain data streams. Although, this arrangement will increase storage demand for blockchain but it will also ensure data non-tampering and availability. This layer primarily maintains 2 kinds of streams, namely, metadata stream and data streams. As briefed in previous layer, metadata stream contains the information extracted from header section while the data streams contain the grouped and optimized form of genomic data. The blockchain layer is responsible for maintaining security, transparency, and auditability.

d) **User Access Layer**

layer consists of authorized stakeholders including researchers, healthcare professionals, genomic data owners, and other organizations. Users need to subscribe to blockchain data streams initially. Once subscribed, user can upload genomic data file and query data streams to fetch genomic data.

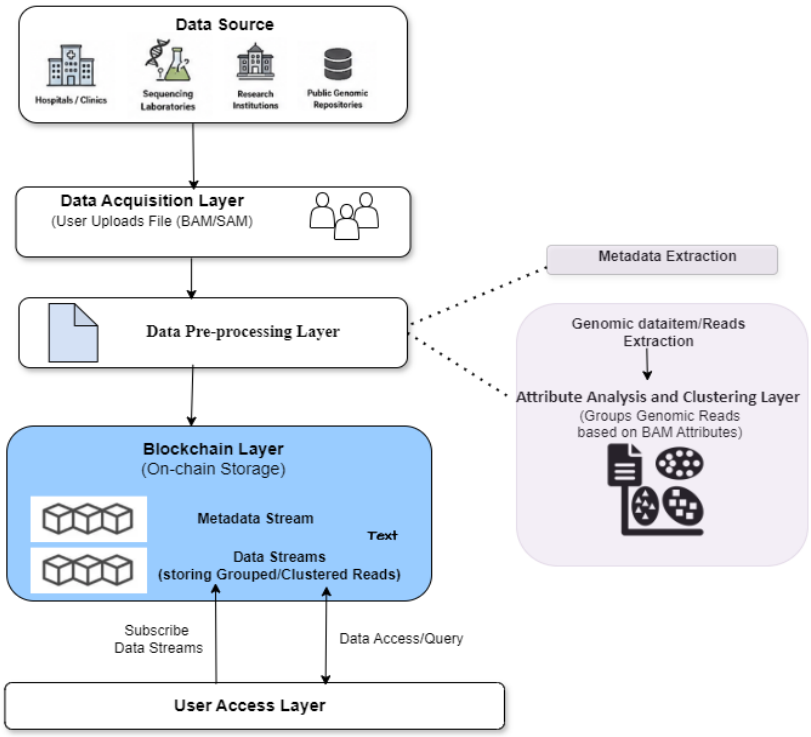


Fig-3: A Framework for Genomic Data Management

4.3 Multichain Data Stream Design Framework

The human DNA comprises of 23 kinds of chromosomes, each varying in length based on the number of base pairs or nucleotides stored in them. To handle this genomic data on blockchain platform, chromosome-wise data management has been opted for better performance in terms of data storage and retrieval. Multichain platform is utilized to manage genomic data due to its suitability for database applications. Streams are used to store data in key-value pairs on multichain platform and characterized by uniform access time. This section presents the structure of streams on multichain platform. To develop the proposed system, 2 categories of stream are created, namely, metadata stream and data stream. Genomic reads, extracted from alignment section, are of 2 types, namely, mapped and unmapped. Mapped reads are aligned to the reference genome, while unmapped reads are not aligned to any genomic location of a chromosome within the reference genome. Thus, the data streams can be further categorised as mapped data streams and unmapped data streams. Figure 4 shows the stream design framework for the blockchain layer.

a) **Mapped Data Streams**

For storing mapped reads, multiple mapped data streams are created chromosome-wise for efficient data retrieval. Maintaining genomic data chromosome-wise helps to reduce search space, especially when query involves a particular chromosome. It also reduces stream loading time since only reads belonging to the queried chromosome will be loaded rather than loading the entire genome in system's memory. Thus, different data streams are created for different chromosomes, that is, chromosome 1 to 24. However, the size of every chromosome is different and in millions. If a complete chromosome is stored in a single data stream, the stream loading time will be different for different chromosomes because of their varying length. In order to provide a uniform stream loading time for every data stream, the size of a data stream must be fixed. This means, the number of streams created for every chromosome depends upon the stream size fixed and maximum length of that chromosome. This arrangement also helps to provide reduce search space within a chromosome. The number of data streams required for a chromosome i, can be calculated using equation 1. For instance, the maximum length of chromosome 1 is 249 million approximately, so for a stream size of 10million reads, the numbers of data streams created for storing chromosome1 data will be 25 (~249 million/10

million). The streams created for a particular chromosome are named using chromosome name followed by stream number, for example chr[i]stream[j]), where i represents chromosome 1 to 24 and j represents stream number.

$$num\_of\_streams_{chromosome[i]} = \frac{Chromosome[i]_{Size}}{Data\ Stream_{Size}} \quad (1)$$

Data stream name also represents the range of genomic locations as per which reads are stored in that data stream. For instance, chr1streamx indicates streamx created for chromosome1 which stores all reads having 'chr1' value in RNAME field and POS field value in the range from stream_len* (x-1)+1 to stream_len * x, where stream_len represents stream size .

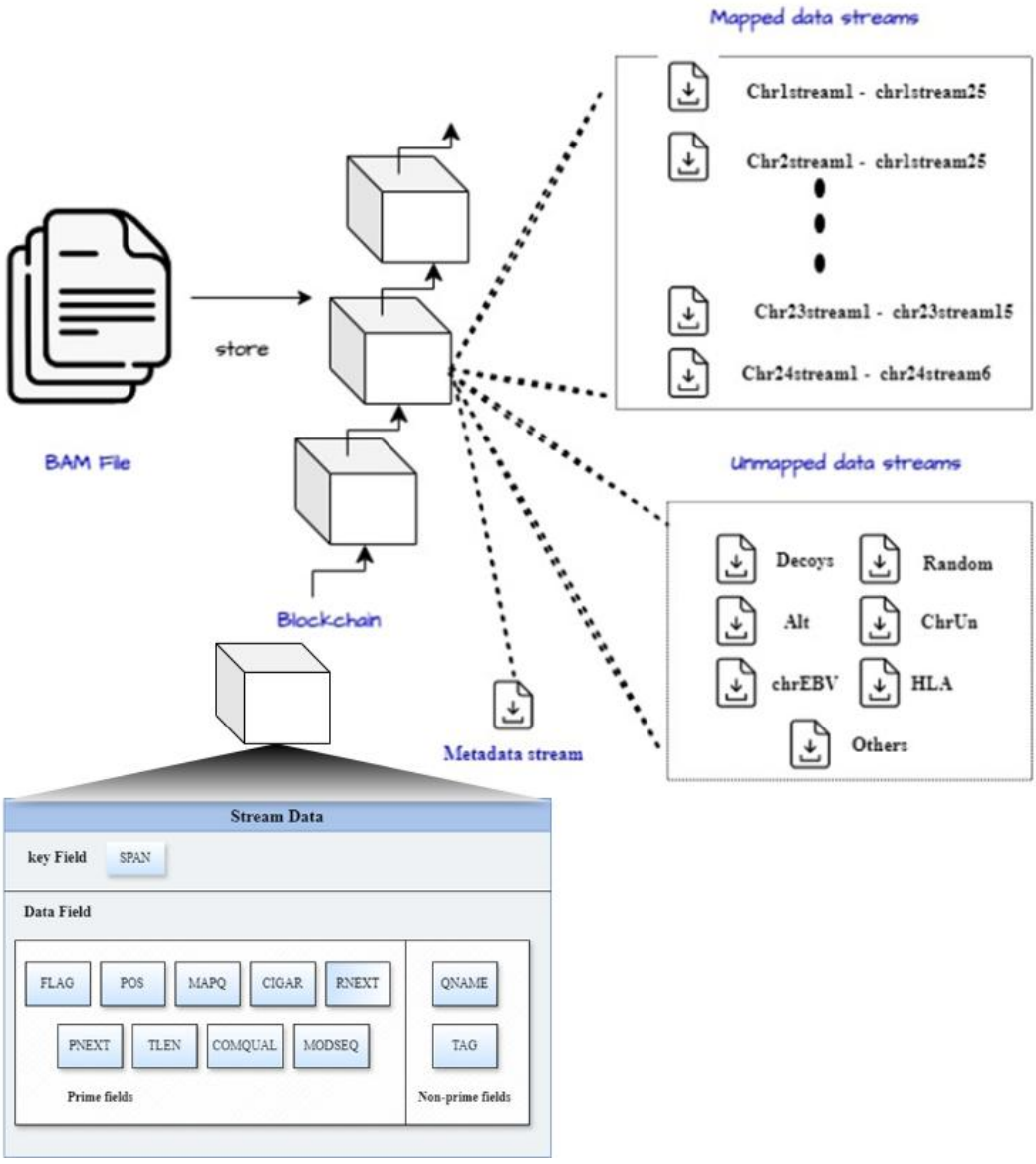


Fig-4: A Framework for Stream Design on Multichain Blockchain

b) **Unmapped Data Streams**

Unmapped genomic reads are those reads that are not aligned to any genomic location of a chromosome within the reference genome. However, they may become mappable in the future with improvements in reference genome, genomic annotations, or alignment methods. These reads are stored in the data streams created according to their

category such as Decoys, Random, Alt, ChrUn, chrEBV, HLA and Others. The category 'Others' refers to reads which doesn't fall in listed categories of unmapped reads. No clustering process is applied on these reads since they are not mapped to any identifiable genomic location.

4.4 Data Stream Structure

In the previous sections, the types of data streams and the distribution of genomic reads among these data streams on the basis of chromosome and genomic location range, has been discussed. This section describes the internal structure of a data stream to provide a better understanding of how internally the BAM attributes are organized when a genomic read is stored. This would help in designing the appropriate query mechanism and the choice of built-in API method.

To store the genomic data from a BAM file, the data is processed in the form of reads. Each read can be represented as a key-value pair on multichain streams. The organization of BAM attributes of a genomic read on these streams is depicted in figure 4. The reads to be stored in a particular stream are filtered by mapping their POS field value to genomic range represented by that stream. In a genomic read, the POS value indicates the starting genomic location in SEQ field. Also, SEQ field doesn't contain sequencing information for a single location, instead it contains value for a range of genomic locations which can be calculated using POS value (i.e. initial location) and the length of SEQ field. That means, if a read whose POS value is nearly in the last series of genomic locations of a data stream, may contain sequencing information for genomic locations represented by the next subsequent data stream for a chromosome. Therefore, a variable 'SPAN' is made part of the key field to indicate if the current stream contains reads having sequencing information of the genomic locations represented by other adjacent or nearby streams. The variable SPAN act as key to fetch all the reads from the identified data stream and adjacent streams, which might contain the queried genomic locations. Note that, using POS value as key to search genomic reads containing queried locations is not possible because POS value indicates only the first genomic location of the genomic range covered by SEQ field in a read. On multichain stream, data can be either loaded completely without key value or partially by matching key values. There is no provision to apply relational operators directly on key field value, it works on equality condition only. Thus, SPAN value can be used to fetch reads from the identified mapped streams as well as from the adjacent streams for the queried locations.

The BAM attributes stored in the data field of streams can be organized into 2 sets, prime and non-prime fields, to optimise genomic data storage using the clustering approach. The prime fields include those BAM fields which are used in matching operation to create clusters for a group of reads belonging to a chromosome and having same POS value. RNAME, QNAME and TAG are not included in prime field list because QNAME is a read identifier, RNAME represents chromosome number part of stream identifier and TAG contain additional data.

Additionally, there is a modification in handling of SEQ field of a genomic read in the proposed method. Instead of 'SEQ' field, 'MODSEQ' field is used to store sequencing data (refer equation4). This field may contain a 'MODCIGAR' value [22] (refer equation 2) representing only modified values in SEQ field position-wise in comparison to reference genome or 'COMSEQ' value which is obtained by compressing SEQ field value (refer equation3).

$$MODCIGAR = \begin{cases} , & if\ C \in \{*, , None\} \\ M(S,C,R), & otherwise \end{cases} \quad (2)$$

Where S is original sequence, C is original CIGAR string, R is reference genome sequence, M(S,C,R) is a functions that generates sequence of mismatches or insertions [22].

$$COMSEQ = B(Z(S) \quad (3)$$

$$MODSEQ = \begin{cases} modcigar, & if\ \ |\ modcigar\ |<|\ comcigar \\ comseq, & otherwise \end{cases} \quad (4)$$

Similarly, instead of ‘QUAL’ field, the 'COMQUAL' field contains compressed 'QUAL' field value (refer equation 5).

$$COMQUAL = B(Z(Q)) \quad (5)$$

Where Q is quality string, Z(x) is zlib compression function and B(x) is base64 encoding function.

Note that, RNAME is not stored in data field since it can be later extracted from stream name which can be identified from RNAME and POS value. This arrangement helps in regenerating original BAM file later.

5. Implementation Details

The proposed work is implemented using Multichain. The data streams in multichain stores data as key-value pairs. The data stored in multichain streams is visible to every node in the network which has subscribed to these streams on primary node. The primary node, which created the streams, can grant and revoke permissions to other nodes with wallet address in the network for read and publish operations on multichain streams.

The whole process of building a blockchain and utilizing it for database operations can be divided into 3 stages namely, Chain creation, Data insertion, and Stream query. Figure 4.6 depicts the workflow of this whole system. The working details of each stage is discussed below.

5.1 Chain Creation

The steps for initializing chain on primary node are specified in algorithm 1 along with API method used. The chain can be created by calling by ‘CreateChain’ function with chain name as parameter. The ‘CreateChain’ function creates a chain and run the daemon process to activate it for various chain operations. The next step will be to create streams for storing mapped and unmapped genomic reads. The steps 6 to 9 shows steps to create mapped data streams. The steps 10 to 13 shows steps to create unmapped data streams. The number of streams (num_of_streams) created in step 6 depends on the size of stream (or stream length) and standard possible count of READs for different chromosomes in a BAM file. The stream length can be changed according to application requirement since it impacts the query processing time.

Algorithm 1: Chain Initialization

```
Procedure CreateChain (ChainName)
set parameters: stream_length, read_length, num_chromosome, chromosome_length
Call multichain-util create ChainName
Call multichaind ChainName -daemon

   // create streams to store mapped READs
for each chromosome i do
  num_of_streams =chromosome_length(i) / stream_length
  for i =1 to num_of_streams do
    Call multichain-cli ChainName create stream chr(i)stream(j)
    Call multichain-cli ChainName subscribe chr(i)stream(j)
  end for

   //create streams to store unmapped READs
```

```
for each x in ['Decoys', 'Random, 'Alt', 'ChrUn', 'chrEBV', 'HLA', 'Others'] do
   Call multichain-cli ChainName create stream 'unmapped'
   Call multichain-cli ChainName subscribe 'unmapped'
end for
return
```

5.2 Inserting data in streams

This section discusses the insertion process of genomic data obtained from BAM file into multichain streams. The algorithm 2 describes the steps to store BAM file data in data streams. During insert operation, the first step comprises of starting the blockchain. Before initiating the process of reading the BAM file, declare dataframe1 and dataframe2 for storing mapped and unmapped reads temporarily before actually storing them on data streams. These dataframes holds reads with same POS value temporarily, before being published on data streams. Now, open BAM file for fetching reads and for each read (see step 5), extract the field values and check if the read is mapped or unmapped (steps 7). If current read is mapped and has same POS value as reads grouped in dataframe1, the identified mapped read is stored in dataframe1 (steps 9 to 11), otherwise the unmapped reads are stored in dataframe2 (step 17). The 'publish' variable is used to check if all reads having same POS value have been stored in dataframe1, before a new read is processed with different POS value. The 'publish' value is set to 1 when a read with different POS value from the reads gathered in dataframe1, is fetched. When the 'publish' is set to 1(step 14), it means that all the mapped reads having same POS values are gathered in dataframe1, then calculate stream_name and key value before calling 'Cluster_Reads' procedure to cluster reads and publish them to the designated stream (steps 21-30). The 'key' value denotes 'SPAN' value which can be used to fetch reads covering queried genomic locations from multiple streams other than designated or mapped stream. Similarly, the unmapped reads in dataframe2 are published according to their category such as Decoys, Random, Alt, ChrUn, chrEBV, HLA and others (steps 31 to 43). Now, empty dataframe1 and dataframe2 to start the same process again for reads with new POS values (steps 44 to 50).

Algorithm 2: Inserting Data in Streams

```
Prodecure Insert (ChainName, Stream_length, BAMfile)
start chain
set publish=0
firstread=true
for each read in BAMfile do
   extract BAM fields
   identify mapped and unmapped reads
   // store mapped reads in a dataframe1 for same 'pos'
   if read is mapped then
      if 'POS' of read ==''POS' value in dataframe1 and firstread=true then
         calculate additional fields 'MODCIGAR', 'COMCIGAR', 'COMQUAL'
         add read to dataframe1
         set publish=0
      else
         set publish=1
      end if
```

```
 // store unmapped reads in a dataframe2 for same 'pos'
else
    add unmapped reads using dataframe2
end if
firstread=false
if publish =1 then
   // publish mapped reads
   if len(dataframe1)>0 then
       stream_num=int(POS/Stream_length)+1
       stream_name= RNAME+'stream'+stream_num
       if POS+len(SEQ) > stream_num* stream_len then
          key= 'F1'
       else
          key= 'F0'
      end if
      call Cluster_Reads(dataframe1, key, stream_name, ChainName)
   end if
   // publish unmapped reads
   if len(dataframe2)>0 then
      for each item in dataframe2 do
         key= QNAME+FLAG
         data= JSON(all BAM fields)
         unmapped=[ Decoys, Random, Alt, ChrUn, chrEBV, HLA, others]
           if type is in RNAME then
              stream= type
           else
              stream='others'
           end if
          call multichain-cli ChainName publish 'stream' 'key' 'data
     end for
  end if
empty dataframe1, dataframe2
set publish=0
if read is mapped then
   add current read in dataframe1 (do step 10,11) or
else
```

```
    add current read in dataframe1
  end if
exit
```

a) **Clustering Process for BAM Reads**

Figure 5 depicts clustering step during the insertion process of BAM file in data streams. The algorithm 3 describes the steps for clustering BAM fields. The BAM reads grouped for same POS value are passed as a python dataframe to the procedure 'Cluster_Reads' for clustering and publishing to the streams. In first step, 3 clusters are declared as c1, c2 and c3. The steps 2 to 5 depict the approach used to calculate or group reads in c1, c2 and c3 according to cluster rules. The remaining steps shows the process to calculate data field value while key value is already passed as a parameter in this procedure. The steps 7 to 12 describe the steps for calculating data field value for publishing c1 cluster data on stream. Data1 represents a string comprising of prime field values separated by colon. Data2 represent python dataframe to store non-prime field values for the grouped reads or read set. Note that, data field calculated for stream publishing is in the form of a dictionary where 'common' attribute stores prime BAM field values compacted in the form of a string and 'other' attribute stores non-prime BAM fields as a python dataframe. The data field needs to be converted in JSON format before publishing on the stream.

Algorithm 3: Cluster and Publish Reads

```
Procedure Cluster_Reads (dataframe, key, stream, ChainName)
df=dataframe
//make clusters c1, c2, c2
c1=extract reads from df having all BAM fields same except QNAME, TAG
// extract remaining reads from df
tmp_df=df-c1
c2= extract reads from tmp_df having some BAM fields same ie. RNAME, POS, MAPQ, CIGAR, MODCIGAR
// extract remaining reads from tmp_df
c3= tmp_df-c2
//publish clustered reads in stream
publishitems=[]

//process cluster1
for each duplicate read_set in c1 do
   data1 = str(cluster_type: POS: FLAG: MAPQ: CIGAR: RNEXT: PNEXT: TLEN:
           MODSEQ: ID_list: COMQUAL)
   data2= read_set['QNAME', 'TAG', 'ID']
   data=JSON({'common':data1, 'other': data2})
   call multichain-cli ChainName publish 'stream' 'key' 'data'
end for

//process cluster2
for each similar read_set on 'FLAG', 'RNEXT' in c2 do
```

```
    data1=str(cluster_type:POS:FLAG:MAPQ:CIGAR:RNEXT:PNEXT_list:TLEN_list:
          MODSEQ: ID_list: COMQUAL_list)
    data2= read_set['QNAME', 'TAG', 'ID']
    data=JSON({'common':data1, 'other': data2})
    call multichain-cli ChainName publish 'stream' 'key' 'data'
end for


//process cluster3
for each read in c3 do
    data1=str(cluster_type: POS: FLAG: MAPQ: CIGAR: RNEXT: PNEXT: TLEN: MODSEQ:
          ID: COMQUAL)
    data2= read['QNAME', 'TAG', 'ID']
    data=JSON({'common':data1, 'other': data2})
    call multichain-cli ChainName publish 'stream' 'key' 'data'
end for
return
```

The steps 13 to 18 and steps 19 to 24 represent the similar steps followed for creating and publishing c2 and c3 cluster data on stream. Note that, the data1 value calculated in steps 8, 14, and 20 having different string patterns as per clustering level. For instance, data1 value for cluster1(c1) has ID_list denoting the list of ID values of reads merged in cluster 1, while other BAM field values doesn't have '_list' suffix attached because they have exactly same value for the merged read group. Thus, '_list' suffix denotes that the associated field has different values for that merged read group. Also, note that for cluster 3, data1 has no merged read (see step20) and every read is published separately due to low similarity. So, it is desirable to have fewer reads assigned to cluster 3 to have maximum storage reduction for a BAM file, but it again depends upon the reads and their field values.

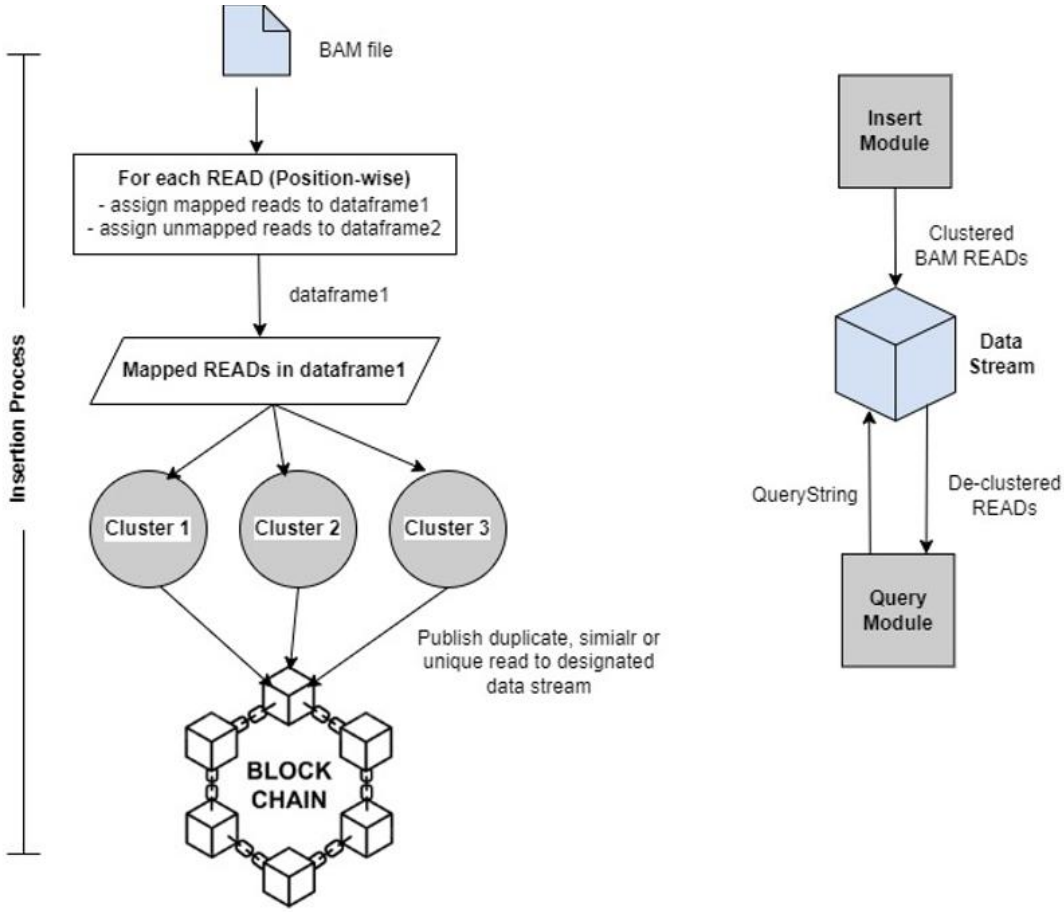


Fig-5: Clustering process during Insertion of BAM file

b) **Maintaining Genomic Coverage Information**

After the insertion process is completed, the insertion procedure collects an additional data about the range of genomic locations covered in genomic data stored on blockchain. Table 1 shows the metadata maintained for a sample BAM file stored on-chain. This information helps in optimizing data storage in future by providing information about the genomic locations for which sequencing information is already available on-chain for a user. It would help in deciding whether the new BAM file should be completely stored on-chain for the same individual. In such cases, only those genomic reads can be stored on blockchain which contains uncovered genomic information or genomic locations for which the sequencing information has changed with respect to reference genome. For instance, the genomic region of Chr1 covered in File1.bam can be avoided when File2.bam is stored on blockchain since there is no change in sequencing information for Chr1.

Table-1: Summary of genomic locations covered on-chain

File name	Total Reads	Read length	Genomic location covered (chromosome-wise)
File1.bam	100,000	150	'Chr': {1: ['56994629-57409724']}
File2.bam	1000,000	150	Chr': {1: ['56994629-58546724'],

			2: ['140231275-142132457'], 3: ['75906529-77649964'] }

5.3 Querying Data Streams

Different API commands are available in multichain for querying data streams. The 'liststreamkeyitems' method is used in this work to load complete stream data for evaluating query performance. Algorithm 4 describes the steps for fetching reads for a query string. First, identify the list of streams which may contain the reads covering the queried genomic location (QueryString) followed by declaration of result_df, a python dataframe to store the results. Then, fetch data from every stream available in stream list 'streams' (steps 3, 4).

Algorithm 4: Querying Data streams

ref_genome: *reference genome for mapping reads*

Sample QueryString: *chromosome: genomic location*

Procedure QueryRead (QueryString, ref_genome, ChainName)

// identify streams

identify streams as per genomic locations in Querystring

declare dataframe result_df

for *each stream in streams* ***do***

data=call multichain-cli ChainName 'liststreamkeyitems' stream

for *each item in data* ***do***

data=extract 'data' field from item

prime_string=extract 'common' field value from data

non_prime_df=extract 'other' field values from data

pos= extract POS field value from prime_string

if *pos is in QueryString* ***then***

identify cluster type

RS=extract BAM field values for read(s) from prime_string as per cluster type

//construct original value for SEQ, QUAL for read set (RS)

R=construct single read or multiple reads by mapping RS and non_prime_df as per cluster type

add R to result_df

end if

end for

end for

return result_df

Now, for every item in fetched data, extract prime and non-prime BAM attributes from data field of fetched data items using reverse logic of clustering approach used during insertion process (steps 5 to 8). In the next step, POS field value (pos) is extracted from prime fields and compared against QueryString to filter reads matching the queried genomic locations (steps 9,10). Then, for all matched read items extract BAM fields and construct reads (R) by associating prime and non-prime BAM fields as per cluster type (steps 11 to 13). Finally, add the reads R to result_df, a python dataframe which allows complex queries on multiple BAM fields with 'and' and 'or' operations. Figure 6 shows the sample genomic data published on stream.

```
read_df:                                        QNAME  ...                                                TAG
0    [A00297:35:HFKWWDSXX:4:1646:27534:28322]  ...  [[('MC', '150M'), ('MQ', 60), ('AS', 140), ('X...
1     [A00404:40:HFGYNDSXX:2:1636:9200:36620]  ...  [[('MC', '150M'), ('MQ', 60), ('AS', 140), ('X...
2     [A00404:40:HFGYNDSXX:1:1144:8847:34976]  ...  [[('MC', '150M'), ('MQ', 60), ('AS', 140), ('X...
3    [A00404:41:HFH25DSXX:2:2669:15863:21167]  ...  [[('MC', '150M'), ('MQ', 60), ('AS', 135), ('X...
4    [A00404:40:HFGYNDSXX:3:2129:11162:24173]  ...  [[('MC', '150M'), ('MQ', 40), ('AS', 140), ('X...
..                                        ...  ...                                                ...
599  [A00297:35:HFKWWDSXX:4:1145:15519:25332]  ...  [[('MC', '150M'), ('MQ', 60), ('AS', 145), ('X...
600   [A00404:40:HFGYNDSXX:1:1251:13476:5165]  ...  [[('MC', '150M'), ('MQ', 60), ('AS', 145), ('X...
601   [A00404:40:HFGYNDSXX:1:1251:13476:5165]  ...  [[('MC', '150M'), ('MQ', 60), ('AS', 145), ('X...
602  [A00404:41:HFH25DSXX:3:1242:10800:18881]  ...  [[('MC', '150M'), ('MQ', 60), ('AS', 145), ('X...
603  [A00404:41:HFH25DSXX:3:1242:10800:18881]  ...  [[('MC', '150M'), ('MQ', 60), ('AS', 145), ('X...
```

Fig-6: Sample output

The next section discusses the results obtained after implementing above discussed algorithms for storing and querying genomic data on multichain blockchain.

6. Results and Discussion

To evaluate the efficacy of above discussed methodology of clustering genomic data for optimizing storage, python3 language is used to implement code on multichain platform. The experiment is performed on node having specifications comprising of Intel(R) Core(TM) i7-8665U CPU, 16 GB of RAM and 1TB of HDD with Ubuntu 20.04 LTS OS and Multichain 2.0.6.

Table-2: List of key parameters

Parameter	Value
No. of Chromosome	25
Stream size	10000000 reads
Default read length	100
Max. key length	256 bytes

For evaluating the performance of the proposed storage approach, it is compared against the storage method used in [22], since until now it is the only relevant research work available to store genomic data on-chain. In order to have true comparison, the proposed approach is tested under similar conditions, parameters, and datasets, as used for simulating the work in [22]. For a comprehensive evaluation and comparison, different queries based on POS values, covering both single genomic locations and ranges of genomic locations, were executed on datasets of varying sizes using both the proposed approach and SAMchain approach [22]. Table 2 shows the list of key parameters affecting storage and query performance.

6.1 Analysing Insertion process

This sub-section discusses the impact of proposed attribute-based clustering inspired storage method on insertion process for storing BAM file on blockchain. The impact on time and memory (RAM) requirement during insertion process is discussed below.

a) **Analysing Time and RAM usage during Inserting data in Streams**

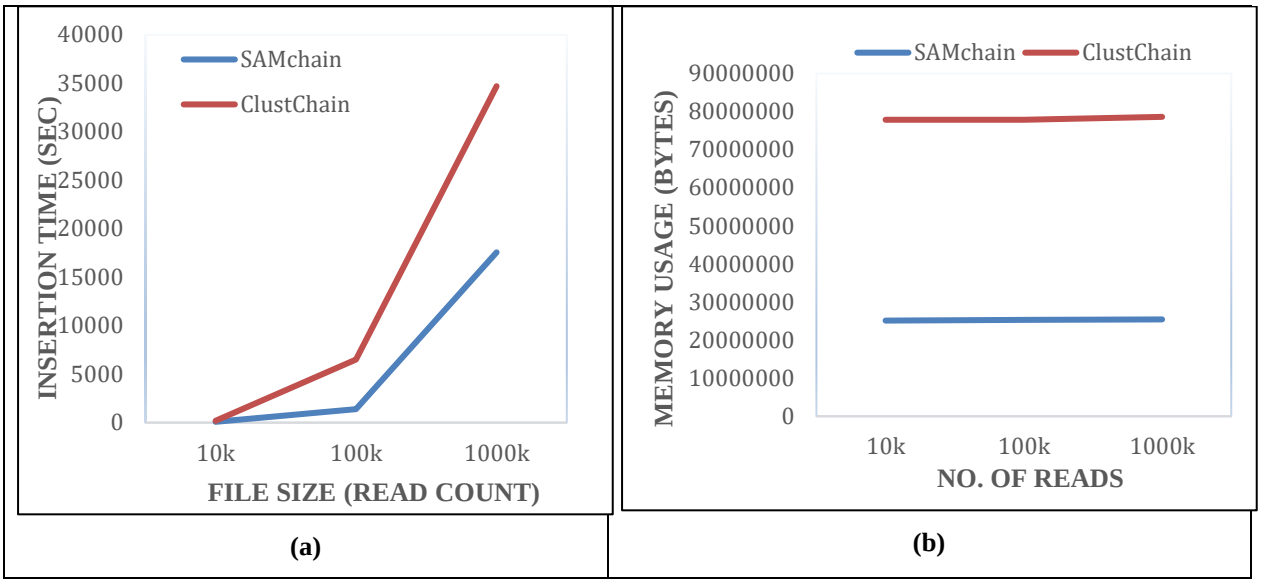


Fig-7: Impact on Time and Memory requirement during insertion process

Figure 7 depcits the impact of the proposed clustering based storage method and SAMchain on time and memory requirements during insertion process in blockchain. The performance of the system can be studied for different file sizes (in terms of read count) to avoid biasness of any kind. It is clear from the Fig. 7a that insert time increases drastically as the file size increases and this increase is even more for GenoChain because of the additional overhead of processing clusters for storing BAM reads. Since, it is a one time process, the increased insertion time is acceptable, given the long-term availability of the genomic data

It has been observed that the memory requirement is uniform for different file sizes, as depicted in Fig. 7b. The reason behind higher memory requirement for ClustChain chain compared to SAMchain is primarily due to the clustering process, which requires storing of multiple reads in memory at the same time for a particular genomic location. Whereas SAMchain process one genomic read at a time, resulting in lowe memory consumption. Although the clustering process demands additional memory overhead, this trade-off is manageable because it can reduce the overall storage requirement of genomic data, as discussed in the next section.

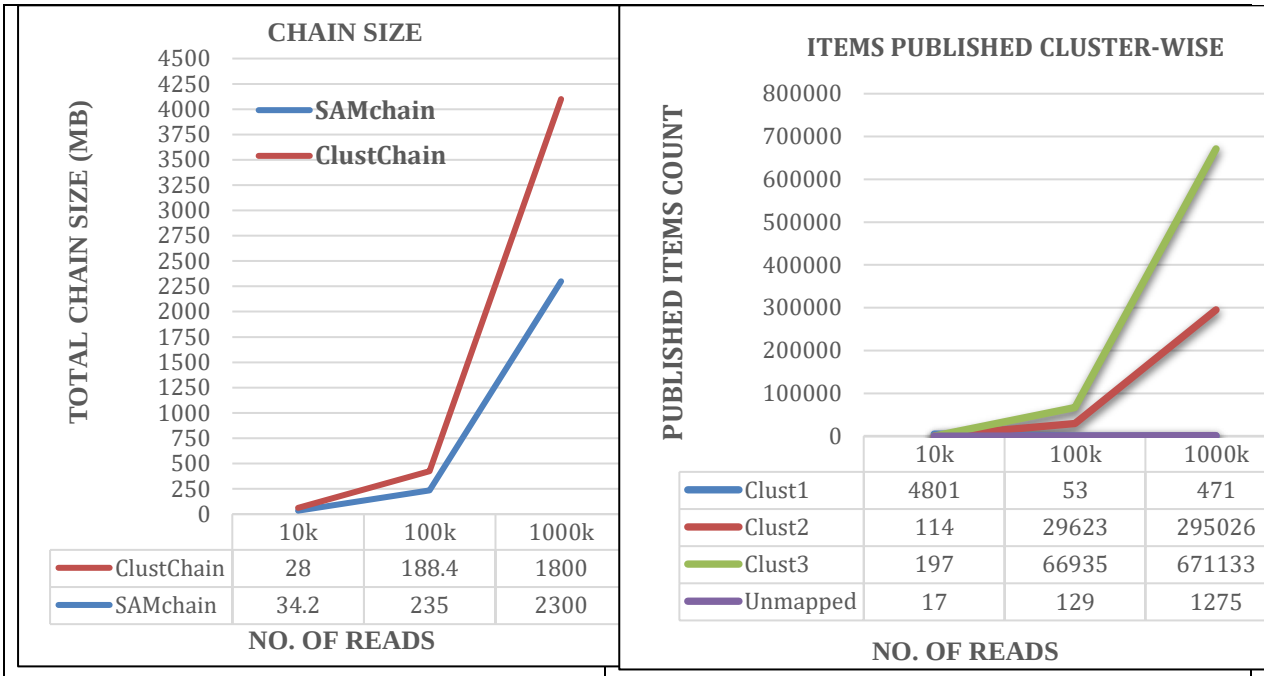


Fig-8: Impact on disk storage after insertion process

b) **Analyzing Disk Storage after Insertion Process**

The impact on disk storage for SAMchain and ClustChain is analyzed in terms of total chain size. Figure 8a indicates a lower disk storage requirement with increasing file size for ClustChain as compared to SAMchain. Table 3 shows the reduction in storage requirement by ClustChain against SAMchain. The clustering process groups genomic reads at 3 different levels of similarity to avoid redundancy in genomic data stored. The size of these clusters for a dataset completely depends upon the genomic reads available in the dataset. The reduction in storage will be higher when majority of the reads are grouped in cluster 1 or cluster 2. On the other hand, if majority of the reads are assigned to cluster 3 or remains unmapped, then reduction in storage will be lower. Figure 8b shows a comprehensive view of the clusters size in terms of data items. It is clear from figure 6b that majority of the reads are assigned to cluster 3 for file sizes of 100k and 1000k. Consequently, a lower storage reduction is achieved than expected for the sample BAM file used in this experiment. Therefore, the effectiveness of the proposed storage optimization approach is strongly influenced by the distribution of reads across different cluster types, impacting storage reduction according to the number of reads assigned to a cluster type.

Table-3: Reduction in Storage Requirement

Read Count	SAMchain	ClustChain	% Reduction in Storage Requirement
10k	34.2 MB	28 MB	18
100k	235 MB	188.4 MB	20
1000k	2300 MB	1800 MB	22

c) **Analyzing Wallet Size after Insertion**

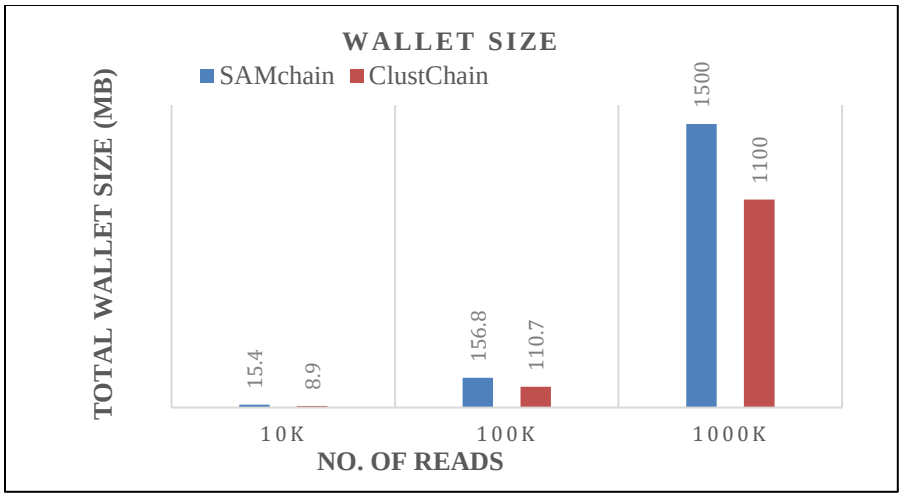


Fig-9: Analysis of Wallet Size after Insertion

Figure 9 shows the impact on wallet size for different number of reads in a file. Typically, a wallet contains keys, transaction records, transactional metadata, UTXOs, indexes, and references to other components. It can be observed that the wallet size of ClustChain is lower than SAMchain. The clustering mechanism utilized by the proposed approach reduces the number of API calls and transactions to record genomic data on blockchain. This leads to reduced transaction metadata in the wallet, resulting in lower transactional overhead and better scalability. Table 4 summarizes the performance analysis of the proposed approach against SAM for insertion process.

Table-4: Summary of Performance Analysis for Insertion Process

Metric	SAMchain	ClustChain
Stream items	Very high	Lower
Transactions	Very high	Lower
Wallet records	Higher	Lower
Chain Size	Higher	Lower
Memory Usage	Lower	Higher
Blockchain growth	Higher	Lower

6.2 Analysing Query Performance

This section discusses the impact of the proposed storage method on query performance in comparison to SAMchain. To analyse the query performance, different genomic locations (single or range) have been included in query covering varying size of result set. The query performance is studied from two perspectives namely, stream load time and query time.

a) **Impact on Stream Loading**

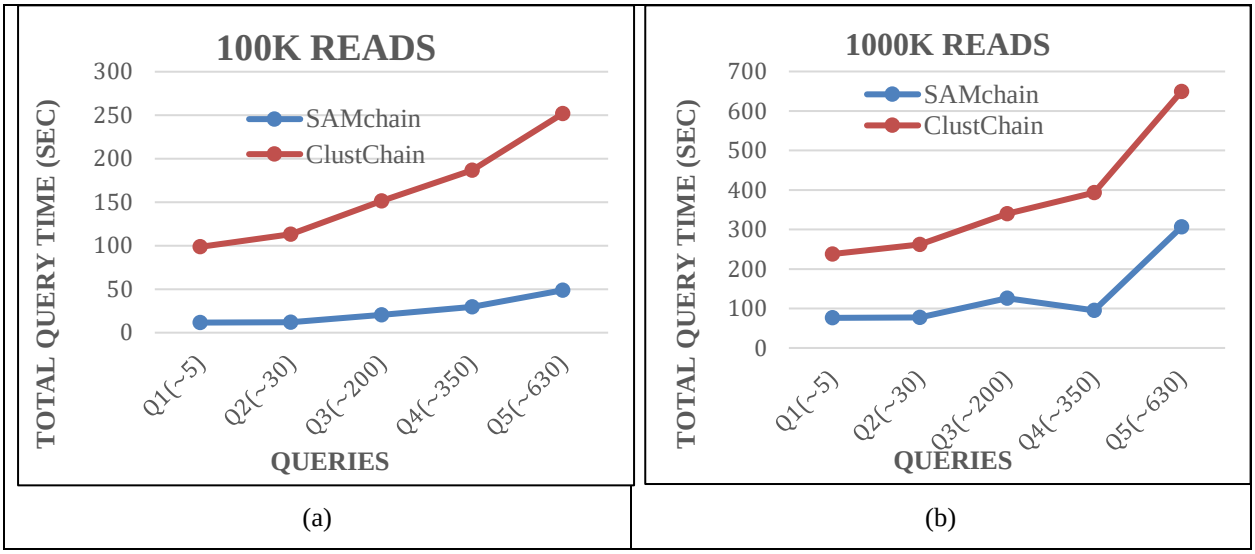
The stream size plays a major role in query processing time since it directly impacts the first step of query processing that is, stream loading. Stream loading refers to fetching data from identified streams as per queried genomic location. Every stream represents a range of genomic locations mapped. When a genomic location is queried, the related stream is completely loaded in memory. If other adjacent streams, basically previous streams as per ordering of genomic location range mapped to them, also covers the queried location in their sequencing field (SEQ), then those streams can be loaded partially that is, load only those reads that can cover the queried locations. So, the larger is stream size, the more time it will take to load stream data which ultimately increases total query time. The impact of stream size, representing number of read stored, on the stream load time can be observed in table 5. From

the observations in table 5, it is clear that increasing stream size (or read count) increases stream load time for SAMchain and decreases stream load time for ClustChain because of reduced size of data (clustered data) stored on ClustChain.

Table-5: Summary of stream load time and loaded items size for different stream size.

Read Count/ stream size	Stream Load Time		Size of Loaded Items	
	SAMchain	**GenoChain**	**SAMchain**	**GenoChain**
10k	0.982388	0.581613	87616	43032
100k	10.00838	14.2151	824512	824456
1000k	71.97773	53.04575	3041880	3012896

b) **Impact on Query Time**



(a) (b)

Fig-10: Impact on total query time for different stream size (read count)

This section analyses the impact of proposed method of clustering BAM reads on query performance. To produce same result as obtained by querying SAMchain which has stored each BAM reads as a single transaction and without any compression and clustering of multiple reads, the query method used for querying ClustChain need to de-cluster or decompress fields (eg: COMQUAL or MODSEQ) wherever required as per the fetched data from streams. Figure 10 depicts the increase in total query time for different stream size or read counts in case of ClustChain as compared to SAMchain. The main reason behind increased query time is primarily due to the de-clustering process for extracting original single reads from the cluster and decompression of few fields (SEQ, QUAL). It has been observed that as the result set size increases, shown along with query name, for instance Q1(~5)), query time also increases. The query method used in the proposed method follows the similar steps as followed in SAMchain [22] producing complete genomic reads, but clustering and compression techniques used in proposed approaches increases query time. So, there is a need to design an effective query method which can retrieve desired results in less time or atleast similar to query time of SAMchain for specific circumstances.

Also, from the above observation, it can be inferred that both stream size and result set size impacts query time. Generally, the requirement of whether stream is completely or partially loaded depends upon the queried location, the larger stream size increases both stream load time and search space which in turn increases query time. The result set size can also increase query time depending upon query type (single or range). However, there is an additional benefit

of allowing multi-attribute queries in the query method implemented in the proposed approach because resultant reads are stored in a dataframe.

```
read_df:                                        QNAME  ...                                                TAG
0    [A00297:35:HFKWWDSXX:4:1646:27534:28322]  ...  [[('MC', '150M'), ('MQ', 60), ('AS', 140), ('X...
1     [A00404:40:HFGYNDSXX:2:1636:9200:36620]  ...  [[('MC', '150M'), ('MQ', 60), ('AS', 140), ('X...
2     [A00404:40:HFGYNDSXX:1:1144:8847:34976]  ...  [[('MC', '150M'), ('MQ', 60), ('AS', 140), ('X...
3    [A00404:41:HFH25DSXX:2:2669:15863:21167]  ...  [[('MC', '150M'), ('MQ', 60), ('AS', 135), ('X...
4    [A00404:40:HFGYNDSXX:3:2129:11162:24173]  ...  [[('MC', '150M'), ('MQ', 40), ('AS', 140), ('X...
..                                        ...  ...                                                ...
599  [A00297:35:HFKWWDSXX:4:1145:15519:25332]  ...  [[('MC', '150M'), ('MQ', 60), ('AS', 145), ('X...
600   [A00404:40:HFGYNDSXX:1:1251:13476:5165]  ...  [[('MC', '150M'), ('MQ', 60), ('AS', 145), ('X...
601   [A00404:40:HFGYNDSXX:1:1251:13476:5165]  ...  [[('MC', '150M'), ('MQ', 60), ('AS', 145), ('X...
602  [A00404:41:HFH25DSXX:3:1242:10800:18881]  ...  [[('MC', '150M'), ('MQ', 60), ('AS', 145), ('X...
603  [A00404:41:HFH25DSXX:3:1242:10800:18881]  ...  [[('MC', '150M'), ('MQ', 60), ('AS', 145), ('X...
```

Fig-11: Sample output for a query

8. Conclusion

The proposed method reveals a significant improvement in storage requirement as the BAM file size increases. The findings of the paper reveal a significant improvement in storage requirement as the BAM file size increases. But it leads to an increase in data access time due to processing overhead caused by clustering method. In summary, there is a trade-off between time and memory requirement for managing genomic data on blockchain. As observed in the implemented blockchain based clustering approach, reducing disk storage can impact both memory and query time for processing BAM data on blockchain. But still the proposed approach is effective for memory-efficient applications. The increased query time due to de-clustering process, provides a future scope to optimize data query speed according to the queried genomic locations while taking advantage of both reduced storage requirement and stream load time. Further, for queries spanning multiple data streams, parallel processing methodology can be utilized to reduce total query time.

Statements and Declarations

Funding

The authors have not received any kind of funding for the work reported in their manuscript.

Authors' contribution statement

A. Arya: Conceptualization, Investigation, Methodology, Project administration, Writing – original draft.

A. Malik: Conceptualization, Project administration, Supervision, Validation, Writing – review & editing.

Conflict of Interest

The authors state that they have no known conflict of interests that would have affected the quality of the work done in this study.

Data Availability Statement

The BAM file used in this manuscript is available at http://files.gersteinlab.org/public-docs/2022/02.23/HG00114.loci.bam.

REFERENCES

1. L. Chouchane, R. Mamtani, A. Dallol, and J. I. Sheikh, "Personalized medicine: A patient - centered paradigm," 2011. doi: 10.1186/1479-5876-9-206.
2. Langreth and Waldholz, "New era of personalized medicine: targeting drugs for each unique genetic profile," Oncologist, 1999, doi: 10.1634/theoncologist.4-5-426.
3. M. E. Klein, M. M. Parvez, and J. G. Shin, "Clinical Implementation of Pharmacogenomics for Personalized Precision Medicine: Barriers and Solutions," J. Pharm. Sci., vol. 106, no. 9, pp. 2368–2379, 2017, doi: 10.1016/j.xphs.2017.04.051.

4. V. Merlo, G. Pio, F. Giusto, and M. Bilancia, “On the exploitation of the blockchain technology in the healthcare sector: A systematic review,” Expert Syst. Appl., p. 118897, 2022.
5. M. Hernaez, D. Pavlichin, T. Weissman, and I. Ochoa, “Genomic Data Compression,” 2019. doi: 10.1146/annurev-biodatasci-072018-021229.
6. Z. Huang et al., “A privacy-preserving solution for compressed storage and selective retrieval of genomic data,” Genome Res., 2016, doi: 10.1101/gr.206870.116.
7. A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman, “MedRec: Using blockchain for medical data access and permission management,” in Proceedings - 2016 2nd International Conference on Open and Big Data, OBD 2016, 2016. doi: 10.1109/OBD.2016.11.
8. K. Fan, S. Wang, Y. Ren, H. Li, and Y. Yang, “MedBlock: Efficient and Secure Medical Data Sharing Via Blockchain,” J. Med. Syst., 2018, doi: 10.1007/s10916-018-0993-7.
9. S. Tanwar, K. Parekh, and R. Evans, “Blockchain-based electronic healthcare record system for healthcare 4.0 applications,” J. Inf. Secur. Appl., 2020, doi: 10.1016/j.jisa.2019.102407.
10. G. Yang and C. Li, “A design of blockchain-based architecture for the security of electronic health record (EHR) systems,” in Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom, 2018. doi: 10.1109/CloudCom2018.2018.00058.
11. S. Kushch, S. Ranise, and G. Sciarretta, “Blockchain Tree for eHealth,” in 2019 IEEE Global Conference on Internet of Things, GCIoT 2019, 2019. doi: 10.1109/GCIoT47977.2019.9058412.
12. Z. Sun, D. Han, D. Li, X. Wang, C. C. Chang, and Z. Wu, “A blockchain-based secure storage scheme for medical information,” Eurasip J. Wirel. Commun. Netw., 2022, doi: 10.1186/s13638-022-02122-6.
13. K. Miyachi and T. K. Mackey, “hOCBS: A privacy-preserving blockchain framework for healthcare data leveraging an on-chain and off-chain system design,” Inf. Process. Manag., 2021, doi: 10.1016/j.ipm.2021.102535.
14. S. Ma, Y. Cao, and L. Xiong, “Efficient logging and querying for blockchain-based cross-site genomic dataset access audit,” BMC Med. Genomics, 2020, doi: 10.1186/s12920-020-0725-y.
15. N. D. Pattengale and C. M. Hudson, “Decentralized genomics audit logging via permissioned blockchain ledgering,” BMC Med. Genomics, 2020, doi: 10.1186/s12920-020-0720-3.
16. G. Gürsoy, R. Bjornson, M. E. Green, and M. Gerstein, “Using blockchain to log genome dataset access: efficient storage and query,” BMC Med. Genomics, 2020, doi: 10.1186/s12920-020-0716-z.
17. M. S. Ozdayi, M. Kantarcioglu, and B. Malin, “Leveraging blockchain for immutable logging and querying across multiple sites,” BMC Med. Genomics, 2020, doi: 10.1186/s12920-020-0721-2.
18. A. Chernomoretz et al., “GENis, an open-source multi-tier forensic DNA information system,” Forensic Sci. Int. Reports, 2020, doi: 10.1016/j.fsir.2020.100132.
19. G. Gürsoy, C. M. Brannon, and M. Gerstein, “Using Ethereum blockchain to store and query pharmacogenomics data via smart contracts,” BMC Med. Genomics, 2020, doi: 10.1186/s12920-020-00732-x.
20. T. T. Kuo et al., “Benchmarking blockchain-based gene-drug interaction data sharing methods: A case study from the iDASH 2019 secure genome analysis competition blockchain track,” Int. J. Med. Inform., 2021, doi: 10.1016/j.ijmedinf.2021.104559.
21. S. J. Lee, G. Y. Cho, F. Ikeno, and T. R. Lee, “BAQALC: Blockchain applied lossless efficient transmission of DNA sequencing data for next generation medical informatics,” Appl. Sci., 2018, doi: 10.3390/app8091471.
22. G. Gürsoy, C. M. Brannon, E. Ni, S. Wagner, A. Khanna, and M. Gerstein, “Storing and analyzing a genome on a blockchain,” Genome Biol., vol. 23, no. 1, p. 134, 2022.
23. E. Petraglio, R. Wertenbroek, F. Capitao, N. Guex, C. Iseli, and Y. Thoma, “Genomic data clustering on FPGAS for compression,” in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2017. doi: 10.1007/978-3-319-56258-2_20.
24. V. Di Gesú, R. Giancarlo, G. Lo Bosco, A. Raimondi, and D. Scaturro, “GenClust: A genetic algorithm for clustering gene expression data,” BMC Bioinformatics, vol. 6, no. 1, p. 289, 2005.
25. A. Ciaramella et al., “Interactive data analysis and clustering of genomic data,” Neural Networks, vol. 21, no. 2–3, pp. 368–378, 2008.
26. R. Li et al., “Gclust: A parallel clustering tool for microbial genomic data,” Genomics. Proteomics Bioinformatics, vol. 17, no. 5, pp. 496–502, 2019.
27. Y. Liu, X. Shen, Y. Gong, Y. Liu, B. Song, and X. Zeng, “Sequence Alignment/Map format: A comprehensive review of approaches and applications,” Brief. Bioinform., vol. 24, no. 5, 2023, doi: 10.1093/bib/bbad320.
28. The SAM/BAM Format Specification Working Group, “SAM/BAMv1: Sequence Alignment/Map Format Specification,” SAM/BAM Format Specif. Work. Gr., 2015.
29. J. Gailly and M. Adler, “Zlib compression library,” 2004.