Storing and analyzing a genome on a blockchain

Gamze Gursoy[1,2], Charlotte Brannon[1,2, *], Sarah Wagner[3, *], and Mark Gerstein[1,2,3]

[1]Computational Biology and Bioinformatics Program, Yale University
[2]Molecular Biophysics and Biochemistry Department, Yale University
[3]Department of Computer Science, Yale University
[*]These authors contributed equally

ABSTRACT

The genomic characterization of individuals promises to be immensely useful for medical research. Moreover, sequencing, analysis, and interpretation of patients' genomes is projected to be a staple of healthcare in the future. A critical barrier to expanding personal genome sequencing is the ability to store genomic data securely and with high integrity. While cloud storage offers solutions to access such data from any place and device, the security, data integrity, and robustness vulnerabilities such as single-point-of failure losses have not yet been addressed. Here, we developed novel tools for decentralized storage, access, and analysis of genome sequencing data on private blockchain networks. Storing and analyzing large-scale data on a blockchain can be challenging because of the slow transaction speed and limitations on querying data stored on-chain. Hence, current genomic blockchain applications only log links to the data. We overcome this challenge by implementing data compression techniques and nested database indexing. Our tools provide open-source blockchain-based storage and access tools for advanced genomic analyses such as variant calling.

Keywords: blockchain, multichain, personal genome, blockchain database

MAIN TEXT

Modern advances in personalized medicine have resulted in an increasing number of individuals willing to sequence their own genome for disease-risk predictions and ancestry analysis, which has brought us closer to an era of genomic data-driven health care and biomedical research (1). Furthermore, understanding the human genomic landscape of millions of diverse individuals is essential for characterizing and investigating rare diseases and genotype-phenotype associations. Given the widespread interest in understanding one's own genomic data, and the promise of these data for advancing biomedical research, it is almost inevitable that genome sequencing will become part of routine clinical care in the future and that the number of sequenced human genomes will continue to grow (2).

Growth of personal genomic data has been limited by bottlenecks in computational requirements and server capacity (3). The NIH and several other institutions are moving toward cloud-computing-based services in order to overcome these bottlenecks (4). However, cloud-based storage and data analysis tools present security concerns, as they are based on a centralized architecture and are therefore vulnerable to single-point-of-failure losses (5; 6). These are critical problems; as genomic data becomes increasingly integral to our understanding of human health and disease, its integrity and security must be a priority when providing solutions to storage and analysis. Corruption, change, or loss of personal genomes could create problems in patient care and research integrity in the future.

An ideal implementation of genomic data storage and access would protect from both loss and manipulation. Blockchain technology could be an ideal solution due to three key properties: decentralization, immutability, and security (5). Decentralization prevents a single entity from controlling the data; immutability guarantees that data cannot be altered; and security is ensured by protecting accounts with enhanced cryptographic methods (5). Already today, there are multiple personalized medicine start-ups that aim to store individuals' genomes in blockchains.

A recent review by Ozercan et al (5) outlines the current status of commercial and academic proposals to share genomic data using blockchain platforms. Among these newly emerged platforms are gene-chain and Zenome, which provide solutions for genomic data distribution for commercial use. Another platform, Nebula genomics, creates tokens as an incentive for individuals to share their genomes with entities using blockchain technology. CrypDist enables synchronized data-sharing among researchers with protection against accidental or intentional data removal. This technology allows for sharing large genomic data files, such as reference aligned genomes (BAM files). However, due to the inefficiency of inserting large datasets to a blockchain, they instead share links to the data rather than the data itself. Additionally, although all of these platforms permit secure storage and sharing of genomic data, none of them offers a solution to perform computation on the data stored in the blockchain. This is a critical gap in the technology; not only do physicians and scientists need secure access to raw genomic data, but they also require secure tools for analyzing the data.

A central caveat to blockchain technology is the inefficiency of storing and querying data due to the potential for chains to reach large sizes. The storage space and computational power required by blockchain is greater than a centralized database application due to the redundancy of storage and network verification protocols. The decentralized system also creates a higher latency (delay in data communication) during storage and retrieval of data. Additionally, transactions in the blockchain network require a cryptographic consensus verification, which makes them slow to publish data to the chain (7).

To overcome these challenges, we developed novel data structures based on nested database indexing, file-format modifications and compression techniques with the open-source blockchain API MultiChain (8). MultiChain is a platform specifically designed for building and deploying private blockchain applications. Moreover, it is well-suited for database applications; it has a 'data stream' feature, which allows users to create multiple key-value, time-series, or identity databases that can be used for data-sharing, time-stamping, and encrypted archiving (8) (see Supplemental Information).

Raw genomics data from sequencers are often stored as compressed binary file types called binary alignment maps (BAM) and/or compressive alignment map s(CRAM) that are derived from sequence alignment map (SAM) files in text format (9). These are the genomics data standard as determined by GA4GH (10). In this paper, we focused on efficient storage and analysis of such files using blockchain technology. We introduce the first open-source, proof-of-concept application for storing and analyzing BAM files in a Blockchain. We provide two tools: the first is SAMChain, which allows users to store sequence alignment maps in a blockchain efficiently, and the second is SCtools, which provides functions such as (a) querying, (c) depth analysis, (d) pile-ups for variant calling, and (e) re-creating BAM files.

MultiChain data streams make it possible for a blockchain to be used as a general purpose database. The data published in every stream is stored by all nodes in the network. Each data stream on a MultiChain blockchain consists of a list of items. Each item in the stream contains the following information, as a JSON object (8): A publisher (string), key:value pairs (between 1-256 ASCII character, excluding whitespace and single/double quotes) (string), data (hex string), a transaction ID (string), blocktime (integer) and Confirmations (integer). When data needs to be queried or streamed, it can be retrieved by searches using the key:value pairs. Publishing a stream item to a data stream constitutes a transaction. When a transaction happens, it is held in the memory pool. After mining of the transaction is complete, the transaction is added to a block. Each block has a maximum transaction size, i.e after a block reaches its maximum size or the time to create a block reaches its limit, the block is sealed and appended to the chain. This means a data stream in MultiChain can span multiple blocks based on the time of the transaction (i.e time of the publishing the data to the blockchain).

Implementation: Our implementation can be found at https://github.com/gersteinlab/SAMChain. We took an approach to maximize the efficiency of storing and querying data. Our goal was to store minimal data while indexing it in a creative way to allow rapid retrieval, thereby reducing the time and memory cost of analysis and increasing the utility of the stored data (Figure 1a). To achieve this goal, we manipulated a) data structures in data streams, and b) data to be stored in BAM files. For (a), we first separated mapped and unmapped reads from a BAM file. First, we created a data stream called AllReadData, in which the key values are the read names and the data values are the modified versions of the BAM features. We then created N number of streams (called chr{i}bin{j}). Each of these N streams represents a bin of genomic coordinates. Based on the location of a read mapped on the human reference genome, we log the read names as keys in chr{i}bin{j} stream. Some reads will span multiple bins. In that case, we store the read in the bin that the beginning of the read maps to and the bin that the end of the read maps to. We then add a boolean key to the chr{i}bin{j} stream that we call FLANK. A FLANK value of 0 indicates that the entire read is in that bin. A FLANK value of 1 indicates that the read is stored in two consecutive streams. The FLANK value tells our retrieval algorithm to search for the particular read in multiple chr{i}bin{j} streams. Our query algorithm can retrieve the keys in the chr{i}bin{j} stream based on the queried location, and use these keys to access the AllReadData stream and retrieve the rest of the features corresponding to that particular read. Our code base allows developers to bin the data according to a desired feature that might be queried by users such as read names, mapping qualities, or alignment scores; these are the features stored as keys in the binned streams. Our implementation uses binning by genomic location, as it is the most commonly queried property for depth analysis or variant calling. Unmapped reads are stored in AllReadData but not in the chr{i}bin{j} streams, and can be queried directly from the AllReadData stream. Binning is done for the purpose of efficient access to keys during analysis. For (b), we were inspired by the data compression techniques in CRAM files, and stored the difference between the read and the reference sequence in the chain instead of the sequences themselves to reduce the size of the data stored on-chain (SI Figure 6). With this approach, our implementation is able to regenerate the sequence of a read by using the reference genome and other features stored in the chain.

We developed SCtools to extract information from SAMchain for downstream analysis. We provide a code base that has the ability to query on a blockchain. The key:value property of the data streams in MultiChain2.0 together with the ability to query on multiple keys provides an opportunity to extract data from the blockchain without the need for costly calculations. Our query module can retrieve data from a chain based on the position in the reference genome. Given our query algorithm, it would be simple to build upon SCtools and add querying by the other keys stored in the binned streams, which we call indexable keys (Figure 1c). Moreover, any user could do this; it would only require writing a python script (no changes would be needed to the underlying blockchain data structure).

If a user queries the chain for reads mapped to a genomic region, our query module first finds the correct streams/bins containing that region. From the bins, it extracts the read name(s) that correspond to the correct position(s), and uses the read name(s) to key into the allReadData stream and return the data. This approach reduces the query time significantly due to the following reasons: (a) Data streams do not allow range searches. If the data were kept in a single stream, then the query would have to iterate over the location range for every single stream item. With binned streams, the query is only done on the streams containing the relevant data. (b) Genomic location is not unique. There are multiple reads that map to the same location. However, the read names are always unique for all mapped reads. Therefore, query uses the binned streams to retrieve the read names and then quickly searches the unique read names in the allReadata stream.

Here we summarize the key functionalities of SCtools. Detailed explanations and flowcharts can be found in the Supplemental Information.

Depth Analysis: This module can be used to query sequencing depth for a given position or a range of positions in the genome. For example, when a user specifies a location in the genome to query sequencing depth, this module first finds the stream containing the reads from that genomic location. It then checks whether the location flanks any other streams (SI Figure 1). After finding the relevant streams, it lists all the stream items and searches for the reads that span the queried locations. After checking each read's CIGAR for clippings or deletions in the queried location, it returns the resulting number of reads as the depth of the queried location (See SI Figure 4 for the flowchart).

Pile-ups for variant calling: SAMtools provides a useful function to determine the pile-ups for a queried location or all of the locations in the genome. Pile-up files contain the number of reads that mapped to a location, the reference allele for that location, and the sequenced nucleotide in each read for that location. This allows users to visualize the genetic variation and calculate allele frequencies for the variants. We developed this functionality to create pile-up files from the blockchain (See SI Figure 4 for the flowchart).

Re-construction of alignment maps: This functionality is for the users to have a copy of an alignment map. This module combines the data from allReadData with unaligned reads and the header tore-construct an alignment file in BAM format, which can be further manipulated using SAMtools (See SI Figure 5 for the flowchart).

The accuracy of our SCtools modules are demonstrated with an example in Figure 2c. We also calculated the time requirement of read streaming, depth analysis and pile-ups empirically. Figure 2d shows how the performance of the querying changes with the increasing amount of the reads.

We envision a real-world scenario in which individuals create private blockchains to store their personal genomes to share with their healthcare providers. Simply with ssh access, healthcare providers and associated genetics researchers can stream or query patients' genomes without needing to download or transfer data. This reduces not only the risk of data corruption, but also non-permissioned access to private data by adversaries. Blockchain provides immutability such that the data cannot be altered, whether intentionally or accidentally.

Our framework is the first open-source application to allow querying and streaming of genomic data from blockchain to the best of our knowledge. This is a substantial improvement over the current biomedical applications of blockchains. With previous implementations, the security of the data is provided by blockchain but the computation on the genomic data is done on plain text. To address privacy concerns, our framework may be extended to store homomorphically encrypted data in the data streams. However, this will add storage and computation overhead to the solution.

While the main benefit of using blockchain for data storage is data security and integrity, blockchain also makes it easy to append data to large data files. For example, in the cases of

BAM files, if a user wishes to add to the data, one could create data streams for these additions (since it is a private chain, only the owner of the chain would have permission to make these changes). Thus, the data owner does not have to deal with opening large-data files, modifying them, and re-indexing them, which creates costly network traffic. Searches by genomic location could also check the new data streams, to determine if the owner has appended any changes to the data.

Our blockchain solution can be generalized to other large-scale data storage and querying problems beyond BAM files. Data including but not limited to electronic health records, vcf files from multiple or single individuals, and somatic mutation datasets from cancer patients can be stored in blockchain using our indexing schemes, allowing for rapid and partial retrieval of the data.

Availability: SAMChain and SCTools can be found at https://github.com/gersteinlab/SAMChain

REFERENCES

[1] Mackey, Tim K and Kuo, Tsung-Ting and Gummadi, Basker and Clauson, Kevin A and Church, George and Grishin, Dennis and Obbad, Kamal and Barkovich, Robert and Palombini, Maria. 'Fit-for-purpose?' - challenges and opportunities for applications of blockchain technology in the future of healthcare. *BMC Med.*, 2019;17(1):68.

[2] Khan, Razib and Mittelman, David. Consumer genomics will change your life, whether you get tested or not. *Genome Biol.*, 2018;19(1):120.

[3] O'Driscoll, Aisling and Daugelaite, Jurate and Sleator, Roy D. 'Big data', Hadoop and cloud computing in genomics. *J. Biomed. Inform.*, 2013;46(5):774–781.

[4] governmentCIO: Transforming government IT. NIH prioritizes cloud migration 2020 it ecosystem plan. https://governmentciomedia.com/nih-prioritizes-cloud-migration-2020-it-ecosystem-plan

[5] Ozercan, Halil Ibrahim and Ileri, Atalay Mert and Ayday, Erman and Alkan, Can. Realizing the potential of blockchain technologies in genomics. *Genome Res.*, 2018;28(9):1255–1263.

[6] Kuo, Tsung-Ting and Kim, Hyeon-Eui and Ohno-Machado, Lucila. Blockchain distributed ledger technologies for biomedical and health care applications. *J. Am. Med. Inform. Assoc.*, 2017;24(6):1211–1220.

[7] Nakamoto S. Bitcoin: A Peer-to-Peer Electronic Cash System. *bitcoin.org/bitcoin.pdf*, 2008.

[8] Greenspan, Gideon. MultiChain Private Blockchain - White Paper. *https://www.multichain.com/download/MultiChain-White-Paper.pdf*, 2015.

[9] Li H, Handsaker B, Wysoker A, Fennell T, Ruan J, Homer N, Marth G, Abecasis G, Durbin R. The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, 2009;25(16):2078-2079.

[10] GA4GH Global Alliance for Genomics and Health https://www.ga4gh.org/

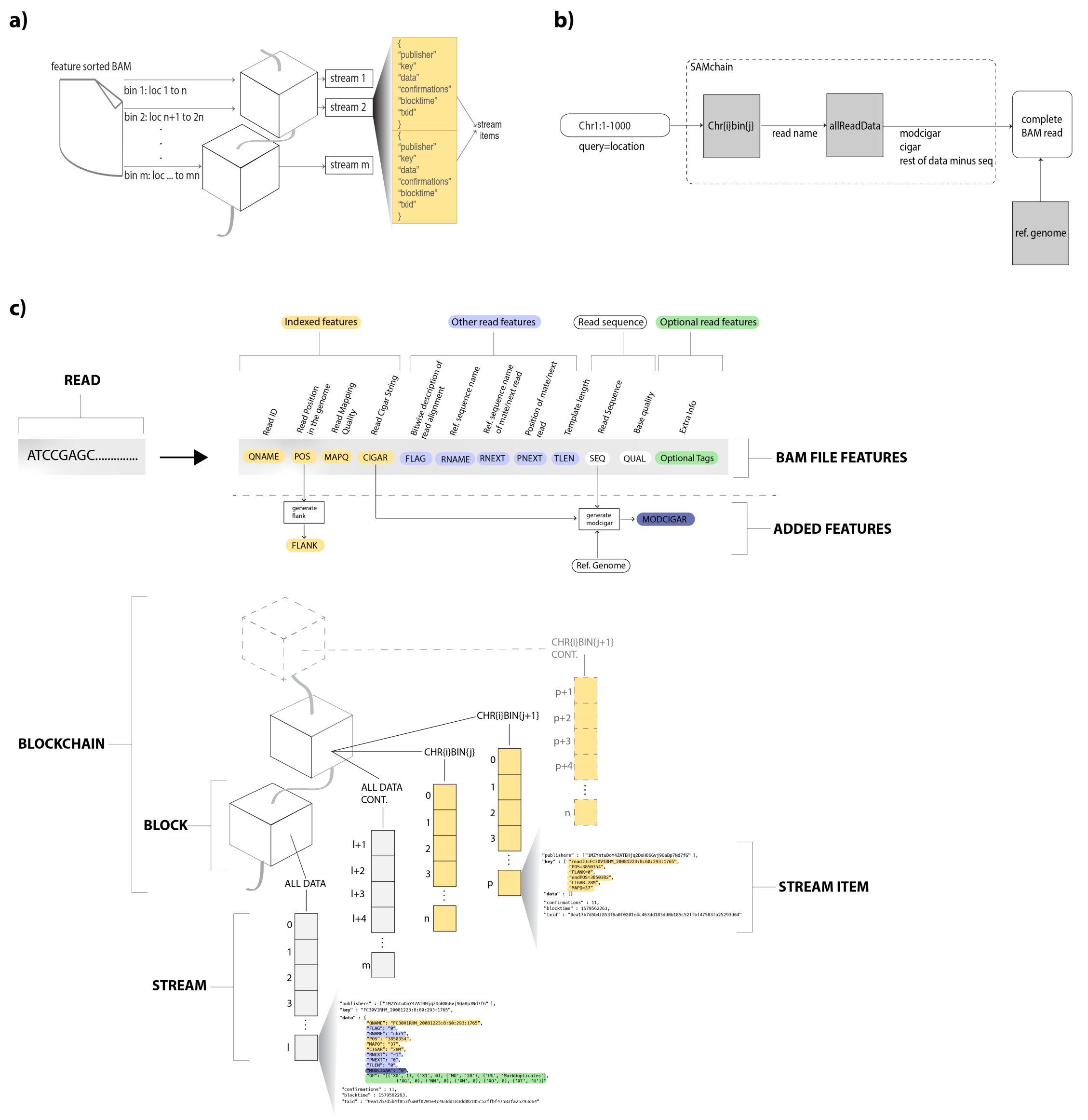
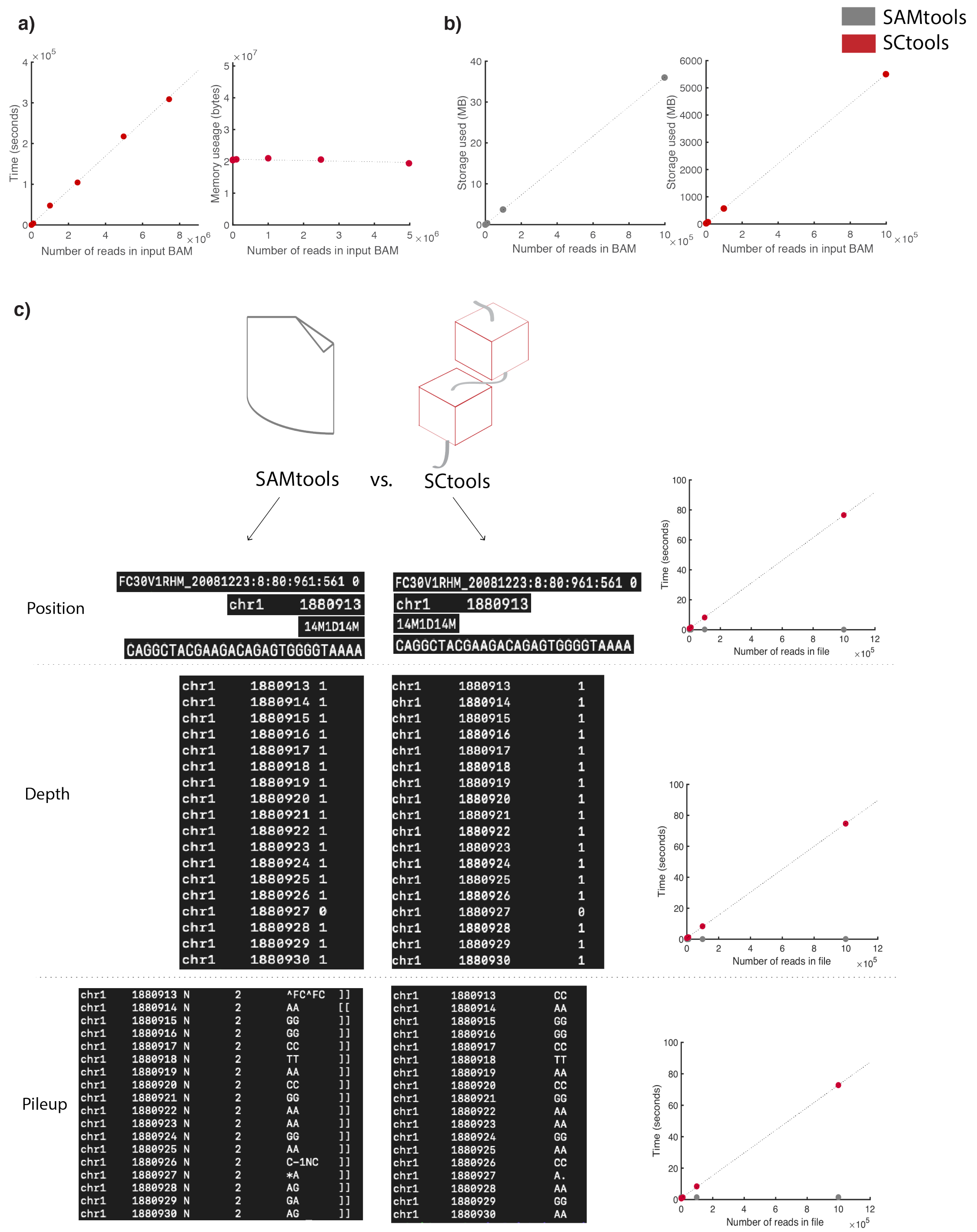


Figure 1. SAMchain design and implementation. (a) An overview of the SAMchain approach. Given a BAM file, we bin the read data into several "streams." MultiChain datastreams permit the use of a blockchain as a database. (b) Overview of the query process. Upon querying a genomic location, our algorithm searches through the binned streams to obtain the read names corresponding to the specified location. It then uses the read name to key into the allReadData stream and retrieve the data. This data, in combination with a reference genome, yields a complete BAM read. (c) Details of data storage in SAMchain. A read is typically stored in a BAM file containing several features. Our data structure makes use of five indexable features. A single stream, named allReadData, contains all of the data for each read in the input BAM, except for the sequence information. Many other streams serve as bins by genomic location, and hold only the indexable features. Stream items correspond to a single read.



```
FC30V1RHM_20081223:8:80:961:561 0
                    chr1    1880913
                           14M1D14M
CAGGCTACGAAGACAGAGTGGGGTAAAA
```

```
FC30V1RHM_20081223:8:80:961:561 0
chr1    1880913
14M1D14M
CAGGCTACGAAGACAGAGTGGGGTAAAA
```

```
chr1    1880913 1
chr1    1880914 1
chr1    1880915 1
chr1    1880916 1
chr1    1880917 1
chr1    1880918 1
chr1    1880919 1
chr1    1880920 1
chr1    1880921 1
chr1    1880922 1
chr1    1880923 1
chr1    1880924 1
chr1    1880925 1
chr1    1880926 1
chr1    1880927 0
chr1    1880928 1
chr1    1880929 1
chr1    1880930 1
```

```
chr1    1880913         1
chr1    1880914         1
chr1    1880915         1
chr1    1880916         1
chr1    1880917         1
chr1    1880918         1
chr1    1880919         1
chr1    1880920         1
chr1    1880921         1
chr1    1880922         1
chr1    1880923         1
chr1    1880924         1
chr1    1880925         1
chr1    1880926         1
chr1    1880927         0
chr1    1880928         1
chr1    1880929         1
chr1    1880930         1
```

```
chr1    1880913 N       2       ^FC^FC  ]]
chr1    1880914 N       2       AA      [[
chr1    1880915 N       2       GG      ]]
chr1    1880916 N       2       GG      ]]
chr1    1880917 N       2       CC      ]]
chr1    1880918 N       2       TT      ]]
chr1    1880919 N       2       AA      ]]
chr1    1880920 N       2       CC      ]]
chr1    1880921 N       2       GG      ]]
chr1    1880922 N       2       AA      ]]
chr1    1880923 N       2       AA      ]]
chr1    1880924 N       2       GG      ]]
chr1    1880925 N       2       AA      ]]
chr1    1880926 N       2       C-1NC   ]]
chr1    1880927 N       2       *A      ]]
chr1    1880928 N       2       AG      ]]
chr1    1880929 N       2       GA      ]]
chr1    1880930 N       2       AG      ]]
```

```
chr1    1880913         CC
chr1    1880914         AA
chr1    1880915         GG
chr1    1880916         GG
chr1    1880917         CC
chr1    1880918         TT
chr1    1880919         AA
chr1    1880920         CC
chr1    1880921         GG
chr1    1880922         AA
chr1    1880923         AA
chr1    1880924         GG
chr1    1880925         AA
chr1    1880926         CC
chr1    1880927         A.
chr1    1880928         AA
chr1    1880929         GG
chr1    1880930         AA
```

Figure 2. Performance and Accuracy of SAMchain and SCtools. (a) Time and memory required to build the chain with an increasing number of reads in the initial BAM file. (b) Storage usage of a BAM file (left) and SAMchain (right) with increasing number of reads stored. (c) Accuracy and speed evaluation of SCtools by comparison with SAMtools output for queries, depth analysis (one-read match) and pileup analysis (two-read match, to demonstrate multiple allele output) of a particular genomic region.