PLOS ONE

RESEARCH ARTICLE

SeqRepo: A system for managing local collections of biological sequences

Reece K. Hart[1]*, **Andreas Prlić**[2]

1 Biocommons, San Francisco, CA, United States of America, **2** Invitae, Inc., San Francisco, CA, United States of America

* reece@biocommons.org

OPEN ACCESS

Citation: Hart RK, Prlić A (2020) SeqRepo: A system for managing local collections of biological sequences. PLoS ONE 15(12): e0239883. https://doi.org/10.1371/journal.pone.0239883

Editor: Ruslan Kalendar, University of Helsinki, FINLAND

Received: September 13, 2020

Accepted: November 13, 2020

Published: December 3, 2020

Data Availability Statement: Source code, including code used to generate manuscript figures, is available at https://github.com/biocommons/biocommons.seqrepo.

Funding: Invitae, Inc. supported both authors in developing SeqRepo. The funder provided support in the form of salaries for authors [RKH, AP], but did not have any additional role in the study design, data collection and analysis, decision to publish, or preparation of the manuscript. The specific roles of these authors are articulated in the 'author contributions' section.

Abstract

Motivation

Access to biological sequence data, such as genome, transcript, or protein sequence, is at the core of many bioinformatics analysis workflows. The National Center for Biotechnology Information (NCBI), Ensembl, and other sequence database maintainers provide methods to access sequences through network connections. For many users, the convenience and currency of remotely managed data are compelling, and the network latency is non-consequential. However, for high-throughput and clinical applications, local sequence collections are essential for performance, stability, privacy, and reproducibility.

Results

Here we describe SeqRepo, a novel system for building a local, high-performance, non-redundant collection of biological sequences. SeqRepo enables clients to use primary database identifiers and several digests to identify sequences and sequence alises. SeqRepo provides a native Python interface and a REST interface, which can run locally and enables access from other programming languages. SeqRepo also provides an alternative REST interface based on the GA4GH refget protocol.

SeqRepo provides fast random access to sequence slices. We provide results that demonstrate that a local SeqRepo sequence collection yields significant performance benefits of up to 1300-fold over remote sequence collections. In our use case for a variant validation and normalization pipeline, SeqRepo improved throughput 50-fold relative to use with remote sequences. SeqRepo may be used with any species or sequence type. Regular snapshots of Human sequence collections are available.

It is often convenient or necessary to use a computed digest as a sequence identifier. For example, a digest-based identifier may be used to refer to proprietary reference genomes or segments of a graph genome, for which conventional identifiers will not be available. Here we also introduce a convention for the application of the SHA-512 hashing algorithm with Base64 encoding to generate URL-safe identifiers. This convention, *sha512t24u*, combines a fast digest mechanism with a space-efficient representation that can be used for any object. Our report includes an analysis of timing and collision probabilities for *sha512t24u*. SeqRepo enables clients to use sha512t24u as identifiers, thereby seamlessly integrating public and private sequence sets.

Competing interests: Invitae, Inc. supported both authors in developing SeqRepo. This does not alter our adherence to PLOS ONE policies on sharing data and materials.

Availability

SeqRepo is released under the Apache License 2.0 and is available on github and PyPi. Docker images and database snapshots are also available. See https://github.com/biocommons/biocommons.seqrepo.

Introduction

Many bioinformatics analysis pipelines require access to biological sequence data. One example is genetic variation data, which requires access to all sequences that are used as references in order to validate sequence bounds and to normalize variants [1,2].

A typical whole genome sequencing sample has between 3.5 and 12 million variants [3]. Variant analysis pipelines for data of such volume need fast, random access to an assortment of genome, transcript, and protein sequences. While network-accessible databases [4,5] are convenient, the latency is prohibitive for this high-throughput setting. Furthermore, dependencies on remote services create risks for privacy, reproducibility, and overall system availability. These were the problems for which we developed SeqRepo in 2016 as a component for the hgvs Python package [6]. Using SeqRepo increases validation and variant projection throughput by nearly 50-fold relative to remote sequence access. We are unaware of tools similar to SeqRepo that enable the management and efficient distribution of sequence collections.

Sequence databases are often highly redundant within and between data providers. Deduplication of sequence datasets may be efficiently achieved using a digest or hash algorithm, such as SEGUID for protein sequences [7]. Here, we propose an approach that uses a variation of the SHA-512 hashing algorithm [8] with Base64 encoding [9] to generate URL-safe identifiers. The sha512t24u convention is a conceptual successor to SEGUID, but is computed approximately twice as fast on 64-bit processors and may be used in URLs without encoding. (SEGUID uses characters that are reserved for URL delimiters.) The GA4GH Variation Representation Specification [10] and the GA4GH refget protocol [11] have adopted the sha512t24u convention to generate computed object identifiers.

SeqRepo, presented here, provides these features:

1. Deduplication and compression of sequences;
2. Fast random access to sequences and sub-sequences;
3. Space-efficient storage and transfers of snapshots;
4. Ability to use of conventional, primary database identifiers and digest-based identifiers;
5. Publicly accessible snapshots of Human sequences;
6. Privacy and availability benefits appropriate for use in clinical settings.

Implementation

The sha512t24u digest

We sought a digest and encoding that is based on existing standards, that may be implemented in prevalent programming languages, and may be readily used in URLs. Our method, dubbed *sha512t24u*, constructs a SHA-512 binary digest, truncated to 24 bytes (192 bits), and represented using "base64url" encoding. Fig 1 shows an implementation in Python; sha512t24u has also been implemented in Go, Java, javascript, and Perl.

```
>>> import base64, hashlib
>>> data = b""
>>> sha512_digest = hashlib.sha512(data).digest()
>>> sha512t24u = base64.urlsafe_b64encode(sha512_digest[:24]).decode("ascii")
>>> sha512t24u
'z4PhNX7vuL3xVChQ1m2AB9Yg5AULVxXc'
```

Fig 1. The sha512t24u digest in Python.

https://doi.org/10.1371/journal.pone.0239883.g001

SeqRepo library

SeqRepo consists of two components: a module, fastadir, that stores sequences non-redundantly using an internal key, and a second module, seqaliasdb, that stores identifiers associated with the internal key. SeqRepo is written in Python 3.

Sequence module: Fastadir. Sequences are stored in FASTA-formatted files and index using Blocked GZipped Format (BGZF) [12]. Fast random access to BGZF files is provided by the PySAM library [13].

When loading a sequence into SeqRepo, an internal *sequence identifier* is generated based on the *sha512t24u* convention. If the sequence does not already exist, it is written in a FASTA file with a timestamped filename using the internal sequence identifier as the FASTA identifier. A sequence manifest database, implemented in SQLite, stores sequence length, alphabet, and the path to the BGZF file that contains the sequence. Sequences in SeqRepo are immutable and not deletable; therefore, sequence files never change and exist in all future SeqRepo snapshots. New releases require only the space for new sequences and manifest db.

Alias module: Seqaliasdb. Once a new or existing sequence is confirmed in the sequence database, SeqRepo loads identifiers for the sequence as provided by the client. Identifiers are stored in a second SQLite database as <*namespace*, *alias*> pairs and are associated with the internal sequence identifier.

Namespaces from identifiers.org are used when available. Aliases are required to be unique within a namespace. If an <*namespace*, *alias*> pair already exists, SeqRepo verifies that the associated sequence identifier matches the sequence being loaded; if it does not, the older association is deprecated in the database and a new association is made. Identifier associations are timestamped, making it possible to see the naming history of any sequences.

For new sequences, SeqRepo generates a set of computed identifiers using computational digests and loads these as with conventional identifiers. Currently, these digests are sha512t24u, MD5, SEGUID, and SHA-1.

Snapshots

Snapshots are created using hard links in the file system. Therefore, new snapshots consume only the space required for the incremental BGZF and database files. Similarly, transfers with rsync transfer only the incremental changes.

Interfaces

SeqRepo includes two interfaces: the native Python interface, and a read-only REST-based interface to be used for local access by programs written in other languages. The SeqRepo package also implements the GA4GH refget protocol to support clients that require that interface. The refget implementation passes the compliance suite provided by the authors. Table 1 summarizes operations supported by these three interfaces.

Table 1. Summary of operations provided by the Python interface, REST interface, and refget protocol interface.

	Python interface	SeqRepo REST interface	refget protocol
get service information	N/A	/v1/ping	/sequence/service-info
put sequence	store(seq, identifiers[])	N/A	N/A
get sequence	fetch(ir, [start], [end]) sr[ir][start:end]	/v1/sequence/:ir ("start" and "end" query parameters optional)	/sequence/:digest ("start" and "end" query parameters optional)
put aliases	store(seq, identifiers[])	N/A	N/A
get metadata	translate_identifier(ir)	/v1/metadata/:ir	/sequence/:digest/metadata

All interfaces support rapid access to slices of chromosome-sized sequences. The SeqRepo provides two mechanisms to fetch sequence slices: A fetch() method, and a dict-style access that permits a SeqRepo instance to be accessed as a Python dictionary. The SeqRepo REST interface and refget protocol are read-only interfaces. "ir" denotes an identifier of the form *namespace*:*alias*; an alias may be used without namespace if it is globally unique. The refget protocol itself currently requires the use of digests for queries.

https://doi.org/10.1371/journal.pone.0239883.t001

SeqRepo interfaces represent identifiers as W3C Compact URIs (CURIEs), mapping <*namespace*,*alias*> to the <*prefix*, *reference*> nomenclature; the converse operation is also supported when CURIE-formatted identifiers are provided as sequence identifiers.

Installation scenarios

SeqRepo supports four installation scenarios:

1. A read-write instance, loaded and maintained locally with the seqrepo command line interface.
2. A read-only instance of the human collection, mirrored and updated using the seqrepo command line tool.
3. A docker data-only container for linking with other application containers. This approach is useful to share a single data container with multiple docker applications.
4. SeqRepo and refget REST interfaces, also available as a docker image.

Detailed instructions for each of these options are available at the GitHub repository (https://github.com/biocommons/biocommons.seqrepo).

Results

sha512t24 timing and statistical analysis

Members of the SHA-1, SHA-2, and MD5 family of digests were compared for timing. Because SHA-512 is specified in terms of 64-bit operations, the increased complexity of this algorithm is offset by the ability to process data twice as fast as with 32-bit operations specified by shorter digest algorithms. SHA-512 provides 512 bits of digest, which is far more than required for even extremely large sets of messages. Because this digest will be used as a key and transmitted widely, a practical tradeoff between key size and collision probability is desirable. Base64 results in encodings that are ceil(4/3) of the size of the input message; therefore, the truncation length should be modulo 3 for efficiency. The probability of collision was evaluated for a series of modulo-3 truncation lengths, and 24 bytes was chosen as a compromise between number of expected messages (sequences) and collision probability. For example, in a corpus of 10e+18 sequences, a key size of 24 bytes is expected to result in a collision probability of <1e-21. See

S1 Code for details on the timing and statistical analyses. Note that the truncated digest used here is not the same as the truncated digest proposed in the SHA-2 family of algorithms; in particular, we use the SHA-512 initialization vector in order to enable prefix abbreviations for larger digests.

Human SeqRepo collections

SeqRepo may be used for sequences of any type from any species. The current Human releases (available on https://dl.biocmmons.org/) consume approximately 12 GB on disk (with indexes) and consist of over 925,000 unique sequences from NCBI35, NCBI36, GRCh37 (with patches), GRCh38 (with patches), JRGv1, JRGv2, RefSeq (NM, NP, XM, XP, NG), and Ensembl (ENST, ENSP). The differential size between snapshots is approximately 1 GB, and is easily incrementally updated using the seqrepo command line interface. The command line interface enables users to easily import fasta files into custom sequence collections.

SeqRepo Python interface

SeqRepo provides a facile interface for accessing sequences, demonstrated in Fig 2.

```
>>> from biocommons.seqrepo import SeqRepo
>>> sr = SeqRepo("/usr/local/share/seqrepo/latest/")

# fetch() retrieves sequences by alias, with optional namespace and coordinates
>>> sr.fetch(alias="chr1", namespace="GRCh38", start=1000000, end=1000020)
'GTGGAGCGCGCCGCCACGGA'

# SeqRepo also supports dictionary-style lookup
>>> sr["refseq:NC_000001.11"][1000000:1000020]
'GTGGAGCGCGCCGCCACGGA'

# Aliases need to be namespace-qualified only if ambiguous
# An exception is raised if ambiguous (e.g., "chr1")
>>> sr["NC_000001.11"][1000000:1000020]
'GTGGAGCGCGCCGCCACGGA'

# Other identifiers for this exact sequence
>>> sr.translate_alias("NC_000001.11")
['GRCh38:1',
 'GRCh38:chr1',
 'GRCh38.p1:1',
 …,
 'refseq:NC_000001.11',
 'SEGUID:FCUd6VJ6uikS/VWLbhGdVmj2rOA',
 'ga4gh:SQ.Ya6Rs7DHhDeg7YaOSg1EoNi3U_nQ9SvO']
```

Fig 2. Examples of the native Python interface. SeqRepo retrieves sequences and metadata using conventional identifiers (*i.e.*, from NCBI, Ensembl, GRCh, LRG, and other sources) and from digest identifiers (*i.e.*, sha512t24u, ga4gh, md5, SEGUID). Identifiers are namespaced, and generally written as "<namespace>:<alias>". Aliases are always unique within a namespace. See the SeqRepo repository for installation instructions and S2 Code for example details.

https://doi.org/10.1371/journal.pone.0239883.g002

```
$ curl 'http://localhost:5000/seqrepo/1/sequence/NC_000001.11?start=100000&end=100020'
ACTAAGCACACAGAGAATAA

$ curl 'http://localhost:5000/seqrepo/1/metadata/NC_000001.11'
{
  "added": "2016-08-27T21:17:00Z",
  "aliases": [
    "GRCh38:1",
    "GRCh38:chr1",
    ...
    "GRCh38.p9:chr1",
    "MD5:6aef897c3d6ff0c78aff06ac189178dd",
    "refseq:NC_000001.11",
    "SEGUID:FCUd6VJ6uikS/VWLbhGdVmj2rOA",
    "SHA1:14251de9527aba2912fd558b6e119d5668f6ace0",
    "sha512t24u:Ya6Rs7DHhDeg7YaOSg1EoNi3U_nQ9SvO",
    "ga4gh:SQ.Ya6Rs7DHhDeg7YaOSg1EoNi3U_nQ9SvO"
  ],
  "alphabet": "ACGMNRT",
  "length": 248956422
}
```

Fig 3. Examples of the SeqRepo REST API. See the SeqRepo repository for installation instructions and S2 Code for example details.

https://doi.org/10.1371/journal.pone.0239883.g003

SeqRepo REST API

SeqRepo also provides a REST API to access sequence slices and metadata from any language while preserving the performance advantages of local sequence collections. An example of using the REST interface from a Linux command line is shown in Fig 3.

Timing comparison with NCBI E-utilities and ENA refget

Data serialization/deserialization and network latency can incur significant performance penalties on high-throughput systems. In order to quantify these factors in the context of sequence retrieval, we measured elapsed time for fetching 1,000 random sequence slices of 1–30 nucleotides from all chromosomes of GRCh38 using NCBI E-utilities [4], the European Nucleotide Archive implementation of refget, and SeqRepo. Results are showing in Table 2.

Table 2. Timing results for remote and local sequence sources.

	NCBI E-Utilities	ENA refget	SeqRepo Python API	SeqRepo REST API
elapsed time (s)	892	245	0.663	1.16
throughput (slices/s)	1.12	4.08	1508	858
relative throughput	≡1	3.64	1346	766

Timings are for 1,000 sequence lookups from 1) NCBI nucleotide sequences using the E-utilities interface, 2) European Nucleotide Archive using the refget protocol, 3) Local SeqRepo using the native Python interface, 4) Local SeqRepo interface using a SeqRepo REST API. Local SeqRepo access offers the best performance; using the SeqRepo REST API adds overhead, but enables access from other programming languages. Details are provided in S3 Code.

https://doi.org/10.1371/journal.pone.0239883.t002

SeqRepo tests were performed on a modern laptop with a solid-state disk. NCBI E-utilities and ENA refget tests were performed on a AWS c4.large instance in us-east. Given the magnitudes of the differences between the methods, we did not seek higher precision timings. See Supplementary Information for methods and timing data.

NCBI imposes rate limiting of 3 queries per second without an API key, or 10 queries per second with an API key, which we used. It was also necessary to implement rate limiting and retry logic in order to successfully retrieve all sequences. A genomic slice can be retrieved in one query, resulting in a theoretical maximum throughput of 10 sequence slices/second. Server-side rate limiting also defeats client-side parallelism.

As can be expected, there is a significant difference in overall throughput between the different approaches of accessing sequence data. The comparison presented here is intended to inform system architecture decisions. In particular for high-performance applications, local access is recommended. We are currently investigating further optimization with in-memory caching.

Conclusion

SeqRepo permits the use of conventional identifiers and digests for accessing and retrieving sequences. It is not necessary to adopt digest-based identifiers when using SeqRepo. This difference makes it possible to easily translate between conventional identifiers and digest identifiers. Furthermore, a locally-maintained SeqRepo instance enables pipelines to transparently mix public and custom sequences, such as masked sequences or alternative assemblies for variant calling.

Since its release in 2016, we have released 19 snapshots and improved namespace support for the GA4GH Variation Representation Specification. While refget and SeqRep can translate from GA4GH sequence identifiers to conventional aliases, SeqRepo is currently the only service that can translate from conventional identifiers to GA4GH sequence identifiers. This service is essential for translating variation to GA4GH standards.

Circular sequences are not directly supported by SeqRepo. However, such sequences may be stored in a linear form. We anticipate adding support for circular sequences in the future. SeqRepo currently implements only the BGZF backend described here. However, generalizing the interface to REDIS, AWS Aurora, or AWS S3 (as used by refget) would be straightforward and is under consideration.

Deciding between remote and local sequence services requires considerations of runtime performance, maintenance effort, data currency requirements, data privacy, and overall system availability. We demonstrated a 600-fold performance improvement for using local sequence storage rather than remote services. Although the relative performance benefit of using local sequences is unsurprising, the magnitude of the difference provides guidance regarding this important choice when designing an analysis pipeline, particularly if high-throughput is desired. SeqRepo significantly lessens the effort required to maintain a local sequence repository and provides significant performance benefits over remote sequence lookup services.

Supporting information

S1 Code. SHA-512 timing and collision analysis (jupyter notebook).
(PDF)

S2 Code. API examples (jupyter notebook).
(PDF)

S3 Code. Timing comparisons (jupyter notebook).
(PDF)

Author Contributions

Conceptualization: Reece K. Hart.

Data curation: Andreas Prlić.

Formal analysis: Reece K. Hart.

Investigation: Reece K. Hart.

Methodology: Reece K. Hart.

Project administration: Reece K. Hart.

Resources: Andreas Prlić.

Software: Reece K. Hart.

Validation: Reece K. Hart.

Writing – original draft: Reece K. Hart.

Writing – review & editing: Reece K. Hart, Andreas Prlić.

References

1. den Dunnen JT, Dalgleish R, Maglott DR, Hart RK, Greenblatt MS, McGowan-Jordan J, et al. HGVS Recommendations for the Description of Sequence Variants: 2016 Update. Hum Mutat. 2016. https://doi.org/10.1002/humu.22981 PMID: 26931183
2. Tan A, Abecasis GR, Kang HM. Unified representation of genetic variants. Bioinformatics. 2015; 31: 2202–2204. https://doi.org/10.1093/bioinformatics/btv112 PMID: 25701572
3. Hwang K-B, Lee I-H, Li H, Won D-G, Hernandez-Ferrer C, Negron JA, et al. Comparative analysis of whole-genome sequencing pipelines to minimize false negative findings. Sci Rep. 2019; 9: 3219. https://doi.org/10.1038/s41598-019-39108-2 PMID: 30824715
4. Sayers E. A General Introduction to the E-utilities. Entrez Programming Utilities Help [Internet] Bethesda (MD): National Center for Biotechnology Information (US). 2010.
5. Ruffier M, Kähäri A, Komorowska M, Keenan S, Laird M, Longden I, et al. Ensembl core software resources: storage and programmatic access for DNA sequence and genome annotation. Database. 2017; 2017. https://doi.org/10.1093/database/bax020 PMID: 28365736
6. Wang M, Callenberg KM, Dalgleish R, Fedtsov A, Fox N, Freeman PJ, et al. hgvs: A Python package for manipulating sequence variants using HGVS nomenclature: 2018 Update. Hum Mutat. 2018. https://doi.org/10.1002/humu.23615 PMID: 30129167
7. Babnigg G, Giometti CS. A database of unique protein sequence identifiers for proteome studies. Proteomics. 2006; 6: 4514–4522. https://doi.org/10.1002/pmic.200600032 PMID: 16858731
8. National Institute of Standards and Technology. Secure Hash Standard (SHS). Gaithersburg, MD 20899–8900: U.S. Department of Commerce; 2015 Aug. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf.
9. Josefsson S, et al. The base16, base32, and base64 data encodings. RFC 4648, October; 2006. Available: https://tools.ietf.org/html/rfc4648.
10. Babb L, Wagner AH, Schuilenburg H, Cline M, Riehle K, Lee J, et al. ga4gh/vr-spec: 1.1. Zenodo; 2020. https://doi.org/10.5281/ZENODO.3344568
11. GA4GH. Refget protocol. 2019 Jul. Available: http://samtools.github.io/hts-specs/refget.html.
12. Li H. Tabix: fast retrieval of sequence features from generic TAB-delimited files. Bioinformatics. 2011; 27: 718–719. https://doi.org/10.1093/bioinformatics/btq671 PMID: 21208982
13. PySAM Developers. PySAM GitHub repository. GitHub; Available: https://github.com/pysam-developers/pysam.