Planning Safety Solutions, Models and Algorithms for Special Databases^{*}

Fanni Magdolna Hízó^a, Attila Kiss^b

^aELTE Eötvös Loránd University, Faculty of Informatics, Budapest, Hungary fanni.hizo@gmail.com

^bELTE Eötvös Loránd University, Faculty of Informatics, Budapest, Hungary kiss@inf.elte.hu

Abstract

DNA sequencing is the process of determining the order of nucleotides in DNA. The rapid speed of sequencing attained with modern DNA sequencing technology has been instrumental in the sequencing of complete DNA sequences, including the human genome. Nevertheless it is a sensitive data which needs safe but efficient storage methods.

The goal in this research was to analyze different models and algorithms to determine which is the most applicable for storage, and query considering the need of user permissions, and encryption.

 $K\!eywords:$ bioinformatics, data science, data protection, data compression, database applications

MSC: 68P15, 92B05, 97R50, 68P20, 97R30

1. Motivation

Due to cheaper methods it is possible for everyone to keep their own DNA on their own device. However these are sensitive data which needs to be secured. Genomes are stored mostly in text files. The size of these files could be even 3GB or more depending on the analyzed species. Secured data management is not solved in these files. By using database managers, the levels of permissions can be easily managed, because many database manager systems have built-in security and encryption possibility.

^{*}The project was supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002).

Copyright © 2020 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

2. Related Works

Previous researches are also available about using database manager systems for storing genotypic data, describing and comparing multiple techniques [1]. One uses the database as a container to store bioinformatic file formats [2]. The authors of this paper explores the array-based technique, showing that it scales to far greater volumes than other existing techniques [1].

In this paper the authors construct a DNA relational database with a simple data model in which one DNA molecule stores one piece of data, [3] and this one introduces the database aspects of DNA computing [4]. This paper discuss the Cassandra NoSQL database approach for storing genomic data [5]. In [6] the authors managed to implement a web interface, to help students to develop informatic thinking skills. When considering possible encryption methods for data security, one of them is the 64-bit block ciphers (e.g Blowfish, 3DES), but these ciphers are known to be susceptible to collision attacks [7]. On the other hand the AES-128 uses 128-bit blocks and gives better protection [8].

3. DNA processing

3.1. DNA

Deoxyribonucleic acid (DNA) is a molecule composed of two chains that coil around each other to form a double helix carrying genetic instructions. Nucleic acids are one of the four major types of macromolecules that are essential for all known forms of life. Each nucleotide is composed of one of four nitrogen-containing nucleobases (cytosine [C], guanine [G], adenine [A] or thymine [T]). DNA sequencing is the process of determining the nucleic acid sequence. In this paper we worked with these sequences.

3.2. MySQL database

Firstly we created a database in local host, followed by a database table. We split the sequences into 1680-character-long pieces and uploaded many different DNA sequences into the table.

```
CREATE TABLE dna (

id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,

seqPosition INT NOT NULL,

sequence TEXT NOT NULL

);
```

Only with this structure, a simple pattern searching SQL query took about 5-10 minutes in MySQL [9].

SELECT * FROM dna

```
where seq like "%GACCGCGGCGCCGAGCGGCAGCCGCGCGGCCGGAC%"
and id=101;
1 row(s) returned 550.953 sec / 0.000 sec
```

3.3. PostgreSQL database

We did the same in PostgreSQL.

```
CREATE TABLE dna (
id SERIAL PRIMARY KEY NOT NULL,
seqPosition INT NOT NULL,
sequence TEXT NOT NULL
);
```

However in PostgreSQL [10] a pattern searching query has never took more than 5 minutes.

3.4. Indexing

A B-tree index can be used for column comparisons in expressions that use the =, >, >=, <, <=, or BETWEEN operators. The index also can be used for LIKE comparisons if the argument to LIKE is a constant string that does not start with a wildcard character. MySQL uses the Turbo Boyer-Moore [11] algorithm to initialize the pattern for the string and then uses this pattern to perform the search more quickly. We examined the query runtime after creating indexes for the table columns. As we mentioned before, a simple selection statement could took minutes to complete. After creating indexes in the database columns, the following statement returned within seconds.

```
SELECT * FROM dna where pos=111;
10 row(s) returned 0.110 sec / 0.015 sec
```

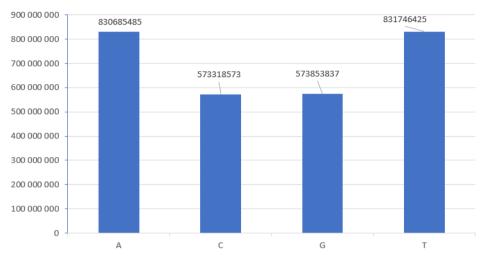
4. K-mers

In bioinformatics, k-mers are subsequences of length k contained within a biological sequence. Primarily used within the context of computational genomics and sequence analysis. Usually, the term k-mer refers to all of a sequence's subsequences of length k, such that the sequence AGAT would have four monomers (A, G, A, and T), three 2-mers (AG, GA, AT), two 3-mers (AGA and GAT) and one 4-mer (AGAT). The maximum number of variations of the possible substrings are 4^k .

4.1. Method

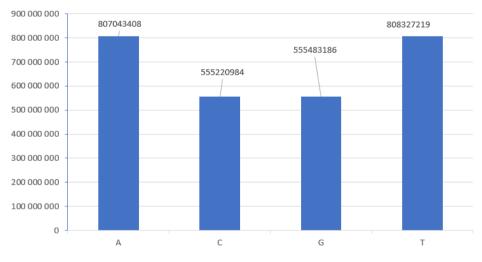
We developed a program in python to determine the number of k-mers. With reading the file line by line it counts the occurring nucleotid variations.

We examined human, hominoidea, plant and bacteria dna sequence too.



human 1mer

Figure 1: Human monomer



bonobo 1mer

Figure 2: Bonobo monomer

The next diagram shows the difference in percentage between each of the genom 3-mers self-compared value of occurrence.

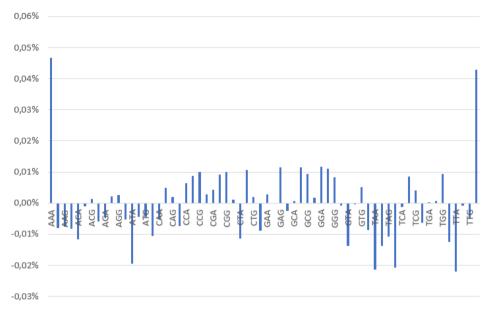


Figure 3: Comparison of a human genom and a bonobo genom 3-mer

4.2. Multiprocessing

We had an idea to use multiprocessing to increase the program efficiency. The basic idea was that if we split the file into smaller pieces, (for example four equal sized files) and if we add each piece into a thread, it should reduce the runtime. In python, multiprocessing is a package that supports spawning processes using an API similar to the threading module. Due to this, the multiprocessing module allows the programmer to fully leverage multiple processors on a given machine. It runs on both Unix and Windows.

The multiprocessing module also introduces APIs which do not have analogs in the threading module. A prime example of this is the Pool object which offers a convenient means of parallelizing the execution of a function across multiple input values, distributing the input data across processes (data parallelism).

We get the following results without and with multiprocessing: The runtime with a 3gb sized dna sequence is more than 24 minutes (1440.026 sec) without parallelism.

The runtime with the same, but divided file is less than 10 minutes (585.83 sec) with parallelism. This means a 59% performance improvement.

For a better representation we tested our program on a 820MB sized DNA sequence. At first we did not use multiprocessing, then split the file into 3/6/11 equal pieces and used multiprocessing for our calculation. The following figure represents the taken time by each run.

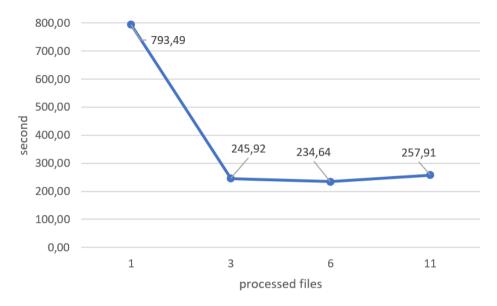


Figure 4: Multiprocessing

5. Compression

The Lempel–Ziv–Markov chain algorithm (LZMA) is an algorithm used to perform lossless data compression. This algorithm uses a dictionary compression scheme somewhat similar to the LZ77 algorithm published by Abraham Lempel and Jacob Ziv in 1977. It is used by 7zip nowadays [12, 13]. We decided to use this algorithm to compress our data because it features a very high compression ratio. The algorithm creates a dictionary while encodes the data and the fact that we are working only four characters provides us a small dictionary. With this method a 3GB sized DNA sequence compressed size is only 831 MB which means more than 73% storage space saving.

6. Safety solutions

In discussing security, it is necessary to consider fully protecting the entire server host (not just the server) against all types of applicable attacks: eavesdropping, altering, playback, and denial of service. Both MySQL and PostgreSQL have the option to create and add privileges with different roles to the database. Which provides the opportunity to determine which user could access the database and which could edit it. Every fresh MySql and PostgreSql system has an already defined role, which is always the superuser.

6.1. MySQL

The privilege system is the most basic security feature MySQL has to offer. It allows defining who can access database. Each user may be assigned a number of privileges. They define what each user can do once they have been authorized to connect. The privileges can be either global, granting specific right without restrictions, or limited to a specific object such as database or table.

MySQL implements an SSL library to provide connection encryption. All endpoints – server and clients – require their own set of private keys and certificates to be able to use encryption.

It offers a number of cryptographic [14] and hashing functions that can be used to either encrypt or decrypt data directly in queries – AES_ENCRYPT(), AES_DESCRYPT(), DES_ENCRYPT(), DES_DECRYPT(), SHA(), etc.

6.2. PostgreSQL

Same as in MySQL, PostgreSQL also use user roles to manage access privileges. Roles can act as users, groups or both. Roles can own database objects and can manage the permissions on these objects. Both of the discussed database management systems the owner of a created object is the user who created it.

PostgreSQL has native support for SSL connections to encrypt client/server communications for increased security.

We would like to mention that it also has the possibility to encrypt data, both one way (e.g passwords) or two way(e.g credit cards). For one way encryption, the crypt function packaged in pgcrypto provides an added level of security above the md5 way. We have several ways to accomplish the two way encryption, these implements the encryption part of the OpenPGP(RFC 4880) standard. However we did not try these methods out.

7. Conclusion

We created an environment where we can manage the user privileges and store the data securely and encrypted. The database manager gives us many options to easily add new users and create user hierarchy, from the principle of least privilege to a user with admin authority. We also managed to implement the first basic version of our processing tool that gives us the opportunity to compare different species genomes. Even started to test the multi-threaded version of the program for faster process of the genomes.

8. Future Work

In the future we want to try new data storage options and compare our results with other database managers, e.g NoSQL. As our reviewers insisted it would be interesting to examine another compression algorithms, and compare them to our results. Moreover, the processing optimization and performance improvement is in our main scope. Implementing a user friendly interface with options to import genomic data, which also capable of creating different graphs and reports based on the results is also one of our main goals.

References

- LICHTENWALTER, R.N., ZORINA-LICHTENWALTER, K., DIATCHENKO, L. Genotypic data in relational databases: efficient storage and rapid retrieval, In: Kirikova, M., Norvag, K., Papadopoulos, G.A. (eds.) *ADBIS 2017. LNCS* vol. 10509 (2017), 408– 421.
- [2] UWE ROEHM, JOSE BLAKELEY Data management for high-throughput genomics (2009), https://arxiv.org/abs/0909.1764
- [3] YAMAMOTO M., KITA Y., KASHIWAMURA S., KAMEDA A., OHUCHI A. Development of DNA Relational Database and Data Manipulation Experiments, In: Mao C., Yokomori T. (eds) DNA Computing. DNA *Lecture Notes in Computer Science* vol. 4287 (2006), 418–427.
- [4] GILLIS J.J.M., VAN DEN BUSSCHE J. A Formal Model for Databases in DNA. In: Horimoto K., Nakatsui M., Popov N. (eds) Algebraic and Numeric Biology. *Lecture Notes in Computer Science* vol. 6479 (2012), 18–37.
- [5] ANICETO, R., XAVIER, R., GUIMARÃES, V., HONDO, F., HOLANDA, M., WAL-TER, M.E., LIFSCHITZ, S. Evaluating the Cassandra NoSQL database approach for genomic data persistency, *Int. J. Genomics* 2015, 1–7.
- [6] RICE, M., GLADSTONE, W., WEIR, M., Relational databases: a transparent framework for encouraging biology students to think informatically, *Cell Biol. Educ.* 3(4) (2004), 241–252.
- [7] BHARGAVAN, K., LEURENT, G., On the practical (in-) security of 64-bit block ciphers: collision attacks on HTTP over TLS and OpenVPN, In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (2016), 456-467.
- [8] BOGDANOV, A., KHOVRATOVICH, D., RECHBERGER, C. Biclique cryptanalysis of the full AES, In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073 (2011), 344–371.
- [9] MySQL Documentation, https://dev.mysql.com/doc/
- [10] PostgreSQL Documentation, https://www.postgresql.org/docs/
- [11] JORMA TARHIO, ESKO UKKONEN Approximate Boyer-Moore String Matching, SIAM Journal on Computing 22(2) (1993), 243–260.
- [12] Lempel-Ziv-Markov chain algorithm, https://en.wikipedia.org/wiki/Lempel-Ziv-Markov_chain_algorithm
- [13] A. D. WYNER AND J. ZIV Fixed data base version of the Lempel-Ziv data compression algorithm. *IEEE Transactions on Information Theory* vol. 37(3) (1991), 878–880.
- [14] QIU, MANYING, AND STEVE DAVIS Database Security Mechanisms and Implementation Issues in Information Systems vol. 3 (2002), 529–534.