The Relational Way To Dam The Flood Of Genome Data

Sebastian Dorok
supervised by Prof. Gunter Saake
Bayer Pharma AG and University of Magdeburg
Germany
sebastian.dorok@ovgu.de

ABSTRACT

Mutations in genomes can indicate a predisposition for diseases such as cancer or cardiovascular disorder. Genome analysis is an established procedure to determine mutations and deduce their impact on living organisms. The first step in genome analysis is DNA sequencing that makes the biochemically stored hereditary information in DNA digitally readable. The cost and time to sequence a whole genome decreases rapidly and leads to an increase of available raw genome data that must be stored and integrated to be analyzed. Damming this flood of genome data requires efficient and effective analysis as well as data management solutions. State-of-the-art in genome analysis are flat-file-based storage and analysis solutions. Consequently, every analysis application is responsible to manage data on its own, which leads to implementation and process overhead.

Database systems have already shown their ability to reduce data management overhead for analysis applications in various domains. However, current approaches using relational database systems for genome-data management lack scalable performance on increasing amounts of genome data. In this thesis, we investigate the capabilities of relational main-memory database systems to store and query genome data efficiently, while enabling flexible data access.

1. INTRODUCTION

Genome analysis is a basic building block of personalized medicine as it allows to deduce impacts of genetic variations on the living organism [13]. Therefore, genome analysis requires access to huge amounts of heterogeneous data [6]. The data ranges from genome sequences, gene annotations, metabolic pathways, clinical lab results to electronic health records. The challenge is to analyze data correctly and to provide an infrastructure that enables analysis in a reliable and efficient way.

Genome-analysis-related data are stored in different data formats. Basic genome data such as raw sequences, aligned sequences and variant calls are mainly stored in flat files.

SIGMOD'15 PhD Symposium, May 31, 2015, Melbourne, Victoria, Australia.

http://dx.doi.org/10.1145/2744680.2744692.

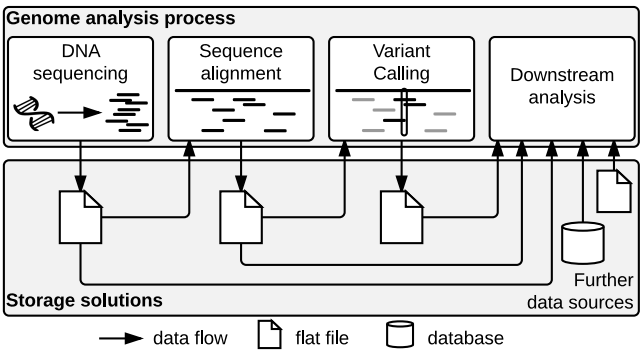

Figure 1: Primary storage solutions of single genome analysis steps. Genome data are mostly stored in flat files.

Relational databases are used to store clinical lab results and graph databases to store metabolic pathways. In Figure 1, we depict the basic steps of genome analysis and the currently used storage solutions. In the DNA sequencing step, a DNA sequencer makes the DNA digitally readable by converting it into strings of A's, C's, G's, and T's producing raw sequences. Due to technical limitations, DNA sequencers are not able to read complete DNA molecules in one run but only small, overlapping sequences of the DNA molecules. To reconstruct the complete DNA sequence from these sequences, in a second step, the small sequences called reads are aligned to a known reference genome. In the variant calling step, differences between sample and reference genome are detected. These differing sites are interesting spots for further downstream analysis. For example, certain mutations of genes indicate a predisposition for cancer [2]. To deduce consequences based on the genetic make up of patients, analysts have to interrelate different kinds of data. In this thesis, we concentrate on management of basic genome data that comprises raw sequences, aligned sequences and variant calls.

1.1 Problem statement

Next-generation DNA sequencing techniques lead to an increase of available raw sequencing data and to an increase of derived information such as aligned sequences and variant calls. Unfortunately, genome analysis is not an exact process but is based on heuristics to reduce computation time and assumptions made by scientists. Scientists often use their own tools to deduce consequences of variations in the

genome. In order to proof analysis results and hypotheses, scientists want to verify their analysis results by inspecting the raw data [3]. For that reason, efficient storage as well as retrieval solutions for raw sequences are required. Additionally, different pipelines of analysis tools have low result concordance [19]. Consequently, scientists have to integrate and analyze results from different analysis pipelines to perform more comprehensive analysis.

Using flat-file-only storage approaches as done today will lead to an increased data-management overhead for analysis applications. Analysis applications – especially downstream analysis applications – do not only have to manage data in various formats but also have to cope with increasing amounts of data while providing reasonable performance.

In order to reduce the data management overhead for data-intensive applications, database researchers have introduced the concept of data independence [8], that decouples applications from knowledge about physical data storage and enables declarative access to data. Approaches using relational database systems (DBSs) were already proposed to facilitate data integration and enable declarative access to genome data for analysis applications [15, 20, 22]. Although these approaches have shown the feasibility of using relational DBSs for genome-data management, scalable query performance as well as storage efficiency remains a challenge [20].

1.2 Research goals

In recent years, new techniques were proposed to speed up relational DBSs such as column-oriented data layouts [1] as well as using the main memory as primary storage [17]. The goal of our research is to investigate the capabilities of relational main-memory DBSs to provide a scalable genome-data management solution. We will develop a demonstrator to evaluate our approaches. To achieve our goal, we have to

- Determine a general set of functionality that our demonstrator should provide to support analysis applications.
- Create a database schema for genome data that enables the required functionality.
- Investigate and evaluate storage and query techniques to provide efficient access to increasing amounts of genome data.

1.3 Paper's structure

The remaining paper is structured as follows. In Section 2, we present related work. Then, we discuss preliminary results of our research in Section 3. Finally, we propose our research plan in Section 4.

2. RELATED WORK

In this section, we provide an overview of approaches that use relational DBSs for genome-data management. Moreover, we discuss strengths and potential weaknesses that we need to consider in our further research.

Relational DBSs provide capabilities such as data security, privacy, and integration that improve genome-data management and analysis [9]. Existing work mainly uses disk-based relational DBSs to deal with genome data. Atlas [22] and Biowarehouse [15] are approaches that use relational DBSs for *data integration* purposes. These approaches integrate genome analysis related data for downstream analysis. They show that using the relational model for modelling genome data and integrating it with further data sources is possible and suitable. The main focus of both approaches is data integration and not scalable storage and analysis of increasing amounts of genome data in a DBS.

Bouffard et al. present an approach to enable *statistical analysis* of genotype data within a relational DBS [4]. They show that using a specialized database schema improves the query performance compared to a naive approach storing the data in a format that corresponds to the flat file structure as done by Atlas or BioWarehouse. They model their problem in a star-schema like fashion with fact and dimension tables and are able to perform statistical analysis on variant calls using standard SQL. The approach by Bouffard et al. is not applicable to arbitrary use cases.

Other approaches provide a more general data model based on storing reads and their alignment. For example, Röhm et al. [20] use a read-centric schema to store genome data that is similar to Atlas and BioWarehouse. Moreover, they show the flexibility of declarative query languages such as SQL to query and analyze genome data. To support genome analysis via SQL, they implement a consensus base caller using genome-specific user-defined functions. Nevertheless, Röhm et al. remark that using user-defined functions can deteriorate query performance as parallelizing them is difficult.

To overcome scalability problems of disk-based DBSs, main-memory relational DBSs seem to be a promising approach [9]. Schapranow et al. provide an approach that enables fast analysis of genome data using state-of-the-art tools running on a main-memory DBS platform [21]. Certainly, the genome data remains in flat-files and is not directly managed by the DBS. Cijvat et al. present an approach to analyze genome data that is stored directly in the relational main-memory DBMS MonetDB [7]. They also use a read-centric approach to store the genome data. Their work shows that such a read-centric approach introduces processing overhead when querying and analyzing single positions in a genome.

Altogether, the relational model is suited to model genome data. Nevertheless, disk-based DBS approaches for genome-data management and analysis do not scale to increasing amounts of genome data. Main-memory DBSs provide a promising platform to solve this problem, but applying main-memory techniques requires further research regarding efficient storage as well as query techniques for genome data.

3. PRELIMINARY RESULTS

In this section, we present preliminary results of our work. First, we identify functionalities that a genome-data management solution should provide. Then, we present our current concept to model and query genome data. Finally, we present preliminary evaluation results regarding query performance and storage efficiency.

3.1 General functionality of genome-data management solutions

From our observations regarding requirements in genome analysis as well as from discussions with genome-analysis experts, we derived three functional requirements that a genome-data management solution should provide:

- Querying and filtering of single bases at specific genome positions to investigate genome regions of interest

	Query and filter	Integration	Raw data
Raw sequences	-	-	+
Aligned reads	+	+	o/+
Variant calls	+	+	-

Table 1: Suitability of data representations to fulfill requirements. Aligned reads are best suited. Legend: - ... not, o ... mainly, + ... most suitable

- Integration of genome data with additional information to facilitate downstream analysis
- Direct access to raw sequencing data to enable validation of downstream analysis results

Fulfilling these requirements depends on the provided functionality of the data management solution as well as on the stored data. The latter aspect is especially important, because the genome of a plant or human can be represented by different kinds of data. We can distinguish raw sequences, aligned sequences, and variant calls. In the following, we discuss what kind of data is most suitable to fulfill the requirements and summarize our results in Table 1.

Raw sequences cannot be used to query and filter single bases in specific genome regions as it is not possible to determine the genome position of unaligned sequences. For that reason, only aligned sequences and variant calls can be used to fulfill this requirements as these are already aligned to a specific genome region.

Almost all data sources needed for downstream analysis provide information regarding specific genome regions or positions. For example, gene annotations are related to a specific gene position in the genome. For that reason, raw sequences are not suitable but aligned sequences and variant calls are suitable to fulfill the integration requirement.

The last requirement - access to raw data - can be easily fulfilled when we just store raw sequences. Variant calls are not suited as they already represent an aggregated view on the genome and do not allow to access the raw sequence information. In case of aligned sequences it depends whether and how much of the original raw sequences can be extracted. Therefore, we have to remove the alignment information. Certainly, this approach is limited by the used alignment algorithm. If the alignment algorithm removes data from the aligned genome data set while processing sequences, not all raw data can be extracted.

Altogether, our assessment shows that aligned sequences are the most suitable data source to fulfill our three requirements, although the fulfillment of the last requirement highly depends on the alignment algorithm.

3.2 Base-centric genome database schema

Now that we know what data to store, we have to decide how to store it. The database schema as well as used database technology are decisive. From Röhm et al. [20], we know that a read-centric database schema introduces overhead when accessing single bases within reads as the reads have to be split on-the-fly during query processing. Such on-the-fly splitting can be avoided by allowing direct access to every single base using standard SQL. From Bouffard et al. [4], we know that a database specific schema design can pay out regarding query performance. Moreover,

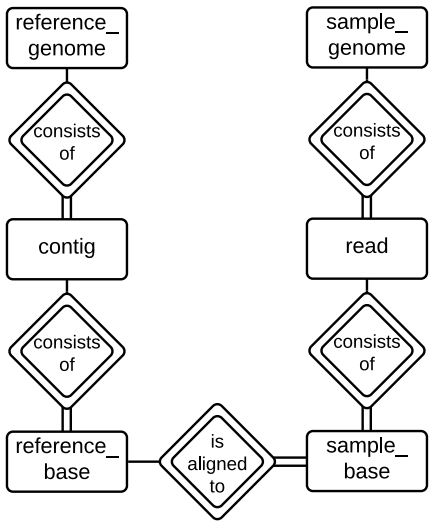

Figure 2: Base-centric genome database schema in entity-relationship model.

if we develop techniques that are based on standard relational operators then we can use the existing highly optimized database operators for efficient query processing. Additionally, an integration into other relational database management systems (DBMSs) should be possible whereas the support for user-defined functions highly depends on the used DBMS. In our approach, we designed a database schema that stores every single base of a read separately. Thus, we have to do the expensive read-splitting task once at import time. This is in contrast to existing approaches, which perform the read splitting during every query that accesses single bases [20] or split reads once and cache the split read [7]. The latter approach has the disadvantage to store the complete genome twice in worst case. In Figure 2, we depict the entity-relationship schema of our *base-centric* genome database schema. Our base-centric schema resembles a snowflake schema typically found in data warehouse applications. The sample bases represent the facts whereas sample genomes and reference genomes as well as their respective substructures describe dimensions.

To physically implement our base-centric genome database schema, we use column-oriented storage layouts as these have shown their superiority in analytical database applications [1]. For a genome-data management system, we expect similar analysis use cases as data are not likely to be updated after import. Moreover, Bouffard et al. show the need for analytical support [4]. Using column-oriented storage layouts has two more advantages that are important for our approach. First, column-stores support light-weight compression schemes effectively [1]. Thus, compression ratios are better than in row-stores, while processing overhead due to not needed decompression during processing does not deteriorate the query performance.

Second, column-stores are well suited for processing data in bulks that has shown to be more cache-efficient than tuple-at-a-time processing [1]. Thus, we are able to fully exploit the processing capabilities of modern hardware. Moreover, we expect that main-memory capacities will further increase in the near future making main-memory DBSs a

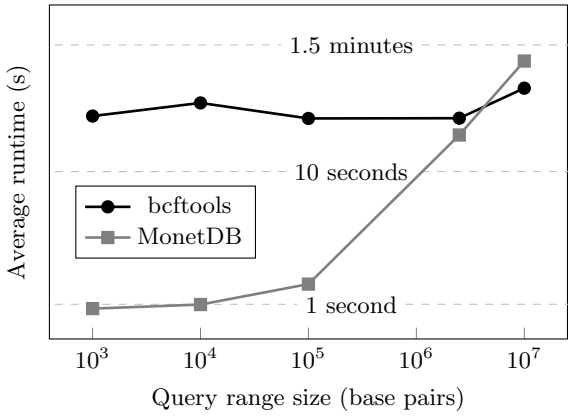

Figure 3: Average runtime of variant calling using bcftools and MonetDB. In small genome regions (< 2.5 million bases), MonetDB provides competitive performance.

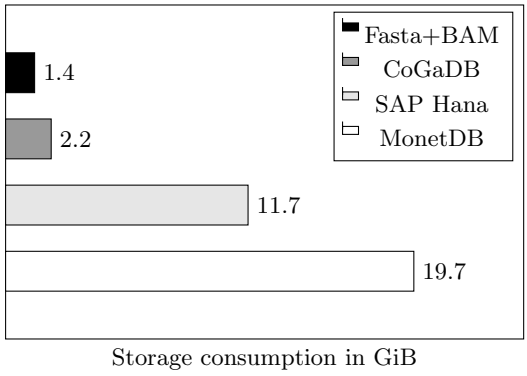

Figure 4: Storage consumption of databases storing one human chromosome 1. Fine tuning of compression schemes in CoGaDB results in similar storage consumption as in flat files.

promising basis to provide an efficient genome-data management solution.

3.3 Querying and storing genome data

To evaluate our approach regarding query and storage efficiency, we used the column-oriented main-memory DBSs MonetDB [23], SAP HANA [12] and CoGaDB [5].

In our previous work, we chose the Feb2013-SP2 release of MonetDB to evaluate the query and analysis performance of our approach [10]. We extended MonetDB with a hand-written user-defined aggregation function to support variant calling. The aggregation function computes the frequency of every base at a single position in the genome and returns the base whose frequency is above a certain threshold [18]. Then, we compare the resulting base with the base of the reference genome at the corresponding position. In case they are different, we call a variant. In Figure 3, we compare the performance of querying and computing variant calls in specific genome regions of one human chromosome 1 with bcftools. Bcftools is part of the samtools tool suite that enables access to aligned read data stored in flat files [16].

In our scenario, bcftools always performs a sequential scan, independent of region size. For that reason, MonetDB provides competitive query runtimes for genome regions with a size below 2.5 million bases. The runtime deteriorates in case of greater region sizes as the selectivity decreases and the intermediate results increase. For that reason, more data have to be processed in downstream operators resulting in performance degradation. Our evaluation shows that it is possible to query and analyze small base pair ranges efficiently. Nevertheless, we acknowledge that our simple variant calling approach using MonetDB does not compete bcftools regarding result quality, but it shows that a detailed exploration of genome data is possible.

Moreover, we investigated compression potentials using our base-centric database schema. First results regarding compression indicate big potentials. In Figure 4, we compare the database size when storing one human chromosome 1 using MonetDB, SAP Hana, CoGaDB and the flat files. The used version of MonetDB only provides dictionary compression of string values without bit-packing. Thus, MonetDB was not able to compress single character base values well. For that reason, we use MonetDB as baseline for evaluating the compression potentials of SAP Hana and CoGaDB.

SAP Hana saves more than 40% of the space that MonetDB requires for storing the database. SAP Hana applies standard light-weight compression schemes such as run-length encoding and dictionary encoding. Hence, SAP Hana compresses columns with low cardinality as expected. We used SAP Hana in version 1.00.60.379234. This version of SAP Hana compressed columns with high cardinality such as primary key columns using dictionary compression leading to storage overhead as the dictionary became as big as the column data. To explore further compression potentials, we imported the same database into CoGaDB and manually tuned the compression schemes for each column. We applied run length encoding and bit-packed dictionary compression. We observe a decrease of storage size by 80% compared to SAP Hana resulting in a similar storage consumption as the flat files.

The results on query performance as well as compression potentials motivate us to further investigate our approach.

4. FUTURE WORK

We implemented and evaluated our base-centric database schema using different main-memory DBMSs. Moreover, we demonstrated a first use case using our developed techniques to analyze plant genome data in a flexible and interactive manner [11]. These initial setups are currently used for analyzing specific parts of genomes. In future work, we intend to investigate further storage and querying techniques to scale our approach to multiple genomes.

First, we will investigate further compression schemes. We will apply genome-specific compression techniques such as reference-based compression because they enable huge compression potentials of aligned genome data [14].

Second, in case we apply genome-specific compression schemes, we also have to investigate efficient querying of such specifically compressed genome data. Thereby, we expect tradeoffs between efficiency of genome-specific compression and query performance. Additionally, we want to investigate different relational schema variants of our base-centric approach and evaluate their impact on storage size as well

as query performance. For example, we plan to denormalize the snowflake schema into a star schema to avoid joins between dimension relations and apply advanced star join techniques such as invisible join [1].

Finally, we research possibilities to integrate genome analysis tasks into a relational main-memory DBS using our base-centric approach. In this context, we evaluate strengths and weaknesses of integrated analysis approaches compared to state-of-the-art analysis using flat files.

5. CONCLUSION

Effective and comprehensive genome analysis requires access to raw sequencing data for validation of analysis results. As current approaches that use relational DBSs for genome-data management do not seem to scale, we suggest to use main-memory database technologies to provide a scalable genome-data management solution. Certainly, this requires an efficient design and implementation of methods and techniques to get maximum impact from main-memory performance potentials. We propose a base-centric genome database schema to store aligned genome data in column-oriented main-memory DBS. Such a database schema enables access to single genome positions using standard SQL as well as standard database operators. Our first results indicate promising potential regarding query performance as well as compression ratios.

6. ACKNOWLEDGMENTS

We thank our advisor Prof.Gunter Saake as well as Prof. Jens Teubner, Dr. Horstfried Läpple and Sebastian Breß for valuable feedback on this work. Moreover, we thank Dr. Uwe Scholz and Dr. Matthias Lange from IPK Gatersleben for their valuable insights into requirements of a genome-data management solution. The work of Dorok is supported by Bayer Pharma AG.

7. REFERENCES

[1] D. J. Abadi, S. R. Madden, and N. Hachem. Column-stores vs. row-stores: how different are they really? In *Conference on Management of Data (SIGMOD)*, pages 967–980, 2008.

[2] A. C. Antoniou, A. B. Spurdle, O. M. Sinilnikova, et al. Common breast cancer-predisposition alleles are associated with breast cancer risk in BRCA1 and BRCA2 mutation carriers. *The American Journal of Human Genetics*, 82(4):937 – 948, 2008.

[3] V. Bafna, A. Deutsch, A. Heiberg, et al. Abstractions for genomics. *Communications of the ACM*, 56(1):83–93, Jan. 2013.

[4] M. Bouffard, M. S. Phillips, A. M. K. Brown, et al. Damming the genomic data flood using a comprehensive analysis and storage data structure. *Database (Oxford)*, 2010.

[5] S. Breß. The design and implementation of CoGaDB: a column-oriented GPU-accelerated DBMS. *Datenbank-Spektrum*, 14(3):199–209, 2014.

[6] Y. Bromberg. Building a genome analysis pipeline to predict disease risk and prevent disease. *Journal of Molecular Biology*, 425(21):3993–4005, 2013.

[7] R. Cijvat, S. Manegold, M. Kersten, et al. Genome sequence analysis with MonetDB: a case study on Ebola virus diversity. In *Datenbanksysteme für Business, Technologie und Web (BTW)*, pages 143–149, 2015.

[8] E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[9] S. Dorok, S. Breß, H. Läpple, and G. Saake. Toward efficient and reliable genome analysis using main-memory database systems. In *Conference on Scientific and Statistical Database Management (SSDBM)*, page 34, 2014.

[10] S. Dorok, S. Breß, and G. Saake. Toward efficient variant calling inside main-memory database systems. In *Workshop on Biological Knowledge Discovery and Data Mining (BIOKDD-DEXA)*, 2014.

[11] S. Dorok, S. Breß, J. Teubner, and G. Saake. Flexible analysis of plant genomes in a database management system. In *Conference on Extending Database Technology (EDBT)*, pages 509–512, 2015.

[12] F. Färber, S. K. Cha, J. Primsch, et al. SAP HANA database: data management for modern business applications. *SIGMOD Record*, 40(4):45–51, 2012.

[13] M. A. Hamburg and F. S. Collins. The path to personalized medicine. *New England Journal of Medicine*, 363(4):301–304, 2010.

[14] M. Hsi-Yang Fritz, R. Leinonen, G. Cochrane, and E. Birney. Efficient storage of high throughput DNA sequencing data using reference-based compression. *Genome Research*, 21(5):734–740, May 2011.

[15] T. J. Lee, Y. Pouliot, V. Wagner, et al. BioWarehouse: a bioinformatics database warehouse toolkit. *BMC Bioinformatics*, 7(1):170, 2006.

[16] H. Li, B. Handsaker, A. Wysoker, et al. The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, 25(16):2078–2079, 2009.

[17] S. Manegold, P. A. Boncz, and M. L. Kersten. Optimizing database architecture for the new bottleneck: memory access. *The VLDB Journal*, 9(3):231–246, 2000.

[18] R. Nielsen, J. S. Paul, A. Albrechtsen, and Y. S. Song. Genotype and SNP calling from next-generation sequencing data. *Nature Reviews Genetics*, 12(6):443–51, 2011.

[19] J. O'Rawe, T. Jiang, G. Sun, et al. Low concordance of multiple variant-calling pipelines: practical implications for exome and genome sequencing. *Genome Medicine*, 5(3):28, 2013.

[20] U. Röhm and J. A. Blakeley. Data management for high-throughput genomics. In *Conference on Innovative Data Systems Research (CIDR)*, 2009.

[21] M.-P. Schapranow and H. Plattner. HIG - an in-memory database platform enabling real-time analyses of genome data. In *BigData*, pages 691–696, 2013.

[22] S. P. Shah, Y. Huang, T. Xu, et al. Atlas - a data warehouse for integrative bioinformatics. *BMC Bioinformatics*, 6:34, 2005.

[23] M. Zukowski, P. A. Boncz, N. Nes, and S. Héman. MonetDB/X100 - A DBMS in the CPU cache. *IEEE Data Engineering Bulletin*, 28(2):17–22, 2005.