Flexible Analysis of Plant Genomes in a Database Management System

Sebastian Dorok
Bayer Pharma AG
University of Magdeburg
Germany
sebastian.dorok@ovgu.de

Sebastian Breß
TU Dortmund University
Germany
sebastian.bress@tu-dortmund.de

Jens Teubner
TU Dortmund University
Germany
jens.teubner@tu-dortmund.de

Gunter Saake
University of Magdeburg
Germany
gunter.saake@ovgu.de

ABSTRACT

Analysis of genomes has a wide range of applications from disease susceptibility studies to plant breeding research. For example, different types of barley have differing characteristics regarding draught or salt tolerance. Thus, a typical use case is comparing two plant genomes and try to deduce which genes are responsible for a certain resistance. For this, we need to find differences in large volumes of aligned genome data, which is already available in large genome databases.

The challenge is to efficiently retrieve the genotypes of a certain range of the genome, and then, to determine variants and their impact on the plant organism. State-of-the-art tools are fixed pipelines with a fixed parametrization. However, in practice, users want to interactively analyse genome data and need to customize the parametrization.

In this demonstration, we show how we can support flexible ad-hoc analyses of arbitrary plant genomes using SQL with a small set of user-defined aggregation functions and dynamic parametrization. Furthermore, we demonstrate how genome analysis workflows for variant calling can be applied to our system and provide insights about the performance of our system.

1. INTRODUCTION

The increasing world population also increases the demand for food. Therefore, the harvest of food plants for humans as well as animals must be increased. Typically, plant species are bred to increase harvest or resilience against pests. Genome analysis allows us to determine differences in plant genomes and to determine which genes affect certain traits. Using this knowledge, a more target-oriented breeding is possible.

Genome analysis comprises sequencing of *deoxyribonucleic acid* (DNA) molecules, alignment or assembly of sequenced reads, and analysis of the reconstructed genomes. Typically, a first analysis is *variant calling*, which determines differences between a sample genome and a reference genome. Knowing the differences of a sample genome provides the starting points for further analyses. As plant genomes are huge, e.g., the barley genome comprises 5 billion base pairs, only a step by step analysis is feasible for scientists. Current tools for variant calling allow to call variants on complete genomes or only in specific regions. Thereby, scientists have to know where to look for differences such as the start and end sites of a gene and use fixed tool pipelines that generate the required results. The used tools are mostly command-line driven and flat-file based and put together using scripts. Such setups have the drawbacks that they miss flexibility and are not interactive.

Missing Flexibility The exchange of analysis tools requires knowledge about and adaptions to the scripts and also requires that the tools are compatible regarding used file formats and conventions.

Non-Interactivity When using a script that runs a defined pipeline of tools, the scientist has to wait until the analysis ends and has no opportunity to interrupt the analysis to start another ad-hoc analysis based on intermediate results.

In previous work, we suggested to use main-memory database systems as future genome analysis platform [2]. Moreover, we presented an approach to integrate variant calling into a relational database system [3]. In this demonstration, we present a system that allows users to interactively query and analyse plant genomes using SQL extended with a small set of genome-specfic aggregation functions.

The paper is structured as follows. In Section 2, we present background information on variant calling and analysis methods on genome data. Then, in Section 3, we describe the basic building blocks of our system. The demonstration setup is presented in Section 4. Finally, we conclude in Section 5.

2. BACKGROUND

In this section, we briefly present background information about genome analysis and approaches to integrate genome analysis steps into *database management systems* (DBMSs).

10.5441/002/edbt.2015.45

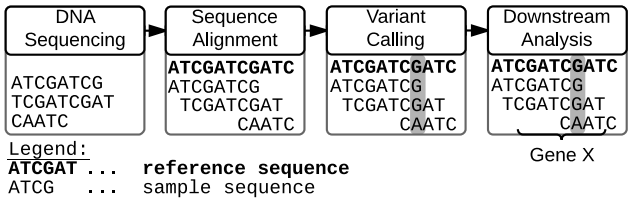

Figure 1: General genome analysis process.

2.1 Genome Analysis

Genome analysis consists of at least four steps. We depict these four steps in Figure 1. In the DNA sequencing step information encoded in DNA molecules is made digitally readable by translating DNA macromolecules into strings of A's, C's,' G's, and T's that encode the nucleo*bases* *A*denine, *C*ytosine, *G*uanine, and *T*hymine, respectively. State-of-the-art DNA sequencers are capable to sequence billions of bases in short time, which increases the amount of genome data to analyse dramatically. Unfortunately, DNA sequencers are not capable to sequence complete DNA molecules at once, but only in small, overlapping pieces. These DNA pieces are called *reads* and have a short length of about 100 base pairs that is rather small compared to the size of a complete genome such as the barley genome with 5 billion base pairs. In order to perform genome analyses, in a second step, DNA molecules must be restored after DNA sequencing from the short reads. After alignment, analysis of genomes starts. Because human or plant genomes are huge, only interesting sites in the genome are analysed at first. To identify such interesting sites, differences between sample genomes and the used reference genome are determined and a variant is called in case a difference is reliable. Such differences can be base exchanges at single genome sites, so called *single nucleotide polymorphisms* (SNPs), or *insertions and deletions* (InDels) of bases regarding the reference genome [6]. At variant sites, further downstream analysis takes place that often requires further data sources to determine the effect of a certain variant. For example, a mutation in a certain barley gene influences the number of spikes of a barley plant [4]. Often, downstream analysis is supported by visualization tools such as IGV that allow scientists to visually inspect and navigate through the genome. In Figure 1, a possible visualization of aligned and annotated genome sequencing data is sketched below the single process steps.

2.2 Related Work

There are several approaches to support genome analysis using database technology. Most of them concentrate on the integration capabilities of DBMSs. For example, Atlas [9] or BioWarehouse [5] integrate heterogeneous data sources using relational database systems and provide specialized APIs to load and access data.

Other approaches aim at integrating genome analysis steps into the database system. Rheinländer et al. present a special join operator that can be used to compute sequence alignments [7]. Wandelt et al. present an approach to perform similarity searches on thousands of genomes [11]. Therefore they use a special index structure that supports fast similarity searches and further allows for impressive compression ratios. In contrast, our approach aims at using minimal-invasive extension mechanisms to provide genome analysis functionality within a relational database system. The approaches can complement each other.

An approach by Rhöm et al. also uses relational databases to primarily store genome data [8]. Therefore, the authors experiment with a special *file wrapper* functionality of SQL Server 2008 to demonstrate how to access data stored in flat files via a DBMS. Additionally, the authors introduce special user-defined functions to manipulate tables and aggregate data items that allow for genome analysis using SQL. Thereby, the authors remark that the performance is problematic due to processing overhead and missing parallelization of user-defined functions. In contrast, our approach is based on main-memory database technology as platform to provide high-performance genome data management. Moreover, we store data directly in the database and use a base-centric database schema to efficiently support genome analysis tasks. Furthermore, we use only user-defined aggregation functions to implement genome-specific analysis tasks.

3. SYSTEM OVERVIEW

In this section, we sketch the basic building blocks of our demonstrator. First, we introduce the database schema used to represent aligned genome data. Then, we present extensions to support genome analysis tasks. Finally, we show how to call SNPs using a custom-aggregation function and how we support dynamic parametrization.

3.1 Database Schema for Aligned Genomes

We use an extended version of the database schema that we presented in previous work for genomes where a reference sequence consists of one contiguous region, such as a single human chromosome or small bacteria genomes [3]. Nevertheless, most animal and plant genomes consist of several contiguous sequences (e.g., chromosomes). This significantly complicates the schema, but is necessary to support variant calling on genomes in general. In Figure 2, we depict the extended entity-relationship schema. We introduced further hierarchies to better represent `reference_genome`s. Not for all genomes a complete reference sequence is known. For example, the reference sequence of barley consists of thousands of known contiguous sequences that have a known order but unknown gaps between them. To represent these contiguous sequences, we introduced the entity `contig`. Moreover, we integrated an entity to represent `sample_genome`s.

An aligned sample genome consists of millions of `read`s, that are aligned to a certain site in a reference genome. Instead of storing the alignment information per read, we split up every aligned read into its single bases and store each base and its mapping to the reference genome separately. In our base-centric database schema, these single bases are represented by the `sample_base` entity. The same idea is used to store the `reference_base`s of a reference genome that consists of contiguous regions instead of reads.

3.2 SQL Genome Extensions

To perform genome analysis tasks using SQL, we have to extend our DBMSs with genome-specific functionality. We use the concept of user-defined aggregation [12] to integrate the required analysis extensions into the DBMS and to make it available via SQL. In combination with our base-centric database schema, we can perform read- or genome-site-specific analyses by grouping and aggregating bases.

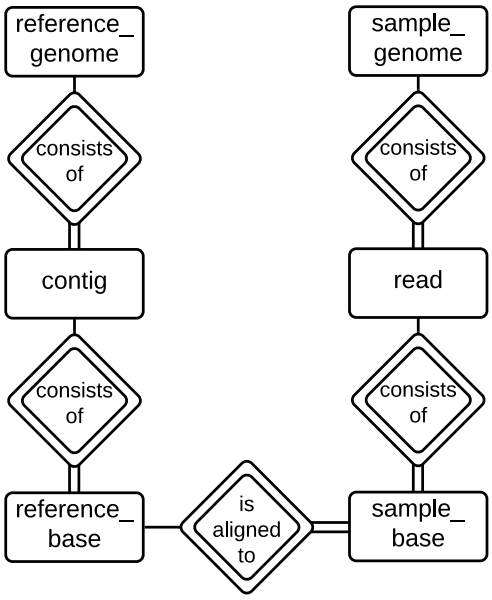

Figure 2: Database schema for aligned genomes showing the association between reference and sample genomes.

In order to call SNPs, we first need to call genotypes. Therefore, we implemented an aggregation function *genotype*, which consumes all bases of a sample genome at a certain position in the genome and determines which genotype is the likeliest at that position. For example, adenine, thymine, cytosine, guanine, or a combination of these bases in case of heterozygous genotypes. We use a frequency based genotype calling algorithm, which is the quasi standard for high coverage genome data [6].

For manual investigation or post processing, it is often necessary that specific reads or the computed genotypes in a certain region can be constructed from the database. Therefore, we propose a user-defined aggregation function *concat_bases*, which concatenates the (computed) bases to reconstruct the sequence, e.g., for the same `read_id`.

Especially, for the analysis of InDels, it is relevant to consider the bases around potential InDels. DNA sequencers have problems reading sequences of bases of the same kind. InDels in such *homopolymer* regions are more likely to be sequencing errors than real variations in the genome. To detect such regions, we integrated a new aggregation function that identifies homopolymer regions.

3.3 SNP Calling

In this section, we explain how to use our database schema and our genome extensions for SQL to call SNPs in a specific genome region.

The goal of SNP calling is to find differences in a sample genome compared to a reference genome, often in coding regions that contain genes. For this, the SNP caller needs to compute the genotype on each position of the aligned sample genome. Our database schema provides simple base-wise access to sample and reference genomes. Therefore, we only have to join the sample_base table with the reference_base table on the reference_id, contig_id, and the position in the reference genome. In order to avoid wrong SNP calls, we exclude reads where the alignment algorithm detected an insertion by filtering sample bases with an insert_offset > 0. Then, we group the sample bases by their position and their respective reference base nd aggregate the sample bases with our aggregation function *genotype*. Finally, we retrieve the variants by comparing the reference base with the computed sample genotype (the called genotype). We illustrate this procedure in an example query in Listing 1.

```
1 SELECT r.position, r.base as reference_base,
2     genotype(s.base) as sample_genotype
3 FROM sample_base s JOIN reference_base r ON
4     r.reference_genome_id = s.reference_genome_id AND
5     r.contig_id = s.contig_id AND
6     r.position = s.alignment_position
7 WHERE s.insert_offset = 0
8 GROUP BY r.position, r.base
9 HAVING reference_base <> sample_genotype;
```

Listing 1: Query for SNP calling

Based on this query, further analysis queries can be derived. For example, the analyst could reconstruct certain reads in a region of interest, which was discovered by the first query, using our *concat_bases* function. In order to concat bases in a sensible way, we have to guarantee the correct order of bases within a group. Therefore, we use a sort-based grouping approach with the knowledge that all bases within one read reside in the correct order in main memory.

3.4 Dynamic Parametrization

Variant callers typically have many parameters, which customize their behaviour. We support dynamic customization in two ways. First, we can express some parameters (e.g., the consideration of InDels) as simple filter conditions in SQL. Second, the user can set environment variables (e.g., the frequency threshold for the genotype function). For result reproducibility, we include the parametrization as meta data in the query result.

3.5 Putting it all together

With the previously introduced building blocks, users have the ability to analyse genome data in a flexible and interactive way using SQL statements. For example, users can call SNPs using the SQL query presented in Listing 1 and apply additional filter criteria to limit the SNP calling to a genome region of interest. Afterwards, users can check the sequencing coverage in the considered genome region or extract some genotype statistics to validate the variant calling result. Additionally, users can extract the sequences of reads in the genome region of interest. We depict an excerpt of possible analysis tasks and their possible combinations in Figure 3.

4. DEMONSTRATION SETUP

During the demonstration, we will remotely access a server, which runs our system. The server has two Intel Xeon CPUs (E5-2609 v2) @ 2.50GHz and 256GB of main memory @1333 MHz. We implemented our flexible variant calling on plant genomes by integrating our database schema and our aggregation functions in CoGaDB, a column-oriented, GPU accelerated, main-memory DBMS [1]. As evaluation database, we use Harrington barley genome data. We will demonstrate the following aspects in our setup:

Minimal Invasive Extensions and Flexibility: We show the audience how database technology can support

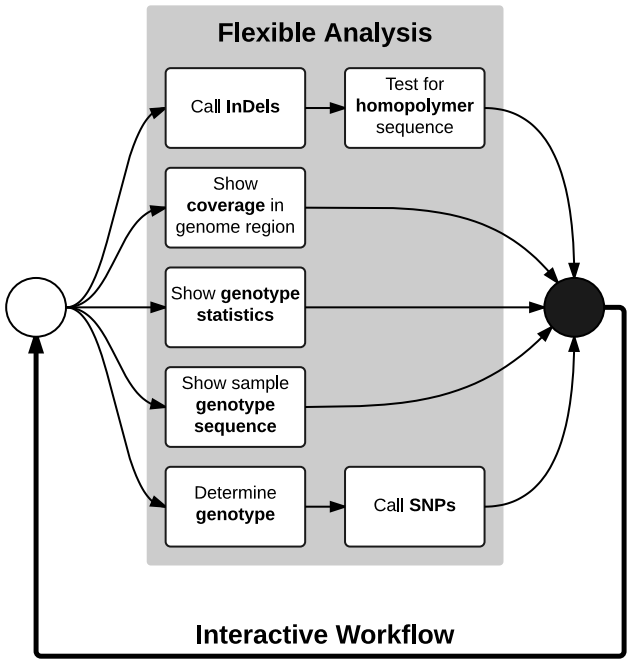

Figure 3: Toolbox for flexible and interactive analysis of genomes.

genome analysis steps such as variant calling. Our intention is to demonstrate that only small extensions of the SQL dialect are sufficient to allow exploration of genome data. We show the flexibility of our system by preparing a list of analysis queries the user can choose from. We assist the user in understanding the analysis queries by annotating each query with information about purpose of the analysis and interpretation of the query result. Additionally, we allow the user to formulate own ad-hoc queries on our database schema.

Simulated Genome Analysis Workflow: The core of the demonstration is to show a typical analysis workflow such as variant calling (c.f. Section 2) that can be performed using our demonstrator. Moreover, the user can experiment with the system on their own to explore the barley genome interactively via SQL. Thereby, our system presents the query results on a SQL commandline interface, but can also function as a backend for other analysis and visualization tools (c.f. Section 2).

Performance of Query Processing: During the interactive analysis, the user can view generated query plans and run-times of single operators, including the specialized genome operators to assess the performance of query processing.

5. CONCLUSION

In this demo, we show how to store aligned genome data in a relational database system and how we can apply real world workflows to our system. Integrating analysis steps into a DBMS and, thus, pushing code to data brings performance advantages due to reduced transfer costs. Moreover, extending SQL with bioinformatics operators combined with improved query performance allows for interactive and ad-hoc querying supporting scientists to prove or disprove hypotheses. This will be demonstrated in the sample workload that we prepare for the demo.

In future work, we want to improve the storage capabilities of our system by applying standard light-weight compression techniques known from database systems such as run-length encoding or dictionary encoding. Moreover, we investigate compression schemes specific for genome data such as referential compression [10] and how to integrate them efficiently into a relational database system. Additionally, we allow the integration of further information such as annotation information.

6. ACKNOWLEDGMENTS

We thank Matthias Lange and Uwe Scholz and their team at the IPK Gatersleben in Germany for fruitful discussions and providing aligned high coverage barley genome data for our experiments.

7. REFERENCES

[1] S. Breß. The Design and Implementation of CoGaDB: A Column-oriented GPU-accelerated DBMS. *Datenbank-Spektrum*, pages 1–11, 2014.

[2] S. Dorok, S. Breß, H. Läpple, and G. Saake. Toward efficient and reliable genome analysis using main-memory database systems. In *SSDBM*, pages 34:1–34:4, 2014.

[3] S. Dorok, S. Breß, and G. Saake. Toward efficient variant calling inside main-memory database systems. In *BIOKDD-DEXA*, 2014.

[4] T. Komatsuda et al. Six-rowed barley originated from a mutation in a homeodomain-leucine zipper I-class homeobox gene. *PNAS*, 104(4):1424–1429, Jan. 2007.

[5] T. J. Lee, Y. Pouliot, V. Wagner, P. Gupta, D. W. J. Stringer-Calvert, J. D. Tenenbaum, and P. D. Karp. BioWarehouse: a bioinformatics database warehouse toolkit. *BMC Bioinformatics*, 7(1):170, 2006.

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

[7] A. Rheinländer, M. Knobloch, N. Hochmuth, and U. Leser. Prefix tree indexing for similarity search and similarity joins on genomic data. In *SSDBM*, pages 519–536, 2010.

[8] U. Röhm and J. A. Blakeley. Data management for high-throughput genomics. In *CIDR*, 2009.

[9] S. P. Shah, Y. Huang, T. Xu, M. M. S. Yuen, J. Ling, and B. F. F. Ouellette. Atlas - a data warehouse for integrative bioinformatics. *BMC Bioinformatics*, 6:34, 2005.

[10] S. Wandelt and others. Data Management Challenges in Next Generation Sequencing. *Datenbank-Spektrum*, 12(3):161–171, 2012.

[11] S. Wandelt, J. Starlinger, M. Bux, and U. Leser. RCSI: Scalable Similarity Search in Thousand(s) of Genomes. *PVLDB*, 6(13):1534–1545, 2013.

[12] H. Wang and C. Zaniolo. Using SQL to Build New Aggregates and Extenders for Object- Relational Systems. In *VLDB*, pages 166–175, 2000.