Toward Efficient Variant Calling inside Main-Memory Database Systems

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

Sebastian Breß
University of Magdeburg
sebastian.bress@ovgu.de

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

***Abstract*—Mutations in genomes indicate predisposition for diseases or effects on efficacy of drugs. A variant calling algorithm determines possible mutations in sample genomes. Afterwards, scientists have to decide about the impact of these mutations. Certainly, many different variant calling algorithms exist that generate different outputs due to different sequence alignments as input and parameterizations of variant calling algorithms. Thus, a combination of variant calling results is necessary to provide a more complete set of mutations than single algorithm runs can provide. Therefore, a system is required that facilitates the integration and parameterization of different variant calling algorithms and processing of different sequence alignments. Moreover, against the backdrop of ever increasing amounts of available genome sequencing data, such a system must provide matured database management capabilities to enable flexible and efficient analyses while keeping data consistent. In this paper, we present a first approach to integrate variant calling into a main-memory database management system that allows for calling variants via SQL.**

I. Introduction

Mutations in human genomes indicate a predisposition for diseases such as cancer [12]. Moreover, genome mutations affect drug efficacy [22]. For that reason, knowledge about genome mutations and their effects on organisms' life improves detection and treatment of diseases.

Genome analysis is a technique to determine genome mutations [3]. Thereby, first, patients' genomes are sequenced by reading the base sequence of the DNA molecule. Due to technical limitations of current DNA sequencers, only small parts of genomes, called *reads*, can be determined at once [13]. To determine the complete base sequence of patients' genomes, the reads are aligned to a known reference genome [11]. Next, possible genome mutations are determined using variant calling algorithms [14]. Thereby, patients' genomes are compared site by site with the respective reference genome used for alignment. Then, especially at sites that differ, further downstream analyses are performed to assess whether a difference is a mutation and what consequences this mutation has for the patient's health.

A diversity of command-line tools exists for every genome analysis step. In a recent study, O'Rawe et al. show that the variant calling result highly depends on the used tools and that common tool combinations have only a result concordance of less than 60% [15]. Moreover, variant calling results depend on the concrete parameterization of the used tools. Thus, to minimize the probability of false negatives, variant calls of different tools and under different parameterization should be considered during downstream analysis [15].

Since downstream analysis approaches that use database management systems (DBMSs) as basis already store alignment data inside the database [17], our idea is to integrate different variant calling algorithms directly into the DBMS pushing code to data. This has the following two advantages: First, a downstream analysis application can request a variant calling at any time with any parameterization. This facilitates and improves downstream analysis as these applications can call variants on demand and are not restricted to already imported variant calls. Second, genome data does not have to be exported for external processing that reduces data transfers. Moreover, this improves data management quality of the genome analysis process as data is under control of the DBMS at any time.

In this paper, we contribute a first approach for variant calling inside a main-memory DBMS. We present a database schema to store genome alignment data, which efficiently supports variant calling using standard database operators and a user-defined aggregation function, which implements a heuristic determining a genotype for each site in the requested range. Moreover, we conduct a first performance study of integrated variant calling.

The remainder of the paper is structured as follows: In Section II, we describe general approaches for variant calling. In Section III, we present our database schema and the integration of variant calling via SQL that we evaluate in Section IV. In Section V, we present related work. Finally, we conclude and present future work in Section VI.

II. Background on Variant Calling

In this section, we explain the general idea of variant calling and describe two common approaches to call variants.

Variant calling tools analyze sequence alignments to determine differences between a sample and a reference sequence [14]. A sequence alignment describes for every read where and how it is aligned to a reference sequence. In Figure 1, we depict an exemplary alignment of five reads to a reference sequence. Thereby, a read can match completely (e.g., Read 3) or with differences (e.g., Read 1, at position 10). Moreover, deletions can occur as in Read 4 where the alignment algorithm inserted a gap at position 11 and 12. Insertions are the reverse situation. Thereby, a read contains bases that are not present in the reference sequence.

Often, additional information is available for every aligned read, such as base call quality values and a mapping quality value. The base call quality is determined by the DNA

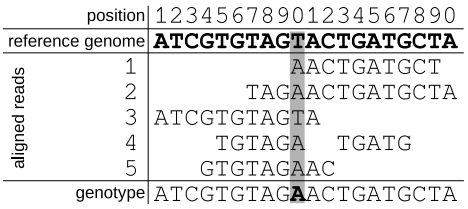

Fig. 1. Variant calling on aligned reads

sequencers before sequence alignment and describes the probability that a base call is wrong. In order to support working with base call error probabilities that are very close to 0, because these are the most important base calls, the base call error probability P is transformed into a quality value Q as follows: $Q = -10 * \log_{10} P$ [5]. Thus, a quality value of 30 represents a base call error probability of 0.001%. A mapping quality value is determined by sequence alignment algorithms to describe the probability that a complete mapping is wrong. Mapping quality values are also derived from error probabilities as described for base call quality values.

The general idea of variant calling is based on determining the genotype for the sample genome and comparing it with the reference sequence site by site. A genotype describes the concrete genetic makeup of an organism. As many reads overlap at the same genome site in the sequence alignment, the genotype is a consensus base determined by aggregating all bases that are aligned to the same site. In Figure 1, the last line represents the determined genotype. To determine the genotype at a specific site, two general types of approaches exist [14]. Probabilistic approaches include base call quality values to compute the genotype with the highest likelihood based on a given set of base calls. Another approach is to determine the relative frequencies of every base value that is aligned to a site (including deleted bases) and to use cut-off rules to decide about the genotype. The genotypes in Figure 1 are determined using cut-off rules. If the relative frequency of one base value is greater or equal to 80%, the genotype equals to this base value. If the relative frequency is less or equal to 20%, the base value is not considered for determining the genotype at all, as it is assumed that the base call is wrong. In between, a polyploid genotype is assumed.

III. Integration of Variant Calling into Main-Memory DBMSs

In this section, we present our concept for integration of variant calling into a main-memory DBMS. Therefore, we first describe important design considerations. Then, we present our approach in detail. Thereby, we describe our database schema and a technique to call variants.

A. Design Considerations

Current research shows that main-memory DBMSs outperform classical disk-based DBMSs in OLTP as well as OLAP applications by orders of magnitudes [8], [16]. The performance boost of main-memory DBMSs relies not only on exploiting huge amounts of main-memory as primary storage but also on using cache-aware algorithms and data structures. Furthermore, column-oriented data layouts are widely used as they provide cache-efficient memory access. Moreover, column-oriented data layouts, allow for better compression than row-oriented layouts as data of the same data type is stored consecutively [1]. Due to the expected performance benefits, we decide to use column-oriented, main-memory DBMSs as basis for efficient variant calling inside a DBMS.

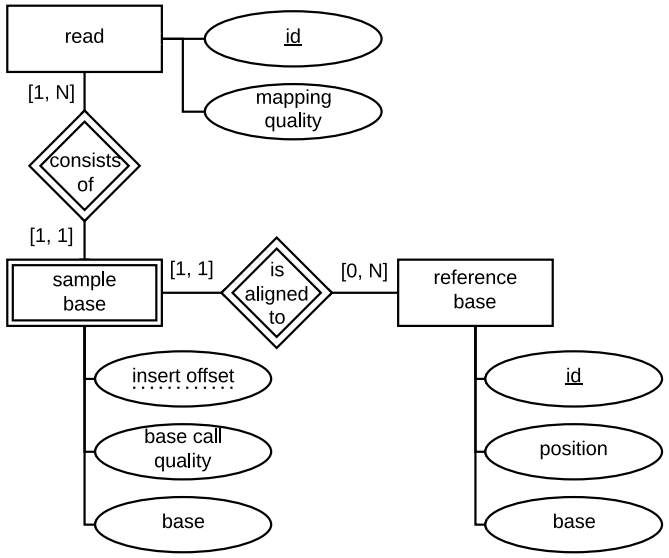

Fig. 2. Alignment database in entity-relationship model

B. Database Schema

In order to perform variant calling, we have to store the aligned reads of a sample genome and the respective reference genome in the database. Inspired by the idea that variant calling is a site by site comparison between a reference genome and the currently processed sample genome (cf. Figure 1), we decide to store all base sequences (i.e., sample reads and reference genome) in a per base manner. Thus, we can directly access single bases of every read and reference genome as needed for variant calling. Moreover, we expect high compression potentials as the base value alphabet is limited to a small set of characters that is beneficial for dictionary encoding. In Figure 2, we depict our database schema as entity-relationship schema. Every *sample base* entity belongs to exactly one *read* entity. Moreover, every *sample base* entity is aligned to exactly one *reference base* entity. To store information about deletions in reads, we introduce a new pseudo base value *X*. This is necessary to avoid false negatives during variant calling as the number of bases aligned to a certain genome site directly influences the genotyping result (see Section II). To include the information about insertions, we introduce a further attribute for a *sample base* entity that indicates a so called *insert offset*. In case a base is inserted between two sites of the reference genome, the respective *sample base* entity is aligned the *reference base* entity with the lower position and the *insert offset* is incremented. To restore original reads, we have to determine all *sample base* entities that belong to a *read* entity and order them by their alignment position (i.e., from the respective *reference base* entity). In case bases were inserted, we have to order all *sample base* entities by their *insert offset*.

Using this database schema, we can easily query for all bases that are aligned to a certain genome site and filter by base call and mapping quality. In Listing 1, we query for all

```
1  SELECT s.base
2  FROM sample_base s JOIN reference_base r
3       ON s.refid = r.id
4       JOIN read re ON s.readid = re.id
5  WHERE r.position = 1000000 AND
6        s.base_call_quality >= 30 AND
7        re.mapping_quality >= 30;
```

Listing 1. Query for aligned bases

bases aligned to position 1,000,000 that have a base call quality value greater or equal 30 and belong to a read that has been mapped with a mapping quality greater or equal than 30.

The drawback of storing sequences in a per base manner is the need to store information about read containment and alignment position for every base explicitly. In common file formats this information is stored more compact. In the SAM file format reads are stored as strings [10]. Thus, just the starting position of the read and a so called CIGAR string is stored to encode all information to describe the alignment. Thereby, the CIGAR string efficiently encodes matching, deleted, and inserted bases. Thus, the alignment position for every base can be derived implicitly and the read containment is obvious. As we designed our database schema to efficiently access all needed information for variant calling via SQL, we cannot avoid these drawbacks. However, we expect the compression capabilities of main-memory DBMSs to decrease storage consumption.

C. Variant Calling via SQL

In order to call variants, standard SQL does not provide all needed functionality. We can access every single base; we can filter by base call quality and mapping quality values; exclude genome sites that are not covered with enough bases; and limit the genome region to process by defining a start and end position, but we cannot determine a genotype with a standard SQL function. Thus, we have to integrate the genotyping functionality into the DBMS. The SQL standard provides different mechanisms to integrate user-defined functionality: stored procedures, stored functions, and external functions [4]. We use the external function mechanism to integrate a user-defined aggregation function, because we want to access the full performance potential of our approach by writing our aggregation function in C. The user-defined aggregation function is called *genotype* and computes the genotype at a specific genome position by evaluating base frequencies with fixed cut-offs of 20 and 80% (see Section II). With this user-defined aggregation function and SQL's *group by* operator, we can determine the genotype via SQL.

In Listing 2, we show the SQL query for variant calling that uses the built-in grouping functionality (Line 9) to compute the genotype at a single genome site (Line 5). Then, the genotype is compared with the reference base at the same genome site (Line 10). In case both bases differ, the position and both base values are returned as variant that can be further investigated (Line 1 – 2).

IV. Evaluation

In this section, we evaluate the main-memory consumption and the variant calling performance of our variant calling approach inside main-memory DBMSs. Moreover, we show first results regarding the result quality of our variant caller.

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

Listing 2. Query for variant calling

A. Experimental Setup

We use MonetDB [7] and SAP HANA [6], both column-oriented main-memory DBMSs, to evaluate our approach. We use MonetDB 5 Feb2013-SP2 release and SAP HANA version 1.00.60.379234. We implement our user-defined aggregation function in MonetDB only as we have access to the source code. Thus, we compile it from source with enabled optimizations. The SAP HANA software is already tuned and installed on our evaluation hardware by SAP. We do not have the possibility to integrate our user-defined aggregation function written in C into SAP HANA as we do not have access to the source code. Thus, we use SAP HANA only to evaluate the compression potentials.

We run our evaluation measurements on two different hardware setups. We install MonetDB on an Amazon Elastic Computing Cloud (EC2) instance[1] with 67 GB main memory and 8 CPU cores. The Ubuntu 13.04 64bit operating system runs on para-virtualized hardware. Para-virtualization allows the guest operating system to directly access certain host hardware components, and thus is more efficient than a fully virtualized environment [2]. SAP AG provides us a virtualized testing server for their SAP HANA software with 24 CPU cores and 141.8 GB main memory. However, since we use SAP HANA to show the compression potential only, the virtualization has no impact on our results.

We use genome data provided by the 1000 Genomes Project as evaluation data set [21]. For our evaluation, we use the aligned sequences of one human sample called *HG00096* that are aligned to human chromosome 1.[2] We use reference genome *grch37*. We developed a tool to convert the reference raw data from 1000 Genomes Project into CSV files that we used for bulk loading. In summary, our test data for evaluation comprises the human reference chromosome 1 and the according alignment data of human sample HG00096. Chromosome 1 is the biggest human chromosome and comprises 249,250,621 reference bases. The alignment data comprises 11,277,558 reads with 1,080,535,160 bases.

B. Main-memory Data Size

In Table I, we show the main-memory sizes of all tables of the relational database schema derived from our entity-relationship schema (cf. Figure 2) in MonetDB and SAP HANA. As MonetDB only supports dictionary encoding on strings with surrogates of at least 1 Byte, the compression potential of MonetDB is limited as the only two string columns in our database schema already have a size of 1 Byte. Thus, we

[1] http://aws.amazon.com/ec2/instance-types/

[2] http://ftp.1000genomes.ebi.ac.uk/vol1/ftp/data/HG00096/alignment/

#	Table	Column	In-memory size in MiB MonetDB	SAP HANA	Compression Ratio in %
1	HG00096	readid	4,122.000	2,662.777	64.599
2		refid	4,122.000	4,495.983	109.073
3		insert_offset	4,122.000	0.076	0.002
4		base	1,030.635	0.771	0.075
5		base_call_quality	4,122.000	1.758	0.043
6	grch37	id	950.875	2,614.746	274.983
7		position	950.875	1,782.782	187.489
8		base	237.875	0.003	0.001
9	reads	id	43.125	107.552	249.396
10		mapping_quality	43.125	0.010	0.024
11		overall	19,744.510	11,666.458	59.087

TABLE I. COMPRESSION RATES OF SINGLE COLUMNS OF SAP HANA COMPARED TO MONETDB

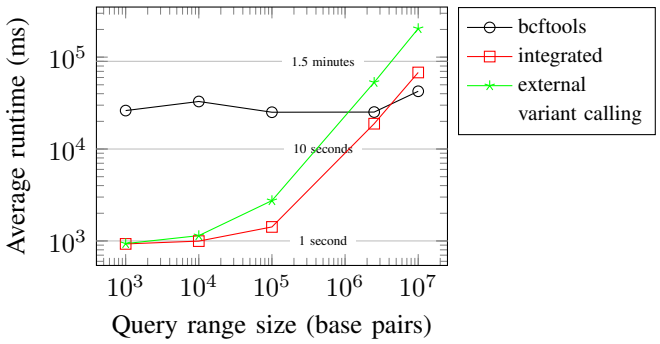

Fig. 3. Average runtime of integrated and external variant calling

use the main-memory sizes of MonetDB as benchmark for SAP HANA. In the last column, we compute the compression ratio of SAP HANA that is the ratio between SAP HANA's main-memory size of the column and MonetDB's column.

On the one hand, SAP HANA has enormous compression potentials (Line 3, 4, 5, 8, 10) but also compression ratios over 100% that means that SAP HANA needs more storage despite compression. Overall, SAP HANA saves up to 40% of the main-memory storage that MonetDB uses. The difference of compression potentials between the single columns can be explained with cardinalities. Columns with low cardinality can be compressed very well with techniques such as dictionary and run-length encoding that SAP HANA supports. The bad compression ratios are for columns with high cardinality, such as unique values as in identifier columns. As SAP HANA always compresses at least with dictionary encoding, the dictionary size increases with every unique value to compress.

The genome data columns show very good compression potential with standard database compression techniques. The problem to solve in the future is a more efficient compression of columns that store identifiers to ensure referential integrity.

C. Variant calling performance

We implemented our user-defined aggregation in MonetDB only (cf. Section IV-A). Thus, we restrict our evaluation of variant calling performance on MonetDB. In Figure 3, we show the variant calling runtime over different genome range sizes. Thereby, we directly compare our integrated approach for variant calling as described in Section III-C with an external approach that only queries for relevant data and then performs the variant calling outside the database. In order to get a measure for acceptable runtimes, we also perform variant calling over the same ranges with *bcftools*, which is part of the samtools tool suite [10].

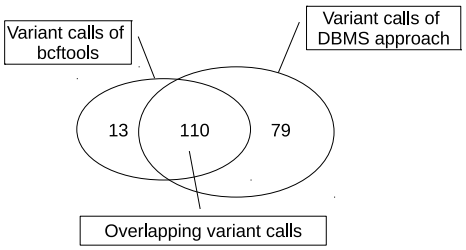

Fig. 4. Venn diagram for single nucleotide variant calls of bcftools and the DBMS approach on an exemplary base range of size 100,000

At least in range sizes below 2.5 million bases, the integrated and external approach for variant calling have acceptable runtimes. If the range sizes increase the runtime deteriorates. Thereby, the integrated approach is always faster than the external approach. The runtime difference between both approaches increases with increasing range size. The reason for this result is that with increasing range size more and more data has to be transferred to the external tool, which shows the benefit of variant calling inside a DBMS. Although the client runs on the same machine as MonetDB, the transfer overhead is significant. The poor runtimes in higher base ranges seem not to be related with our user-defined aggregation function as the runtime is also low in our external approach. Our profiling indicates that these observations are related to the internal processing of MonetDB or to the virtualized environment, which we could not determine further. For now, efficient variant calling on small range sizes of up to 2.5 million base pairs is possible.

D. Result quality

In Figure 4, we show a Venn diagram for one exemplary base range with a size of 100,000 bases to compare the variant calls of bcftools and our DBMS approach. The Venn diagram shows how many variant calls of bcftools and of our DBMS approach for variant calling on this genome region overlap and how many are unique to each tool. Our DBMS approach uses simple cut-off rules to determine genotypes. In contrast, bcftools uses a probabilistic approach to determine genotypes. Our DBMS approach calls 110 variants of the 123 variants called by bcftools that is 89.4%. 13 are not called. As the recall compared to bcftools is rather high, the precision of our approach is only 58.2% as it calls 79 variants that are not recognized by bcftools. Thereby, our additional calls are not wrong but bcftools uses a more restrictive heuristic. We just exclude reads with a mapping quality less than 30 and bases with a base call quality less than 30.

V. RELATED WORK

In this section, we provide an overview of existing approaches that use DBMSs to support genome analysis.

Several systems exist that use database technologies to support genome analysis. Thereby, the majority of systems

focuses on supporting downstream analysis tasks leveraging the data integration features of database systems such as Atlas [19], BioWarehouse [9], and BioDWH [20]. These systems provide tools to extract, transform, and load data into an integrated database, but do not integrate analysis tasks such as variant calling. Instead, they rely on upstream processing of sequence alignment and variant calling and integrate the result data. Moreover, the systems provide specialized interfaces to export data for external processing.

To the best of our knowledge, there is only one approach by Röhm et al. that integrates genotyping using user-defined functions into a DBMS [17]. Thereby, genome sequences are stored as strings and a user-defined function is needed to split genome sequences on the fly in order to compute the genotype from all bases at a specific genome site. Compared to this work, Röhm et al. do not provide a comprehensive performance and result quality evaluation of their approach. However, they show that storing genome sequences as strings generates additional overhead when processing single sites of a genome, because the parallelization of their user-defined functions is challenging and deteriorates the performance. In contrast, to Röhm et al., we store aligned reads in a per base manner eliminating the need for splitting reads on the fly. Thus, we have to store more information explicitly, but good compression ratios on single columns reduce this effect.

All the above approaches use classical disk-based DBMSs such as MySQL or Oracle. To the best of our knowledge, the only approach for genome analysis that uses main-memory DBMSs is from Schapranow et al. [18]. In their approach, they incorporate existing applications for genome analysis to work on their main-memory storage platform. In contrast to this work, they do not integrate the analysis functionality into the main-memory DBMS.

VI. Conclusion and Future Work

Variant calling is an important step in the genome analysis pipeline to determine for each site of a reference genome, whether the patients genome has the same genotype. The following downstream analysis can then use this information to determine the effect of a mutation on the patient's health. Since different downstream analysis applications have different needs, it would be more beneficial for them to request a customized variant calling. As downstream analysis is conducted more and more inside database systems, and already has to store alignment information inside the DBMS, it simplifies the downstream analysis if a variant calling can be computed inside the DBMS.

In this work, we present the first approach for variant calling inside a DBMS using SQL. We propose a database schema to store aligned reads for efficient per site access and showed that it is feasible to perform variant calling in all SQL compliant system that support user-defined aggregation functions. Our evaluation shows that our approach for variant calling can be used efficiently for range sizes of up to 2.5 million bases inside MonetDB, a popular open source DBMS. Furthermore, our database schema is designed to mainly have attributes with very low cardinality, which allows for efficient compression.

In future work, we will increase the efficiency of our variant-calling approach and extend it by other variant calling algorithms with the capability to configure the algorithms using the DBMS. The same reasoning for integrating variant calling inside DBMSs can be applied on the alignment step of the genome analysis pipeline as well. This would allow the application to request different alignments and variant callings. In fact, integrating variant calling inside a DBMS is a first step toward this goal. Finally, storing and processing genome data in a database system at all times would put the complete genome analysis process under the supervision of a DBMS, which would guarantee data consistency, and compliance; properties that are not met using existing pipelines based on command-line tools.

VII. Acknowledgements

The work of Dorok is supported by Bayer HealthCare AG. Moreover, we thank Veit Köppen for reviewing the manuscript.

References

[1] Daniel J. Abadi et al. Column-stores vs. row-stores: how different are they really? In *SIGMOD*, pages 967–980, 2008.

[2] Paul Barham et al. Xen and the art of virtualization. In *SOSP*, pages 164–177, 2003.

[3] Y. Bromberg. Building a genome analysis pipeline to predict disease risk and prevent disease. *J. Mol. Biol.*, 425(21):3993–4005, 2013.

[4] Andrew Eisenberg. New standard for stored procedures in SQL. *SIGMOD Rec.*, 25(4):81–88, 1996.

[5] Brent Ewing and Phil Green. Base-calling of automated sequencer traces using phred. II. Error probabilities. *Genome Res.*, 8(3):186–194, 1998.

[6] Franz Färber et al. SAP HANA database: data management for modern business applications. *SIGMOD Rec.*, 40(4):45–51, 2012.

[7] Stratos Idreos et al. MonetDB: Two decades of research in column-oriented database architectures. *Data Eng. Bull.*, 35(1):40–45, 2012.

[8] Robert Kallman et al. H-store: a high-performance, distributed main memory transaction processing system. *PVLDB*, 1(2):1496–1499, 2008.

[9] Thomas J. Lee et al. BioWarehouse: a bioinformatics database warehouse toolkit. *BMC Bioinformatics*, 7(1):170, 2006.

[10] Heng Li et al. The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, 25(16):2078–2079, 2009.

[11] Heng Li and Nils Homer. A survey of sequence alignment algorithms for next-generation sequencing. *Brief. Bioinform.*, 11(5):473–483, 2010.

[12] Nasim Mavaddat et al. Cancer risks for BRCA1 and BRCA2 mutation carriers: results from prospective analysis of EMBRACE. *J. Natl. Cancer Inst.*, 105(11):812–822, 2013.

[13] Michael L. Metzker. Sequencing technologies - the next generation. *Nat. Rev. Genet.*, 11(1):31–46, 2009.

[14] Rasmus Nielsen et al. Genotype and SNP calling from next-generation sequencing data. *Nat. Rev. Genet.*, 12(6):443–51, 2011.

[15] Jason O'Rawe et al. Low concordance of multiple variant-calling pipelines: practical implications for exome and genome sequencing. *Genome Medicine*, 5(3):28, 2013.

[16] Hasso Plattner. A common database approach for OLTP and OLAP using an in-memory column database. In *SIGMOD*, pages 1–2, 2009.

[17] Uwe Röhm and José A. Blakeley. Data management for high-throughput genomics. In *CIDR*, 2009.

[18] Matthieu-P. Schapranow and Hasso Plattner. HIG - An in-memory database platform enabling real-time analyses of genome data. In *BigData*, pages 691–696, 2013.

[19] Sohrab P. Shah et al. Atlas - a data warehouse for integrative bioinformatics. *BMC Bioinformatics*, 6:34, 2005.

[20] Thoralf Töpel et al. BioDWH: A data warehouse kit for life science data integration. *JIB*, 5(2), 2008.

[21] Marc Via et al. The 1000 Genomes Project: New opportunities for research and social challenges. *Genome Medicine*, 2(1):3, 2010.

[22] Hong-Guang Xie and Felix W Frueh. Pharmacogenomics steps toward personalized medicine. *Pers. Med.*, 2(4):325–337, 2005.