Datenbank Spektrum (2015) 15:185–191
DOI 10.1007/s13222-015-0198-x

SCHWERPUNKTBEITRAG

# Genome sequence analysis with MonetDB

## A case study on Ebola virus diversity

**Robin Cijvat · Stefan Manegold · Martin Kersten · Gunnar W. Klau · Alexander Schönhuth · Tobias Marschall · Ying Zhang**

Published online: 12 October 2015


**Abstract** Next-generation sequencing (NGS) technology has led the life sciences into the big data era. Today, sequencing genomes takes little time and cost, but yields terabytes of data to be stored and analyzed. Biologists are often exposed to excessively time consuming and error-prone data management and analysis hurdles. In this paper, we propose a database management system (DBMS) based approach to accelerate and substantially simplify genome sequence analysis. We have extended MonetDB, an open-source column-based DBMS, with a BAM module, which enables *easy*, *flexible*, and *rapid* management and analysis of sequence alignment data stored as Sequence Alignment/Map (SAM/BAM) files. We describe the main features of MonetDB/BAM using a case study on Ebola virus genomes.

## 1 Introduction

Next-generation sequencing (NGS) technology has confronted the life sciences with a "DNA data deluge" [14]. Thanks to its massively parallel approach, NGS allows for generating vast volumes of sequencing data, which, in comparison to conventional 'first-generation' sequencing methods, happens at drastically reduced costs and processing times. Consequently, biologists now need to invest in the design of data storage, management, and analysis solutions. Ever more often, improvements in this area are no longer an option, but a pressing issue.

As per common NGS-based "re-sequencing" work flows, short DNA fragments are sequenced and subsequently aligned to a reference genome, which aims at determining the differences between the sequenced genome and the reference genome. Thereby, the resulting alignment data files, most often stored in the Sequence Alignment/Map (SAM) format or its binary counterpart BAM [8] format, quickly reach the terabyte mark. Complementary software libraries, e.g., SAMtools [8], provide basic functionality, such as predicates-based data extraction. For more complex data exploration, scientists usually resort to writing customized software programs.

However, the traditional file based approach has several drawbacks. First, it requires users to manually manage file repositories. This quickly becomes a tedious process as file-based analysis tools generate files as output, that in turn are used as input to other file-based tools, and so on. Second, existing file-based tools usually only work properly with data that fits in main memory. Thus, researchers are often left with the non-trivial tasks of partitioning data into optimally fitting pieces and constructing final results from partial results. Moreover, having to repeatedly reload such large files is undesirable. Third, software development and maintenance are extremely time-consuming and error-prone tasks. They require highly specific knowledge of programming languages and applications. Finally, performance is paramount for big data analysis. Therefore, scientists have been increasingly enforced to become "hard-core" programmers, to exploit

R. Cijvat (✉) · M. Kersten · Y. Zhang
MonetDB Solutions,
Amsterdam, The Netherlands
e-mail: robin.cijvat@monetdbsolutions.com

S.Manegold · M. Kersten · Gunnar W. Klau · A. Schönhuth · Y. Zhang
Centrum Wiskunde & Informatica,
Amsterdam, The Netherlands

T. Marschall
Saarland University & Max Planck Institute for Informatics,
Saarbrücken, Germany
e-mail: marschal@mpi-inf.mpg.de

the full computational power of modern hardware, such as multi core CPUs, GPUs, and FPGAs.

Although Hadoop systems have recently gained much interest in big data processing, they are no ideal candidates to solve the aforementioned problems. Hadoop systems are primarily designed for document-based data processing [3]. They can be extremely fast in executing simple queries on large number of documents, e.g., distributed grep. But Hadoop systems quickly suffer from serious performance degradation, when they are used to process more complex analytical queries that often involve aggregations and joins [10].

The problems mentioned above have been extensively tackled by database management systems (DBMS), which are designed and built to store, manage, and analyze large-scale data. By using a DBMS for genome data analysis, one can significantly reduce the data-to-knowledge time. A major advantage of using a declarative language such as SQL is that the users only need to state *what* they want to analyze, but not *how* exactly to analyze. The DBMS takes care of efficient execution of the queries, e.g. choosing the best algorithms, optimizing memory usage, facilitating parallel execution where possible, and making use of the aforementioned modern hardware, while hiding the heterogeneity of underlying hardware and software systems. In this way, scientists can reap the fruits of more than 50 years of work of the database community on optimizing query processing, so as to spend more time on their primary research topics.

However, so far DBMSs have not been widely adopted in the life sciences for algorithmically difficult problems like sequence alignment, variant calling, and statistical processing of variant call data. This is because so far DBMSs do not yet cater to the requirements of data analysis in modern life sciences. There is a general lack of DBMS functionality and operations to directly query genomic data already stored in files. For example, the current common practice for getting SAM/BAM data into a DBMS is to first convert it into CSV files, then load them into a DBMS. This conversion step does not only incur a high data-to-query time, but also substantially increases storage requirements, especially if the original data are compressed. Moreover, it is extremely difficult to keep duplicate data consistent, when there are updates. Finally, although genomic data are encoded in strings, a standard DBMS data type, they have particular semantics. Without dedicated functions, it is not trivial to express even the most basic operations on genomic data using SQL.

### 1.1 MonetDB/BAM

Our first step towards a solution for big data analysis in life sciences is to get rid of the discrepancies between modern life science data analysis requirements and current DBMS functionalities. Therefore, we have extended the open-source column-based database system MonetDB[1] with a BAM module[2], which allows *in-database* processing of SAM/BAM files. The software is available as of the Oct2014 release of MonetDB.

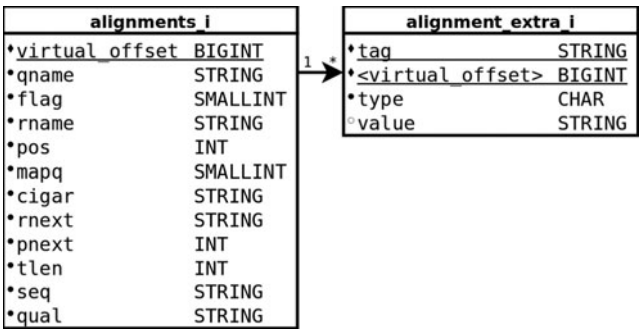



**Fig. 1** Sequential storage schema

MonetDB is primarily designed for data warehouse applications, such as data mining and Business Intelligence [9]. These applications are identified by their use of large data sets, which are mostly queried to provide business intelligence or decision support. Similar applications also appear frequently in the big data area of e-science, where observations are collected into a warehouse for subsequent scientific analysis. This makes MonetDB a good candidate to provide a data management solution for such applications.

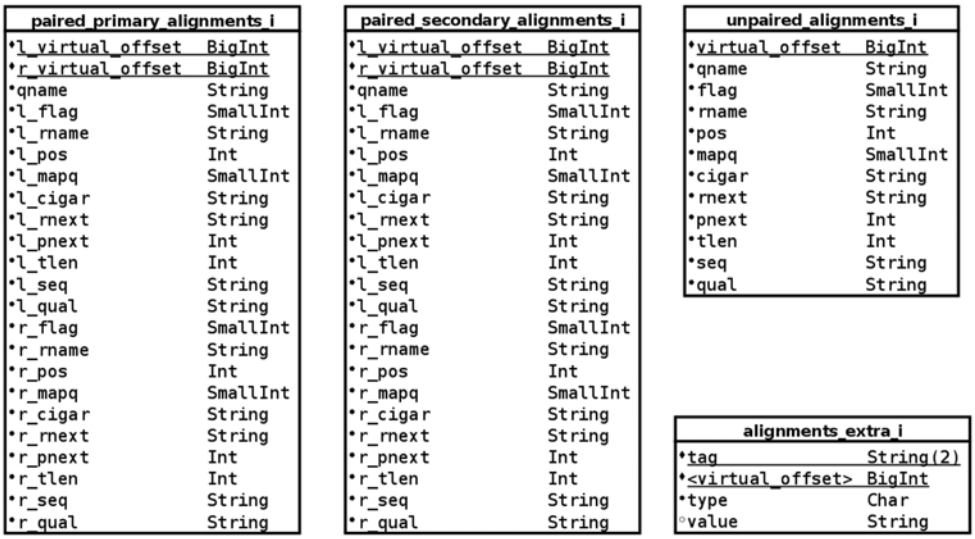
The main features of MonetDB/BAM include:

1. SQL loading functions to load individual SAM or BAM files, or a repository of them into a predetermined database schema.
2. SQL export functions that allow saving query results as SAM formatted files.
3. SQL functions for common operations on sequence alignment data, e.g., computing reverse complements of DNA strings and the actual lengths of aligned sequences.
4. Automatic reconstruction of primary and secondary read pairs, which accelerates analyses on paired-end read alignments.
5. Hierarchical clustering of loaded SAM and BAM files, which enables simple and efficient analyses on groups of SAM and BAM files.
6. R-support to perform in-database analyses on SAM and BAM files.

The loader of MonetDB/BAM loads SAM and BAM files into different relational storage schemas. Currently, users can choose from either the sequential (Fig. 1) or the pairwise storage schema (Fig. 2).

[1] https://www.monetdb.org/
[2] https://www.monetdb.org/bam/

**Fig. 2** Pairwise storage schema




**Sequential storage schema.** The sequential storage schema is a straightforward mapping of aligned reads in BAM files to columns in a database table. In the sequential schema, aligned reads of one BAM file are stored in two tables `alignments_i` and `alignment_extra_i`, where *i* is the internal ID of the BAM file. Each tuple in `alignments_i` contains all main fields of one aligned read, i.e., `qname`, `flag`, `rname`, `mapq`, `cigar`, `rnext`, `pnext`, `pos`, `seq` and `qual`. The `EXTRA` field of the aligned reads are parsed and stored in `alignment_extra_i` as `<tag,type,value>` tuples. The `virtual_offset` is used to link the tuples in these tables. In SAM files, this `virtual_offset` is the aligned read number, while in BAM files it is an actual offset into the BAM file that can be used to retrieve a specific aligned read.

**Pairwise storage schema.** The sequential schema is suboptimal for sequence analyses that operate on alignment pairs, because the expensive alignment pairs reconstruction has to be done repeatedly for every such analysis. Therefore, MonetDB/BAM comes with the option to load files into a pairwise storage schema, that explicitly stores primary and secondary alignment pairs. To reconstruct the read pairs, MonetDB/BAM uses the information that is readily available in several fields of the aligned reads, i.e. `flag`, `rnext` and `pnext`. All alignments that are not part of any alignment pair are stored in a separate table. In the pairwise schema, the table storing the extra information from the alignments is exactly the same as before. Its foreign key relation can now however not be connected to a physical table since the alignments are scattered across multiple physical tables. The pairwise schema can be extremely useful in use cases, such as calculating alignment ratios or statistics on the gaps in between aligned read pairs.

### 1.2 Earlier work

This paper is based on the work presented in [2]. That work focused mainly on preliminary analyses steps on SAM and BAM files, i.e., calculating aggregates, histograms, and subsets of SAM and BAM files, which may not necessarily lead to another file format such as the Variant Calling Format (VCF). We have run many experiments on a file repository of BAM files ranging from 22 MB to 100 GB. To make a comparison with file-based methods, several analyses were implemented both in C with the SAMtools API, and in SQL that could be run on MonetDB. The experiments were run on a machine with 32 Intel® Xeon® E5-2650 0 @ 2.00 GHz processors and 256 GB DIMM DDR3 1600 MHz (0.6 ns) RAM. Results of the experiments show that even though there is the overhead of loading the files into the DBMS, many queries run significantly faster inside MonetDB/BAM than using the file-based methods. This is especially true if only a small portion of the data is needed by the computation. In general, the more complex a query is, the more beneficial it is to do the computation inside MonetDB/BAM. Moreover, the experiments show trends that indicate that MonetDB/BAM scales better than using the file-based methods. Experiments also show that the storage requirements for using MonetDB/BAM lie in the same order of magnitude as using the uncompressed SAM format.

### 1.3 Related work

Several prototypes exist that also use DBMSs for genome data analysis. For instance, Röhm and Blakeley [12] propose a hybrid data management approach, which relies on file stre-

aming features of SQL Server 2008 to process gene sequence data stored as binary large objects (BLOBs). When faced with large data files however, loading data on-the-fly will suffer from the same performance problems as the file-based approaches. Moreover, this work does not consider additional DBMS functionality to facilitate genomic data analysis.

Schapranow and Plattner [13] describe an in-memory DBMS platform, HIG, in which existing applications are incorporated for genome analysis. But HIG does not integrate the analysis functionality *into* the DBMS.

The work of Dorok et al. [4] is closest related to our work. It uses a column oriented main memory DBMS to perform in-database variant calling using simple SQL. It proposes both a DBMS schema to store genome alignment data and an integrated user-defined function `genotype` (in MonetDB) to enable variant calling using simple SQL. However, this work mainly focuses on supporting variant calling, and does not mention how the raw data is ingested by the DBMS. Moreover, it proposes a base-oriented DBMS schema that, in contrast to our read-oriented schema, hampers analyses on read data. For example, extracting the actual sequence data of a subset of aligned reads would require the reconstruction of all these reads from the individual bases. Since this is a fairly common task for scientists who are used to working with e.g. SAMtools, this might give quite a performance overhead for their analyses. With MonetDB/BAM, we try to target more general genome data analysis tasks by providing easy-to-use loading functionality, combined with multiple SAM/BAM specific SQL functions, that provide means for efficient read-oriented analyses. Many examples of read based analyses can be found in [2].

### 1.4 This paper

In this paper, we demonstrate how MonetDB/BAM can be used to facilitate genome sequence alignment data analysis, by conducting a case study on a current and highly relevant topic: studying the genetic diversity of the Ebola virus.

## 2 Ebola virus diversity: a case study

Viruses populate their hosts as swarms of related, but genetically different, mutant strains, each defined by its own, characteristic genomic sequence. Analyzing such "mutant clouds", often called *viral quasi species*, is of clinical importance, as it explains virulence, pathogenesis, and resistance to treatment. Exploring the composition of sequences and their relative frequencies, the *genetic diversity* of a quasi species, based on NGS is a current, central issue in virology [1, 15].

We demonstrate how MonetDB/BAM can be used to explore the genetic diversity of Ebola infections. Although it has recently been established that Ebola is a highly diverse and rapidly mutating virus [5], conclusive insights are yet to be made. Our example exploration is done on a small file repository. For different experiments on larger file repositories, our earlier work can be consulted ([2]). We use BAM files containing sequence fragments from viral quasi species of the actual (2014) Ebola outbreak in Sierra Leone [5].

### 2.1 Preparing and loading data

First, we retrieved 32 files containing Ebola virus genome sequences (SRP045416) from [5]. Together they contain 6,786,308 reads and take 390 MB on hard disk. Then, we used the Burrows-Wheeler Aligner [7] to align the reads with the Zaire reference string (NC_002549.1) [16]. In this way, we were able to align 15.6% of the reads in a file to the reference string on average. The results are stored in 32 BAM files containing an entry for every read (so also the unmapped reads are stored), with a total size of 500 MB.

All BAM files are loaded into a MonetDB database with the SQL query Q1 as shown in Fig. 3. The first argument is the path to the repository of BAM files. The second argument chooses the storage schema: 0 for sequential (Fig. 1), 1 for pairwise (Fig. 2). For this use case we use the sequential schema. For all queries in this work, we have defined a special `MERGE TABLE alignment_all`, containing `alignment_`*i* tables of all loaded BAM files.

### 2.2 Multi-file analyses

A major advantage of using MonetDB/BAM is that it is simple to conduct analyses on multiple files at once. MonetDB comes with a built-in *Merge Table* technique, which allows defining virtual tables that combine data from as many sub tables (which contain e.g., SAM/BAM files) as desired. One can not only create merge tables over regular tables, but also add merge tables to other merge tables. The latter effectively enables creating hierarchical groups of sequence alignments. Since a merge table has the same signature as its partition tables (e.g., the `alignments_`*i* tables), SQL queries for certain analyses only have to be designed once, since they can then be applied to single files, groups of files, or even groups of groups of files. Meanwhile, the MonetDB *merge table optimizer* ensures efficient processing of the data that are divided over the different tables.

For the use cases in this paper, we created a merge table `alignments_all` (Q2), which we populate with data of all BAM files we have loaded into the database (Q3). In this way, whenever we add a file to the database, we simply add the new `alignments_`*i* table to `alignments_all`. Then, by re-running the queries on `alignments_all`, new aligned reads are automatically included in the analysis.

```
1 CALL bam_loader_repos('/path/to/ebola-bam-repo', 0)                    (Q1)
```

```
1 CREATE MERGE TABLE alignments_all (
2     virtual_offset BIGINT PRIMARY KEY,
3     qname          STRING,
4     flag           SMALLINT,
5     rname          STRING,
6     pos            INT,
7     mapq           SMALLINT,
8     cigar          STRING,
9     rnext          STRING,
10    pnext          INT,
11    tlen           INT,
12    seq            STRING,
13    qual           STRING
14 );                                                                    (Q2)
```

```
1 ALTER TABLE alignments_all ADD TABLE alignments_1;
2 ALTER TABLE alignments_all ADD TABLE alignments_2;
...                                                                      (Q3)
```

```
1 SELECT s.value AS refpos,
2        seq_char(s.value, al.seq, al.pos, al.cigar) AS seq_char,
3        COUNT(*) AS cnt
4 FROM generate_series(0, 18960) as s
5      JOIN (SELECT pos, seq, cigar FROM alignments_all WHERE pos > 0) AS al
6      ON s.value BETWEEN al.pos AND al.pos + seq_length(al.cigar)
7 GROUP BY refpos, seq_char ORDER BY refpos, seq_char                    (Q4)
```

```
1 SELECT refpos, SUM(cnt) AS cnt FROM positional WHERE seq_char IS NOT NULL
2 GROUP BY refpos ORDER BY cnt LIMIT k                                   (Q5)
```

```
1 SELECT refpos - refpos 1000 AS grp_start,
2        refpos - refpos 1000 + 1000 AS grp_end, AVG(cnt) AS average
3 FROM coverage GROUP BY grp_start, grp_end ORDER BY average DESC LIMIT k (Q6)
```

```
1 SELECT refpos, coverage.cnt AS coverage, diversity.cnt AS diversity,
2        CAST(diversity.cnt AS FLOAT) / coverage.cnt * 100 AS diversity_perc
3 FROM coverage JOIN (
4          SELECT refpos, SUM(cnt) AS cnt FROM base
5          WHERE seq_char IS NOT NULL AND seq_char <> SUBSTRING(ref, refpos, 1)
6          GROUP BY refpos
7      ) diversity USING (refpos)
8 ORDER BY diversity_perc DESC, coverage DESC, diversity DESC            (Q7)
```

**Fig. 3** Use case queries

### 2.3 Use case 1: computing positional data

Query Q4 in Fig. 3 shows how to compute the count of all characters that occur on all positions in MonetDB/BAM. The MonetDB-specific function `generate_series` generates a one-column table with a row for every integer number in the range. We use this in Line 4 to create a table with an entry for every position in the reference genome (NC_002549.1), with a length of 18960 [16]. Line 5 selects the position, the sequence, and the CIGAR string for all aligned reads. We join the series table with the aligned reads (lines 4–6). A result tuple is produced if the sequence of the read overlaps with a position in the series table (line 6). The join results are grouped and ordered on the reference positions of the reads and the characters that are found on these positions (line 7). Values of these characters are extracted in the `SELECT` clause (line 2). Finally, from the grouped result, we select the reference positions, the characters on these positions, and their occurrence counters (lines 1–3). Applying Q4 on the Ebola alignment data produces results as shown in Table 1, which reveals that, e.g., on position 46, there is one aligned read with an A and one with a C.

**Table 1** Result Q4

| refpos | seq_char | cnt |
|---|---|---|
| ... | ... | ... |
| 46 | A | 1 |
| 46 | C | 1 |
| 47 | A | 8 |
| ... | ... | ... |

### 2.4 Use case 2: computing coverage and diversity

Assume that the results of Q4 are stored in a table `positional`, query Q5 in Fig. 3 shows how to create a top-$k$ of positions that have the highest coverage, i.e., the highest number of aligned reads that overlap with these positions.

**Table 2** Result Q5

| refpos | cnt |
|---|---|
| 6239 | 9340 |
| 6240 | 9337 |
| 6245 | 9196 |
| 1571 | 9191 |
| ... | ... |

**Table 3** Result Q6

| grp_start | grp_end | average |
|---|---|---|
| 1000 | 2000 | 7053.6699999999992 |
| 3000 | 4000 | 6694.4919999999984 |
| 6000 | 7000 | 6681.6100000000024 |
| 4000 | 5000 | 6150.8489999999983 |
| ... | ... | ... |

The results of Q5 in Table 2 show that the reference position 6239 has the highest number of overlapping aligned reads, i.e., 9340.

Assuming the result of Q5 is stored in a view `coverage`, a next step is to calculate a top-*k* of regions with the highest average coverage, as Q6 in Fig. 3. The results of Q6 are in Table 3, which shows that the region 1000–2000 has the highest average coverage.

Diversity is another interesting analysis we can do with the results of Q4 and Q5, i.e., computing the percentage of aligned reads that differ from the reference genome on each position. Query Q7 in Fig. 3 produces a list of positions with their coverage and diversity, with decreasing diversity percentages. In Q7, we have loaded the reference genome string (NC_002549.1) [16] in the SQL variable `ref`. The function `SUBSTRING` returns a single character at the given `refpos`. Q7 first computes similar intermediate data as in Q5 (lines 4–6), except filtering out the positions with matching characters with the reference genome (line 5). Then, we join the `coverage` table with the just computed diversity information on the reference position (lines 3–7). This gives us for every position: i) the total number of overlapping aligned reads, and ii) the number of overlapping aligned reads that differ from the reference genome. Finally, we select the reference position, the count of both the coverage and the diversity sub results, and calculate the diversity percentage for all reference positions as the number of differing aligned reads divided by the total number of aligned reads for these positions (lines 1,2). The results of Q7 are in Table 4, which e.g. shows that all aligned reads at reference positions 721 and 7029 differ from the reference genome.

### 2.5 Query performance

We run all five queries on a moderate laptop (i7 Quad Core CPU, 4GB RAM, SSD disk). Fig. 4 shows the query execution times on different data sizes. The x-axis denotes both the number of files and the file size for each data set. All

**Table 4** Result Q7

| refpos | coverage | diversity | diversity_perc |
|---|---|---|---|
| 721 | 1471 | 1471 | 100 |
| 7029 | 1469 | 1469 | 100 |
| 5639 | 1131 | 1127 | 99.6463307 |
| ... | ... | ... | ... |

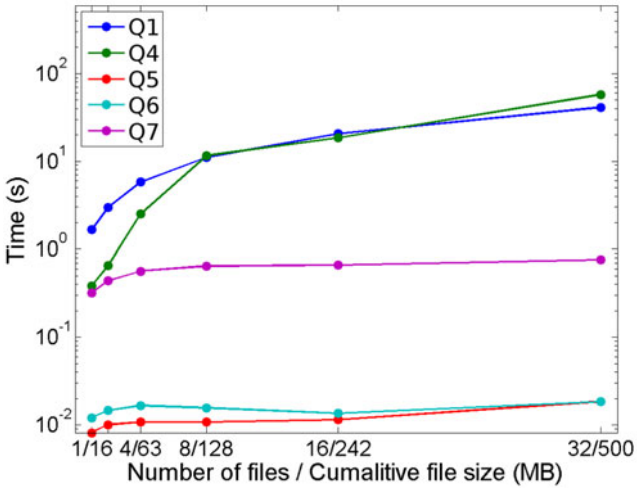



**Fig. 4** Execution times of all queries Q1, Q4, Q5, Q6, and Q7

results are averages of 10 runs. The execution times show a linear behaviour with growing data size. Loading (Q1) and computing positional data (Q4) are the most time consuming tasks. Q1 spends most time on decompressing the BAM files and passing values. The execution times of Q4 include the time to store its results in the `positional` table, which serves as a pre processing step for the remaining queries. [3] Once data are loaded and pre processed, the further analysis queries (Q5–Q7) are done in milliseconds, and the execution times are hardly affected by growing number of files and data sizes.

Note that the files used for these experiments are rather small, compared to e.g. BAM files containing data of human genomes. For analyses of MonetDB/BAM on larger file repositories, see [2].

## 3 Future work

As mentioned earlier, the current state of MonetDB/BAM is a first step towards facilitating exploration of vast amounts of genetic data. Many plans exist to further advance the state of MonetDB/BAM and plenty of open issues call for consideration. For instance, the performance and scalability of

[3] For this use case, we do not benefit from the read oriented storage that MonetDB/BAM uses. However, [2] shows many use cases for which it does.

MonetDB/BAM must be extensively evaluated and improved. We should stress the system with both BAM files of larger genomes, such as human or plant genomes, and terabytes scale file repositories. Also, we should compare the performance of our approach with existing analysis tools, such as BEDTools [11].

Besides optimizing MonetDB/BAM for large file repositories, we need to reduce the data-to-query time to enable users to immediately start exploring their files, without long loading times. Therefore, just-in-time loading techniques have been applied to MonetDB/BAM, that conform to the framework described in [6]. These techniques, together with experimental results, are presented in [2]. This also greatly reduces the storage overhead of using MonetDB/BAM.

Moreover, a bigger part of genetic analyses pipeline should be natively supported by MonetDB/BAM. For example, supporting a base-oriented database schema as described by Dorok et al. [4], and building in support for loading reference and/or index files, enables users to run in-database variant calling algorithms. Furthermore, implementing in-database support for VCF files eliminates yet another part of the genetic analyses pipeline that is normally file-based. However, it would also be interesting to investigate the possibilities of doing the alignment itself in the database. We would need to address which parts of the alignment process could be translated into native database operators and which operations should be delegated to third party software. The usage of the third party software should then be seamlessly integrated into MonetDB.

Finally, work flow support is a must for scientific exploration. MonetDB already provides various client connections (e.g., JDBC, ODBC), a REST interface, and seamless integration with the R software suite for statistical computing[1]. There is also ongoing work on integration with Python by mapping internal database columns to NumPy arrays. Therefore, MonetDB can be easily integrated into existing work flow systems, such as Taverna [17]. Moreover, there is ongoing work on a MonetDB-supported interface to SAMtools, which we refer to as "DAMtools"; a command line tool that implements a similar interface to SAMtools, but instead runs on top of MonetDB/BAM. With DAMtools, replacing parts of genetic pipelines that use SAMtools becomes effortless.

## 4 Conclusion

In this paper, we showed how to use MonetDB/BAM to facilitate exploration of the genetic diversity of the Ebola virus. Our study shows clearly how powerful MonetDB/BAM can be for this use case. First of all, we demonstrate how MonetDB Merge Tables can be used to group loaded data together, resulting in ways to apply single SQL query to one file, a group of files, or even hierarchical groups of files. Furthermore, we show that conceivable analyses on genome sequence alignment data can be easily expressed as SQL queries, provided the added functionality of MonetDB/BAM.

## References


1. Beerenwinkel N et al (2012) Challenges and opportunities in estimating viral genetic diversity from next-generation sequencing data. Front Microbiol 3:329
2. Cijvat R (2014) Bridging the gap between big genome data analysis and database management systems. Master's thesis, CWI and Utrecht University
3. Dean J, Ghemawat S (2004) MapReduce: simplified data processing on large clusters. Paper presented at the 6th Symposium on Operating System Design and Implementation, San Francisco, December 2004
4. Dorok S et al (2014) Toward Efficient Variant Calling Inside Main-Memory Database Systems. BIOKDD-DEXA Workshops, pp. 41–45
5. Gire SK et al (2014) Genomic surveillance elucidates Ebola virus origin and transmission during the 2014 outbreak. Science 345(6202):1369–1372
6. Kargin Y, Kersten ML, Manegold S, Pirk H (2015) The DBMS—your big data sommelier. Proceedings of IEEE International Conference on Data Engineering 2015 (ICDE 31)
7. Li H, Durbin R (2009) Fast and accurate short read alignment with burrows-wheeler transform. Bioinformatics 25:1754–1760
8. Li H et al (2009) The Sequence Alignment/Map format and SAMtools. Bioinformatics 25:2078–2079
9. Manegold S et al (2009) Database architecture evolution: mammals flourished long before dinosaurs became extinct. PVLDB 2(2):1648–1653
10. Pavlo A et al (2009) A Comparison of Approaches to Large-Scale Data Analysis. SIGMOD, pp. 165–178
11. Quinlan A Hall I (2010) BEDTools: a flexible suite of utilities for comparing genomic features. Bioinformatics 26(6):841–842
12. Röhm U, Blakeley JA (2009) Data management for high-throughput genomics. CIDR, pp. 97–111
13. Schapranow MP, Plattner H (2013) HIG - An in-memory database platform enabling real-time analyses of genome data. BigData, pp. 691–696
14. Schatz MC, Langmead B (2013) The DNA data deluge. IEEE Spectrum 50(7):28–33
15. Toepfer A et al (2014) Viral quasispecies assembly via maximal clique enumeration. PLoS Comput Biol 10(3):e1003515
16. Volchkov VE et al (1999) Characterization of the L gene and 5' trailer region of Ebola virus. J Gen Virol 80(Pt2):355–362
17. Wolstencroft K et al (2013) The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud. Nucleic Acids Res 41(Web Server issue):W557–W561