**Master Basismodule**

# Seamless Integration of Data Management and Statistical Computations

Claudio Brasser

Matrikelnummer: 14-921-746

Email: claudio.brasser@uzh.ch

August 9, 2019

supervised by Prof. Dr. Michael Böhlen and Oksana Dolmatova

University of Zurich[UZH]

Department of Informatics

# 1 Introduction

Classical database management systems are highly optimized for supporting massive amounts of data and their manipulation using relational operators such as joins or aggregations. However database systems typically lack support for analytical procedures used in data science for knowledge discovery and data analysis such as matrix algebra. Although it is achievable through constructs such as user defined functions (UDF) in SQL, the development using those is cumbersome and time intensive compared to statistical programming environments such as R or Python. Modern data base systems often have embedded interpreters for high-level programming languages like R or Python, however those typically suffer from data copying overheads [1]. Data Science packages as available in R/Python offer simple syntax and high level packages with support for large amounts of data science/machine learning algorithms but they can not deliver the same performance with large data sets that is to be expected from data base systems. Multiple solutions to close the gap between data base systems and analytical applications have been proposed. Grosse et al. [2] classify possible approaches into three groups:

1. Enhancing statistical software packages with support for big data
2. Enriching data base systems with support for analytical functionality
3. Improving the cooperation between a data base system and a statistical software package used for data analysis

The goal of this project is to study different approaches to combine relational database operations and analytical operations typically used in data science. In the following sections we aim to place all considered approaches into one of these categories as well as discuss their strengths and weaknesses. Finally we will demonstrate two different approaches to solving a linear regression on an artificial example data set.

# 2 Relational Matrix Algebra

This solution proposed by Dolmatova et al. [3] aims to utilize the strengths of a traditional relational database system such as MonetDB[1] in handling large amounts of data with relational operations. They implemented linear algebra matrix operations commonly seen in data science applications directly into the data base management system (DBMS). The direct integration of the matrix operations allows for optimization directly within the internal optimizer of the DBMS, with respect to the internal data structure of the DBMS. However, it also brings direct internal changes to the core of MonetDB and thus could be laborious to generalize for non column-store DBMS. Clearly, this proposition falls into the second aforementioned category of enriching a data base system with functionality for analytical operations.

MonetDB uses Binary Association Tables (BATs) as its internal data structure. Each BAT contains the values for one attribute of a relation. A BAT holds pointers to a head and a tail array as well as some metadata. The head array holds tuple identifiers (OID) which are unique per tuple within a relation. The tail array holds the attribute values corresponding to the tuple with the OID at the same array index in the head array. BATs can be combined to relations by merging them according to the OIDs. Relational operations in MonetDB consist of a number of operations on BATs. RMA also operates directly on BATs and thus does not require any preliminary modification of the data structures inside MonetDB. Dolmatova et al. [3] extended the SQL parser of MonetDB such that it understands matrix operations within the FROM clause of SQL statements. This allows for a seamless integration of matrix and relational operations within SQL. All operations have been directly integrated into the optimizer. The implementation and availability of most common matrix operations such as matrix multiplication, inverse computation, singular value decomposition etc. is an important factor of this solution as it allows the user to develop their own algorithms relying on efficient matrix algebra. The fact that this solution is integrated directly into SQL comes with several implications. Firstly, data analysis algorithms can be coupled directly to very efficient and highly expressive data exploration using the standard relational operations offered by the data base system. However the user has to write SQL queries to utilize the system, which do not offer the high-level syntax and expressiveness that is present in e.g. R[2] or Python[3]. Additionally, machine learning/data science algorithms often rely on iterative procedures, e.g. for optimization problems, which can typically be expressed better in procedural programming languages.

[1] https://www.monetdb.org/
[2] https://www.r-project.org/
[3] https://www.python.org/

# 3 Abstraction for Advanced In-Database Analytics

D'Silva et al. [1] proposed AIDA (Abstraction for Advanced in-database Analytics) as a solution for integrating a general-purpose data science workflow into a DBMS. They argue that existing solutions to procedural and/or incremental programs in DBMS don't offer the usability provided by popular statistical software packages such as R or Python. Thus they aimed to address not only efficiency issues in copying data from and to the DBMS but also efficiency in time needed for the programmer. Their solution was to extend the embedded Python interpreter of a DBMS (MonetDB) to provide an API with support for directly using data stored in the DBMS without copying it to the address space of the client language (Python). The authors argue that this leverages portability of the system to other DBMS as embedded Python interpreters are common in modern DBMS [1]. The defining factor of AIDAs architecture is that all data transformations are done on the server site inside the DBMS, invoked from the Python client through remote method invocations (RMI). This brings benefits with it such as the possibility to leverage the processing power of the server-grade hardware through low power client systems or the possibility to integrate AIDA into fully distributed computing environments with full transparency for the client [1]. Client-server communication may introduce concerns on data transfer bottlenecks or connection stability. AIDA respects this by only transferring remote references between client and server unless the client specifically needs access to the data, e.g. for manual inspection.

## 3.1 Tabular Data

AIDA uses *Tabular Data* objects as an abstraction between the matrix format used by Python/Numpy and relations inside the DBMS. All references returned to the client point to TabularData objects and AIDA transparently moves the representation of the data of such objects, such that both DBMS and Python can use it. At initialization, a TabularData object only contains information on the source relation(s) and the operations performed on them to get the desired set of records. Once a relational operation is invoked onto a TabularData object, it creates a Python dictionary with relation attributes as keys and attribute values as values. This makes zero-copy integration with MonetDB possible, as the underlying data structure for the value arrays in Python dictionaries are c-style arrays, the same structure used for BATS in MonetDB. If the content of a TabularData object is requested for matrix algebra operations, the data is loaded into a matrix format that can be used by Python. Once one of the two representations has been initialized, it remains in memory for future access. One of the main benefits in terms of usability with AIDA is the possibility to write custom operations on TabularData objects.

These take in a TabularData object and an arbitrary number of other parameters. Within a custom operation, any construct provided by the Python programming language can be used. The only restriction being that the operations needs to return a result either as a Python matrix or dictionary, the two data structures from which a new TabularData object can be generated within AIDA.

# 4 MonetDB in R

Several DBMS have interfaces for programming languages such as R or Python which are often used for data science tasks. Through these interfaces a client can query the DBMS to get copies of the desired data. However since data science tasks often require an iterative and explorative workflow, this can get cumbersome for the programmer and cause a large data copying overhead as the number of queries onto the DBMS grows [4]. Lajus et al. [4] proposed MonetInR to address the latter issue that is the large amount of data copied from the DBMS. As suggested by the name, the authors integrated MonetDB into the R statistical framework. They utilize the fact that both MonetDB and R use c-style arrays as a low-level storage data structure in a similar way as described in Section 3. Their solution can use data stored in MonetDB within R without having to copy and transform it to a format accepted by R.

## 4.1 Data Structures

R uses *Symbolic Expressions* (SEXP) as its internal data structure. All computations in R are done in a vectorized manner and thus there is no concept of singular values in R [4]. For example, a vector of integers (INTSXP) holds the R SEXP header with information on the type of data *directly* followed by an array of integers.

This structure is fairly similar to the BAT structured used in MonetDB, as described in Section 2. The tail array from a BAT can be used within R without any transformation, the only problem being the different headers used by both systems. R requires the SEXP header to be stored directly before the array of primitives. Thus, in order to use the values of a BAT within R, the authors modified the memory management of MonetDB in order to guarantee enough free space in front of the tail array. Once the BAT is accordingly initialized within MonetDB, MonetInR hands a reference to the tail of the BAT that should be analyzed to R. There, the system decrements the pointer reference by the size of the SEXP header (which can be determined by the computer hardware architecture) and writes the header into the free memory space. From this point on, the values can be used natively for all operations in R. This zero-copy integration offers advantages in both usability and performance. Firstly, the authors showed that their system outperformed multiple solutions which offer the same functionality without a zero-copy mechanism in various benchmarks, especially in test suites with large data selections. D'Silva et al. [1] state that usability is a highly influential factor for data science platforms, arguably even higher in priority than raw performance. MonetInR offers a prototype where a data science engineer is able to rely on standard R functionality, without having to worry about data conversion overheads.

# 5 RICE

Grosse et al. [2] proposed RICE, an integration of R into the the SAP In-Memory Computing Engine (IMCE). IMCE is a data management platform which stores most of the data in memory in a highly compressed format and offers interfaces for both pure SQL as well as SAP applications [2]. The authors identify one of the main problems within the collaboration of statistical software packages and DBMS to be the communication of data between both systems. Specifically, most common database interfaces for such communication like JDBC/ODBC provide query results to the client in a row-based manner. This is beneficial for certain streaming-based operations which rely on receiving complete tuples and can start operating on single tuples or parts of a data set. However, it may cause transformation overheads for systems with column-based data structures such as R [2]. Similar to MonetInR [4], Grosse et al. [2] proposed a solution working with shared memory (SHM) to reduce the communication overhead from sending query results to only sharing references to the result sets in memory. The result set is placed in the shared memory sector and stored in a format from which R can easily create its internal data structure, a dataframe.

The second solution proposed by Grosse et al. [2] integrates R directly into IMCE as a database operation. IMCE can describe logical database execution plans through data flow graphs with operations as nodes. The defining factor for this approach is the possibility offered by IMCE to use either standard relational operations as nodes or define custom operations. R-Op introduces the R operator as a node in the data flow graph. Next to an arbitrary number of input relations, it receives an R-Script as a string input argument which is executed in a separate R runtime once the execution plan reaches said node. The data is supplied to the R runtime using the SHM mechanism described earlier and thus no data copying is necessary. In contrary to SHM, this leaves the control flow on the DBMS side. The authors accentuate this fact, as it allows for leveraging the parallel execution plans of the DBMS by invoking multiple R runtimes without having to incorporate parallelism within the R script itself.

# 6 Experiments

In order to examine the differences between two possible approaches to the combination of relational database operations and statistical machine learning algorithms, we implemented a linear regression using two different approaches. The data for our experiments uses the same relational schema as the example provided in Section 2 of RMA [3] and all relations have been supplied with synthetic data. The first of which is using RMA [3] and thus handles all computations within MonetDB. The operation consists out of multiple relational operations for data preparation followed by the regression operation and is fully implemented in SQL (Listing 6.1). We perform multiple aggregations beforehand and isolate the dependent and independent variables through two WITH clauses and temporarily store them in *u1* and *u2*. Finally, the SOL operation included in RMA is invoked in order to perform the matrix operations required for the linear regression.

Listing 6.1: RMA linear regression

```
with u1(date, fault, length) as
(select t.date, avg_fault, length
from (
    select avg(fault) as avg_fault, date
    from fault
    group by date) t
join shift on t.date=shift.date
),
u2(profit, date) as (
select avg(profit) as avg_profit, date
from profit
group by date)

select * from
    (u1 on fault, length order by date)
    sol (u2 on profit order by date);
```

The second approach is done in the R statistical environment, using only native functionality. In order to perform the same data preprocessing and selection within R, we use dataframe functionality such as *merge* and *subset*. While not offering the same level of expressiveness as the SQL counterpart, these operations lead to the same result. In order to stay as close as possible to the ordinary least squares approach to linear regression as presented by Dolmatova et al. [3], we explicitly wrote the required matrix operations step-by-step (Listing 6.2).

Listing 6.2: R Linear Regression

```
u1 <- merge(
  x = setNames(
    aggregate(bakery.fault[, 2:2], list(bakery.fault$date), mean),
    c('date','fault')),
  y= subset(bakery.shift, select=c('date', 'length')),
  by='date')

u2 <- setNames(
  aggregate(bakery.profit[, 4:4], list(bakery.profit$date), mean),
    c('date', 'profit'))

u3 <- crossprod(
  data.matrix(subset(u1, select=c('fault','length'))),
  data.matrix(subset(u1, select=c('fault', 'length')))
)
u4 <- solve(u3)
u5 <- crossprod(
  data.matrix(subset(u1, select=c('fault', 'length'))),
  data.matrix(subset(u2, select=c('profit')))
)
u <- u4 %*% u5
```

Both solutions are implemented within approximately 20 lines of code. We argue that data aggregation in this case follows a cleaner syntax in the RMA implementation as it reflects a natural use case for SQL. As the ordinary least squares linear regression is fully implemented in RMA, it can be achieved in a single line of code using the *sol* operator. While this could be achieved in R as well either using available software packages or by encapsulating the matrix operations into a custom function, we decided against it for comparability. We conclude that in such a use case it would be superior to rely on the efficient data management offered by the DBMS, as the algebraic part of the implementation is readily available here. However this verdict may be suspect of change for more complex and especially iterative algorithms, where procedural programming comes to shine. In this case, solutions which focus on the cooperation between DBMS and statistical software packages might be preferable as they offer both high-level procedural programming as well as conventional DBMS functionality, as shown by Lajus et al. [4], Grosse et al. [2], and D'Silva et al. [1].

# Bibliography

[1] Joseph Vinish D'silva, Florestan De Moor, and Bettina Kemme. Aida: Abstraction for advanced in-database analytics. *Proc. VLDB Endow.*, 11(11):1400–1413, July 2018.

[2] Philipp Große, Wolfgang Lehner, Thomas Weichert, Franz Färber, and Wen-Syan Li. Bridging two worlds with rice integrating r into the sap in-memory computing engine. *PVLDB*, 4(12):1307–1317, 2011.

[3] Oksana Dolmatova, Nikolaus Augsten, and Michael H. Böhlen. A relational matrix algebra and its implementation in a column store.

[4] Jonathan Lajus and Hannes Mühleisen. Efficient data management and statistics with zero-copy integration. In *Proceedings of the 26th International Conference on Scientific and Statistical Database Management*, SSDBM '14, pages 12:1–12:10, New York, NY, USA, 2014. ACM.