Best of Both Worlds – Relational Databases and Statistics

Hannes Mühleisen
Database Architectures Group
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands
hannes@cwi.nl

Thomas Lumley
Department of Statistics
University of Auckland
Auckland, New Zealand
t.lumley@auckland.ac.nz

ABSTRACT
Statistics software packages and relational database systems possess considerable overlap in the area of data loading, handling, and transformation. However, only databases are mainly optimized towards high performance in this area. In this paper, we present our approach on bringing the best of these two worlds together. We integrate the analytics-optimized database MonetDB and the R environment for statistical computing in a non-obtrusive, transparent and compatible way.

1. INTRODUCTION
We live in a world where many decisions both in the commercial as well as in the political context are driven by insights gathered from analyzing raw data. Decision makers rely on accurate statistics and visualizations of their environment to be informed. However, the amount of raw data to be analyzed is constantly growing and has long overwhelmed either the capabilities of standard data analysis tools or the patience of their users waiting for results. For large-scale data analysis (or "business intelligence", as some like to call it), a multitude of different solutions has been proposed, for example specialized "data warehouse" systems and their corresponding user front-ends. However, these solutions require statisticians to part with their standard analysis tools and – unfortunately – often also require them to adapt their questions to the capabilities of the large-scale analysis system. To improve this undesirable situation, we propose a tight but unobtrusive integration of these two worlds – statistical tools and relational databases. One of the increasingly popular tools to create derived *information* from raw data is *GNU R*. R is "a language and environment for statistical computing and graphics" [11]. Most prominently, R features a collection of tools for data analysis and the corresponding facilities for data handling and graphics creation. Moreover, the *CRAN* collection contains thousands of user-contributed packages for R, which can be easily installed and greatly expand R's capabilities [10].

The typical work flow for data analysis in R is to first load the raw data from a file, then to select and transform raw data into a form suitable for statistics calculation, and then apply a statistical algorithm and visualization. However, the amount of data that can be analyzed using this process is limited by the amount of available memory on the system on which R is run, which are typically desktop computers. A logical next step to mend this problem is to store the raw data in a relational database system, similar to the process standard IT applications went through long ago. The standard process is now modified by not loading the raw data into R, but instead to load it into a database. Then, one can "outsource" the selection of data relevant to the analysis as well as basic calculations and aggregations to the database system. For example, the `RPostgreSQL` and `ROracle` CRAN packages allow R to load data from PostgreSQL and Oracle databases, respectively [2, 8]. However, in order to tell the database which data is to be transferred to R, a user still is required to write queries in the standardized Structured Query Language (SQL), which breaks work flows and increases training requirements.

This unsatisfactory current situation brings us to our research question: Can R make use of the powerful selection and transformation features of relational databases in a non-disruptive, transparent and compatible way? As a methodology, we have chosen to attempt this integration and verify its usefulness with a real-world statistics issue, survey analysis. The dataset being used for our experiments is the American Community Survey (ACS) [12]. The three-year ACS dataset consists of nine million rows and is too large to be loaded into R in reasonable time on typical desktop computers. While the R database interface (DBI [9]) provides a generic way of communicating with a relational database from R, not all relational databases are equally well suited to support R. In the described common scenario, few queries touching a large part of the stored data are sent to the database. Also, transformation procedures and simple calculations make recommending a database optimized for "On-line analytical processing" (OLAP) rather obvious. Furthermore, R handles tabular data as a list of columns, and calculations are typically performed column-wise, and – in the case of ACS and in most others – only a fraction of columns is actually analyzed at a given time. These factors together suggest a column-oriented database design. For our proposed approach, we have chosen MonetDB [3], an open-source column-oriented database system. We aim at pushing all reasonably supported operations into the database, as this will have the highest chance of substantially reducing the amount of data that has to be transferred between processes.

In the remainder of this paper, we describe our approach to integrating R and MonetDB in Section 2. In Section 3, we introduce our use case, survey analysis. In Section 4, we present our experiments on the ACS dataset. Section 5 describes related work, and Section 6 concludes this paper and gives an outlook on future work.

2. MARRYING MONETDB AND R

Before we can discuss the user experience when dealing with data stored in the database from the R environment, we have to enable communication with the database itself. As mentioned, the R package `DBI` provides a generic interface to send SQL queries to relational databases and retrieve the query result table as a R object, the `data.frame`. `DBI` can best be compared to JDBC, which is the generic database interface for Java.

To allow this interaction with MonetDB, we have implemented a new MonetDB-specific driver [7]. This driver is written in native R code for high compatibility with existing environments and ease of installation. The driver uses R's socket facilities to connect to the database, which is listening on a TCP port, and supports MonetDB's MAPI protocol, which is used to send queries and receive their results.

However, as discussed in the introduction, the ability to send SQL queries to a relational database from the R environment only takes us part of the way. Users are still required to "think database" when interacting with the data stored there. To solve this issue, another layer of connectivity is required, which we describe in the following section.

2.1 Virtual Data Object

One of the most widely used data types in R is the `data.frame`, which consists of a named list of individually typed columns. Once constructed, this type supports various operations, such as selecting a subset of rows or columns, arithmetical operations on numeric columns, comparisons with constants and many more. Many of the contributed packages are able to operate directly on `data.frame` objects, for example, the `ggplot2` graph plotting library can take a `data.frame` as data source. Obviously, this type is very close to data stored in relational tables or general query results from a relational database system. For example, the `DBI` database abstraction layer specifies `data.frame` objects to be returned by data-retrieving functions such as `dbGetQuery`.

A fully functional method to manipulate database tables in R would now be to use `DBI` to load a database table into a `data.frame`, and use R's built-in operators to select, reshape, and generally manipulate the data. However, this is of course very unpractical for large datasets, as the entire table has to be transferred from the database into R. A more sensible solution is a *virtual data object*, or *proxy object*. This object lives in the R environment, and behaves almost exactly like a materialized `data.frame`. All the calls that the database in the background supports are translated there, and only when the database is out of its depth the potentially much smaller and already aggregated data is transferred into R, where the user can now run more complex calculations.

Since R allows overloading of functions and operators through its class system, such a solution was implemented in `monet.frame`. This object is initialized with a database table. Rather than reading the data, it will only determine the names and types of the columns in the table and its size. In the case of MonetDB as well in many other database systems, this information comes from the database *catalog*, which is independent of the amount of data that is actually in the table. The user can now manipulate this object the very same way as a local object. In contrast however, her actions do not trigger actual data transfer, but rather a reformulation of the SQL query that backs the virtual data object. For example, consider the example R interaction in Listing 1:

Listing 1: Interaction with virtual data object

```
c <- dbConnect(MonetDB.R(),
        "monetdb://localhost/db1")
mf <- monet.frame(c, "t1")
mean(subset(mf,c1 > 42)$c2)
```

We can see how the `monet.frame` object is constructed with a connection and the name of the database table to be wrapped. Now, a prototypical R interaction is performed on the object: First, the `subset` method is used to select a subset of the data based on a comparison. Then, the `$` operator is used to select a single column from the result. Finally, a statistical aggregate is calculated, which is then typically used as the basis of more complex analyses. While no SQL statements are visible to the user, they are still being created in the background. In particular, through overloading `subset`, `$` and `mean`, the query was rewritten over several steps as seen in Listing 2, with only the last query being executed:

Listing 2: Generated SQL Query – Rewriting steps

```
SELECT * FROM t1
SELECT * FROM t1 WHERE (c1>42)
SELECT c2 FROM t1 WHERE (c1>42)
SELECT AVG(c2) FROM t1 WHERE (c1>42)
```

To clarify this translation process, R allows the definition of new classes of objects. Many R-internal operations are defined as so-called *generics*, which means that they will not directly call an implementation, but rather search the space of defined methods for one matching the class name. Per convention, the specific implementation of a generic is named with the generic function name appended with the name of the class. For example, the `length` method for our virtual data object is defined with the name `length.monet.frame` and will be called if a user applies the `length` method to a monet.frame object. In this case, the function returns a simple scalar value. However, there are also methods which return a new `monet.frame` object such as the `$` operator. Here, the wrapped SQL query is changed internally, as seen in Listing 2 (Lines 2 to 3).

Since it is possible for the user to inspect the intermediate result at every step, this allows exploratory usage of data stored in the database from potentially very large tables without big performance penalties. Furthermore, these objects can now also be transparently used by extension packages that provide specialized calculations, without them being required to support database operations explicitly. Often, we can simulate calculations not provided by the database through a combination of other functions. For example, `acosh` can be calculated as $x + \sqrt{x^2 - 1}$, which is supported. However, there are other calculations such as the factorial or gamma functions that have to be executed within R. To support this, the `monet.frame` contains explicit operators that materialize the wrapped database objects into the R environment.

However, overloading operators is only one side of the equation. The next question is in which language the calls should be translated. MonetDB supports multiple front-ends beside SQL, and – as in many database systems – actual query execution is expressed in a procedural language, *MAL*. While we are translating to standard SQL code for now, which also allows our virtual data object to co-operate with databases other than MonetDB, we have the possibilities of either translating direct to MAL with potential additional performance gains, or implement a R-supporting database front-end to the same purpose.

3. USE CASE: SURVEY ANALYSIS

In order tho show the usefulness of the virtual data object and the MonetDB – R connection, we have chosen a use case from survey analysis. The American Community Survey (ACS) [12] is a yearly survey that is performed in all U.S. states. Data is collected from around 3.5 million randomly-selected addresses. Among others, the survey collects information on income, health, education, transportation and housing. The main goal of ACS is to provide authorities with information about their residents. However, it is difficult to provide information about the entire population of the U.S. through observation of only a random fraction of people. To allow statistically valid inferences while still protecting the privacy of the included people, replication-based estimation is used in ACS.

To correctly estimate for example the average age of the U.S. population from the ACS data, one cannot simply average the corresponding column from the data set. Instead, the overall weight for the correct value and the replicate weights for correct standard error estimation have to be considered. To stay in the example, the average age $\overline{age}$ is calculated by using the overall weight column w. To obtain the standard error (SE) for this estimator, one has to repeat this calculation with all the replicate weight columns. In the case of ACS, there are 80 replicate weights w_r.

$$\overline{age} = \frac{\sum (age \times w)}{\sum w} \tag{1}$$

$$SE(\overline{age}) = \sqrt{\frac{4}{80} \sum_{r=1}^{80} (w_r - \overline{age})^2} \tag{2}$$

Performing both calculations on the ACS data requires a considerable part of the data to be included. However, the mathematical operators required on the bulk of these values are only basic algebra and aggregation, which are supported by most relational databases and of course MonetDB. Therefore, we can take advantage of the high degree of optimization in these systems to calculate the desired estimators quickly. Apparently, the average American is 37.1 years old with a standard error of 9.2×10^{-6}.

4. EXPERIMENTAL RESULTS

In this section we present two experiments over three differently sized datasets with three different implementations. The first experiment is based on the simple calculation on a column in the data from Listing 1. The second experiment is the calculation the weighted average of people's ages in ACS and the corresponding standard error from the preceding Section 3. These experiments were repeated for three different scenarios: 1) *Plain R*, where the R-internal CSV parser is used to load the data file, and the calculations are performed on the materialized `data.frame`, 2) *Manual*, where SQL queries are directly sent to the database, and 3) *Virtual*, where the virtual data object is used. The implementation of the survey analysis was realized by the R packages `sqlsurvey` [5] for the manual method, `survey` [6] for the plain R method, and a new implementation using our virtual data object.

Our general hypothesis is that the *Manual* method will be the fastest, since its manually constructed SQL queries will minimize the amount of round-trips to the database. We also assume that the virtual method will be close in performance, since no data is actually being materialized in R until absolutely necessary. Finally, we expect the *Plain R* method to show a very poor performance in comparison, partly due to R's single-threaded design. We have used three different datasets based on the ACS data: 1) *Small*, 142.982 tuples in a 91 Megabyte CSV file representing the data from the state of Alabama, 2) *Medium*, 1.060.060 tuples in a 690 Megabyte CSV file representing the data from California, and 3) *Large*, 9.093.077 tuples in a 6 Gigabyte CSV file with the data from the entire U.S. Both experiments were run on two standard desktop computers, one with a quad-core processor and 16 GB of main memory (*Fast*) and one with a dual-core processor and 2 GB of memory (*Slow*).

System	Exp.	Method	Dataset		
			Small	Medium	Large
Fast	DB Import		1.1 s	35.4 s	702.2 s
	Baseline	Plain R	4.5 s	92.5 s	–
		Manual	0.1 s	0.1 s	0.1 s
		Virtual	0.1 s	0.2 s	0.2 s
	Survey	Plain R	14.2 s	93.7 s	–
		Manual	0.1 s	0.9 s	3.6 s
		Virtual	0.9 s	1.5 s	4.2 s
Slow	DB Import		3.0 s	180.1 s	2520.3 s
	Baseline	Plain R	7.3 s	–	–
		Manual	0.1 s	0.1 s	0.2 s
		Virtual	0.3 s	0.4 s	0.5 s
	Survey	Plain R	8.2 s	–	–
		Manual	0.3 s	1.1 s	124.5 s
		Virtual	14.0 s	15.4 s	80.0 s

Table 1: Experimental Results

The measured completion times of all experiments can be seen in Table 1. For each combination of System, Experiment and Method the times for the three datasets are given. Measurements were aborted (–) if the process did not finish within one hour. In the table, we can see how the *Plain R* method consisting of loading the CSV file from disk into the R process space requires a increasingly long time to complete. Contrary, the *Manual* SQL query method is consistent in its speed. The *Virtual* method using `monet.frame` shows very the expected results as well, a bit more time is spent compared to manual SQL queries, but far less than plain R operations. We can also see how the time it takes to load the data into MonetDB is also considerable. However, this has only to be done once, whereas this is necessary every time R is started for the native R approach. Furthermore, the timings are remarkable considering the fact that MonetDB has not only to read the data from the CSV file, but also creates the persistent database files on disk.

For the baseline experiment, we can see that the virtual data object incurs costs only marginally larger than those shown by manually constructing SQL queries. Furthermore, loading a common CSV file into MonetDB is faster than a one-time load into the R address space, even in a situation where enough physical memory is available. It should be also noted that the *Plain R* method could me more competitive at least on small datasets when slow loading is amortized by a large number of fast calculations. Similar results can be seen in our results for the survey analysis experiments. Here, the performance of the *Native* method was again clearly dominated by the time it took to load the CSV file. For the *Manual* method realized through `sqlsurvey` we can see outstanding performance, with only 3.6 seconds taken to analyze the entire dataset with the help of MonetDB. Comparing the results between the *Manual* and *Virtual* scenario, we only see a increase in time to 4.2 seconds. For our implementation, this delta can then be regarded as the overhead incurred by the convenience of using the virtual data object. Comparing between the two systems the experiments were run on, we can see how the Slow system, where the data does not fit into main memory at all, the Plain R method suffers most, with both the Manual and Virtual methods still being practical. Still, the Slow system shows a much higher overhead for the virtual data object, which could be attributed to a higher number of round trips between R and MonetDB.

5. RELATED WORK

In [4], a method of embedding a R environment inside row-oriented relational database PostgreSQL is presented. Interestingly, they already note the potential issue that arises from applying R's column-based operations to row-based data in the database. In this approach, calls to R code have to be defined as user-defined-functions in the database. This is far from being transparent to the user.

Another embedding method is presented in [1]. They present a method of embedding R into the MySQL database system. However, in addition to that, a CORBA interconnect is used to connect the embedded R environment with the local R environment on the user's workstation. This also allows distributed computations, as the interconnect can be use to attach more than one remote database. Through modifying R itself (in particular the interpreter for its internal programming language), they also achieve transparency of computations and data access. However, this method is rather intrusive and suffers from similar issues as the previous one due to the row-based nature of the underlying database.

The most recent and most related approach to the one presented here is *RIOTDB* [13]. Here a virtual proxy object is also used to allow the user to work with data from a relational database (MySQL) from within R. Each virtual object (there referred to as "strawman") is mapped to a database table or view. Operations such as additions create an additional view. Unfortunately, this approach requires a modification to the R core, rendering it less practical.

6. CONCLUSION AND OUTLOOK

In this paper, we have described our vision and present our solution for a non-disruptive, transparent and compatible way to integrate the R programming environment and the relational column-oriented database system MonetDB. We have argued that a non-intrusive and compatible method to access data in relational databases from within the R environment is very beneficial. We have constructed such a method with the MonetDB-backed virtual data object. From our experiments with the rather large ACS dataset, it has become clear that the performance penalties when compared to hand-written SQL queries can be justified by the increased simplicity of usage. We will continue to expand the capabilities of the virtual data object towards maximal compatibility with existing R packages, without compromising compatibility of our approach with R itself. Modifying the core of this environment is prohibitive in most environments.

For our future work, we would like to consider four main research directions: First, as mentioned, is extending the virtual data object. Second, increasing performance by either providing a R-supporting front-end to MonetDB or translating directly in the internal query execution language. Third, further explore the possibilities of embedding an R execution environment inside a relational database as pioneered in previous work [1, 4]. Fourth, we also would like to embed a MonetDB database kernel into the R environment, again without compromising compatibility. A database kernel in this environment promises zero-copy operations, but more importantly an even smoother user experience, who currently still has to manually install and run MonetDB on her computer.

The described MonetDB–R connector including the virtual data object `monet.frame` is available on CRAN today [7] and is already in use by the statistics community. For example, the package has been used to analyze the Medicare Claims Public Use Files (BSA PUFs) containing 97 Million records. We invite all those interested to try this powerful combination of the best of both worlds on their own datasets and push the envelope on what is practical in data analysis.

Acknowledgments

We would like to thank Martin Kersten, Sjoerd Mullender, Fabian Groffen and Anthony Damico for their various work. Also, we thank the anonymous reviewers for their feedback. H. Mühleisen is supported by the *COMMIT/* project funded by NWO.

7. REFERENCES

[1] F. Chen and B. D. Ripley. Statistical computing and databases: Distributed computing near the data. In *Proceedings of the 3rd International Workshop on Distributed Statistical Computing (DSC 2003)*, 2003.

[2] J. Conway, D. Eddelbuettel, T. Nishiyama, S. K. Prayaga, and N. Tiffin. RPostgreSQL: R interface to the PostgreSQL database system. `http://cran.r-project.org/web/packages/RPostgreSQL/`. checked 2013-02.

[3] S. Idreos, F. Groffen, N. Nes, S. Manegold, K. S. Mullender, and M. L. Kersten. MonetDB: Two decades of research in column-oriented database architectures. *IEEE Data Engineering Bulletin*, 35(1):40–45, 2012.

[4] D. T. Lang. Scenarios for using R within a relational database management system server. `http://www.omegahat.org/RSPostgres/Scenarios.pdf`. checked 2013-02.

[5] T. Lumley. sqlsurvey: SQL-based analysis for large surveys. `http://sqlsurvey.r-forge.r-project.org`. checked 2013-02.

[6] T. Lumley. survey: Analysis of complex survey samples. `http://cran.r-project.org/web/packages/survey/`. checked 2013-02.

[7] H. Mühleisen, T. Lumley, and A. Damico. MonetDB.R. `http://cran.r-project.org/web/packages/MonetDB.R/`. checked 2013-02.

[8] D. Mukhin, D. A. James, and J. Luciani. ROracle: OCI based Oracle database interface for R. `http://cran.r-project.org/web/packages/ROracle/`. checked 2013-02.

[9] R Special Interest Group on Databases. DBI: R Database Interface. `http://cran.r-project.org/web/packages/DBI/`. checked 2013-02.

[10] The R Foundation for Statistical Computing. The Comprehensive R Archive Network. `http://cran.r-project.org`. checked 2013-02.

[11] The R Foundation for Statistical Computing. What is R? `http://www.r-project.org/about.html`. checked 2013-02.

[12] United States Census Bureau. American Community Survey. `http://www.census.gov/acs/www/`. checked 2013-02.

[13] Y. Zhang, H. Herodotou, , and J. Yang. RIOT: I/O-efficient numerical computing without SQL. In *Proceedings of the 2009 Conference on Innovative Data Systems Research*, 2009.