MonetDBLite: An Embedded Analytical Database

Mark Raasveldt
CWI
Amsterdam
m.raasveldt@cwi.nl

Keywords Embedded Databases, Machine Learning, OLAP

ACM Reference format:
Mark Raasveldt. 2018. MonetDBLite: An Embedded Analytical Database. In *Proceedings of 2018 International Conference on Management of Data, Houston, TX, USA, June 10–15, 2018 (SIGMOD'18),* 3 pages.
https://doi.org/10.1145/3183713.3183722

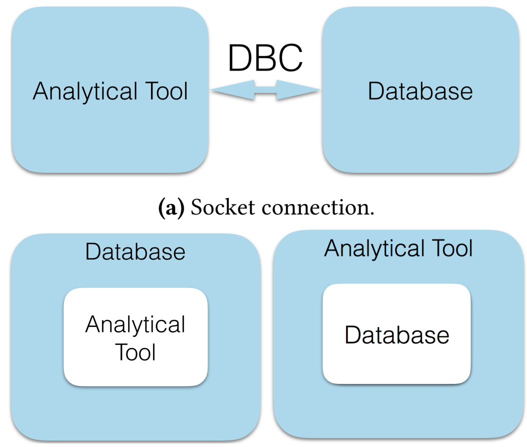
There is a disconnect between data-intensive analytical tools and traditional database management systems. Data scientists using these tools often prefer to manually manage their data by storing it either as structured text (such as CSV or XML files), or as binary files [5]. This approach of managing data introduces a lot of problems, especially when a large amount of data from different sources has to be managed. Flat file storage requires tremendous manual effort to maintain, and is often difficult to reason about because of the lack of a rigid schema. Furthermore, the data is prone to corruption because of lack of transactional guarantees and atomic write actions.

Another consequence of this disconnect is that data scientists have re-implemented many common database operations inside popular scripting languages rather than using a database to perform these actions. Libraries such as dplyr [10] and Pandas [6] re-implement most standard database operations, such as joins and aggregations. However, these libraries suffer from having to load all required data and intermediates into memory. This leads to frequent out of memory problems or poor performance due to swapping.

All these issues could be solved by combining an efficient analytical RDBMS with these tools. The RDBMS can prevent data corruption through ACID properties, it can automatically manage data storage for the user and make data easier to reason about by enforcing a rigid schema. In addition, the RDBMS can perform efficient execution on larger-than-memory data by only loading required columns.

SIGMOD'18, June 10–15, 2018, Houston, TX, USA

ACM ISBN 978-1-4503-4703-7/18/06.
https://doi.org/10.1145/3183713.3183722



(a) Socket connection.

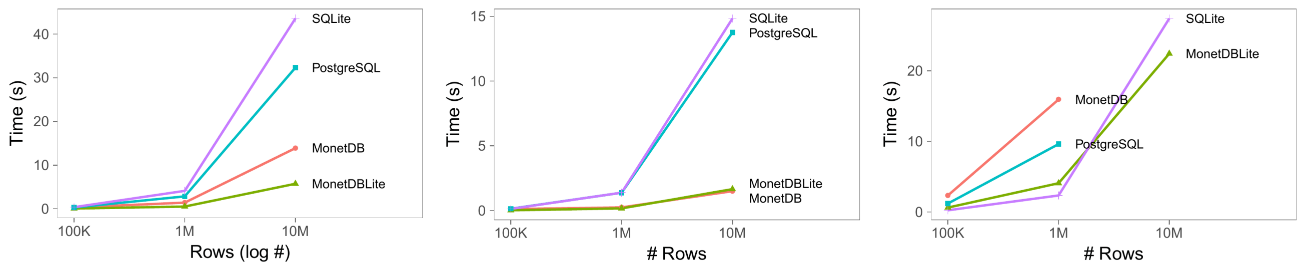
(b) In-database processing. **(c)** Embedded database.

Figure 1. Different ways of connecting analytical tools with a database management system.

However, the current methods of using standard RDBMSes in conjunction with analytical tools are lacking. The standard approach is to run the database as a separate process (the "database server") and connecting the analytical tool with it through a socket connection (as a "database client"). The analytical tool can then issue queries to the database, after which the server will transfer the query results to the client through the socket. This approach has several issues. Firstly, maintaining a database server requires significant manual effort from the user. The database server must be installed, tuned and continuously maintained. Secondly, communicating with a database through a socket connection falls short when a large amount of data is involved. The data has to be transferred to and from the analytical tool through a socket connection, which is inefficient in current major database systems [8], even when the database server and the analytical tool reside on the same machine.

Additionally, writers of scripts in analytical languages such as R or Python prefer writing portable scripts that they can share with other data scientists. Scripts containing references to external tools such as database management systems are challenging to port to other systems, and as such cannot be included in these scripts.

An alternative solution is to use in-database processing methods [7]. By executing the analysis pipelines inside the database, the overhead of data export can be avoided. While



(a) Transfer data from database to client. **(b)** Run TPCH-Q1 inside the database. **(c)** Move data from the client to the database.

Figure 2. Experimental results.

this approach removes the data transfer overhead between the scripting language and the database, it still requires the user to run and manage a separate database server. These user-defined functions also introduce new issues. They force users to rewrite code so the code fits within the query workflow, are difficult to debug [3] and introduce safety issues as arbitrary code can now run within the database kernel.

Another solution is to embed the database directly into the scripting language. As the database lives in the same address space as the scripting language, data can be transferred between the two systems without any overhead. Embedded databases are popular, mainly because of the omnipresent SQLite [2]. However, SQLite is designed for OLTP workloads. While popular analytical tools do have SQLite bindings, it does not perform well when used for analytical purposes.

In this work, we introduce MonetDBLite[1], an Open-Source embedded database based on the popular columnar database MonetDB [4]. It is an in-process analytical database that can be run directly from within popular analytical tools without any external dependencies. It can be installed through the default package managers of popular analytical tools, and has bindings for C/C++, R, Python and Java. In addition, because of its in-process nature, data can be transferred between the database and these analytical tools at zero cost.

Evaluation. In order to test the effectiveness of our system we compare it against current solutions for combining analytical tools with database systems in three different important areas:

1. Transfer of data from the database to the client process.
2. Execution of analytical queries within the database.
3. Transfer of data from the client process to the database.

The systems used for comparison are (1) the databases PostgreSQL [9] and MonetDB [4] connected through a client connector, and (2) SQLite [2] running embedded inside the client process. The benchmarks were run using an R shell as the client process, and were run on a machine running Fedora 26 with an Intel i7-2600K with 8 Cores running at 3.4 GHz and 16GB of Main Memory.

[1]The source code of MonetDBLite is available here: https://github.com/hannesmuehleisen/MonetDBLite

Transfer To Client. In this benchmark, we transfer the `lineitem` table from the TPC-H benchmark [1] from the database to the client process.

In Figure 2a, the transfer time from the database to the client process is shown. We can see that MonetDBLite performs an order of magnitude better than the competing systems. This because it both runs inside the client process, meaning data does not have to be transferred over a socket, and data is stored in columnar format much like it is inside the scripting languages. This allows for fast transfer of data.

MonetDB shows good performance on this benchmark compared to the other databases. This is because MonetDB uses a client protocol optimized for bulk transfer of data in columnar format [8]. Meanwhile, both SQLite and PostgreSQL are doing poorly because they have to convert from a row-based to a columnar format.

Execution of analytical queries. In this benchmark, we run Q1 of the TPC-H benchmark inside the database server and transfer the result to the client.

In Figure 2b, the execution time of TPC-H Q1 within the database is shown. This query has a small result set, hence transfer time from the database to the client is not a bottleneck. Because of that MonetDBLite and MonetDB have identical performance. We can see that PostgreSQL and SQLite perform significantly worse than MonetDB. This is because they are row-store databases designed for OLTP workloads.

Transfer To Database. In this benchmark, we again transfer the `lineitem` table, but this time from the client to the server and store it persistently within the database.

In Figure 2c, the results of this benchmark are shown. We can see that both MonetDB and PostgreSQL perform very poorly here. This is because the data is transferred over a socket connection and individual rows are transferred using `INSERT INTO` statements, which then have to be parsed back into binary data. Both SQLite and MonetDBLite perform much better on this benchmark, and show very similar performance. The main bottleneck for these systems is writing the data to disk.

References

[1] 2013. *TPC Benchmark H (Decision Support) Standard Specification.* Technical Report. Transaction Processing Performance Council. http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v2.17.1.pdf

[2] Grant Allen and Mike Owens. 2010. *The Definitive Guide to SQLite* (2nd ed.). Apress, Berkely, CA, USA.

[3] Pedro Holanda, Mark Raasveldt, and Martin Kersten. 2017. Don't Hold My UDFs Hostage - Exporting UDFs For Debugging Purposes. In *Proceedings of the 28th International Conference on Simpósio Brasileiro de Banco de Dados, SSBD 2017, UberlÃćndia, Brazil.*

[4] Stratos Idreos, Fabian Groffen, Niels Nes, Stefan Manegold, Sjoerd Mullender, and Martin Kersten. [n. d.]. MonetDB: Two decades of research in column-oriented database architectures. *IEEE Data Eng. Bull* ([n. d.]), 2012.

[5] Sean Kandel, Andreas Paepcke, Joseph M. Hellerstein, and Jeffrey Heer. 2012. Enterprise Data Analysis and Visualization: An Interview Study. *IEEE Transactions on Visualization and Computer Graphics* 18, 12 (Dec. 2012), 2917–2926. https://doi.org/10.1109/TVCG.2012.219

[6] Wes McKinney. 2010. Data Structures for Statistical Computing in Python. In *Proceedings of the 9th Python in Science Conference*, Stéfan van der Walt and Jarrod Millman (Eds.). 51 – 56.

[7] Mark Raasveldt and Hannes Mühleisen. 2016. Vectorized UDFs in Column-Stores. In *Proceedings of the 28th International Conference on Scientific and Statistical Database Management, SSDBM 2016, Budapest, Hungary, July 18-20, 2016.* 16:1–16:12.

[8] Mark Raasveldt and Hannes Mühleisen. 2017. Don't Hold My Data Hostage: A Case for Client Protocol Redesign. *Proc. VLDB Endow.* 10, 10 (June 2017), 1022–1033. https://doi.org/10.14778/3115404.3115408

[9] Michael Stonebraker and Greg Kemnitz. 1991. The POSTGRES Next Generation Database Management System. *Commun. ACM* 34, 10 (Oct. 1991), 78–92. https://doi.org/10.1145/125223.125262

[10] Hadley Wickham. 2017. Package 'dplyr': A Grammar of Data Manipulation. (2017). https://cran.r-project.org/web/packages/dplyr/dplyr.pdf