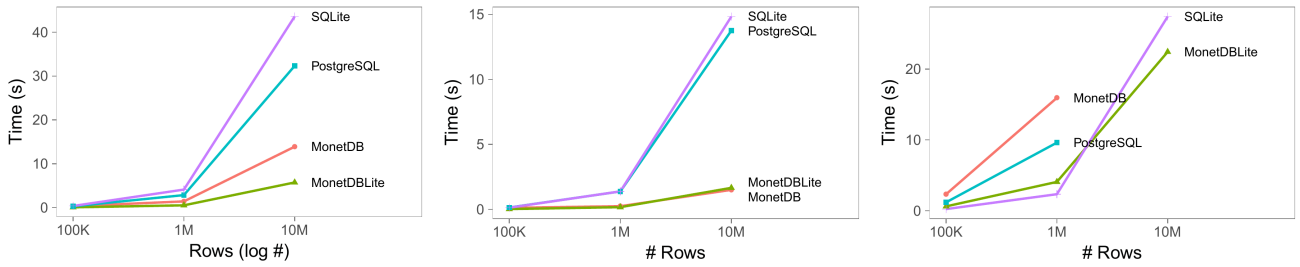

(a) Transfer data from database to client. **(b) Run TPCH-Q1 inside the database.** **(c) Move data from the client to the database.**

Figure 2: Experimental results.

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 has bindings for C/C++, R, Python and Java, and can be installed through their default package managers. 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.

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.

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

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. https://cran.r-project.org/web/packages/dplyr/dplyr.pdf