While for very small data sets all systems will show comparable behavior, only DuckDB will be able to continue functioning for larger ones. SQLite will begin to suffer from its row-based execution model and MonetDBLite begins to suffer from excessive intermediate result materialization due to its bulk processing model. While HyPer is extremely fast in processing queries, it will not be able to transfer result sets as quickly as DuckDB using its socket client protocol [12].

For the "drilldown" scenario, we invite the audience to propose their own query to be configured into the benchmark computers. This will allow direct appraisal of DuckDB's performance, without the demonstration authors being able to cherry-pick queries where DuckDB excels. Again, the audience member that has proposed the query will then be able to turn the dial to increase the amount of data read by the query and observe the impact on the four systems in real-time.

4 CURRENT STATE AND NEXT STEPS

As of this writing, DuckDB runs all TPC-H queries and all but two TPC-DS queries. We expect complete TPC-DS coverage by the time the demonstration is presented. DuckDB also already completes most of SQLite's SQL logic test suite that contains millions of queries.

Immediate next steps for DuckDB are completion of DataBlocks storage scheme and cardinality estimating. A buffer manager is also not yet implemented, but will be. DuckDB already supports inter-query parallelism but intra-query parallelism will be added as well. We plan to implement a work stealing scheduler to balance resources between short and long running queries. A special consideration is also to allow balancing resource usage with the host application, a special issue for embedded operations.

A more advanced future direction is self-checking. We have learned to distrust the hardware the database is running on. This is particularly relevant in the edge computing use case, where hardware failures are to be commonplace. One approach is to keep checksums on all persistent and intermediate data and piggy-back checksum verification on scan operators. This might be possible without a significant performance impact. A vectorized engine is particularly suited for this since a chunk of data typically fits in the CPU cache and additional passes are not requiring RAM access. Another approach to increasing trust in the hardware is inspired by video game developers which periodically run sanity check computation to ensure correct operation of CPU and RAM.

Acknowledgements

We would like to thank all past, current and future contributors to DuckDB at the CWI Database Architectures Group and elsewhere. We are also particularly indebted to the TUM database group for their papers on query optimization, window functions, storage and concurrency control that we used to implement DuckDB.

REFERENCES

[1] Peter A. Boncz, Marcin Zukowski, and Niels Nes. 2005. MonetDB/X100: Hyper-Pipelining Query Execution. In *CIDR 2005, Second Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2005*. 225–237. http://cidrdb.org/cidr2005/papers/P19.pdf

[2] Lukas Fittl. 2019. C library for accessing the PostgreSQL parser outside of the server environment. https://github.com//fittl/libpg_query.

[3] Richard Hipp. 2019. Database File Format. https://www.sqlite.org/fileformat.html.

[4] Richard Hipp. 2019. Most Widely Deployed and Used Database Engine. https://www.sqlite.org/mostdeployed.html.

[5] Harald Lang, Tobias Mühlbauer, Florian Funke, et al. 2016. Data Blocks: Hybrid OLTP and OLAP on Compressed Storage using both Vectorization and Compilation. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*. 311–326. https://doi.org/10.1145/2882903.2882925

[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] Guido Moerkotte and Thomas Neumann. 2008. Dynamic programming strikes back. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*. 539–552. https://doi.org/10.1145/1376616.1376672

[8] Thomas Neumann. 2011. Efficiently Compiling Efficient Query Plans for Modern Hardware. *PVLDB* 4, 9 (2011), 539–550. https://doi.org/10.14778/2002938.2002940

[9] Thomas Neumann and Alfons Kemper. 2015. Unnesting Arbitrary Queries. In *Datenbanksysteme für Business, Technologie und Web (BTW), 16. Fachtagung des GI-Fachbereichs "Datenbanken und Informationssysteme" (DBIS), 4.-6.3.2015 in Hamburg, Germany. Proceedings*. 383–402. https://dl.gi.de/20.500.12116/2418

[10] Thomas Neumann, Tobias Mühlbauer, and Alfons Kemper. 2015. Fast Serializable Multi-Version Concurrency Control for Main-Memory Database Systems. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*. 677–689. https://doi.org/10.1145/2723372.2749436

[11] Thomas Neumann and Bernhard Radke. 2018. Adaptive Optimization of Very Large Join Queries. In *Proceedings of the 2018 International Conference on Management of Data (SIGMOD '18)*. ACM, New York, NY, USA, 677–692. https://doi.org/10.1145/3183713.3183733

[12] Mark Raasveldt and Hannes Mühleisen. 2017. Don't Hold My Data Hostage - A Case For Client Protocol Redesign. *PVLDB* 10, 10 (2017), 1022–1033. https://doi.org/10.14778/3115404.3115408

[13] Mark Raasveldt and Hannes Mühleisen. 2018. MonetDBLite: An Embedded Analytical Database. *CoRR* abs/1805.08520 (2018). arXiv:1805.08520 http://arxiv.org/abs/1805.08520

[14] Hadley Wickham, Romain François, Lionel Henry, and Kirill Müller. 2018. *dplyr: A Grammar of Data Manipulation*. https://CRAN.R-project.org/package=dplyr R package version 0.7.8.