Designing Query Optimizers for Big Data Problems of The Future

Nga Tran, Sreenath Bodagala, Jaimin Dave
HP/Vertica, Cambridge, MA
{ntran@vertica.com, sbodagala@vertica.com, jdave@Vertica.com}

ABSTRACT

The Vertica SQL Query Optimizer was written from the ground up for the Vertica Analytic Database. Its design, and the tradeoffs we encountered during implementation, support the case that the full power of novel database systems can be realized only with a custom Query Optimizer, carefully crafted exclusively for the system in which it operates.

1. INTRODUCTION

The Vertica Analytic Database (Vertica) [3] is a modern, commercially successful RDBMS. It contains a SQL query optimizer, written from scratch, especially for the Vertica Storage System and Execution Engine. We wrote our own optimizer, despite a countervailing industry trend to reuse or wrap existing optimizers [1, 4] in new database systems.

The choice to write an optimizer from scratch was not taken lightly and it delayed our product's initial introduction to the market. However, as the Vertica founders envisioned, writing a custom optimizer was the only sure way to take full advantage of Vertica's columnar storage system and distributed execution engine. We believe our experience and success in the marketplace validates our decision. Furthermore, the choice to make our optimizer a set of extensible modules makes it possible to quickly extend the optimizer to address newer problems. We evaluate this premise within our Design History, described next.

2. DESIGN HISTORY

Vertica founders knew that their optimizer would need to be aware of data compression and its effects on the CPU and I/O cost of database operations. With this and other requirement in mind, they listed the major optimizer decisions as 1) which projections to use for a given query, 2) how to prune the plan search space to a feasible size, 3) at what points in the plan they should materialize columns, and 4) the optimizer needs to be data distribution aware. The Vertica Query Optimizer addresses all of these decisions and many more, but the final design did not fully emerge on our first attempt. The Query Optimizer used in Vertica today is actually our third version, and the history of how we arrived at the current design highlights some the challenges faced by industrial optimizer implementers.

1. **StarOpt**. The initial Vertica Optimizer was a Kimball-style query optimizer[2], which assumed any interesting customer schema and query could be modeled as a star or snowflake. When this design assumption met real-world problems, it was clear that most customer data did not exactly conform to the star ideal. For a variety of technical and non-technical reasons we couldn't expect customers to change their schemas. Instead, we needed a different optimizer with a different set of assumptions. StarOpt also applied distribution and sort-algorithm decisions **after** the join order was chosen, significantly affecting the optimizer's ability to minimize data movement for complex plans running against complicated distributed schemas. As we learned, an industrial optimizer must handle all the vagaries of messy schemas and data, however nonsensical they may seem to the implementors.

2. **StarifiedOpt**. The second generation optimizer was a modification of StarOpt. Internally, this optimizer forced non-star queries to look like a star, solely for the purpose of applying existing StarOpt algorithms. This approach was far more effective for non-star queries than we could have reasonably hoped, and it bought us sufficient time to design and implement the third generation optimizer: the custom built, though poorly named, V2Opt.

3. **V2Opt**. This optmizer is completely aware of distribution, sortedness, and non-star schemas in all its decisions. It has been the default Vertica Optimizer since version Vertica 4.0 and has served us well at thousands of customer installations and in planning hundreds of millions of queries.

2.1 Why Modular Optimizer?

After developing two generations of optimizers, StarOpt and StarifiedOpt, two major goals existed for the next generation (1) adding all of the features missing in the first two optimizers, and (2) designing V2Opt as fully extensible for ongoing improvements and future Big Data requirements. Developed as a set of extensible modules, V2Opt lets us change key elements of the optimizer, without rewriting lots

Proceedings of the VLDB Endowment, Vol. 6, No. 11

of code. In fact, V2Opts inherently-extensible design has already enabled us to incorporate knowledge gleaned from our end-user experiences, without significant engineering efforts.

V2Opt plans a query by categorizing and classifying the physical properties associated with the query, such as column selectivity, projection column sort order, projection data segmentation, prejoin projection availability, and integrity constraint availability. The degree of importance for each physical property is measured in different ways through experiments on the customers' real-life data sets and queries. These physical property heuristics are combined with a cost-model pruning strategy, based on compression-aware I/O, CPU and Network transfer costs. Combined, all of this information becomes instrumental in helping the optimizer (1) control the explosion in search space, while continuing to explore optimal plans and (2) account for data distribution and bushy plans during the join order enumeration phase.

With an easily extensible design, whenever a new physical property becomes prominent and affects the choice of selecting optimal query plans, we can swiftly add a new ranking module to V2Opt without changing the entire optimizer. Moreover, living in the era of Big Data, the effects of physical properties inevitably change. For example, data segmentation becomes more important as customers increase their database cluster size. In that case, the modular optimizer design will let us easily adjust the degree of importance given both segmentation and cluster size.

3. CONCLUSION

Even though the success of Vertica in the marketplace has validated the accuracy of selecting optimal query plans with the Vertica Query Optimizer, we continue to make ongoing improvements. With its modular design, our engineering team can quickly adjust or modify the optimizer to provide customers the power to generate optimal query plans in the explosion of Big Data.

4. ACKNOWLEDGMENTS

Special thanks to Kanti Mann who has helped us edit this paper.

5. REFERENCES

[1] Y. Chen, R. L. Cole, W. J. McKenna, S. Perfilov, A. Sinha, and E. Szedenits, Jr. Partial Join Order Optimization in the Paraccel Analytic Database. In *SIGMOD*, 2009.

[2] R. Kimball and M. Ross. *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*. Wiley, John & Sons, Inc., 2002.

[3] A. Lamb, M. Fuller, R. Varadarajan, N. Tran, B. Vandiver, L. Doshi, and C. Bear. The Vertica Analytic Database: C-store 7 Years Later. In *VLDB*, 2012.

[4] S. Shankar, R. Nehme, J. Aguilar-Saborit, A. Chung, M. Elhemali, A. Halverson, E. Robinson, M. S. Subramanian, D. DeWitt, and C. Galindo-Legaria. Query Optimization in Microsoft SQL Server PDW. In *SIGMOD*. ACM, 2012.