Towards Creating Application-Specific Database Management Systems

Alvin Cheung
University of Washington

1. INTRODUCTION

Specialization is an effective way to improve performance. That principle has manifested time and again in database management systems (DBMSs), with custom data storage engines such as column stores and streaming engines outperforming traditional DBMSs in specific application and architecture domains. Unfortunately, designing custom data storage engines is a tedious task. Developers need to first analyze numerous applications and their hosting platforms to identify common patterns in manipulating persistent data. This is followed by a painstaking and trial-and-error process of devising efficient implementations for the data operations on the target platform. We argue that advances in program analysis and software synthesis can help in automating this design process. In the following we describe opportunities where understanding the application can improve functionalities provided by the DBMS.

2. RESEARCH OPPORTUNITIES

Contextual query processing. Having accurate workload information is essential to efficient query processing. As many queries are now programmatically generated by applications rather than entered free-form by users, analyzing the application will give us insights about the issued queries. For example, a medical record application issues many queries to render a patient dashboard. Such programmatically-issued queries have highly deterministic structure, for instance query parameters are derived from program variables. Analyzing the application source allows us to create *application contexts*. Each context represents a program path in the code and describes the queries that are issued. It also includes the application code that manipulates the query results, the relationship among the query parameters, the frequencies of each query (which can be approximated from the code or query logs), and the actions that trigger each query such as a button click.

Application contexts are useful in many scenarios. First, the DBMS can use application contexts to combine queries and prefetch a subset of the results. Moreover, the correlations among the query parameters allow the DBMS to improve the generated query plans. For instance, if the context indicates that the same program variable is passed into two different queries as parameters, then once the value of the program variable is known, the DBMS can estimate the selectivity of both queries accurately. In addition, the DBMS can make use of context information to determine when to evict query results from the buffer pool when they are no longer needed.

Infer functional dependencies. Besides optimizing read queries, application contexts can be used to learn functional dependencies. For instance, after a user puts in a city name in a form, the application issues a query to fetch the given city's current population from the DBMS. After that, the application computes a prediction of the city's future population, shows it to the user for validation, and stores the (city name, predicted population) pair in another table. There is an obvious correlation between the current and predicted populations stored in the DBMS. Discovering such correlation from the logs is difficult as it involves application code, and the log might include queries issued by multiple concurrent users. We can instead use the application context for this purpose. For instance, in this case the context will inform us that predicted populations are derived from another table that stores the current values.

In general, we can use contexts to learn rich functional dependencies of the form $X \rightarrow f(Y)$, where some part of f might be implemented in the application code. Such dependencies are very difficult to infer using statistical techniques, especially those that involve continuous data values, such as converting between different measurement units, and computing interest rate given credit score. One idea is to use program synthesis [1] to derive a succinct representation of f from the source code. Once f is derived, we can retrieve statistics about X from the DBMS and propagate through f to estimate the distribution of Y.

Improve query optimization. Classical query optimization focuses on finding efficient implementations of each operator in the query tree. Given application contexts, we can treat each contextual query as a user defined function (UDF) that includes both queries and application code that processes the results. We can then specialize the optimizer to find efficient implementations of each UDF given the physical design.

This insight raises new research opportunities. Since UDFs are traditionally treated as black boxes in query optimization, it will be interesting to optimize UDFs using classical program optimization techniques [2]. One challenge is to extend such techniques to be I/O cost-aware in addition to the traditional goal of minimizing the number of instructions, and consider implementing DBMS functionalities in hardware functional units such as flash controllers and FPGAs. Addressing these challenges will advance both data management and programming systems research.

3. CONCLUSION

Rather than manually re-designing DBMSs for new architecture or application domains, we should create tools that can automatically tune DBMS implementations to make them application aware. In this abstract we illustrated different aspects of DBMS functionalities that can be tuned using application information, and described new research challenges in achieving that vision.

4. REFERENCES

[1] R. Bodík et al. Algorithmic program synthesis: introduction. *International Journal on Software Tools for Technology Transfer*, 15(5-6):397–411, 2013.

[2] H. Massalin. Superoptimizer: A look at the smallest program. In *Proc. ASPLOS*, pages 122–126, 1987.

7th Biennial Conference on Innovative Data Systems Research (CIDR '15) January 4-7, 2015, Asilomar, California, USA.