The DBToaster White Paper

The DBToaster Consortium
c/o EPFL IC DATA, Station 14, CH-1015 Lausanne, Switzerland
dbtoaster@epfl.ch

What is DBToaster?

DBToaster constitutes a quantum leap in database technology, achieving seemingly impossible performance figures.

DBToaster is a compiler for database queries. The DBToaster compiler turns SQL queries into highly efficient C++ or Scala code that can be run stand-alone, without a need for DBToaster or a database system. It generates self-contained main-memory query engines that can be easily embedded into user applications.

At the heart of the DBToaster compiler is a transform that turns SQL queries into code that performs an extremely aggressive form of incremental view maintenance.

Code generated from SQL queries by DBToaster is usually by 1000 to 10000 times faster at keeping SQL views fresh than state-of-the-art database and data stream processing systems.

DBToaster provides a unique blend of the advantages of data stream/complex event processing (CEP) engines and relational database systems. Like stream engines, DBToaster supports continuous queries: It keeps a fresh query result available at all times. Like relational database systems, DBToaster provides rich support for analytical queries using the SQL language. DBToaster does not suffer from the semantic limitations of stream engines: it does not impose window semantics. At the same time it does not suffer from the high query latencies of relational database systems. High volume data streams can be seamlessly and efficiently combined with historical data.

DBToaster was designed with applications such as (database) monitoring and low-latency trading in mind. These are domains in which analytical queries arise but where window semantics is just not acceptable: Take, for example, algorithmic trading with order book data. An order book is a table of orders waiting to be executed on an exchange. This table is updated very frequently; however some orders persist for a relatively long time span before they are executed or removed. There is a need for relatively sophisticated analytics, often involving relatively large volumes of further data (historical financial data, sentiments from the internet, etc), but the analyses are only useful if they can be obtained with extremely low latency to enable effective buying and selling decisions.

DBToaster is the first data management system to support such applications. It allows to express analytics in a declarative language such as SQL, greatly improving programmer productivity compared to the state-of-the-art, which is to implement such analyses in a high-level programming language to provide sufficient speed.

How well does it perform?

DBToaster has been experimentally evaluated in a peer-reviewed publication to be presented at the VLDB Conference in 2012:

DBToaster: Higher-order Delta Processing for Dynamic, Frequently Fresh Views.
Yanif Ahmad, Oliver Kennedy, Christoph Koch, and Milos Nikolic.
Proc. VLDB 2012, Istanbul, Turkey.
http://www.dbtoaster.org/papers/pvldb2012-dbtoaster.pdf

The benchmark we used consists of analytical SQL queries on financial (order book) and TPC-H datasets. Our results show that for our benchmark, DBToaster outperforms state-of-the-art relational DBMS and data stream processing engines by 4 orders of magnitude or more. That means, the latency and update rates of views are frequently improved by a factor of 10000 and more over state-of-the-art systems. For details on the benchmark, experimental setup, and results, see the paper.

We do not claim that DBToaster is the right solution for everyone, but you should check out DBToaster if you need to

* maintain materialized views of complex SQL queries,
* read these views and care about very high refresh rates / low refresh latencies,
* work with standing (aka continuous) rather than ad-hoc queries, i.e. you want to monitor the changing result of a given query over time, as the data changes, and
* do not work with extremely large datasets (this is a temporary restriction until we release our parallel/secondary storage runtimes).

DBToaster may be also right for you even if you do not care so much about low view refresh latencies: DBToaster turns a set of queries into efficient specialized code for processing just these queries. DBToaster generates code that you can link into your applications. No further software (such as a separate database server or CEP engine) is required. Thus DBToaster is a very lightweight way of including fixed (parameterized) SQL queries in your applications.

How does it work?

In a nutshell, what DBToaster does is express SQL queries in an internal calculus that allows to compute -- in the same sense as for mathematical functions -- the (discrete) derivative of any query, even the derivatives of derivatives. A derivative captures how a query result changes when the database is updated. DBToaster essentially maintains a query via the sequence of all its derivatives (that is, the derivative, the derivative of the derivative, the derivative of the derivative of the derivative, and so on). Using these, a materialized view can be kept fresh under database updates by summing up derivative function values, with no need for classical query operations such as joins.

The details of the DBToaster technology are not easy to explain, and did require groundbreaking progress in the theory of databases and query algebras as well as in query optimization and compiler technology.

The DBToaster technique performs provably better at keeping materialized views fresh than any previous technique. It even performs asymptotically better: the improvement is not by a constant factor, but grows even on a relative scale as one considers larger and larger data. This was shown in the paper

Christoph Koch: Incremental query evaluation in a ring of databases. Proc. PODS 2010: 87-98.
http://www.dbtoaster.org/papers/pods2010-ring.pdf

(Beware that this is a paper written for researchers in theoretical computer science.)

Features

Speed: See above, **how well does it perform?**

Materialized views of nested queries: DBToaster supports efficient materialized views of nested SQL queries. Many commercial database systems support materialized views / incremental view maintenance, but no other system does so for nested SQL queries, even though they are essential for complex analytics. Nesting refers to the presence of select-statements (SQL queries) in the SELECT, FROM, or WHERE clauses of SQL queries.

C++ and Scala code generation: DBToaster is able to generate both C++ an Scala code that can be integrated into applications written in these languages (as well as Java, since Scala lives in the Java ecosystem and compiles to Java Bytecode that can be linked with Java applications).

License

Benchmarking and experimentation

You are invited to perform and report on benchmark results and experimental comparisons of your system against DBToaster, but we ask you for fairness and to run DBToaster with settings that allow it to perform at its best. Specifically, the DBToaster interpreter mode is provided for your convenience and is not to be used for timing purposes. For the purpose of comparison with other data management systems, please use the C++ rather than the Scala backend. Please contact us if you have questions about how to optimize the performance of DBToaster.

Can I use DBToaster right away in my commercial/production environment?

No. See licensing above. You need to contact us if you want to use DBToaster for anything other than evaluation, research, and learning purposes.

Furthermore, while we are excited about the performance of DBToaster, it is still a project in an early stage, and it may not be ready yet for your production purposes.

However, the license allows you to find out whether DBToaster is relevant to you. We would love to hear from you what features you would need to make DBToaster useful to you.

Continuity and long-term plans of the project

DBToaster is a research project and not a commercial venture. However:

* We consider DBToaster a long-term project that is still in an early phase. Expect follow-up releases that will substantially improve its features, quality, and performance.

* If there is considerable interest in deploying DBToaster, we plan to create a support organization, either by open-sourcing it, by commercializing it, or both.

* Despite its character as a research project, we do have the resources to create high-quality and ultimately production-quality software. We intend to provide support in the form of fixing reported bugs, (within reasonable limits) respond to questions on the use of DBToaster, and implement requested features if they are consistent with our project vision. We are committed to furthering knowledge and making our results publicly available with all the resources currently available to us: this is not in conflict with making our work relevant to as many users as possible.

* The current release, DBToaster Beta1, contains features and APIs that are not considered stable. We do not provide any guarantee of backwards compatibility of applications you build using the current DBToaster APIs. Our APIs will stabilize, and we will make such commitments in the future. Starting with the release of DBToaster1 (expected in the fall of 2012), we plan to provide a matrix of stable vs. experimental features.

For researchers

The principles underlying DBToaster are presented in a number of technical papers available at

http://www.dbtoaster.org/index.php?page=research

Currently we do not give away the source code of the DBToaster compiler (the runtime system has been open-sourced already).

We provide a binary of the DBToaster compiler plus a source code distribution of the runtime libraries (which are kept simple to facilitate integration with various applications).

However, researchers can obtain, upon request, additional resources for reproducing our experiments and comparing DBToaster to other systems.

Current limitations

Limited SQL support: Currently DBToaster does not support all of SQL. Features not supported include:

* NULL values and outer joins.
* The ORDER BY clause.
* The aggregates MIN and MAX. These can be expressed using nested SUM aggregates, but this is not a particularly efficient solution. (All SQL aggregate functions will be supported soon.)
* A number of functions for manipulating strings and dates.

We are working on providing more complete support of the SQL standard.

Parallelization and scalability: The runtime system we currently provide is single-core and maintains data in main memory. Parallelizing and optionally secondary-storage runtimes are under development and will be released soon. Furthermore, a PL-SQL code generator backend for DBToaster is under development that will allow tight integration with relational DBMS such as Oracle or Postgres.

Code generators and runtime system: DBToaster excels at producing efficient query evaluation code. Still, there is much left to be done, and when we look at the code we generate, we spot plenty of opportunities, many very obvious, to improve code and data structure generation. Thus, many opportunities of further improvement at left, and we expect to improve the performance of generated C++ code by another 100x on the same hardware within the near future. One particular untapped opportunity is the direct generation of C data structures rather than the current use of Boost.

The DBToaster roadmap

Milestone 1: (Sept 2012)

* Implement the remaining missing SQL query features to support TPC-H, minus ORDER-BY, MIN/MAX, and NULL values/outerjoins.
* Further performance improvements.

Milestone 2: (Dec 2012?)

* A parallel runtime for DBToaster.

Milestone 3: (June 2013?)

* Rewrite the backend optimizer (code fusion, beta-reduction, etc.) and the C++ code generator. Generate custom data structures and maintenance. Phase out Boost. This should lead to considerable efficiency improvements.
* Synthesize (tree) data structures for efficiently processing and indexing theta-joins.
* Support order. ORDER-BY, MIN, and MAX.

Milestone 4: (Sept. 2013?)

* Frontends for APL-style array processing languages (The R analytics language; Matlab; Q). That is, DBToaster will be able to compile analytical queries expressed in such languages in addition to SQL, with similar performance.