The DBO Database System (598)

Florin Rusu, Fei Xu, Ravi Jampani, Luis Perez, Mingxi Wu, Chris Jermaine, Alin Dobra
University of Florida, Gainesville, FL, USA

ABSTRACT

We demonstrate our prototype of the DBO database system. DBO is designed to facilitate scalable analytic processing over large data archives. DBO's analytic processing performance is competitive with other database systems; however, unlike any other existing research or industrial system, DBO maintains a statistically meaningful guess to the final answer to a query from start to finish during query processing. This guess may be quite accurate after only a few seconds or minutes, while answering a query exactly may take hours. This can result in significant savings in both user and computer time, since a user can abort a query as soon as he or she is happy with the guess' accuracy.

1. INTRODUCTION

Data warehousing and analytic processing have been active areas of database research and development for nearly two decades, and many experts now consider relational query processing in these domains to be solved. However, an argument can be made that users and databases have simply reached an uneasy truce with regard to analytic processing. If users avoid ad-hoc, exploratory queries that might take days to execute, then the database performs just fine.

The limitations of the state of the art are evident if one carefully examines the latest published TPC-H benchmark results. One recent result describes a system that achieves interactive-speed query processing on a 100GB database only by applying an incredible amount of hardware to the problem. To process 100GB of data – which could be stored on a single hard disk – the system requires *eighty four* 36-GB disks and 128GB of RAM. The benchmark also demonstrates that interactive speeds over larger databases are simply impossible using existing technology. For example, another result describes a system that stores a 30TB database using more than $7 million in hardware, yet still requires 14 hours to complete query 18 during the TPC-H throughput test!

In this demonstration, we will exhibit a prototype system being developed at the University of Florida called *DBO*[1]. DBO is designed to facilitate interactive analytic processing over the largest databases. Like traditional relational database systems, DBO can produce exact answers in a scalable fashion. However, unlike any existing research or production system, DBO uses randomized algorithms to produce a statistically meaningful guess for the final query answer at all times throughout query execution. The guess is always bounded by approximation guarantees, such as "With a probability of 95%, the true answer is between 12.2 and 13.6." As more data are processed, the guess is made more and more accurate. A user can stop execution whenever satisfied with the guess' accuracy, which may translate to dramatic time savings during exploratory processing, or else he/she can run the query to completion. In this way, DBO can render interactive, ad-hoc analytic query processing a reality, even over the largest databases.

Building DBO has posed two sets of fundamental challenges: one set related to the nearly total database system redesign that is required and second set related to the resulting statistical issues. Classic database system design cannot support the randomized algorithms required by DBO because traditionally, relational operations run in isolation, and do not share internal state. The statistical challenges presented by DBO are significant because of of the difficulty of analyzing the effect of DBO's randomized algorithms on joins, set subtractions, and other operations.

2. CAN'T OTHER SYSTEMS DO THIS?

While previous systems (such as Bell Labs' AQUA system [1]) have been proposed in the data management research literature that attempt to make use of statistical synopses of the data to provide a user with a guess to the answer to his or her query, almost all previous proposals provide a *fixed precision* estimate for the final query result. That is, the information contained in the synopsis is fixed at the time that the query is issued, and if the user is not happy with the result, he or she is out of luck. It is unlikely that any such system can truly supplant existing database technology as the preferred query processing option, which is the goal of the DBO project.

The project that produced results closest to the DBO system is the UC Berkeley CONTROL project, which resulted in the idea of *online aggregation* [2, 3]. In online aggregation as in DBO, if the user is not happy with a guess' accuracy, he or she can simply wait for the guess to become more

[1]The technical aspects of DBO's query processing engine were described in detail in a SIGMOD 2007 paper [4].

ACM SIGMOD '08 Vancouver, BC, Canada

accurate. However, the problem with online aggregation algorithms is that they are not scalable – as soon as they process enough data that the data cannot be stored in main memory, they become unusable. In contrast, DBO continues to produce better and better guesses throughout query execution, and is just as fast as most production database systems.

3. QUERY PROCESSING IN DBO

In designing DBO, the goal is to achieve scalability in analytic processing, while at the same always time giving the user an accurate guess for the final answer to a query. To realize this, the various relational operations running within the DBO engine are constantly communicating with one another, checking whether tuples produced as partial query result can combine to form tuples that will actually appear in the query result set. Since tuples are always processed in a statistically random (or pseudo-random) order, whether or not DBO is "lucky" enough to find such tuples can be used to provide for a statistically meaningful guess for the final answer to the underlying query.

This search for such "lucky" output tuples requires fundamental design changes in the query processing engine compared to a traditional database system. Traditionally, query processing operations such as joins and table scans operate in isolation as black boxes with little or no communication of their internal state to the outside world. In contrast, in DBO all operations constantly communicate internal state in order to facilitate the search for "lucky" output tuples. The reason for this requirement is clear: if a query involves multiple relations, it is impossible to guess the answer if we exclude information provided by an operation that is exclusively processing one of the relations. It is for this reason that query processing in DBO is centered around the basic abstraction of a *levelwise step*. A levelwise step consists of all of the operations that are performed at a single level of a query plan. It provides for communication of internal state so that "lucky" tuples can be located, and it allows careful control of the progress of the underlying relational operations so that the statistical properties of the search for "lucky" tuples can be characterized mathematically.

The process of evaluating a query from startup through completion in DBO for a query plan containing an eight-way join is depicted in Figure 1. In this example, DBO's engine begins by executing the first levelwise step, where each join at the bottom level of the plan is evaluated concurrently. DBO maintains all of the data in the base relations in a random order on disk, so that at all times the set of tuples scanned by the first levelwise step is a statistically random sample of the various input relations. At all times, the various joins communicate with one another, looking for lucky output tuples. This communication allows the levelwise step to maintain an online estimator N_1 for the final answer to the query. As the levelwise step progresses and more data has been seen, N_1 achieves more and more accuracy. Eventually the first step begins to produce output tuples from each of its constituent operations. These tuples are produced using a statistically random ordering provided by a hash function, so that the tuples can be pipelined into the second levelwise step and are also used by the step to produce a second statistically meaningful online estimator for the final query result, called N_2. Eventually, the first levelwise step completes, and N_1 becomes frozen. At all times,

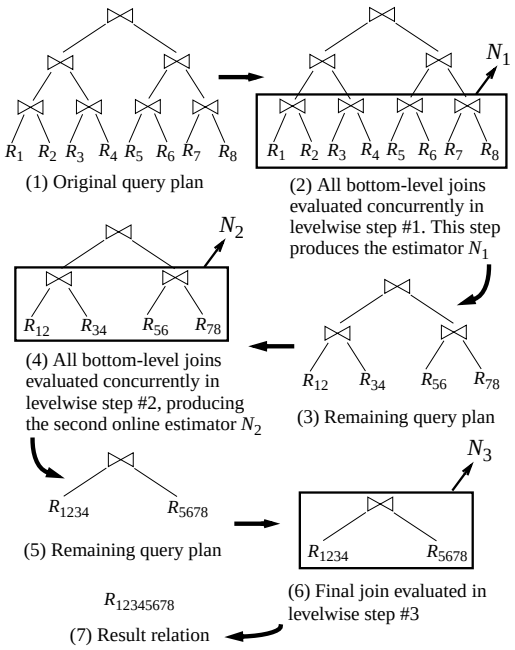

Figure 1: Levelwise query evaluation in DBO.

the current value for N_2 is combined with N_1 to produce a single estimate for the final answer to the query. Finally, the last levelwise step is executed, which produces an online estimator N_3. Again, as this step progresses, N_3 is combined with both N_1 and N_2 (both eventually frozen) to produce an estimate for the answer to the query. As the end of query execution approaches, N_3 will approach (and eventually become equal to) the correct query result.

4. DBO'S SOFTWARE ARCHITECTURE

Section 3 gave an overview of how query processing is performed in DBO; this subsection describes the basic software architecture for the system, which is depicted above in Figure 2. There are five major software components in the DBO system: the query compiler, the controller module, the levelwise step module, the in-memory join, and the statistics module. In addition, there are modules that perform the standard functionality required by every database system: storage management, record management, and so on. Currently, there is no query optimizer in DBO, and queries are supplied to the system using a special-purpose specification language that resembles relational algebra. "Queries" in this language are not declarative, and describe the query plan exactly. Adding an optimizer to DBO that can handle standard SQL queries is one of the research problems that will be considered by the proposed project, and the problem of adding an optimizer is considered subsequently.

These components work together to evaluate a query as follows. Before query execution begins, the query compiler compiles the query and uses the resulting query plan to creates instances of the four query processing components: the controller, the required levelwise steps, the in-memory join, and the statistics module. After this startup phase, control

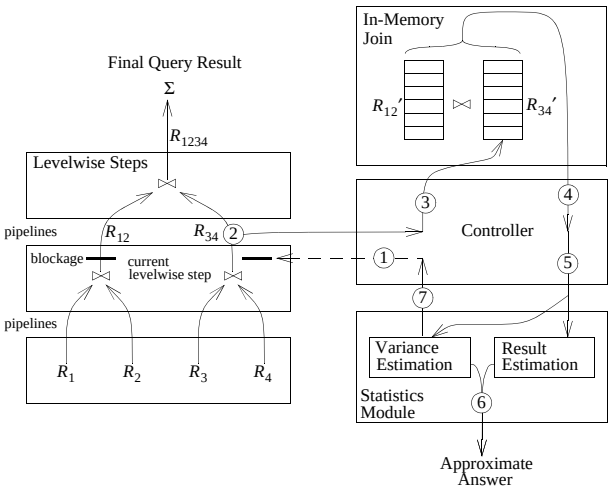

Figure 2: DBO's query processing components and information flow; the basic functionality of these components is described in detail in Section 4.

of query processing is passed to the controller. The controller directs data flow throughout the system, and makes sure that all software components operate in a synchronized fashion.

Consider Figure 2 which depicts the state of the query processing engine after the first levelwise step (consisting only of table scans) has completed and the joins comprising the second levelwise step have begun to produce output tuples. In DBO, one thread is associated with each of the joins. These threads place output tuples into data pipelines that send the tuples into the next levelwise step. Conceptually, the controller places a "blockage" into each of the output pipes from the current levelwise step. This blockage allows the operations in the step to continue their operation until they produce enough output tuples that the pipeline "backs up" – that is, its buffer fills – and then each blocked operation must wait until the blockage on its pipe is removed by the controller.

When the controller is ready to process more data, it temporarily removes the blockage from one of the output pipes, which is depicted as step (1) in Figure 2. This allows a set of random tuples from $\mathtt{R}_{34}$ to flow into the next levelwise step, where they are processed. A fork in the pipeline also directs copies of those tuples into the controller (step (2)), where they are forwarded into the in-memory join (step (3)). In Figure 2, this sample or current subset of $\mathtt{R}_{34}$ is denoted as $\mathtt{R}'_{34}$. It is the task of the in-memory join to efficiently look for any "lucky" output tuples that happen to contribute to the final query result. The in-memory join buffers random subsets of each of the output relations from the current levelwise step in a set of hash table indexes. When the in-memory join receives a new set of tuples from one of the output relations, it discards the old set from that output relation and replaces it with the new tuples. The in-memory join then uses the hashed index to efficiently join all of the buffered tuples (step (4)); the result of this in-memory join is then sent to the controller.

The controller then forwards those lucky output tuples to the statistics module (step (5)), where they are used to update the current estimate for the query result, as well as to update the statistical model for the variance (or inaccuracy) of the current estimate. After this, the statistics module updates the user interface with the current estimate (step (6)). The controller then uses information from the statistics module to decide which pipe to unblock next (and for how long) in order to speed convergence (step (7)) – thus completing the loop.

5. THE DBO DEMONSTRATION

SIGMOD participants who attend the DBO demonstration will see a side-by-side comparison of Postgres (a widely-used, standard relational engine) and DBO for executing queries over the TPC-H schema. We plan to have two identical desktop machines sitting side-by-side at the demonstration, both containing scale-factor three (six GB) and scale-factory ten (20 GB) instantiations of the TPC-H database. One of the machines will be running DBO, and the other Postgres. Several queries over both databases will be "ready to go"; from very simple, one-table scans with a `SUM` on top to multi-table joins with `GROUP BY`s and complex aggregate functions.

The demonstration attendee can choose one of the queries (or modify one of the queries as he or she wishes) and then execute the query in parallel on both database systems. DBO will immediately spawn a graphical user interface that shows its current estimate (or estimates in the case of a `GROUP BY`) as well as the associated confidence bounds that describe the accuracy of the guess. DBO plots these as a function of time, so that the bounds are shown narrowing as the query progresses. For many queries, the bounds can become exceptionally narrow (indicating high accuracy) after only a very short time. Postgres, in contrast, will give no feedback until it completes execution – typically significantly later than DBO finishes execution.

On the smaller, scale-factor three instantiation of the TPC-H schema, most queries will finish in a few minutes, which will provide an informative contrast of the relative speed of the two systems. On the larger, scale-factory ten instantiation, some queries will take an hour or more to finish (which is beyond the attention span of the average demonstration attendee), but this larger database instance will demonstrate how after just a few minutes, DBO can still produce a very useful guess as to the final query result.

6. REFERENCES

[1] Swarup Acharya, Phillip B. Gibbons, Viswanath Poosala, and Sridhar Ramaswamy. The aqua approximate query answering system. In *SIGMOD Conference*, pages 574–576, 1999.

[2] Peter J. Haas and Joseph M. Hellerstein. Ripple joins for online aggregation. In *SIGMOD Conference*, pages 287–298, 1999.

[3] Joseph M. Hellerstein, Peter J. Haas, and Helen J. Wang. Online aggregation. In *SIGMOD Conference*, pages 171–182, 1997.

[4] Christopher M. Jermaine, Subramanian Arumugam, Abhijit Pol, and Alin Dobra. Scalable approximate query processing with the dbo engine. In *SIGMOD Conference*, pages 725–736, 2007.