Approximate Query Engines: Commercial Challenges and Research Opportunities

[Keynote]

Barzan Mozafari

University of Michigan, Ann Arbor
mozafari@umich.edu

ABSTRACT

Recent years have witnessed a surge of interest in Approximate Query Processing (AQP) solutions, both in academia and the commercial world. In addition to well-known open problems in this area, there are many new research challenges that have surfaced as a result of the first interaction of AQP technology with commercial and real-world customers. We categorize these into deployment, planning, and interface challenges. At the same time, AQP settings introduce many interesting opportunities that would not be possible in a database with precise answers. These opportunities create hopes for overcoming some of the major limitations of traditional database systems. For example, we discuss how a database can reuse its past work in a generic way, and become smarter as it answers new queries. Our goal in this talk is to suggest some of the exciting research directions in this field that are worth pursuing.

Keywords

Approximation; Analytics; Interactive Response Times

1. INTERACTIVITY: AN ELUSIVE GOAL

Social media, mobile devices, and wireless sensors continue to create massive volumes of data at unprecedented rates. Given the abundance of rich datasets, the race for deriving instantaneous insight has become the landmark of a new Gold Rush. Ironically, data analysis tools are now themselves the bottlenecks of many data-driven activities. When faced with sufficiently large datasets, traditional query engines can easily take minutes or hours to answer the simplest of queries [22]. Even with modest-sized datasets, traditional engines are easily pushed to their limits when running a large number of concurrent queries in a shared cluster—a increasingly popular deployment pattern in large organizations.

As a result, query response times are in many cases unacceptable to users. For example, in data exploration tasks, data scientists form an initial hypothesis (e.g., by visualizing the data), consult the data, adjust their hypothesis accordingly, and repeat this process until they discover a satisfactory answer. Slow and costly interactions with data can therefore severely inhibit their productivity, engagement, and even creativity.[1]

While in-memory databases and columnar formats seemed like an ultimate solution to performance problems a decade ago, achieving interactive response times has largely remained an elusive goal. There are many contributing causes, ranging from software inefficiencies (e.g., excessive copying and serialization overheads in modern applications), to hardware limitations (e.g., the Memory Wall [15]), to the explosion of data volumes and the rising number of shared infrastructures and concurrent applications.

Driven by the growing market for interactive analytics, database vendors have invested heavily in various optimization techniques. These techniques aim at exploiting parallelism, indexing, materialization, better query plans, data compression, columnar formats, and in-memory and in-situ processing. While new advances are constantly made in each of these areas, these techniques are in essence no different than the mainstream database research on query optimization over the past four decades, which has been bound by three characteristics:

C1 the goal is to *efficiently* access **all** tuples relevant to the current query and avoid tuples that are irrelevant (e.g., indexing, materialization, compression, caching);

C2 the optimization decision is about choosing the cheapest query plan among a set of plans that are all **equivalent** and **correct**; and

C3 the work (both I/O and computation) performed for answering a query is **wasted afterwards**, i.e., the answer to a previous query is rarely useful for speeding up future queries.

Despite its many merits, pursuing this traditional approach is in many cases **unnecessary** and even **impractical**. First, one can make perfect decisions in many cases without having perfect answers. This is because query results are only useful inasmuch as they enable the best decisions, and thus, as long as our estimates of their results lead us to the same decisions, computing their precise values is unnecessary. There are numerous examples, such as A/B testing, root-cause analysis, hypothesis testing, feature selection, and visualization.

[1]Studies show that, for a human's interactive engagement, the computer's response time must not exceed 2 secs [16].

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

SIGMOD'17 May 14-19, 2017, Chicago, IL, USA

© 2017 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-4197-4/17/05…$15.00

DOI: http://dx.doi.org/10.1145/3035918.3056098

Moreover, as long as we insist on processing all relevant tuples, all our query optimization techniques are eventually destined to fail, because of the exponential growth of data which has already surpassed Moore's law [30, 32]. It is true that the set of tuples queried by the user (a.k.a. *working set*) is usually much smaller than the entire dataset, e.g., most queries might focus on the last week's worth of data. However, the data generated each week is also growing at an exponential rate; thus, even if we know which tuples will be accessed *a priori*, keeping them entirely in memory will eventually be too expensive (even after compression). In other words, since the price improvement of memory (and CPU) is lagging exponentially behind the growth rate of data, our inability to provide interactive response times at an affordable cost will only increase. Eventually, when the datasets are sufficiently large, no optimization technique will be able to provide interactive response times unless we relax our goal of processing all relevant tuples (C1).

This is the primary reason for the recent surge in Approximate Query Processing (AQP), which forgoes C1 as a goal. In AQP, only a small fraction of the relevant tuples is processed in order to provide fast, approximate answers. We argue that an AQP engine can also forgo C2 by *deliberately* pursuing *multiple query plans that are not necessarily equivalent* in order to obtain several approximations of the same quantity. These various estimates can then be carefully calibrated and combined into a more accurate answer [17]. Interestingly, in an AQP context, one can even overcome the ultimate limitation of the traditional approach, namely C3. Instead of wasting the work done for answering each query, recent results show that an AQP engine can change this paradigm and reuse much of the previous work based on an idea called *Database Learning* [26].

Despite its long history in academic research [34], AQP has only recently begun to play a role in commercial products. Shortly after the adoption of BlinkDB [9] and G-OLA [35] by Databricks [1], several other vendors have started to add AQP features to their query engines. Examples include Facebook's Presto [3], Infobright [13], Yahoo's Druid [2], SnappyData [4], and more recently, Oracle [33].

2. NEW CHALLENGES

There are long-standing challenges that are well-known in the AQP literature (e.g., the difficulty of supporting adhoc joins [7, 14]). However, the first interactions of AQP technology with commercial and real-world customers [4, 22, 29] have revealed new challenges that have been previously overlooked in our academic endeavors. These new challenges can be categorized into three main categories.

Deployment Challenges — The commercial adoption of AQP engines is still largely limited by the reluctance of major vendors to modify their database internals. However, every AQP solution to date has required some modification of an existing database engine. Even when AQP engines store their samples as relational tables, they still modify the database internals to compute and control the approximation error [6, 9, 10, 11, 12, 24, 28]. For example, BlinkDB and G-OLA rely on overriding the default aggregate implementations of the database [35] or augmenting each tuple with Poissonized multiplicities [8].

Moreover, AQP engines return error estimates along their output tuples. These additional columns are incompatible with existing BI (Business Intelligence) tools, which assume standard relational outputs from their data warehouse (e.g., Tableau, Oracle BI, GoodData, Sisense, InsightSquared and Zoho Reports). Consequently, existing users relying on these BI tools are also reluctant to use an AQP engine unless (i) it is compatible with their existing BI tool, and (ii) it can be used as a drop-in replacement for their existing data warehouse.

One possible approach to overcome this adoption barrier is to design a middleware-based AQP architecture that does not rely on any internal modifications of the underlying database. We have started an academic prototype of such a system, called Verdict [5, 17]. Ensuring the generality and efficiency of such an approach is itself an interesting area of new research. To circumvent the compatibility obstacles, we have also begun to explore the idea of substituting low-level error estimates with higher-level accuracy contracts [21].

Planning Challenges — Predicting the resource requirements and actual latency of database queries has always been a difficult problem even for traditional engines [19, 18]. However, query optimizers have traditionally relied on (rough) cost estimates to distinguish between various query plans, without the need to predict the actual query latencies. Likewise, *online* AQP engines do not need to predict query latencies *a priori*; instead; they gradually refine their approximate answers as they read more data, and stop whenever the user is satisfied or their time budget is exhausted. Unfortunately, this is not an option for *offline* AQP engines, which must choose an appropriate sample size among a fixed number of available samples. Here, if users specify a deadline for their query, the engine must be able to predict the query's latency for different sample sizes quite accurately, e.g., to choose the largest sample that can be processed within 2 seconds.

Another planning challenge is the lack of an approximation-aware scheduler in the literature. During execution, an approximate query is treated the same as an exact one. In other words, the approximation decision in current systems is made prior to the scheduling decision. Ideally, approximation requirements must be exposed to the scheduler to enable dynamic decisions of which tasks to prioritize and when to stop an execution. Achieving this goal requires an enhanced scheduler interface to capture the approximation quality of a query given its partial execution. The scheduler can then holistically moderate the progress of all (exact and approximate) queries in the system.

Another planning challenge is that of deciding which samples to build in advance. In fact, a common criticism of stratified sampling [9, 10] is its reliance on the prior knowledge of query patterns. However, exploratory workloads tend to be quite unpredictable in practice. Consequently, a set of stratified samples that were previously optimal may quickly become sub-optimal and brittle for future queries. Here, adapting a robust optimization framework, such as CliffGuard [20], might be a promising direction. CliffGuard takes the effect of workload uncertainties into account in order to ensure a graceful degradation of performance when future workloads deviate from the past.

Finally, there is a need for an approximation-aware code generation logic. Modern databases rely heavily on vectorization and dynamic code generation [23] to remove virtual function calls, maximize the use of CPU registers, and im-

prove branch predictions. While these techniques are well-understood for common query operators, the same ideas must be adjusted to complex AQP operators.

For instance, the computation logic in Poissonized bootstrap [8] is based on a "tuple stream" model and is hence not optimized for the modern day multi-core systems. Likewise, Java byte codes enjoy significant performance benefits when the Hotspot JIT compiler turns them into optimized machine code instructions based on the specific processor in use (e.g., using SIMD instructions). However, such dynamic compilations are only triggered for large iteration counts. These counts can be significantly lower when the AQP engine works on small samples of the data.

Interface Challenges — While a statistician might be able to use and interpret complex error statistics, a typical database user or an application developer may find a large number of error statistics associated with an approximate answer simply overwhelming. Online AQP engines have advocated the use of gradually shrinking error bars [12, 35]. However, many users may still find confidence intervals hard to interpret. While range-based error guarantees are easier to interpret [27], they also tend to be quite loose in practice. For visualization tasks, another idea is to define error as the visual proximity of the approximate results to the original pixels [25]. SnappyData [22] uses high-level accuracy contracts (HAC) to shield users from low-level error statistics, offering them a single-valued error guarantee. Despite all these efforts, the design of creative user-interfaces for AQP engines remains an open area of research.

3. NEW OPPORTUNITIES

Once the door to approximation is opened, many interesting optimization opportunities arise that would never be possible in a traditional database with precise answers.

Database Learning — Traditional databases have limited opportunities to reuse past query answers. This is because, unless a query is an exact subset of another, the output of one cannot be used for answering the other. However, this limitation does not apply to an AQP engine because of the following observation: *the answer to each query reveals some* fuzzy knowledge *about the answers to other queries, even if each query accesses a different subset of tuples and columns.* This is because the answers to different queries stem from the same (unknown) underlying distribution that has generated the entire dataset. In other words, each answer reveals a piece of information about this underlying but *unknown distribution.*

Note that having a concise statistical model of the underlying data can have significant performance benefits. In the ideal case, if we had access to an incredibly precise model of the underlying data, we would no longer have to access the data itself. In other words, we could answer queries more efficiently by analytically evaluating them on our concise model, which would mean reading and manipulating a few kilobytes of model parameters rather than terabytes of raw data. While we may never have a perfect model in practice, even an imperfect model can be quite useful. Instead of using the entire dataset (or even a large sample of it), one can use a small sample of it to quickly produce a rough approximate answer, which can then be calibrated and combined with the model to obtain a more accurate approximate answer to the query. *The more precise our model, the less need for actual data, the smaller our sample, and consequently, the faster our response time.* In particular, if we could somehow continuously improve our model—say, by *learning* a bit of information from every query and its answer—we should be able to *answer new queries using increasingly smaller portions of data, i.e., become smarter and faster as we process more queries*

This idea is called *Database Learning* [26], as it is reminiscent of the inferential goal of Machine Leaning whereby past observations are used to improve future predictions.

Active Database Learning — Instead of passively waiting for users to issue queries and then learn, the AQP engine itself could pro-actively issue queries and learn from them (e.g., whenever the database is idle). This is akin to the idea of active learning (AL) in machine learning, whereby the learner itself participates in deciding which observations are most beneficial to obtain [31]. There are two main strategies in AL: *uncertainty* measures seek observations for which the learner is least certain, while *informativeness* measures seek observations that, if obtained, will have the largest impact on the learner's prediction power for other data points. In an AQP context, the first strategy would mean exploring tuples/columns that have not been accessed by previous queries, whereas the second one would entail active querying of regions that, although accessed by previous queries, are still worth further exploration given their extreme popularity.

Stochastic Query Planning — Unlike traditional databases that choose the best query plan from a set of equivalent alternatives, an AQP engine can *deliberately* pursue multiple query plans that are *not* necessarily equivalent. In addition to executing the original query on a small sample, the AQP engine can concurrently obtain several other approximations using column-wise, tuple-wise, and temporal correlations present in the data. When available, one can even invoke existing regression models to estimate a specific column's marginal or conditional distribution given a combination of other columns. These various estimates can then be carefully calibrated and combined to produce a single, more accurate answer. We call this general idea *Stochastic Query Planning* [17].

4. ACKNOWLEDGEMENTS

This work is in part supported by National Science Foundation (grants 1544844, 1629397, and 1553169).

5. REFERENCES

[1] Databricks. http://databricks.com/.
[2] Fast, approximate analysis of big data (yahoo's druid). http://yahooeng.tumblr.com/post/135390948446/data-sketches.
[3] Presto: Distributed SQL query engine for big data. https://prestodb.io/docs/current/release/release-0.61.html.
[4] SnappyData Inc. http://snappydata.io.
[5] Verdict. http://verdictdb.org/.
[6] S. Acharya, P. B. Gibbons, and V. Poosala. Aqua: A fast decision support system using approximate query answers. In *VLDB*, 1999.
[7] S. Acharya, P. B. Gibbons, V. Poosala, and

S. Ramaswamy. Join synopses for approximate query answering. In *SIGMOD*, 1999.

[8] S. Agarwal, H. Milner, A. Kleiner, A. Talwalkar, M. Jordan, S. Madden, B. Mozafari, and I. Stoica. Knowing when you're wrong: Building fast and reliable approximate query processing systems. In *SIGMOD*, 2014.

[9] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. BlinkDB: queries with bounded errors and bounded response times on very large data. In *EuroSys*, 2013.

[10] S. Chaudhuri, G. Das, and V. Narasayya. Optimized stratified sampling for approximate query processing. *TODS*, 2007.

[11] A. Dobra, C. Jermaine, F. Rusu, and F. Xu. Turbo-charging estimate convergence in dbo. *PVLDB*, 2009.

[12] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In *SIGMOD*, 1997.

[13] Infobright. Infobright approximate query (iaq). https://infobright.com/introducing-iaq/.

[14] F. Li, B. Wu, K. Yi, and Z. Zhao. Wander join: Online aggregation via random walks. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, 2016.

[15] S. A. McKee. Reflections on the memory wall. In *Proceedings of the 1st Conference on Computing Frontiers*, 2004.

[16] R. B. Miller. Response time in man-computer conversational transactions. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*. ACM, 1968.

[17] B. Mozafari. Verdict: A system for stochastic query planning. In *CIDR, Biennial Conference on Innovative Data Systems*, 2015.

[18] B. Mozafari, C. Curino, A. Jindal, and S. Madden. Performance and resource modeling in highly-concurrent OLTP workloads. In *SIGMOD*, 2013.

[19] B. Mozafari, C. Curino, and S. Madden. DBSeer: Resource and performance prediction for building a next generation database cloud. In *CIDR*, 2013.

[20] B. Mozafari, E. Z. Y. Goh, and D. Y. Yoon. CliffGuard: A principled framework for finding robust database designs. In *SIGMOD*, 2015.

[21] B. Mozafari and N. Niu. A handbook for building an approximate query engine. *IEEE Data Eng. Bull.*, 2015.

[22] B. Mozafari, J. Ramnarayan, S. Menon, Y. Mahajan, S. Chakraborty, H. Bhanawat, and K. Bachhav. Snappydata: A unified cluster for streaming, transactions, and interactive analytics. In *CIDR*, 2017.

[23] T. Neumann. Efficiently compiling efficient query plans for modern hardware. *PVLDB*, 2011.

[24] N. Pansare, V. R. Borkar, C. Jermaine, and T. Condie. Online aggregation for large mapreduce jobs. *PVLDB*, 4, 2011.

[25] Y. Park, M. Cafarella, and B. Mozafari. Visualization-aware sampling for very large databases. *ICDE*, 2016.

[26] Y. Park, A. S. Tajik, M. Cafarella, and B. Mozafari. Database Learning: Towards a database that becomes smarter every time. In *SIGMOD*, 2017.

[27] N. Potti and J. M. Patel. DAQ: a new paradigm for approximate query processing. *PVLDB*, 8, 2015.

[28] C. Qin and F. Rusu. Pf-ola: a high-performance framework for parallel online aggregation. *Distributed and Parallel Databases*, 2013.

[29] J. Ramnarayan, B. Mozafari, S. Menon, S. Wale, N. Kumar, H. Bhanawat, S. Chakraborty, Y. Mahajan, R. Mishra, and K. Bachhav. Snappydata: A hybrid transactional analytical store built on spark. In *SIGMOD*, 2016.

[30] E. Russo. Applying moore's law to data growth. https://www.datavail.com/blog/applying-moores-law-data-growth/.

[31] B. Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2010.

[32] I. Stoica. For big data, mooreâĂŹs law means better decisions. https://amplab.cs.berkeley.edu/for-big-data-moores-law-means-better-decisions/.

[33] H. Su, M. Zait, V. Barrière, J. Torres, and A. Menck. Approximate aggregates in oracle 12c, 2016.

[34] S. Vrbsky, K. Smith, and J. Liu. An object-oriented semantic data model to support approximate query processing. In *Proceedings of IFIP TC2 Working Conference on Object-Oriented Database Semantics*, 1990.

[35] K. Zeng, S. Agarwal, A. Dave, M. Armbrust, and I. Stoica. G-OLA: Generalized on-line aggregation for interactive analysis on big data. In *SIGMOD*, 2015.