Demonstration of VerdictDB, the Platform-Independent AQP System

http://verdictdb.org

Wen He Yongjoo Park Idris Hanafi Jacob Yatvitskiy Barzan Mozafari
University of Michigan, Ann Arbor
{hewen,pyongjoo,ihanafi,yatvij,mozafari}@umich.edu

ABSTRACT

We demonstrate VerdictDB, the first platform-independent approximate query processing (AQP) system. Unlike existing AQP systems that are tightly-integrated into a specific database, VerdictDB operates at the driver-level, acting as a middleware between users and off-the-shelf database systems. In other words, VerdictDB requires no modifications to the database internals; it simply relies on rewriting incoming queries such that the standard execution of the rewritten queries under relational semantics yields approximate answers to the original queries. VerdictDB exploits a novel technique for error estimation called *variational subsampling*, which is amenable to efficient computation via SQL.

In this demonstration, we showcase VerdictDB's performance benefits (up to two orders of magnitude) compared to the queries that are issued directly to existing query engines. We also illustrate that the approximate answers returned by VerdictDB are nearly identical to the exact answers. We use Apache Spark SQL and Amazon Redshift as two examples of modern distributed query platforms. We allow the audience to explore VerdictDB using a web-based interface (e.g., Hue or Apache Zeppelin) to issue queries and visualize their answers. VerdictDB is currently open-sourced and available under Apache License (V2).

CCS CONCEPTS

• **Information systems → Query optimization**; **Online analytical processing engines**;

KEYWORDS

Approximate query processing, data analytics

ACM Reference Format:
Wen He, Yongjoo Park, Idris Hanafi, Barzan Mozafari, Jacob Yatvitskiy. 2018. Demonstration of VerdictDB, the Platform-Independent AQP System: http://verdictdb.org. In *SIGMOD'18: 2018 International Conference on Management of Data, June 10–15, 2018, Houston, TX, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3183713.3193538

SIGMOD'18, June 10–15, 2018, Houston, TX, USA

ACM ISBN 978-1-4503-4703-7/18/06…$15.00
https://doi.org/10.1145/3183713.3193538

1 INTRODUCTION

Approximate query processing (AQP) allows users to obtain query answers much faster at a negligible cost to accuracy [21]. Currently, however, only a handful of query engines offer approximation features. Universal approximate query processing [26] aims to offer AQP capabilities regardless of the specific SQL platform by the user (e.g., Hive, Spark SQL, Impala, Amazon Redshift, Presto, etc.). In this demonstration, we present the first Universal AQP system, called VerdictDB, which can work with a wide variety of distributed query engines. Before describing our demonstration scenarios, we first describe our motivation and the challenges involved in achieving Universal AQP.

Motivation for Universal AQP. Existing AQP techniques require substantial changes to the standard query evaluation logic implemented in relational engines. For example, previous work has either intimately integrated the error estimation logic into the scan operator [8, 14, 22] or has overridden the relational operators altogether [23, 34]. Others, such as AQUA [7], have used query-rewriting techniques but have relied on the underlying engine's ability to enforce PK-FK (primary-key, foreign-key) relationships—an assumption that does not hold in modern SQL-on-Hadoop engines. Due to the reluctance of database vendors in modifying their internal implementation, adoption of AQP solutions has been slow [17]. Only recently, a few vendors have started to include limited forms of approximation features in their products [2, 4, 13, 30, 31]. To widen the reach of AQP technology and accelerate its adoption, VerdictDB aims to provide a *Universal AQP* solution: powerful and efficient AQP capabilities without any modifications to existing query engines.

To achieve this universality, VerdictDB acts as a middleware; it rewrites incoming queries, such that the standard execution of the rewritten queries under relational semantics would yield approximate answers to the original queries. This requires the entire AQP process to be encoded in SQL, which poses several challenges.

Challenges in Universal AQP. First, correct error estimation must consider inter-tuple correlations introduced as a result of joining multiple sample tables. Previous work has achieved this goal by modifying the internal query evaluation of the database [14, 33], using special join algorithms [15], or restricting joins to PK-FK joins [7]. However, as a middleware, VerdictDB can neither change the query evaluation nor use non-standard join algorithms. On modern SQL-on-Hadoop systems, we cannot enforce FK constraints either.

Second, the generality of Universal AQP should not come at a great cost to computational efficiency. It must remain sufficiently

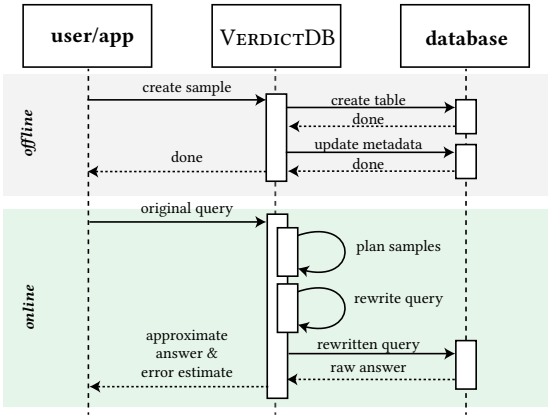

Figure 1: VerdictDB's offline and online workflow: sample preparation (in gray) and query processing (in green).

efficient compared to exact query processing to justify the use of approximation in the first place. Analytical error estimation strategies that modify query evaluation [10, 14, 22] are not applicable. In contrast, resampling-based approaches [8, 28] can be implemented without modifying the DBMS; however, they are computationally prohibitive when expressed in SQL.

VerdictDB's Approach. Although resampling-based error estimation techniques [8, 28] support a wide class of queries, the cost of constructing resamples often becomes a major performance bottleneck. VerdictDB relies on a novel alternative called *variational subsampling* [26], which yields provably-equivalent asymptotic properties as traditional subsampling [29]. The key idea in variational subsampling is that, instead of running the same aggregate query on multiple subsamples, we can achieve the same result through a single execution of a carefully rewritten query on the sample table itself. Our rewritten SQL query treats different resamples separately by relying on a *resample-id* assigned to each tuple.

Next, we provide a brief overview of VerdictDB and refer the interested reader to [26] for further details on variational subsampling and VerdictDB's architecture.

2 VERDICTDB OVERVIEW

In this section, we discuss VerdictDB's deployment scenario, its workflow, and the types of queries it supports.

2.1 Deployment

Users can use any interface for issuing their SQL queries and any off-the-shelf query engine that can return exact answers to SQL queries.[1] VerdictDB is deployed as a middleware between the user (or query interface) and the query engine. Specifically, VerdictDB operates at the driver-level and accepts JDBC/ODBC connections. VerdictDB communicates with the query engine in SQL using its standard interface, i.e., JDBC for Hive, Impala, Redshift, and

[1] The query engine must support `rand()`, a hash function (e.g., `md5`, `crc32`), `create table ... as select ...`, and window functions (e.g., `count(*) over ()`).

Table 1: Types of queries *supported* by VerdictDB.

aggregates	`count`, `count-distinct`, `sum`, `avg`, `quantile`, user-defined aggregate (UDA) functions
table sources	derived tables or base tables joined via equi-joins; the derived table can be a select statement with or without aggregate functions.
selections (filtering)	expr comp expr (e.g., `price > 100`), expr comp subquery (e.g., `price > (select ...)`), logical `AND` and `OR`, etc.
other clauses	`group by`, `order by`, `limit`, `having`

`spark.DataFrame` for Spark SQL. VerdictDB accesses the query engine using the same credentials as the user (e.g., ID/password, a Kerberos ticket). Thus, VerdictDB can only access the data that the user is authorized to access.

2.2 Workflow

The workflow in VerdictDB consists of two stages: sample preparation and query processing, depicted in Figure 1 as gray and green boxes, respectively. During the sample preparation stage, VerdictDB builds multiple sample tables for various base tables. By default, VerdictDB collects certain statistics about each base table to determine if and what types of samples to build for that base table. VerdictDB currently supports simple random samples, stratified samples, and hash-based samples. However, the user can also manually specify which types of samples to build for each table. The created sample tables—including their metadata—are stored in the query engine itself. The user can also define a High-level Accuracy Contract (HAC) [21] to specify his/her error tolerance.

When the user issues a query, VerdictDB intercepts it and decides whether it can be approximated and sped up without violating the HAC. If not, it simply reroutes the unmodified query to the query engine and returns the exact answers back to the user. Otherwise, VerdictDB determines a combination of sample tables that can minimize the approximation error. It then sends a rewritten query to the query engine that uses those sample tables instead of the original (base) tables. Once VerdictDB obtains the raw answer from the query engine, it applies necessary adjustments to the answer and returns an approximate modified answer along with error estimates to the user [26].

2.3 Supported Queries

As previously mentioned, when VerdictDB cannot speed up the query without violating HAC requirements, it simply passes down the unmodified query to the query engine. Queries that can be efficiently approximated by VerdictDB include *non-extreme* aggregate queries (i.e., `count`, `sum`, `avg`, `percentile`). The *extreme* aggregates (i.e., `min` and `max`) are not currently supported.

VerdictDB supports equijoins, comparison subqueries (e.g., `where sales < (select avg(sales) ...)`), and other selection predicates (e.g., `IN list, LIKE regex, <, >`). To support comparison subqueries, VerdictDB converts them into a join. Table 1 summarizes the types of queries supported by VerdictDB.

Figure 2: The query on the left is issued directly against Spark SQL while the one on the right is issued against Spark SQL through VerdictDB. The results are nearly identical while the latter is faster.

3 DEMONSTRATION SCENARIO

We demonstrate VerdictDB using an interactive web interface (Section 3.1). We use two large-scale datasets (Section 3.2), to showcase three aspects of VerdictDB: platform-independence, speedup, and correctness (Sections 3.3 to 3.5).

3.1 Query Interface

Our demonstration will use Apache Zeppelin [1] for issuing queries and visualizing the answers returned by the underlying engine with and without VerdictDB. An example screen is shown in Figure 2. Each page in Apache Zeppelin will contain multiple *notebooks*. In each notebook, the user will issue a single SQL statement. By juxtaposing two notebooks issuing the same query with and without VerdictDB, the user will be able to easily compare the latencies and the query answers.

In other words, our demo will have pairs of pre-created notebooks. The notebooks on the left side of the page will be connected to off-the-shelf SQL engines (e.g., Spark SQL, Amazon Redshift) while the notebooks on the right side will be connected through VerdictDB to those same engines. This will allow the audience to visually compare the approximate and the exact answers, while noticing their latency difference.

3.2 Datasets

We will use a real-life dataset, `insta` [3], as well as a well-known benchmark, `TPC-H` [5]. The `insta` dataset is a 100× scaled version of a publicly available sales records of an online grocery store, called Instacart. We also use the standard TPC-H with a scale factor of 500 (i.e., 500 GB). Recall that VerdictDB's workflow consists of sample preparation and query processing. Considering the limited time of the demonstration, we will prepare 1% samples of the large fact tables in advance. During the live demonstration, the audience will simply issue queries without any preparation.

3.3 Platform-Independence

To showcase VerdictDB's platform-independence—one of VerdictDB's salient features—we will use multiple query engines, including Spark SQL, Amazon Redshift, and Apache Hive. We will have video recordings of our demonstration using other SQL engines on our website, which also host our open-source release [6]. Figure 2 shows a screenshot example of our demonstration using Spark SQL.

3.4 Speedup

A key benefit of AQP in general, and VerdictDB in particular, is to significantly speedup the query processing. We will demonstrate the speedup benefits of VerdictDB using the elapsed times displayed by Apache Zeppelin. For each query engine, comparing the elapsed times with and without VerdictDB will allow the audience to appreciate the massive speedups brought by VerdictDB.

For example, Figure 2 is a query that analyzes how the order frequencies of different types of products change throughout the day in an online grocery store. Here, Spark SQL took 1.15 hours while SparkSQL-plus-VerdictDB took only 50 seconds. In other words, VerdictDB sped up Spark SQL by 82.8× faster, while incurring only 0.6% error.

Given that the queries directly issued against these query engines can take excessively long, we will run them in advance and keep their results and latencies on the display. During the live demonstration, we will guide the audience to avoid querying the underlying engines directly and instead interact with the notebooks that send queries through VerdictDB.

3.5 Correctness

We demonstrate that VerdictDB's approximate answers are highly accurate for many types of complex analytical queries. Figure 2 shows an example. In each notebook, we will visualize both the

exact as well as the approximate answer, allowing the audience to visually observe that VerdictDB's answers are in most cases indistinguishable from the exact ones returned by the underlying query engine. In the example of Figure 2, the audience can see the same correlation between order frequencies and the time of day.

In addition to comparing the overall trend of the visualized results, the audience will also see the actual error bounds computed by VerdictDB. VerdictDB returns error bounds using an additional column; clicking a special icon will display both the estimated and the exact errors numerically.

4 RELATED WORK

AQP has been a subject of great interest over the past decades. For example, STRAT [11] uses a single stratified sample, while BlinkDB [9] creates multiple stratified samples based on different column sets. Quickr [14] uses on-the-fly sampling strategies to support complex and adhoc queries. Online Aggregation techniques continuously refine their answers during query execution [24, 32]. However, these systems require modifications of the database's internals, and are therefore tied to a specific query engine. Likewise, Aqua relies on CLT-based closed-forms, which requires independent random variables. Therefore, it can only support PK-FK joins [7]. Also, due to its use of closed-forms, Aqua cannot support UDAs. VerdictDB uses middleware and query rewriting approaches to achieve its universality and platform-independence. In the past, we have used query rewriting to enforce security policies transparently from the users [12]. Likewise, we have used a middleware approach to speed up visualization workloads [25] and to speed up future query processing by reusing past query answers [27]. In VerdictDB, we use a middleware architecture to achieve platform-independence for AQP.

5 CONCLUSION AND FUTURE WORK

Our demonstration focuses on the user experience and how they can benefit from using VerdictDB on top of their favorite query engine without having to modify the engine or their application. By using several popular query engines (e.g., Spark SQL, Amazon Redshift, and Hive), we showcase VerdictDB's great generality across different platforms, as well as its statistical correctness and efficiency. The core features of VerdictDB are currently open-sourced under Apache License, Version 2, allowing both researchers and practitioners to freely test and deploy VerdictDB in their own environment. Additional videos and documentations can be found on our website [6].

Currently, we are actively working on adding a physical designer to automatically decide which samples to build for more complex and adhoc workloads that change over time [20], integrating Database Learning to enable faster query processing [16, 27], and adapting our machine learning-based latency prediction techniques [18, 19] to estimate the runtime of a query on a given sample.

6 ACKNOWLEDGEMENT

This research is in part supported by National Science Foundation through grants 1629397, 1544844, and 1553169.

REFERENCES

[1] Apache zeppelin. https://zeppelin.apache.org/. Accessed: 2017-09-17.
[2] Fast, approximate analysis of big data (yahoo's druid). http://yahooeng.tumblr.com/post/135390948446/data-sketches. Accessed: 2017-09-17.
[3] Instacart Orders, Open Sourced. https://www.instacart.com/datasets/grocery-shopping-2017. Accessed: 2017-09-17.
[4] Presto: Distributed SQL query engine for big data. https://prestodb.io/docs/current/release/release-0.61.html. Accessed: 2017-09-17.
[5] TPC-H Benchmark. http://www.tpc.org/tpch/. Accessed: 2017-09-17.
[6] VerdictDB. http://verdictdb.org/. Accessed: 2017-09-17.
[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. Agarwal, A. Panda, B. Mozafari, A. P. Iyer, S. Madden, and I. Stoica. Blink and it's done: Interactive queries on very large data. *PVLDB*, 2012.
[11] S. Chaudhuri, G. Das, and V. Narasayya. Optimized stratified sampling for approximate query processing. *TODS*, 2007.
[12] K. Eykholt, A. Prakash, and B. Mozafari. Ensuring authorized updates in multi-user database-backed applications. In *USENIX Security Symposium*, 2017.
[13] Infobright. Infobright approximate query (iaq). https://infobright.com/introducing-iaq/. Accessed: 2017-09-17.
[14] S. Kandula, A. Shanbhag, A. Vitorovic, M. Olma, R. Grandl, S. Chaudhuri, and B. Ding. Quickr: Lazily approximating complex adhoc queries in bigdata clusters. In *SIGMOD*, 2016.
[15] 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.
[16] B. Mozafari. Verdict: A system for stochastic query planning. In *CIDR, Biennial Conference on Innovative Data Systems*, 2015.
[17] B. Mozafari. Approximate query engines: Commercial challenges and research opportunities. In *SIGMOD*, 2017.
[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] B. Mozafari and C. Zaniolo. Optimal load shedding with aggregates and mining queries. In *ICDE*, 2010.
[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, B. Mozafari, J. Sorenson, and J. Wang. VerdictDB: universalizing approximate query processing. In *SIGMOD*, 2018.
[27] Y. Park, A. S. Tajik, M. Cafarella, and B. Mozafari. Database Learning: Towards a database that becomes smarter every time. In *SIGMOD*, 2017.
[28] A. Pol and C. Jermaine. Relational confidence bounds are easy with the bootstrap. In *SIGMOD*, 2005.
[29] D. N. Politis and J. P. Romano. Large sample confidence regions based on subsamples under minimal assumptions. *The Annals of Statistics*, 1994.
[30] 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.
[31] H. Su, M. Zait, V. Barrière, J. Torres, and A. Menck. Approximate aggregates in oracle 12c, 2016.
[32] S. Wu, B. C. Ooi, and K.-L. Tan. Continuous Sampling for Online Aggregation over Multiple Queries. In *SIGMOD*, pages 651–662, 2010.
[33] K. Zeng, S. Gao, J. Gu, B. Mozafari, and C. Zaniolo. ABS: a system for scalable approximate queries with accuracy guarantees. In *SIGMOD*, 2014.
[34] K. Zeng, S. Gao, B. Mozafari, and C. Zaniolo. The analytical bootstrap: a new method for fast error estimation in approximate query processing. In *SIGMOD*, 2014.