Build your own OctopusDB: Blinktopus Edition

Pavlo Shevchenko, Ali Hashaam, Guzel Mussilova, Ali Memon
Otto-von-Guericke-University, Magdeburg
firstname.lastname@st.ovgu.de

***Abstract*—What is this paper about?**

I. Introduction

Over the last decades we are witnessing that modern enterprises need to pick only specialized DBMSs(e.g. OLAP, OLTP, streaming systems and etc.) each tailored to their specific use-case. Consequently, it leads to additional costs in terms of licensing, maintenance, integration and man-hours for DBAs. Although, it is affordable for some companies, to adapt these integrated solutions to constantly changing workloads and requirements, it may still be a challenging and non-trivial task to achieve. Thus, in order to cope with these problems an implementation of a new all-purpose system could be a perfect solution.

Nowadays there exists a great variety of systems that claim to solve the aforementioned problems and yet their cost might be quite prohibitive. Some traditional DBMSs(e.g., Microsoft SQL Server, Oracle, ...) have already included the support of both analytical (OLAP, which is characterized by long-running queries over all the values of few columns) and transactional (OLTP, characterized by short-lived transactions that affect all attributes of few rows) workloads. Meanwhile, in the last 15 years these systems have been observed to be inefficient for new memory-rich architectures. As a consequence, exploiting the benefits from larger memory new DBMSs have been proposed, which have a simpler architecture than traditional disk-based ones and already proved to be more efficient than those. Among these recent solutions are the column-stores(e.g., C-Store[13], MonetDB[8], ...) and the row-stores(e.g., Hekaton[7], H-Store[14], MemSQL[15], ...) that are particularly designed for analytical and transactional processing, respectively.

Still, following the *one size does not fit all* observation these systems have mainly specialized either for OLAP or for OLTP[6]. Thus, it has lead the various vendors to try to build the comprehensive solutions, namely Hybrid Transactional/Analytical Processing (HTAP) systems(the term HTAP was defined by Gartner[17]). Some examples including SAP HANA[9] which has the engines that are optimized for OLAP workloads, at the same time it also supports ACID transactions. HyPer[10] is another example, which has a hybrid approach that uses memory snapshots based on process forking. Other examples include Peloton[11], OctopusDB[1] and SnappyData[16] that also belong to HTAP systems.
One of the solutions that most radically departs from existing architectures was proposed by Jens Dittrich and Alekh Jindal - a new type of database system, coined OctopusDB[1]. By

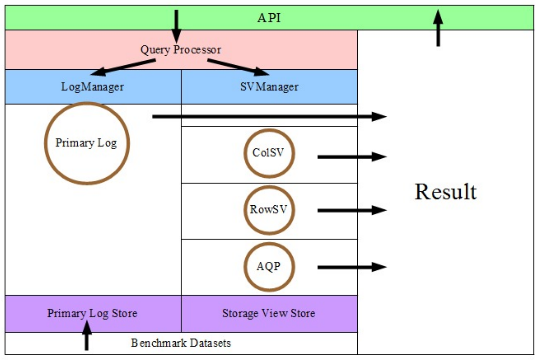

Fig. 1. Blinktopus Architecture.

dynamically mimicking several types of systems(OLAP, OLTP, Hybrid of OLAP and OLTP and etc.) and using logs as a primary storage, OctopusDB shows a considerably better performance. Moreover, depending on the use-case it may also emulate data stream management systems by replacing some of the stores with streaming windows.

Another important goal of HTAP systems(aside from maintaining different system components compatible in order to create an illusion of a single system) is to support OLAP queries for analysis over real-time data. The fact that HTAP systems might reduce the need for transforming data from one system to another(via ETL), seems like a good step towards the support of real-time data. We believe that the exploration of the techniques related to more interactive queries, can contribute to the real-time characteristics of HTAP systems. Among the techniques that can handle more interactive queries Approximate Query Processing(AQP) have recently gained a substantial attention. By processing the compact synopsis of the data rather than the whole dataset, the methods for AQP are often the only viable solution to provide interactive response times when exploring massive datasets and handling high speed data streams[12]. Several successful examples(e.g. BlinkDB, SnappyData on Facebook's Presto) have already proved that there are benefits to be gained by integrating approximation features into existing DBMSs[5].
In this paper we want to evaluate the role that AQP can play as an architectural addition, in a HTAP system like Octopus DB, for facilitating real-time queries on the latest data. We believe that when it is plausible for a system to retrieve results approximately rather than exactly, AQP could be a reasonably good fit to further improve the performance of query processing (especially, OLAP queries). It could also

enhance HTAP by answering the OLAP queries over new data that even has not been included yet and when it is guaranteed that a given amount of incoming data will not change the error of an estimation. Combining OctopusDB and AQP techniques could be feasibly good solution for improving HTAP *freshness* and the performance of our system. In this paper we provide an early evaluation on these aspects.

Our contributions are:

- We review the ideas of OctopusDB and AQP along with the concepts of AQP main data synopses(Section 2).
- We propose a novel concept of a new system called Blinktopus and explain why exactly one type of AQP data synopsis was chosen for our system(Section 3).
- We provide an experimental evaluation on the benefits of Blinktopus's functionality based on the results of the tests performed on the Storage Views(SVs) with the different physical layouts, namely LogSV, ColSV and RowSV(Section 4).
- We discuss related work (Section 5) and future directions for our research(Section 6).

II. Fundamentals

Firstly, we discuss the core idea of OctopusDB, its motivation and architecture(Subsection A). Afterwards we explain the main concept of AQP and elaborate AQP main data synopses such as samples, histograms, sketches, wavelets and etc(Subsection B).

A. OctopusDB

OctopusDB is one of the representatives of *one-size-fits-all* architecture systems. It builds upon 2 ideas:
- Logs as a primary structure (which is similar to Samza[18], and other streaming systems).
- A storage engine with a programmable interface that allows users to specify how data will look like (e.g., RodentStore[23]).
All data in OctopusDB is stored in the central log named *primary log*. The data is being collected into the log through the creation of logical log-entries of the insert/update operations, each record is identified by internally unique log sequence number *lsn*. OctopusDB exploits the write-ahead loging(WAL) protocol and stores the primary log on durable storage(HDD or SSD). Depending on the workload, a so-called Storage View (SV) can represent the entire central log or some of its parts in different physical layouts(e.g., row-store, column-store, PAX, index etc.). For instance, for OLTP queries OctopusDB can create Row SV, Col SV - for OLAP. Moreover, it can optionally decide to create other materialized view such as Index SV or even to mimic streaming systems. At the same time, primary log is another SV for OctopusDB.

Another component of Octopus DB called *Holistic Storage View Optimizer* solves a non-trivial optimization task of *storage view selection*(i.e., to determine automatically which type of SV is proper to be created). It maintains all SVs including primary log as well as it is responsible for creation and maintenance of secondary SVs. By means of scanning the indices and whole tables, later depending on cost model the optimizer resolves either to create new SV or to maintain an existing one or even to scan the log when none of SVs can answer the query. Furthermore, by applying a transformation cost model it might transform different SVs one into another. Additionally, it can remove from system all SVs. Thus, based on the workload and arbitrary occurring use-cases the holistic SV optimizer operates a *storage view lattice*(i.e., the dependency graph between SVs), within the Storage View Store of the OctopusDB.

OctopusDB can be recovered by easily copying the primary log from durable storage to main memory. Meanwhile all SVs that were in the OctopusDB before the system crash will be re-created.

Furthermore, OctopusDB supports ACID. For instance, *Consistency* can be ensured by validating the set of integrity constraints at commit time and *Durability* holds because OctopusDB keeps WAL. The *Isolation* algorithm of OctopusDB is represented via optimistic concurrency control. Whose basic concept is to store all committed or uncommitted changes in the primary log and to write only committed data in secondary SVs. Committed data is available for *read* by uncommitted transactions from log or any secondary SV. Moreover, the latter modifications are possible by adding records to the log but these data will not be further propagated to the secondary SVs for a while. To that end, *Atomicity* is guaranteed by storing in SVs only committed transactions for their latter considerations by other transactions.

B. Approximate Query Processing

It is already evident that nowadays an enormous amount of data is being generated, processed and stored. Even the fastest systems can get stuck for hours in answering simple queries and this response time is less than satisfactory for users and applications. At the same time, in many cases(e.g., a/b testing, exploratory analytics, big data visualization and etc) providing the exact answers to a user query is not vitally important as long as estimations can result in the right decisions. AQP methods can achieve interactive response times(e.g., sub-second latencies) over a massive amount of data by processing only small fraction of the relevant records.

AQP systems differ according to their methodology and design options. AQP operates the datasets through its predefined types of summaries(e.g., *samples, histograms, wavelets, sketches* and so on) that capture main features of initial data while using less space. Such aspects as accuracy, range of applicability, space and time efficiency, error bounds on approximate answers strongly depend upon the chosen types of data synopses and their parameters. Most AQP systems that tailored for accelerating a diverse predefined set of queries, exploit sketches, wavelets and histograms, while sample-based approximations are usually used by all-purpose AQP systems. As a rule of thumb, AQP systems that combine several data synopses seem to demonstrate considerably efficient performance. In addition, another issues in terms of data summaries(i.e., offline or online data summarization, various storage and maintenance strategies and etc) also need to be taken care of.

Samples perhaps are the most researched and extensively implemented type of data summaries. Their prevalence was induced by many reasons. As one the oldest concepts in statistics, there exists a great variety of schemes to extract and maintain samples of data varying in precision and accuracy, such as *Bernoulli sampling, stratified sampling, random sampling with and without replacement* and others.
By using the same schema as the original relation, most queries can be performed over a sample(i.e. small "representative" subset of data) with slight or no alterations to existing system, so they can answer the widest range of queries. Accordingly, an immense diversity of queries can be evaluated using sampling methods, performing MIN, MAX, top-K and COUNT DISTINCT queries on a sample is quite impractical. At the same time, by not exploiting *AVI(Attribute Value Independence)* assumption, most sampling algorithms support high quality selectivity estimation[19].

Moreover, sampling-based approximations do not suffer from "curse of dimensionality", i.e., their accuracy does not deteriorate with the increasing number of data dimensions. However, some samples are not suitable for handling outliers in skewed distributions.

Finally, adding the estimation from several samples can incrementally enhance an imprecise estimate of a query result in interactive exploration of large dataset in "Online aggregation" algorithms. Notwithstanding, when the data is constantly updated in the massive data streams and the majority of future queries are not known in advance, it is crucial to make certain of samples being kept optimal and up to date.

Wavelets are another means of data synopses utilized by AQP systems over large datasets. The core idea of wavelet-based approximations is to transform the input relation in order to acquire a compact data summary that will consist of a small set of wavelet weights or coefficients. While by capturing significant features of the massive dataset wavelets can ensure substantial data compression, they as histograms have the same "curse of dimensionality" limitation. Though this issue can be tackled by efficiently built wavelet decomposition, most implementations revolve around the one-dimensional cases.

Furthermore, an appropriately-defined AQP algebra, which manages the domain of wavelet coefficients, assures answering range and more general SPJ(select, project, join) queries. However, the error guarantees provided by wavelets seem not to be always meaningful in the context of AQP[22].

As an example of the wavelet synopsis *Haar Wavelet Transform(HWT)* is probably conceptually the easiest and therefore the most widely implemented wavelet transformation. Based on recursive pairwise averaging and differencing the resulting wavelets are straightforwardly computed and have been observed to show practically acceptable performance for numerous number of applications(e.g., image editing, querying to OLAP and streaming-data).

Histograms are another type of data synopses for summarizing the frequency distribution of an attribute or sets of attributes. Basically, a histogram groups input data values into subsets(i.e., "buckets") for each bucket it computes a small summary statistics in order to use it further for approximate reconstruction of the data in the bucket. In designing histograms the following aspects such as *partition rule, construction algorithm, value approximation, frequency approximation, error guarantees* are to be carefully considered[20].
Partition rule contains:
- *Partition class* displays whether any restrictions on the buckets exist. For a serial class depending on some parameter(the next characteristic) the buckets are needed to be non-overlapping, while end-biased subclass needs only one non-singleton bucket.
- *Sort parameter* is a parameter whose value for every element in the data distribution is obtained from the corresponding attribute value and frequencies. Attribute value(V), frequency (F) and area (A) are some examples of sort parameters.
- *Source parameter* uses the property of the data distribution and along with the next characteristic identifies a unique partitioning. Spread(S), frequency (F) and area (A) are the most widely exploited source parameters.
- *Partition constraint* is used as a mathematical constraint for a source parameter which solely determines a single histogram within its partition class. There have been defined few partition constraints such as equi-sum, v-optimal, max-diff and compressed.
Construction algorithm builds histograms in compliance with the provided partition rule. Sometimes for the same class histogram there might exist several construction algorithms which differ in efficiency.
Independently of the histogram partition rule *value approximation* shows how attribute values are approximated in the bucket. The continuous value assumption and uniform spread assumption imply that the values are uniformly spread within the certain range in the bucket.
Frequency approximation indicates how frequencies are approximated within the bucket range. The uniform distribution assumption is the main method which assumes that all elements in the bucket are the same and equal to mean value of the actual frequencies.
Based on the information maintained in the histogram *error guarantees* are the upper bounds on the errors of estimates generated by a histogram.

As a result of an active research in the area of histograms over the last decades, a great diversity of histograms schemes have been proposed. They vary in the choice of the buckets, statistics stored per bucket, intra-bucket approximation scheme, classes of queries to be supported and how estimations are extracted. As long as they also barely support more than one data dimensions, one-dimensional histogram is the most preferable type.
The simplicity in implementation and interpretation contributed to the popularity of *equi-width* and *equi-depth* his-

tograms. Equi-width histograms do not overlap among the ranges of attribute values within the bucket and independent of their value frequencies have the same range size (or the number) of values in every bucket[21]. Since they store a greater amount of information than trivial ones, therefore often demonstrate better estimations. The reverse of equi-width histograms - equi-depth (or *equi-height*) class of histograms has the same the sum of the frequencies of the attribute values with respect to each bucket, independent of the range size(i.e., the number) of these values. According to the studies of Piatetsky-Shapiro and Connell who also gave this class of histograms its name, equi-depth histograms have lower worst-case and average error values for a diverse set of selection queries than equi-width histograms. Beyond aforementioned types, more others have been introduced such as *serial*[21], *end biased* and *high biased*, *maxdiff* and other generalizations.

Histograms efficiently answer range-sum queries and their variations. Furthermore, they can be utilized to approximate more general classes of queries(e.g., aggregations over joins), as well as real-valued data and set-valued queries.

Since histograms can be incorporated into existing database system without obtaining an additional storage overhead, they are mainly exploited by query optimizers of almost all DBMSs to produce cost and selectivity estimations of various query plans. However, wrong estimations in query optimizers continue to be an important problem in database systems.

III. Blinktopus

In this section we introduce a novel concept of our system called Blinktopus and give an explanation for the chosen type AQP data synopsis implemented in Blinktopus.

A. Architecture

IV. Experimental Part

Evaluation of the test results:

1. Comparison of runtimes of different types of SVs. 2. Comparison of the diff SVs by means of percentage representation.

V. Related Work

Some related work has to be mentioned here.

VI. Conclusions

Sum up what happened and provide future work

Acknowledgement

Thanks to Gabriel(DBSE). Also we would like to thank Jindal, Dittrich and Mozafari for their awesome ideas that really inspired this project.

References

[1] Dittrich, Jens, and Alekh Jindal. ”Towards a One Size Fits All Database Architecture.” CIDR. 2011.

[2] Jindal, Alekh. ”OctopusDB: flexible and scalable storage management for arbitrary database engines.” (2012).

[3] Jindal, Alekh. ”The mimicking octopus: Towards a one-size-fits-all database architecture.” VLDB PhD Workshop. 2010.

[4] Mozafari, Barzan. ”Approximate query engines: Commercial challenges and research opportunities.” SIGMOD, 2017.

[5] Mozafari, Barzan, and Ning Niu. ”A Handbook for Building an Approximate Query Engine.” IEEE Data Eng. Bull. 38, no. 3 (2015): 3-29.

[6] M. Stonebraker and U. Cetintemel. ”One Size Fits All”: An Idea Whose Time Has Come and Gone. In ICDE, pages 211, 2005.

[7] C. Diaconu, C. Freedman, E. Ismert, P.-A. Larson, P. Mittal, R. Stonecipher, N. Verma, and M. Zwilling. Hekaton: SQL Servers memory-optimized OLTP engine. In SIGMOD, pages 12431254, 2013.

[8] P. Boncz, M. Zukowski, and N. Nes. MonetDB/X100: Hyper-Pipelining Query Execution. In CIDR, 2005.

[9] F. Frber, N. May, W. Lehner, P. Groe, I. Mller, H. Rauhe, and J. Dees. The SAP HANA Database An Architecture Overview. IEEE DEBull 35(1):2833, 2012.

[10] A. Kemper and T. Neumann. HyPer A Hybrid OLTP'&'OLAP Main Memory Database System Based on Virtual Memory Snapshots. In ICDE, pages 195206, 2011.

[11] A. Pavlo, J. Arulraj, L. Ma, P. Menon, T. C. Mowry, M. Perron, A. Tomasic, D. V. Aken, Z. Wang, and T. Zhang. Self-Driving Database Management Systems. In CIDR, 2017.

[12] Cormode, Graham, Minos Garofalakis, Peter J. Haas, and Chris Jermaine. ”Synopses for massive data: Samples, histograms, wavelets, sketches.” Foundations and Trends in Databases 4, no. 13 (2012): 1-294.

[13] Mike Stonebraker, Daniel Abadi, Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amerson Lin, Sam Madden, Elizabeth O'Neil, Pat O'Neil, Alex Rasin, Nga Tran and Stan Zdonik. ”C-Store: A Column-oriented DBMS.” VLDB, pages 553-564, 2005.

[14] H-Store. http://hstore.cs.brown.edu/

[15] MemSQL. http://www.memsql.com/

[16] SnappyData. https://www.snappydata.io/

[17] https://www.gartner.com/doc/3179439/predicts–inmemory-computingenabled-hybrid

[18] https://www.confluent.io/blog/turning-the-database-inside-out-with-apache-samza/

[19] B. Babcock and S. Chaudhuri. Towards a robust query optimizer: A principled and practical approach. In SIGMOD, 2005.

[20] Y. E. Ioannidis, The history of histograms (abridged), in Proceedings of the International Conference on Very Large Data Bases, pp. 1930, 2003.

[21] Y. E. Ioannidis, V. Poosala, ”Histogram-Based Solutions to Diverse Database Estimation Problems.” IEEE Data Eng. Bull. 18 (1995) 10-18.

[22] S. Guha and B. Harb. Wavelet synopsis for data streams: minimizing non-euclidean error. In KDD, 2005.

[23] Philippe Cudr-Mauroux, Eugene Wu, and Samuel Madden. The Case for RodentStore: An Adaptive, Declarative Storage System. In CIDR. www.cidrdb.org, 2009.