Flying KIWI: Design of Approximate Query Processing Engine for Interactive Data Analytics at Scale

Sung-Soo Kim, Taewhi Lee, Moonyoung Chung and Jongho Won
Electronics and Telecommunications Research Institute (ETRI)
218 Gajeong-ro, Yuseong-gu, Daejeon
South Korea
{sungsoo, taewhi, mchung, jhwon}@etri.re.kr

ABSTRACT

This paper introduces the design of hybrid SQL-on-Hadoop system, which supports *dual-mode* (interactive and deep) analytics. We present an architecture of approximate query processing engine using *horizontal* and *vertical* sampling of the original database for interactive big data analytics. A key novelty of our approach is that we can support interactive analytics as well as deep analytics at scale in the *unified* manner.

CCS Concepts

•**Data management systems engines** → Parallel and distributed DBMSs; •**Parallel and distributed DBMSs** → Relational parallel and distributed DBMSs;

Keywords

SQL-on-Hadoop, Approximate Query Processing; Big Data Analytics; Random Sampling

1. INTRODUCTION

A *data warehouse* (DW) is the collection of processes and data whose overarching purpose is to support the business with it analysis and decision-making. The term "Big Data" refers to the continuing massive expansion in the data *volume* and *variety* as well as the *velocity* and *veracity* of data processing. SQL-on-Hadoop is a class of "Big Data" analytics systems that combine established SQL-style querying with Hadoop-based data warehouse. SQL-on-Hadoop systems include Impala, Presto, Tajo, Impala, Drill, Hadapt, HAWQ, BigSQL, Stinger, and Hive on Tez [4].

For *interactive queries* that require a few seconds (or even milliseconds) of response time, MapReduce (MR) is the wrong choice, since MR is often suitable for *batch processing*. In order to provide an effective environment for big data analysis, we believe that SQL-on-Hadoop systems will need to support both *interactive* analytics and *deep* analytics based on massively *batch* processing. *KIWI* is a SQL-on-Hadoop system, which runs on hundreds of machines in existing Hadoop cluster. It is decoupled from the underlying storage engine, unlike traditional database management systems.

BIGDAS 2015 Jeju Island, Republic of Korea

DOI: 10.475/123_4

Given a database $\mathcal{D}$ and a query $\mathcal{Q}$, let $\mathcal{Q}(\mathcal{D})$ denote the exact answer of evaluating $\mathcal{Q}$ on $\mathcal{D}$. In approximate query processing (AQP) [1, 2], an approximate answer can be obtained by (i) extracting from $\mathcal{D}$ a random sample $\mathcal{D}_s$, (ii) evaluating a potentially modified version of $\mathcal{Q}$ (say q) on $\mathcal{D}_s$, and (iii) using $q(\mathcal{D}_s)$ with error bars as an approximation of $\mathcal{Q}(\mathcal{D})$. Generally, error estimation in AQP can be categorized into two main approaches, such as *closed-form* and *bootstrap* [3]. Closed-form error estimation is often used for common aggregate functions in a SQL query, such as `COUNT, SUM, AVG`, and so on. Bootstrap consists in a simple Monte Carlo procedure, however, this requires the computational overhead. *Analytical bootstrap* has been proven equivalent to the simulation-based bootstrap, but avoids the overhead.

The ultimate goal of our work is to provide a SQL-on-Hadoop system, which can support interactive analytics as well as deep (batch) analytics. *Flying KIWI* is an AQP engine in order to support interactive analytics. Flying KIWI provides an approximate answer with an error bar, which is calculated by error estimation.

Main contributions: The contributions of our work can be summarized as follows.

- *Dual-Mode Analytics*: The proposed SQL-on-Hadoop system supports *interactive* analytics as well as MR-based *batch* processing in the unified KIWI architecture.
- *Approximate Query Processing*: We introduce a progressive error estimation method in order to support online aggregation using the *Flying KIWI*.

2. SYSTEM ARCHITECTURE

This section focuses on the overall architecture for approximate query processing engine and describes two main components in the system architecture. Figure 1 shows the high-level architecture of our system, which can be divided into two main blocks, such as *query compilation* block and *query execution* block.

Query Compilation Block: This block performs transparently compiling, checking and rewriting the query to support error estimation using closed forms or analytical bootstrap. The compiler generates a query plan expressed in relational algebra from the a given SQL query. The query rewriter takes into consideration the user-specified quality measures, and rewrites the plan into a new logical query plan according to the logical optimization rules.

Sampling refers to the commonly used technique of evaluating the queries from a small random sample of the original database. In order to create samples, the sampling construction module exploits two sampling methods, such as, *horizontal* sampling (or row sampling) and *vertical* sampling (or column sampling).

Query Execution Block: When a query arrives at runtime, it is re-written to run against the sample tables instead of the original database. The execution block evaluates the query augmented with sample selection operations at runtime, and providing the accuracy measures in user-specified metrics.

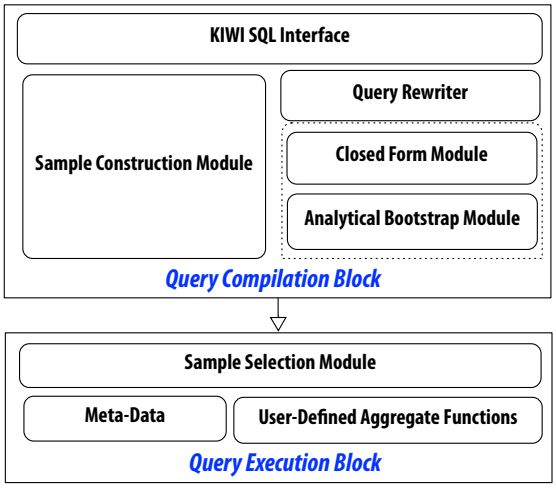

Figure 1: Flying KIWI Architecture.

2.1 Sample Construction and Selection

Given a table T with r rows $R_1, ..., R_n$ and c columns $C_1, ..., C_m$, in horizontal sampling, let $S_h = \{R_i, R_{i+1}, ..., R_{i+l}\}$, where $i \leq i + l \leq r$, denote a *row set* that consists of l rows in T. Our system can construct various row sets according to sampling algorithms such as stratified sampling [3] or Monte-Carlo sampling with user-specified sampling rate. In vertical sampling, let $S_v = \{C_j, C_{j+1}, ..., C_{j+k}\}$, where $j \leq j + k \leq c$, denote a *column set* that consists of k columns in T. A query q often need to scan fully or partially all data items in a row set or a column set S_q of T. If data items in S_q have been materialized, for q, need to scan only materialized items instead of full table T. In case of column sampling, because the number of columns in S_q is often much smaller than c, scanning would be done much faster.

When queries are issued at runtime, the query execution block rewrites the queries to run against sample tables rather than the original tables referenced in the queries. Let $\xi(T, S)$ be the memory space needed to store all data items in a column set S of a table T. Let φ be the storage system's space limit for materialized column sets. Let ω be possible column sets of table T. The sum of the memory space of possible column sets, $\sum_{\forall S_i \in \omega} \xi(T, S)$ is exponentially large. Let Q_p is the set of queries issued in the past. Let $\mathcal{V}(T, S_i)$ be the value obtained for future queries if S_i is materialized.

Problem Definition: Given a table T and a query Q, find a collection of optimal column sets, $S_{opt} = \{S_1, ..., S_k\}$ consisting of k column sets, such that $\sum_{\forall S_i \in \omega} \xi(T, S_i) \leq \varphi$ and $\mathcal{V}_{opt} = \sum_{\forall S_i \in \omega} \mathcal{V}(T, S_i)$ is maximized.

Our Approach: From the set of historical queries Q_h extracts a set of distinct column sets S_a that appear in Q_h. $\forall S_i \in S_a$, compute $\xi(T, S_i)$, remove from S_a if $\xi(T, S_i) > \varphi$. $\forall S_i \in S_a$, compute the appearance frequencies $f(S_i)$, remove from S_a if $\xi(T, S_i) > \varphi$. Let n be the number of column sets in S_a. For an arbitrary column set S, $\xi(T, S)$ can be approximated as:

$$\xi(T, S) = r \times \sum_{i=1}^{|S|} \mathcal{I}(C_i) \quad (1)$$

where r denotes the number of rows in T, $|S|$ denotes the number of columns in S and $\mathcal{I}(C_i)$ denotes the average size of a data item in C_i (e.g., if data type of C_i is double, then $\mathcal{I}(C_i)$ is 8 bytes).

Algorithm 1 Find optimal column sets (S_{opt}).

```
1:  procedure FindOptimalColumnSets(S_a, Q)
2:      List<S> L_s = constructColumnSets(S_a, Q);
3:      L_s.sortByAppreanceFrequency(DESENDING);
4:      for each node S_j ∈ L_s do
5:          if Σ_{∀S_i ∈ S_opt} ξ(T, S_i) + ξ(T, S_j) > ω then return
6:          else
7:              S_opt.add(S_j);
8:              S_a.remove(S_j);
9:          end if
10:     end for
11: end procedure
```

The appropriate sample table(s) to use for a given query Q are determined by comparing Q with the metadata annotations for the samples. **Algorithm 1** shows how optimal column sets, S_{opt}, can be evaluated progressively for a given query Q. The time complexity of this algorithm is $O(n \log n)$, where n is the size of the database.

3. CONCLUSION

We present a hybrid SQL-on-Hadoop system architecture, which supports approximate query processing. Our main contribution in this work has been to propose a new unified approach for supporting *dual-mode* (interactive and deep) analytics at scale.

Our work concludes with the following take-away messages: (1) It is beneficial to have an *unified* query processing engine in the KIWI SQL-on-Hadoop system, (2) *Flying KIWI* is a general purpose system that implements error estimation features of approximate answers for interactive analytics, and (3) the sampling methods are intuitive to use.

Acknowledgments

This work was supported by ETRI R&D program ("*Development of Big Data Platform for Dual Mode Batch Query Analysis*, 15ZS1400") funded by the government of South Korea.

4. REFERENCES

[1] 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 *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 481–492. ACM, 2014.

[2] 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 *Proceedings of the 8th ACM European Conference on Computer Systems*, EuroSys '13, pages 29–42. ACM, 2013.

[3] S. Chaudhuri, G. Das, and V. Narasayya. Optimized Stratified Sampling for Approximate Query Processing. *ACM Trans. Database Syst.*, 32(2), June 2007.

[4] A. Floratou, U. F. Minhas, and F. Özcan. SQL-on-Hadoop: Full Circle Back to Shared-nothing Database Architectures. *Proc. VLDB Endow.*, 7(12):1295–1306, Aug. 2014.