Poster Paper

Sweet KIWI: Statistics-Driven OLAP Acceleration using Query Column Sets

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

KIWI is a SQL-on-Hadoop system enabling batch and interactive analytics for big data. In database systems, materialized views, stored pre-computed results for queries, are one of the most commonly used techniques to improve the query processing speed. However, the key challenge in using materialized views is maintaining their freshness as base data changes. This paper introduces a new approach for accelerating OLAP query processing using query workload statistics and *query column sets* instead of materialized views. We present an architecture of SQL-on-Hadoop system using query column sets of original tables in database. The experimental results demonstrate that our system can provide improved performance by 1.77x on average in terms of TPC-H query processing.

Keywords

Column Sets; SQL-on-Hadoop; OLAP; Big Data Analytics

1. INTRODUCTION

Data warehouse (DW) on Hadoop has rapidly gained popularity and is now being used intensively by *business intelligence* (BI) users in enterprises as well as scientific institutions. SQL-on-Hadoop (SoH) is a class of "Big Data" analytics systems that combine established SQL-style querying with Hadoop-based data warehouse [4]. Most of the Online Analytical Processing (OLAP) query workloads in BI applications are long-running batch workloads that are read-mostly and run repeatedly [1]. *Materialized views* are widely used to facilitate fast queries on large datasets. However, one of the most challenging aspects of using materialized views is maintaining their freshness as base data changes.

Sampling refers to the commonly used technique of evaluating the queries from a small random sample of the original database [1, 3]. Typical OLAP query processing approaches exploit two sampling methods to construct the samples, such as, *horizontal* sampling (or row sampling) and *vertical* sampling (or column sampling) [2]. 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. 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. In our work, we select vertical sampling method to accelerate OLAP query processing on large-scale dataset.

KIWI[1] is the proposed SoH system, which runs on hundreds of machines in existing Hadoop cluster. The ultimate goal of our work is to provide a SoH system, which can support interactive analytics as well as deep (batch) analytics. *Sweet KIWI* is a statistics-driven query processing engine in order to support deep analytics at scale.

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

- *Dual-Mode Analytics*: The proposed SoH system supports *interactive* analytics as well as MapReduce-based *batch* processing in the unified KIWI architecture.
- *Statistics-Driven OLAP Acceleration*: We introduce a OLAP query acceleration method using query column sets to support deep analytics at scale.

2. SYSTEM OVERVIEW

This section focuses on the overall architecture for query processing engine using query column set and describes two main components in the proposed system architecture.

Query Workload Analyzer: One common assumption about query workloads is that future queries will be similar to historical queries [2]. This component is responsible for analyzing a set of historical query workloads to classify the frequently used queries in the past. In order to construct the query column sets, we extract the metadata for query column sets over the entire original tables. The set of query column sets are updated both with the arrival of new data, and when the query workloads changes.

Query Column Sets Constructor: This block maintains the query column sets as cache tables, and manages the mapping data between the original tables and the cache tables. Query column sets are created, and updated based on statistics collected from the base data and historical queries. When a query arrives at runtime, it is re-written to run against the cache tables instead of the original tables. The KIWI workload manager evaluates the query augmented with cache table selection operations at runtime.

Figure 1 illustrates query processing workflow in our system using the query column sets. In the first step, the query workload

[1]KIWI is the abbreviation for "Key Impact on data Warehouse Infrastructure", which is our project code name.

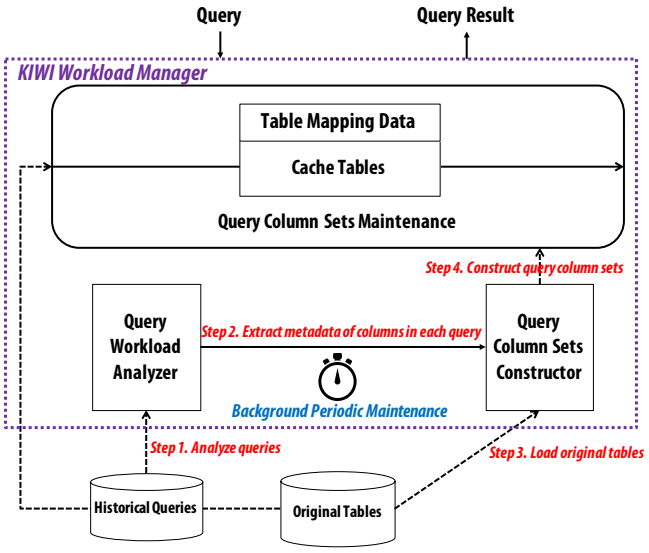

Figure 1: Sweet KIWI Architecture.

analyzer performs historical query analysis. In the second step, it extracts metadata of columns in each query. In the third step, the query column sets (QCS) constructor loads original tables in database to create query column sets. Finally, the QCS constructor inserts the QCS tables and the table mapping data into the database.

2.1 Query Column Sets

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 $\sum_{\forall S_i \in \omega} \xi(T, S_i) \leq \omega$ and $\mathcal{V}_{opt} = \sum_{\forall S_i \in \omega} \mathcal{V}(T, S_i)$ is maximized.

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(DESCENDING);
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
```

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 the memory space $\xi(T, S_i)$, remove from the column set S_a if $\xi(T, S_i) > \omega$. $\forall S_i \in S_a$, compute the appearance frequencies $f(S_i)$, remove from the column set S_a if $\xi(T, S_i) > \omega$. 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** 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. EXPERIMENTAL RESULTS

To evaluate the performance of the KIWI for analytic workloads, we loaded the industry-standard TPC-H data set at scale factor 10 on a node. The server has dual 2.66GHz Intel Xeon CPUs with 128GB RAM, and runs Mac OS X. We compared the wall time of each TPC-H query between original DB and QCS DB.

Table 1: Runtime for TPC-H queries (unit: seconds)

Query	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8
Original DB	137	18.6	130	181	28	174	91	176
QCS DB	68.8	11.6	67.9	123	17	84.4	62	87.9

The experimental results demonstrate that our system can provide improved performance in terms of query processing speed on TPC-H 10GB dataset. There were performance improvements of 1.77x on average compared to the original DB as shown in Table 1.

4. CONCLUSION

We present a statistics-driven OLAP acceleration in SQL-on-Hadoop system architecture for data-intensive applications. 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) *Sweet KIWI* is a general purpose system that constructs the query column sets of historical queries for deep analytics, and (3) the vertical sampling method using *query column sets* is intuitive to use.

Acknowledgments. This work was supported by ETRI R&D program ("*Development of Big Data Platform for Dual Mode Batch · Query Analytics*, 16ZS1400") funded by the government of South Korea.

5. 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.