Pixels: Multiversion Wide Table Store for Data Lakes

Haoqiong Bian
bianhaoqiong@gmail.com

ABSTRACT

Data lake can be considered as the place where massive data is continuously ingested, stored, evolved and consumed. As a storage, it is decoupled from computation and shared by various analytic applications. In practice, a data lake can be built upon massively scalable distributed file system (e.g. HDFS) or cloud storage (e.g. Amazon S3).

In data lakes, wide tables with hundreds or even thousands of columns are often seen. They can be naturally wide [2, 3, 4] or denormalized from non-wide tables [6]. Wide table is inevitable for: (1) it eliminates distributed joins, which are the major bottleneck of performance and scalability in distributed environment. (2) it is flexible to structure the raw data. However, there are still some critical problems to be addressed: (1) a storage model should adapt efficiently to the rapid evolution of application needs in data lakes. (2) wide table may change access pattern in current column stores from sequential dominated to random dominated [4], so that dedicated layout optimization is needed.

Focusing on these problems, we present the idea of building a multiversion store for data lakes which is optimized for wide tables. It can provide more than one order of magnitude improving query performance on wide tables compared to state-of-the-art column stores. Also, it supports efficient evolution of both metadata and user data according to the changing application needs. Although optimized for wide tables, it is also suitable for normal non-wide tables.

Data Versioning and Evolution

We are inspired by multiversion schema warehouse [5] in supporting efficient evolution of wide tables. In our solution, we consider *metadata as a kind of data* and apply a uniform versioning mechanism on both. Each group of data records (a.k.a. row group) or version of metadata is assigned a global ascending timestamp (version) at birth.

On a table, inserts, updates and deletes of data records are considered as new data version and packed into row groups. For metadata, most updates (such as adding or dropping columns, updating column order) are *logical*. They only write a new version of metadata which is only applied to newborn row groups. Queries only see the latest version by their start timestamps. During query execution, each row group can find its corresponding version of metadata and get interpreted. *Physical* updates such as converting a column type from varchar to integer may need extra verification and transformation. But we can still do these along query execution if filling corrupted values by defaults is acceptable.

By such versioning mechanism, we can support not only efficient data evolution but also ACID transactions on row group level. We also plan to build a catalog service based on it to ease data sharing between applications.

Storage Optimization

Another core target of wide table store is to provide high read performance while guarantee timely data ingestion. We have finished a prototype named Pixels for this target and open sourced it at https://github.com/pixelsdb/pixels/.

In Pixels, we realize the aforementioned goal based on two methods: (1) compact 2^n row groups into each storage block and allow each query task to read a contiguous range of 2^m row groups from a block ($0 \leq m \leq n$ and $m, n \in N$, m is variable for each query) according to its access pattern, so that we have more opportunity for data layout optimization without harming data ingestion timeliness and computation parallelism. (2) design a dedicated column cache backed by shared memory for column store, which is more efficient than OS page cache or other general-purpose cache system.

Evaluations results show that under the same workload and environment, Pixels improves query performance by 4x on average using the first method, compared to our previous work [4] on Parquet [1]. By caching 20% data in column cache, the performance can be further improved by 3.7x on average. We have integrated Pixels with mainstream query engines and plan to implement versioning in Pixels.

1. REFERENCES

[1] Parquet. http://parquet.apache.org/.
[2] Sloan digital sky survey. https://www.sdss.org/.
[3] Uber's big data platform: 100+ petabytes with minute latency.
[4] H. Bian, Y. Yan, W. Tao, L. J. Chen, Y. Chen, X. Du, and T. Moscibroda. Wide table layout optimization based on column ordering and duplication. In SIGMOD, 2017.
[5] M. Golfarelli, J. Lechtenbörger, S. Rizzi, and G. Vossen. Schema versioning in data warehouses. In ER, 2004.
[6] Y. Li and J. M. Patel. Widetable: An accelerator for analytical data processing. PVLDB, 7(10), 2014.

This article is published under a Creative Commons Attribution License (http://creativecommons.org/licenses/by/3.0/), which permits distribution and reproduction in any medium as well allowing derivative works, provided that you attribute the original work to the author(s) and CIDR 2020.
10th Annual Conference on Innovative Data Systems Research (CIDR '20). January 12-15, 2020, Amsterdam, Netherlands