Advanced Science and Technology Letters
Vol.86 (Ubiquitous Science and Engineering 2015), pp.5-7
http://dx.doi.org/10.14257/astl.2015.86.02

Column Partitioning to Improve Data Warehouse Query Performance

Jong Wook Kim*, Sae-Hong Cho**, Il-Min Kim***

*Dept. of Media Software at Sangmyung University
20-Gil, Hongji-dong, Jongno-gu, Seoul, Korea
jkim@smu.ac.kr
**Dept. of Multimedia Engineering at Hansung University
389 Samsun-Dong 3-Ga, Sungbuk-Gu, Seoul, Korea
chosh@hansung.ac.kr
**Dept. of Computer Engineering at Hansung University
389 Samsun-Dong 3-Ga, Sungbuk-Gu, Seoul, Korea
ikim@hansung.ac.kr

Abstract. In this paper, we propose a novel data partitioning scheme to improve data warehouse query performance in column-oriented data stores. The proposed partitioning method leverages column access patterns of a given workload in order to find the best multi-column partitions. Our proposed approach is able to compute a set of multi-column partitions that yield the best performance for the given entire query workload, when storage is constrained.

Keywords: Multi-column partition, column store, data warehouse query

1 Introduction

Today's commercial data management systems support creation and use of both indexes and materialized views in order to boost query performance. Indexes and materialized views are physical structures which aim to reduce IO costs [1, 2, 3, 4, 5, 6, 7]. Generally, indexes speed up query processing by avoiding the access to irrelevant data, while with materialized views, the reduced query processing times are achieved by exploiting pre-computed result sets. These techniques have produced impressive results in OLTP-style applications. The same, however, cannot be said for OLAP style applications where it is very common that a query workload contains aggregation operations and nested sub-queries. The use of indexes is limited when the query workload has aggregation operations, because they requires full table scans. Furthermore, the presence of nested sub-queries makes it difficult to rewrite a given query with materialized views because in most commercial systems, a materialized view is not allowed to have nested sub-queries.

ISSN: 2287-1233 ASTL

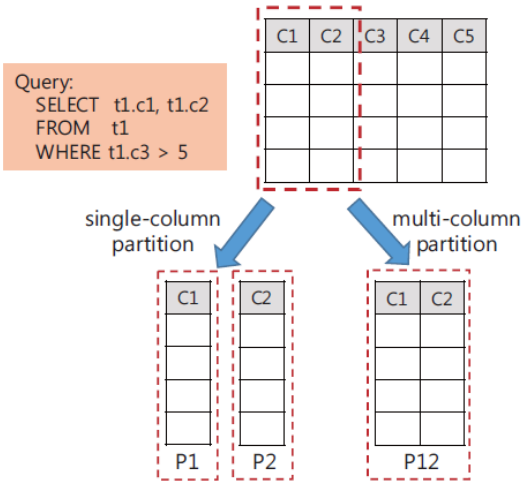

Fig.1. Single-column partition (P1,P2) vs. multi-column partition (P12)

The goal in this paper is to improve data warehouse query performance in column-oriented data stores. As shown in the example in Figure 1, given a user query where both c1 and c2 appear in select-list, using multi-column partition, P12, that consists of c1 and c2 is more efficient strategy than using two separate single-column partitions, P1 and P2. Based on this observation, in this paper, we develop an efficient column partitioning scheme to speed up query processing in column-oriented data stores. In particular, given a storage constraint, our method is able to compute a multicolumn partition configuration which yields the best performance for the given entire query workload. The rest of this paper is structured as follows: In the next section, we present the proposed column partitioning technique for efficiently processing data warehouse query performance in column-oriented data stores. In Section 3, we conclude the paper.

2 Workload-based Column Partitioning in Column-oriented Data Store

In this section, we describe the workload-based partitioning scheme for efficiently processing data warehouse query in column-oriented data stores. As described in Section 1, if a subset of columns in a table are frequently accessed together, query processing time can be significantly reduced by defining and using multi-column partitions. In this paper, we leverage this intuition to develop the column partitioning scheme:

1. Based on a given query workload, first we generate a set of candidate multi-column partitions which potentially are useful for queries.
2. Then, in storage-constrained environments, the next step is to compute the multi-column partition configuration that is a subset of candidate multi-column partitions and yields the best performance for the given entire query workload.

The above two steps need to be performed for each table in a database so that there is one multi-column partition configuration per table. The multi-column partitions identified in this section, along with the single-column partitions, are physically stored on a disk. At query planning time, to access specific table columns referenced by a given query, either single-column partitions or multi-column partitions are selected as the access methods on a cost-basis by the query optimizer. Like indexes and materialized views, the proposed method in this paper incurs overheads in terms of maintenance and disk space. We, however, note that (a) column-oriented storages are most suitable for read-optimized data warehouses, and thus the overhead associated with maintenance can be ignored, and (b) the price of disks has been failing rapidly in these days.

3 Conclusion

In this paper, we proposed a novel workload-based multicolumn partitioning scheme to improve data warehouse query performance in column-oriented data stores. Our proposed approach is able to compute a set of multi-column partitions that yield the best performance for the given entire query workload, when storage is constrained. Future work will include extension of the multi-column partitioning scheme presented in this paper to consider the use of multi-column partitions which are partially referenced by a query.

References

1. Abadi, D.J., Madden, S.R. and Hachem, N.: Column-stores vs. rowstores: how different are they really? In Proceedings of the ACM SIGMOD international conference on Management of data, 2008.
2. Agrawal, S., Chaudhuri, S. and Narasayya, V.R.: Automated Selection of Materialized Views and Indexes in SQL Databases. In Proceedings of the 26th International Conference on Very Large Data Bases, 2000.
3. Kimura, H., Huo, G., Rasin, A., Madden, S. and Zdonik, S.B.: CORADD: Correlation Aware Database Designer for Materialized Views and Indexes. In Proceedings of the 366th International Conference on Very Large Data Bases, 2010.
4. Boncz, P.A., Zukowski, M. and Nes, N.: MonetDB/X100: Hyper- Pipelining Query Execution. In Proceedings of the Conference on Innovative Data Systems Research, 2005.
5. Stonebraker, M., Abadi, D.J., Batkin, A., Chen, X., Cherniack, M., Ferreira, M., Lau, E., Lin, A., Madden, S., O'Neil, E., O'Neil, P., Rasin, A., Tran, N., and Zdonik, S.: C-store: a column-oriented DBMS. In Proceedings of the international conference on Very large data bases, 2005.
6. Lemire, D. and Kaser, O.: Reordering Columns for Smaller Indexes, Information Sciences 181 (12), 2011.
7. Abadi, D.J., Boncz, P.A. and Harizopoulos, S.: Column oriented Database Systems. PVLDB 2(2): 1664-1665 (2009)

