CALF: Comparison of Attribute Layouts on Flash

Satish Kumar Kotha Priyananda Shenoy
Department of Computer Sciences
University of Wisconsin, Madison, WI
{satish,shenoy}@cs.wisc.edu

Abstract

Solid state drives(SSDs) are increasingly being used for database applications, due to their low seek latency and low power consumption. Solid state drives differ fundamentally in their characteristics from disk drives, which mandates that some core database design decisions, such as column layout, needs to take into account the specific characteristics of SSDs for optimal performance. This paper reexamines the question of column layout models for Flash based databases. We propose a flexible data storage model which partitions attributes based on a given workload, taking into account both reads and writes. We evaluate the performance of this intelligent partitioning against NSM and C-Store storage models.

1 Introduction

In recent years, the cost-per-byte of Flash drives has fallen to an extent where it is feasible to build fairly large applications on Flash storage. The lack of mechanical parts, fast random access, low power consumption and durability are some of the features which make it a storage medium of choice. While disk drives will be around for a while, Flash has the potential to replace magnetic disks in most applications.

As we will see in Section 3, Flash drives are inherently different from disk drives in many respects. A significant number of architecture and design decisions behind current database systems have (implicit or explicit) assumptions that work for disk based systems, but not for Flash drives. In this paper, we explore the decision of how columns of a relation are stored in secondary storage, and examine how it can be optimized for Flash databases. Instead of statically deciding column layout, we propose a way of coming up with a layout based on a given workload.

1.1 Organization of the paper

Section 2 briefly introduces the various column-organization alternatives available. Section 3 gives a short introduction to characteristics of Flash devices and how they are different from the drives. Section 4 gives the analytical cost modeling of various operations. Section 5 describes the experimental organization and evaluation. Section 6 describes the results of evaluating the performance of our layout with row- and column- based page layout.

2 Related Work

Column organization techniques can be roughly divided into two groups: *Horizontal* or *row-major* storage models, where all attributes of a tuple are stored together, and *Vertical* or *column-major* storage models in which columns are stored together.

The typical row-major storage model in use is the *N-ary Storage model*(NSM). All the attributes of a tuple are laid out contiguously in a page. Each page maintains a *slot table* which maintains the offset of the beginning of each record. NSM is inefficient when only a few columns of the relation are being accessed, since all the attributes are loaded irrespective of whether they are used or not. CPU Cache performance is also bad, since loading unnecessary data pollutes the cache.

Decomposition Storage Model (DSM) [CK 85] was one of the first column-major layouts. DSM split a n-column relation into n sub-relations, duplicating the primary key or record-id in each sub relation. This method saves on data transfer when only a few columns are accessed, but pays a significant penalty joining the relations when multiple attributes are accessed in the query.

C-Store [SAB05] differs from DSM in that it doesn't explicitly store a record-id per sub relation. C-Store maintains a collection of columns, where an attribute of a column can be duplicated in multiple columns. Different columns can store the same attribute values in different orders, which can help in different queries.

Partition Attributes Across(PAX) [ADH01] improves upon the cache behavior of NSM by splitting a page into minipages, and storing the tuples in column-major order, each column in one minipage. While this improves cache performance, this has no impact on I/O transfer costs, since this just reorganizes data within a page, not across pages.

Data Morphing(DM) [HP 03] improves upon the cache performance of PAX, by making use of locality between attributes to group concurrently-accessed attributes together. As with PAX, DM only optimizes in-page layout, and doesn't help in reducing I/O transfer costs. This paper applies the same approach followed in DM to column layouts across pages to reduce I/O transfer costs as well.

Multiresolution Block Storage Model(MBSM) [ZR 03] extends the PAX model to include multiple pages or blocks. Blocks are arranged to superblocks, which are in turn grouped into Megablocks. Each block corresponds to a column. The relation is partitioned onto multiple superblocks. This approach has good cache performance and (somewhat) takes care of the data transfer problem. This model is no better than column store in systems with inexpensive random reads, since most of the organization is to make use of fast sequential access of disk drives.

There have been comparative studies of row-store vs. column-store organizations. Holloway et al. [HD 08] found that in most read-heavy cases, column stores beat row stores when I/O was the bottleneck. [HBN06] also found similar results, and proposed enhancing the row-storage model with some ideas from column-storage. All these comparisons were for disk drives based databases, hence the results may not hold true for Flash databases.

[SHW08] briefly touches upon storage models for Flash, and mentions that column-major layout is faster than row-major, for read queries. The authors also touch upon the fact that write queries pose a problem for column major layouts, incurring a heavy cost on inserts.

3 Characteristics of Flash

Flash drives have several traits that make them attractive for read-mostly enterprise applications such as web-page serving and search. Flash drives are divided into blocks and each block is further divided into pages. The memory can be read or programmed a byte or word in random access fashion at a time but the unit of erasure is a block. Because of these characteristics, flash drives offer more random read I/Os per second, offer comparable sequential bandwidth, and use a tenth of the power when compared to disks. Flash is also cheaper than DRAM and is non-volatile. Moreover, flash continues to get faster, cheaper, and denser at a rapid pace. However, flash drives are limited by their write endurance. Flash memory cells often wear out after 1000 to 10000 write cycles. Each write cycle requires erasing a super block before writing the actual data. Techniques exist to exploit wear levelling exist to extend the lifetime of the cells but an overhead of remapping fragments is incurred and this technique is useful only when there is a free super block available for the write. Flash memory also allows multiple memory locations to be erased or written in one programming operation and this is another major advantage with respect to disks. Flash memories also have improved on reliability which was one of it's major drawbacks during initial stages.

4 Our Contribution

We plan to re-evaluate various storage models described in Section 2 for flash disks using quickstep, an architecture conscious research project built to investigate and evaluate data storage and query processing techniques. Much work so far has been focused on exploiting random reading offered by flash disks. We also plan to focus on various workloads including writes and analyze relative performance of various storage techniques. The next section describes our cost model to start with and it has following assumptions or limitations which we plan to get rid of as time progresses.

- There are no variable length attributes. For example, if datatype is defined as *varchar(255)*, we assume that it is implemented as fixed size of 255 bytes.
- Database application does not have control on how to write a random page. Choosing an empty block instead of erasing and writing the block could minimize the costs associated. We consider average write cost in all our equations.
- No indexes on any of the columns. We plan to get rid of this assumption as early as possible, since it can have significant effect on the performance equations.

4.1 Cost Modeling

Some definitions:

- R : A relation in the database.
- A : The set of all columns $\{a_1, a_2, \cdots, a_n\}$.
- $sizeof(a_i)$: The maximum number of bytes a column can take.
- G : A *group* is a subset of columns $G \subseteq A$.
- $\mathcal{G}$: A *partition* is a set of disjoint groups $\{G_1, G_2, \cdots, G_k\}$ such that $\bigcup_{\mathcal{G}} G_i = A$.
- $cost_r$: The cost to read a random page.

- $cost_w$: The cost to write a random page.
- *pagesize*: The number of bytes in a page.
- N : number of tuples in the relation.

The costs are assumed to be constant. More details about how these costs are estimated are given in the next section. Note that when $|\mathcal{G}| = 1$, this would reduce to N-ary storage model, and when $|\mathcal{G}| = |A|$, this reduces to column-store model.

The *records per page* for a group measures the number of tuples which can be stored in one page, considering only the columns in that group.

$$rpp(G) = \left\lfloor \frac{pagesize}{\sum_{a \in G} sizeof(a)} \right\rfloor$$

To ensure that each page holds at least one tuple, we can impose the constraint

$$\sum_{a \in G} sizeof(a) \leq pagesize.$$

For a given partition $\mathcal{G}$, the cost of a query is calculated as follows.

- Q is of the form *select * from R*

  $$cost(Q) = cost_r \times \sum_{G \in \mathcal{G}} \left\lceil \frac{N}{rpp(G)} \right\rceil$$

  .

- Q is of the form *select* $x_1, x_2, \cdots, x_p$ *from R*

  $$cost(Q) = cost_r \times \sum_{G \in \mathcal{G}} cost'(G, \{x_1, x_2, \cdots, x_p\})$$

  where,

  $$cost'(G, X) = \begin{cases} 0 & X \cap G = \varnothing \\ \left\lceil \frac{N}{rpp(G)} \right\rceil & otherwise \end{cases}$$

- Q is of the form *select* $x_1, x_2, \cdots, x_p$ *from R where predicate*$(w_1, w_2, \cdots, w_q)$

  Let *row selectivity* of a predicate be defined as

  $$rowsel(p) = Pr_{tuple\ r \in R}\left[p(r) = true\right].$$

  Then the cost of query Q is

  $$cost(Q) = cost_r \times (\sum_{G \in \mathcal{G}} cost'(G, \{w_1, \cdots, w_p\}) + rowsel(p) \times \sum_{G \in \mathcal{G}} cost'(G, \{x_1, \cdots, x_q\}))$$

- Q is of the form *insert into R values*$(v_1, \cdots, v_n)$

  $$cost(Q) = (cost_r + cost_w) \times |\mathcal{G}|$$

- Q is of the form *update table R set* $x_1 = v_1, \cdots$ *where predicate*$(w_1, \cdots, w_q)$ Then the cost of query Q is

  $$cost(Q) = cost_r \times \sum_{G \in \mathcal{G}} cost'(G, \{w_1, \cdots, w_p\}) + cost_w \times rowsel(p) \times \sum_{G \in \mathcal{G}} cost'(G, \{x_1, \cdots, x_p\})$$

- Q is of the form *delete table R where predicate*$(w_1, w_2, \cdots, w_q)$

  $$cost(Q) = cost_r \times \sum_{G \in \mathcal{G}} cost'(G, \{w_1, \cdots, w_p\}) + cost_w \times rowsel(p) \times |\mathcal{G}|$$

The *optimal partitioning* of a relation R for a given workload $\{Q_i\}$ is given by

$$\mathcal{G}^* = argmin_{\mathcal{G}} \sum_{Q_i} cost_{\mathcal{G}}(Q_i).$$

References

[GRA07] Goetz Graefe. The Five-minute Rule: 20 Years Later and How Flash Memory Changes the Rules. In *Proceedings of the Third International Workshop on Data Management on New Hardware*, 2007.

[HD 08] Allison Holloway, David DeWitt. Read-Optimized Databases, In depth. In 34^{th} *International Conference on Very Large Data Bases*, 2008.

[SHW08] Mehul Shah, Stavros Harizopoulis, Janet Wiener, Goetz Graefe. Fast Scans and Joins using Flash Drives. In *Proceedings of the Fourth International Workshop on Data Management on New Hardware*, 2008.

[HP 03] Richard Hankins,Jignesh Patel. Data morphing: An adaptive,cache-conscious storage technique. In *Proceedings of the 29th VLDB Conference*, 2003.

[ZR 03] Jingren Zhou, Kenneth Ross A Multi-resolution Block Storage Model for Database Design. In *Proceedings of the 2003 IDEAS Conference*, 2003.

[SAB05] Mike Stonebraker, Daniel Abadi, Adam Batkin et al. C-Store: A Column-oriented DBMS. In *Proceedings of the 31st VLDB Conference*, 2005.

[CK 85] George Copeland, Setrag Khoshafian. A decomposition storage model. In *Proceedings of the 1985 ACM SIGMOD international conference on Management of data*, 1985.

[ADH01] A. Ailamaki, David DeWitt, Mark Hill, and M. Skounakis. Weaving relations for cache performance. In *Proceedings of VLDB Conference*, 2001.

[HBN06] Alan Halverson, Jennifer Beckmann,Jeffrey Naughton,David DeWitt. A Comparison of C-Store and Row-Store in a Common Framework *Manuscript*, 2001.