Lecture #10: Storage Models and Data Layout

15-721 Advanced Database Systems (Spring 2018)
http://15721.courses.cs.cmu.edu/spring2018/
Carnegie Mellon University
Prof. Andy Pavlo

1 Data Organization

One can think of an in-memory database as a large array of bytes. The schema tells the DBMS how to convert the bytes into the appropriate type. Each tuple is prefixed with a header that contains its meta-data.

Store tuples with fixed-length data makes it easy to compute the starting point of any tuple. Mapping virtual memory pages to database pages.

Memory Pages

- The OS maps physical pages to virtual memory pages.
- The CPU's *memory management unit* (MMU) maintains a *translation lookaside buffer* (TLB) that contains the physical address of a virtual memory page.

Transparent Huge Pages

One potential optimization for an in-memory DBMS is to use "huge" memory pages. The default page size is 4KB. Using larger pages means that it takes fewer TLB entries to map to a larger amount of memory.

Linux supports a *transparent huge pages* optimization that allows applications to automatically take advantage of huge pages.

- All pages are automatically set to either 2MB or 1GB (depending on what the CPU supports).
- Each page has to be a contiguous block of memory. The OS can fall-back to the default 4KB page if a large segment of memory is not available.

Most DBMSs tell you to disable huge pages because they can stall processes that attempt to access a memory page while the OS is reorganizing layout.

2 Data Representation

This determines how a DBMS stores the actual bits for a value in-memory.

All integers are stored in their "native" C/C++ as specified by the IEEE-754 standard [1].

Variable Precision Numbers

- Inexact, variable-precision numeric type that uses the "native" C/C++ types specified by IEEE-754 standard.
- Faster than arbitrary precision numbers because the CPU can execute instructions on them directly.
- Example: `FLOAT`, `REAL`

Fixed Point Precision Numbers

- Numeric data types with arbitrary precision and scale. Typically stored in exact, variable-length binary representation with additional meta-data.
- Used when rounding errors are unacceptable.
- Example: `NUMERIC`, `DECIMAL`

NULL Data Types

Approach #1 – Special Values:

- Designate a value to represent `NULL` for a particular data type (e.g., `INT32_MIN`).
- DBMS needs to make sure that the application does not try to insert a tuple with that value.

Approach #2 – Null Column Bitmap:

- Store a bitmap in the header of every tuple that specifies which of its attributes are `NULL`.
- The length of the bitmap is usually the number of attributes in the tuple (even if those attributes are not null-able).
- This is the most widely used approach in both disk-oriented and in-memory DBMSs.

Approach #3 – Per Attribute Flag:

- Store a prefix in-memory for an attribute that represents whether that value is `NULL`.
- Although the flag only needs to be a single bit (i.e., true/false), this approach has to use more space than just a single bit because this messes with word alignment. For example, if the CPU fetches a value that is not word-aligned, it has to do extra work to provide the application with the correct value:
  - Execute two reads to load the appropriate parts of the data word and reassemble them. This is the approach used in all Intel CPUs and newer ARM CPUs.
  - Read some unexpected combination of bytes assembled into a single word.
  - Throw an exception.
- Only used by MemSQL.

3 Storage Models

N-Ary Storage Model (NSM)

The DBMS stores all of the attributes for a single tuple contiguously. Also known as a "row store". This approach is ideal for OLTP workloads where transactions tend to operate only an individual entity and insert heavy workloads.

Advantages:

- Fast inserts, updates, and deletes.
- Good for queries that need the entire tuple.

Disadvantages:

- Not good for scanning large portions of the table and/or a subset of the attributes. This is because it pollutes the CPU cache by fetching data that is not needed for processing the query.

There are two different ways to organize a NSM database:

- **Heap Organized Tables:** Tuples are stored in blocks called a heap, and the heap does not necessarily define an order.

- **Index-Organized Tables:** Tuples are stored in the primary key index itself, but different from a clustered index.

Decomposition Storage Model (DSM)

The DBMS stores a single attribute for all tuples contiguously in a block of data. Also known as a "column store". This model is ideal for OLAP workloads where read-only queries perform large scans over a subset of the table's attributes. The DBMS is also able to use the vector-at-a-time iterator model.

Advantages:

- Reduces the amount of wasted work during query execution because the DBMS only reads the data that it needs for that query.
- Enables better compression because all of the values for the same attribute are store contiguously in a single column.

Disadvantages:

- Slow for point queries, inserts, updates, and deletes because of tuple splitting/stitching.

4 Hybrid Storage Models

Single logical database instance that uses different storage models for hot and cold data. Data is "hot" when first entered into the database, as it is more likely to updated again near the future. Thus, the DBMS can store new data in NSM for fast OLTP operations. It then migrates data to DSM as it ages for more efficient OLAP.

Approaches for Categorizing Data:

- **Manual:** DBA specifies what tables should be stored in DSM.
- **Off-Line:** DBMS monitors access logs off-line and them makes decisions about what data to move to DSM.
- **On-Line:** DBMS tracks access patterns at runtime and then makes decision about what data to move to DSM.

Approach #1 – Separate Execution Engine:

Run separate "internal" DBMSs that each only operate on DSM or NSM data. The system will combine of query results from both engines to appear as a single logical database to the application. This requires a synchronization method (e.g., two-phase commit) if a transaction spans execution engines.

Fractured Mirrors: [3]

- Store a second copy of the database in a DSM layout that is automatically updated.
- All updates are first entered in NSM then eventually copied into DSM mirror.
- Examples: **Oracle In-Memory Column Store**, **IBM DB2 BLU**.

Delta Store:

- Stage updates to the database in an NSM table.
- A background thread migrates updates from delta store and applies them to DSM data.
- Examples: **SAP HANA**, **MemSQL**, **Vertica**

4.1 Approach #2 – Adaptive Storage

The DBMS uses single execution engine that is able to operate on both NSM and DSM data [2].

- No need to store two copies of the database.

- No need to sync multiple database segments.
- Note that a DBMS can still use a delta-store approach with this single-engine architecture.

References

[1] *IEEE standard for binary floating-point arithmetic*. Institute of Electrical and Electronics Engineers, New York, 1985.

[2] J. Arulraj, A. Pavlo, and P. Menon. Bridging the archipelago between row-stores and column-stores for hybrid workloads. In *SIGMOD '16 Proceedings of the 2016 International Conference on Management of Data*, pages 583–598, June 2016. doi: https://dl.acm.org/citation.cfm?id=2915231.

[3] R. Ramamurthy, D. J. DeWitt, and Q. Su. A case for fractured mirrors. In *VLDB '02 Proceedings of the 28th international conference on Very Large Data Bases*, pages 430–441, August 2002. doi: https://dl.acm.org/citation.cfm?id=1287407.