Lecture #04: Query Execution & Processing I

15-721 Advanced Database Systems (Spring 2024)
`https://15721.courses.cs.cmu.edu/spring2024/`
Carnegie Mellon University
Prof. Andy Pavlo

1 Operator Execution

In OLAP systems, sequential scans are the primary method for query execution. The idea is to scan through large chunks of data and apply necessary operations to extract the desired results.

The goal is to minimize the amount of data fetched from the disk or a remote object store and maximize the use of hardware resources for efficient query execution.

Andy's (unscientific) top three execution optimization techniques:

- **Data Parallelization (Vectorization):** Breaking down a query into smaller tasks and running them in parallel on different cores, threads, or nodes.
- **Task Parallelization (Multi-threading):** Breaking down a query into smaller, independent tasks and executing them concurrently. This allows the database system to take full advantage of multi-core processors and distributed systems, thereby improving query execution time.
- **Code Specialization:** Code generation involves creating specialized code for specific queries to improve performance. This can be done through techniques like just-in-time (JIT) compilation or precompiled parameters.

There are three techniques for speeding up the queries:

- **Reduce Instruction Count**: Use fewer instructions to do the same amount of work. Usually by specializing or organizing the code to reduce the number of instructions to execute the operators.
- **Reduce Cycles per Instruction**: Execute more CPU instructions in fewer cycles. This means maximizing locality by reducing cache misses and stalls due to memory load/stores.
- **Parallelize Execution**: Use multiple threads to compute each query in parallel.

2 Query Execution

Query Execution in a database is carried out based on a generated query plan. Some terminology associated with query plans:

- A query plan is a DAG of *operators*.
- An *operator instance* is an invocation of an operator on a unique segment of data.
- A *task* is a sequence of one or more operator instances.
- A *task set* is the collection of executable tasks for a logical pipeline.

3 MonetDB/X100 Analysis

The paper by Boncz et al [1] provides a low-level analysis of execution bottlenecks for in-memory DBMSs on OLAP workloads. Their results show how DBMSs at that time were designed incorrectly for modern CPU architectures.

3.1 CPU Overview for Databases

CPUs organize instructions into **pipeline stages**, whose goal is to keep all parts of the processor busy at each cycle by masking delays from instructions that cannot be completed in a single cycle. Super-scalar CPUs support multiple pipelines. They can execute multiple instructions in parallel in a single cycle if the instructions are independent. It is categorized as "Single Instruction stream, Single Data stream (**SISD**)" in Flynn's Taxonomy.

3.2 DBMS/CPU Problems

For DBMSs, there are two problems with CPU pipelining.

- **Dependencies**: If one instruction depends on another instruction, then the CPU cannot push it immediately into the same pipeline.
- **Branch Prediction**: The CPU tries to predict what branch the program will take and fill in the pipeline with its instructions. If it gets it wrong, then it has to throw away any speculative work and flush the pipeline. Two examples are **Branch Misprediction** caused by selection scan and **Excessive Instructions** caused by the support of different data types.

4 Processing Model

A DBMS's processing model defines how the system executes a query plan. There are different trade-offs for different workloads. In addition, there are two plan processing directions:

- **Top-to-Bottom**: Start with the root and "pull" data up from its children. This approach always passe tuples with function calls.
- **Bottom-to-Top**: Start with leaf nodes and push data to their parents. This approach allows for tighter control of caches/registers in pipelines.

4.1 Iterator Model

The iterator model is also called the **Volcano** or **Pipeline** Model. Each query plan operator implements a next function. On each invocation of the next function, the operator returns either a single tuple or a null marker if there are no more tuples. The operator implements a loop that calls next on its children to retrieve their tuples and then process them.

This model is used in almost every DBMS as a general approach. It allows for tuple pipelining, but some operators have to block until their children emit all of their tuples, such as "joins", "subqueries", and "order by". Output control works easily with this approach.

4.2 Materialization Model

In the materialization model, each operator processes its input all at once and then emits its output all at once. The operator "materializes" its output as a single result. The DBMS can push down hints to avoid scanning too many tuples and return only a single column. The output can be either whole tuples (NSM) or subsets of columns (DSM).

Since this approach has lower execution/coordination overhead and fewer function calls, it is good for OLTP workloads as the queries only access a small number of tuples at a time. On the other hand, it is not good for OLAP queries with large intermediate results.

4.3 Vectorized / Batch Model

In the vectorized model, each operator implements a next function like the iterator model but emits a batch of tuples instead of a single one. The operator's internal loop processes multiple tuples at a time. The size of the batch can vary based on hardware or query properties.

The vectorized model is considered ideal for OLAP queries because it greatly reduces the number of invocations per operator. It allows operators to use vectorized (SIMD) instructions to process batches of tuples.

5 Plan Processing Direction

The query plan can be processed in two ways - push and pull.

5.1 Approach 1: Top-to-Bottom (Pull)

When processing a query plan, it starts at the root node and ”pulls” data from the children nodes.

- Start with the root and ”pull” data up from its children.
- Tuples are always passed between operators using function calls (unless it's a pipeline breaker).

5.2 Approach 2: Bottom-to-Top (Push)

When processing a query plan, it starts at the leaf nodes and ”pushes” computed results to parent nodes.

- Start with leaf nodes and ”push” data to their parents.
- Can ”fuse” operators together within a for-loop to minimize intermediate result staging.

6 Filter Representation

When processing vectors, we may continue to pass along dead or invalid tuples. We need some logical representation to identify tuples that are valid and need to be processed to include in the materialized result. There are two approaches to do this - Selection Vectors and Bitmaps.

6.1 Approach 1: Selection Vectors

Selection vectors are used to store the indices or identifiers of tuples that are valid and should be considered for further processing.

- Dense sorted list of tuple identifiers that indicate which tuples in a batch are valid.
- Pre-allocate selection vector as the max size of the input vector, we do not want to make memory allocation calls and dynamically size the selection vector.

6.2 Approach 2: Bitmaps

Bitmaps are created to indicate whether a tuple is valid and can be used as an input mask for further processing.

- Positionally-aligned bitmap that indicates whether a tuple is valid at an offset.
- Some SIMD instructions natively use these bitmaps as input masks.

References

[1] P. A. Boncz, M. Zukowski, and N. Nes. Monetdb/x100: Hyper-pipelining query execution. In *CIDR 2005, Second Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2005, Online Proceedings*, pages 225–237, 2005. URL `http://cidrdb.org/cidr2005/papers/P19.pdf`.