DBMS

Week 11

Part Seven

Query Processing and Optimization

Chapter 15 query processing

Query processing refers to the range of activities involved in extracting data from a database. The activities include translation of queries in high-level database languages into expressions that can be used at the physical level of the file system, a variety of query-optimizing transformations, and actual evaluation of queries.

Overview

Parsing and Translation

- Query processing begins with the parsing and translation phase.
- The system translates a user's query (typically expressed in SQL) into an internal form suitable for processing.
- This translation process includes syntax checking, verification of relation names, and the creation of a parse-tree representation of the query.
- Queries involving views are also handled by replacing view references with their underlying relational-algebra expressions.

Optimization

- Once the query is in an internal form (often based on extended relational algebra), the system needs to determine how to execute it efficiently.
- Different queries can be expressed in various ways, and there can be multiple ways to translate a query into a relational-algebra expression.
- The optimization phase aims to find the most efficient way to evaluate the query by choosing an execution plan.
- The choice of execution plan depends on factors like data statistics and available indices.
- An execution plan includes the relational-algebra expression annotated with instructions on how to evaluate each operation, known as evaluation primitives.
- Query optimization is a critical task, as it significantly impacts query performance.

Evaluation

- Once the execution plan is determined, the system proceeds with the evaluation phase.
- The query execution engine takes the query-evaluation plan and executes it, generating the result of the query.
- Execution plans can involve multiple relational-algebra operations, each of which may have different evaluation algorithms.
- The evaluation plan specifies how these operations are coordinated to obtain the final result.
- The output of the query is returned to the user.

Cost Estimation

- To optimize a query effectively, the query optimizer needs to estimate the cost of each operation in the execution plan.
- The cost depends on various parameters, such as available memory and data statistics.
- Accurate cost estimation is challenging, but rough estimates help in selecting efficient execution plans.

Measures of Query Cost

Multiple Evaluation Plans

- When executing a query, there can be multiple possible evaluation plans, and it's essential to compare these alternatives in terms of their estimated cost to choose the best plan.

Cost Metrics

- The cost of query evaluation can be measured in various ways, including disk accesses, CPU time, and, in parallel and distributed database systems, the cost of communication.

Historical Focus on I/O Cost

- Traditionally, the I/O cost (input/output operations) was a dominant factor in estimating the cost of query operations, particularly when data was stored on magnetic disks.

Changing Storage Landscape

- With the advent of larger and more affordable solid-state drives (SSDs) and increased main memory sizes, the I/O cost is no longer the sole dominant factor. CPU costs also play a significant role in query evaluation.

CPU Cost Estimation

- While the passage simplifies by not including CPU costs in the model, databases often use estimators for CPU costs. These estimates may include costs per tuple, costs for processing index entries, and costs per operator or function.

Disk I/O Costs

- The passage discusses the factors affecting disk I/O costs, such as block transfer time (tT) and block access time (tS). These values need to be calibrated for the specific disk system used.

SSD Considerations

- SSDs have lower tS compared to magnetic disks but have an overhead for initiating I/O operations. The passage provides typical values for SSDs and PCIe SSDs.

Memory-Based Processing

- When data is in main memory, the costs are significantly lower than when data is on disk. Reads from memory have lower transfer and latency times. Main memory size (M) is a crucial parameter for estimating costs.

Buffer Residence

- The passage mentions that the presence of data in the buffer (cache) can reduce actual disk-access costs during execution. However, this effect is challenging to estimate accurately.

Response Time vs. Resource Consumption

- Estimating the response time of a query plan is challenging due to various factors, including buffer state and disk distribution. Instead, optimizers focus on minimizing the total resource consumption of a query plan, which may not necessarily correlate with the fastest response time.

Resource-Based Cost Models

- Database optimizers typically use resource-based cost models, like the one described in the passage, to estimate the cost of query evaluation plans. These models help in selecting efficient query execution plans based on expected resource utilization.

Selection Operation

File Scan

- A file scan is a low-level database operation used to access and retrieve records from a database file. It involves scanning the entire file to locate and retrieve records that meet certain selection criteria.

Selection Operation

- In a database query context, a selection operation involves choosing specific records from a table that satisfy a given condition or set of conditions.

Linear Search (A1)

- A linear search is a straightforward search algorithm that scans each block or record in a file sequentially and tests each record against a selection condition. It can be relatively slow for large datasets but can be used with any file organization.

Clustering Index (A2)

- A clustering index is a type of index that organizes records in a database table based on a specific key. It allows for efficient retrieval of records that match an equality condition on the key.

Secondary Index (A4)

- A secondary index is an index that is not the primary means of organizing data in a table but provides an alternative path to retrieve records based on a specific key or attribute.

Buffer Tree

- A buffer tree is an index structure that associates a buffer (memory storage) with each internal node of a tree-based index, such as a B+-tree. It aims to reduce the number of disk I/O operations during queries.

Bitmap Index Scan

- A bitmap index scan is an algorithm that uses bitmaps to efficiently retrieve records based on multiple conditions, such as conjunctions and disjunctions of conditions.

Conjunction

- A conjunction is a logical operation in which two or more conditions must all be true for an overall condition to be considered true. In the context of selections, it involves combining multiple conditions with "AND."

Disjunction

- A disjunction is a logical operation where at least one of multiple conditions must be true for an overall condition to be considered true. In selections, it combines conditions with "OR."

Negation

- Negation is a logical operation that reverses the truth value of a condition. In selections, it retrieves records that do not satisfy a given condition.

Sorting

Sorting in Databases

- Sorting is essential in database systems for two main reasons. First, SQL queries often request sorted output, and second, some relational operations, such as joins, are more efficient when performed on sorted data.

Logical vs. Physical Sorting

- When sorting data, it's possible to create an index on the sort key and use that index to retrieve records in sorted order. However, this is a logical sorting process through an index, not a physical reordering of data on disk. Physical sorting involves arranging the records on disk in the desired order to minimize disk accesses during retrieval.

External Sorting

- External sorting is the technique used for sorting large datasets that cannot fit entirely in memory. It's based on the external sort-merge algorithm, which consists of two main stages: run creation and run merging.
  - Run Creation: In this stage, the dataset is divided into multiple sorted runs. Each run represents a sorted subset of the data. The process involves reading a portion of the data into memory (limited by the available memory buffer size) and sorting it in memory. The sorted data is then written to separate run files. This process is repeated until all data has been processed.
  - Run Merging: In this stage, the sorted runs are merged to produce the final sorted output. The merge operation reads one block from each run into memory, selects the smallest tuple among them (based on sort order), writes it to the output, and removes it from the buffer block. This process continues until all input buffer blocks are empty.

Multiple Passes

- If the number of runs generated in the first stage is greater than the available memory buffer size (M), the merge operation is done in multiple passes. Each pass further reduces the number of runs by merging a subset of them. The process continues until there are fewer runs than the buffer size, and a final pass produces the fully sorted output.

Cost Analysis

- The cost of external sorting is evaluated in terms of disk-access cost and disk-seek cost. The disk-access cost depends on the number of blocks containing records in the relation (br), the number of runs generated, and the number of merge passes. The disk-seek cost accounts for the seeks required for reading and writing data during run generation and merge passes.
- Disk-Access Cost: $br(2\lceil \log\lfloor M/bb \rfloor - 1(br/M)\rceil + 1)$
- Disk-Seek Cost: $2\lceil br/M \rceil + \lceil br/bb \rceil (2\lceil \log\lfloor M/bb \rfloor - 1(br/M)\rceil - 1)$

Join Operation

Nested-Loop Join

- A join algorithm that involves two nested loops to compare each tuple from one relation (outer relation) with every tuple from another relation (inner relation). It is a straightforward join method that doesn't require indices but can be inefficient for large datasets.

Theta Join

- A join operation that combines two relations based on a specified condition (θ), which can be any conditional expression. It's a generic form of join that includes various types like equi-join (equality condition), natural join (based on matching attribute names), and more.

Natural Join

- A type of theta join where the join condition is based on matching attributes with the same name in both relations. It eliminates duplicate attributes in the result.

Block Nested-Loop Join

- An optimization of the nested-loop join where data is read and processed in blocks rather than individual tuples. It can reduce the number of I/O operations and improve performance when memory is limited.

Merge Join

- A join algorithm primarily used for natural joins or equi-joins. It requires both input relations to be sorted on the join attributes. The merge-join algorithm combines the sorted input relations efficiently, similar to the merge stage in the merge-sort algorithm.

Hash Join

- A join algorithm that uses hash functions to partition the input relations into buckets based on the join attributes. It's efficient for large datasets and can be further optimized with techniques like recursive partitioning and hybrid hash join.

Recursive Partitioning

- A technique used in hash join when memory is limited. It involves partitioning the input relations into smaller subsets recursively until each subset fits in memory, allowing efficient hash join operations.

Hybrid Hash Join

- An optimization of the hash join algorithm that leverages available memory to keep a portion of the build input in memory. This can significantly reduce I/O operations and improve performance when memory is sufficient but not large enough to hold the entire build input.

Hash-Table Overflow

- Occurs in hash join when a hash partition or hash index becomes too large to fit in memory. Overflow resolution or avoidance techniques are used to handle this situation.

Indexed Nested-Loop Join

- A join method that utilizes indexes on one or both input relations to efficiently perform join operations. It can be faster than nested-loop joins when suitable indexes are available.

Conjunctive Condition

- A complex join condition that combines multiple conditions using logical AND (&&) operators. For example, $\theta 1 \wedge \theta 2 \wedge \theta 3$ represents a conjunctive condition with three sub conditions.

Disjunctive Condition

- A complex join condition that combines multiple conditions using logical OR (||) operators. For example, θ1 ∨ θ2 ∨ θ3 represents a disjunctive condition with three sub conditions.

Other Operations

Duplicate Elimination

- Duplicate elimination can be implemented through sorting. Identical tuples are grouped together after sorting, and duplicates can be removed efficiently.
- External sort-merge can eliminate duplicates before writing sorted runs to disk.
- Alternatively, hashing can be used. The relation is partitioned using a hash function, and an in-memory hash index is built. Tuples are inserted into the hash index only if they are not already present.

Projection

- Projection can be implemented by applying projection to each tuple, possibly resulting in duplicate records.
- Duplicate elimination techniques can be used to remove duplicates if needed.
- If the attributes in the projection list include a key of the relation, no duplicates will exist.

Set Operations (Union, Intersection, Set-Difference)

- Set operations can be implemented by sorting both input relations and scanning them concurrently.
- The result is produced by merging the sorted relations, and duplicates are removed during the process.
- The cost of these operations depends on the size of the relations and the sorting process. Hashing can also be used for set operations.

Outer Join

- Outer join operations, such as left outer join, can be implemented by first computing the corresponding join (e.g., using merge-join or hash-join).
- Tuples from one of the relations that do not have matching tuples in the other relation are padded with null values and added to the result.
- Merge-join and hash-join algorithms can be extended to handle outer joins efficiently.

Aggregation

- Aggregation operations, like avg, sum, min, max, and count, can be implemented similarly to duplicate elimination.
- Grouping attributes are used to group tuples together.
- Aggregation functions are applied to each group to compute the result.
- On-the-fly aggregation techniques can optimize memory usage and reduce the need for writing tuples to disk when all tuples fit in memory.

Evaluation of Expressions

Materialization

- Materialization in the context of query processing refers to the process of creating and storing intermediate results of operations in temporary relations (or tables) within a database. These temporary results are used as inputs for subsequent operations in a query evaluation. Materialization can involve writing data to disk and can be resource-intensive.

Pipelining

- Pipelining, in the context of query evaluation, is an approach that aims to reduce the need for storing intermediate results by passing data directly from one operation to the next in a sequence without storing it in temporary relations. This approach can help improve query processing efficiency and reduce the use of disk storage.

Demand-Driven Pipeline

- A demand-driven pipeline is a type of data processing pipeline where operations are triggered to produce output only when requested. It pulls data through the pipeline as needed and computes results on-demand.

Producer-Driven Pipeline

- A producer-driven pipeline is another type of data processing pipeline where operations produce data eagerly, pushing it downstream as it becomes available. This approach can lead to more efficient processing, particularly in parallel systems.

Continuous Query

- A continuous query is a type of query used in real-time or streaming data processing. It continuously monitors and analyzes incoming data streams, often with the goal of identifying patterns, aggregating data, or generating alerts as new data arrives.

Tumbling Windows

- Tumbling windows are time-based intervals used in continuous query processing. They divide the data stream into fixed-size windows, and operations like aggregation are performed separately on each window. Once a window expires, its results are typically emitted.

Punctuations

- Punctuations are markers or signals in a data stream that indicate the end of a particular data window or the fact that all future data will have timestamps greater than a specified value. Punctuations help in correctly handling time-based operations in continuous queries.

Query Processing in Memory

Cache-Conscious Algorithms

- These are query processing algorithms designed to make efficient use of CPU cache memory, which is significantly faster to access than main memory. Modern CPUs have multiple levels of cache, and cache-conscious algorithms aim to minimize cache misses. Some strategies for cache-conscious query processing include:
  - Optimizing sorting algorithms to use cache efficiently.
  - Partitioning relations into smaller pieces that fit in cache for operations like hash joins.
  - Arranging attributes within a tuple to optimize cache usage based on access patterns.

Query Compilation

- Traditional query processors interpret query plans at runtime, which can introduce overhead due to interpretation and function calls. Query compilation involves translating query plans into machine code or intermediate-level byte-code, allowing for more efficient execution. Benefits of query compilation include:
  - Computing attribute offsets at compile time to minimize lookup overhead.
  - Combining code for multiple functions to reduce function call overhead.
  - Achieving significant speedup compared to interpreted code, sometimes up to a factor of 10.

Column-Oriented Storage

- In some data analytic scenarios, column-oriented storage can be advantageous. Instead of storing data as rows, it stores data as columns, making it more efficient for selection operations on single or a few attributes. Key features of column-oriented storage include:
  - Efficient access to individual attributes.
  - Utilization of vector-processing capabilities of modern processors, enabling parallel operations on multiple attribute values.
  - Compilation of query plans into machine code with vector-processing instructions for performance optimization.

Class activity

- Query processing
- Evaluation primitive
- Query-execution plan
- Query-evaluation plan
- Query-execution engine
- Measures of query cost
- Sequential I/O
- Random I/O
- File scan
- Linear search
- Selections using indices
- Access paths
- Index scans
- Conjunctive selection
- Disjunctive selection
- Composite index
- Intersection of identifiers
- External sorting
- External sort–merge
- Runs
- N-way merge
- Equi-join
- Nested-loop join
- Block nested-loop join
- Indexed nested-loop join
- Merge join
- Sort-merge join
- Hybrid merge join
- Hash-join
  - Build
  - Probe
  - Build input
  - Probe input
  - Recursive partitioning
  - Hash-table overflow
  - Skew
  - Fudge factor
  - Overflow resolution
  - Overflow avoidance
- Hybrid hash-join
- Spatial join
- Operator tree
- Materialized evaluation
- Double buffering
- Pipelined evaluation
  - Demand-driven pipeline (lazy, pulling)
  - Producer-driven pipeline (eager, pushing)
  - Iterator
  - Pipeline stages
- Double-pipelined join
- Continuous query evaluation