

Query Evaluation Techniques for large DB

Original slides by Daniela
Stasa

Modified by Rachel Pottinger



Purpose

- To survey efficient algorithms and software architectures of query execution engines for executing complex queries over large databases

Steps

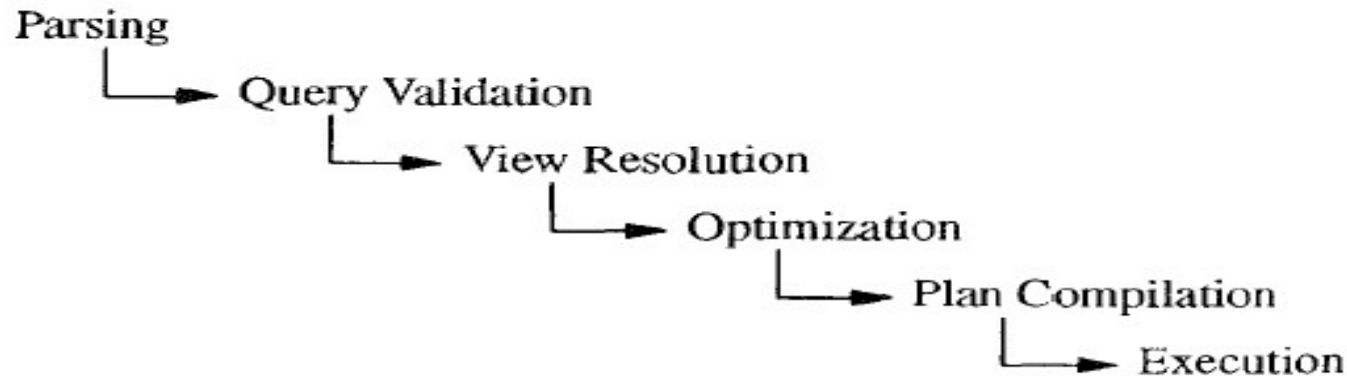


Figure 2. Query processing steps.

- Translate logical query from SQL to query tree in logical algebra.
- Query tree in logical algebra is translated into a physical plan
- Optimizer expands search space and finds best plan.
- Optimal physical plan copied out of optimizer's memory structure and sent to query execution engine.
- query execution engine executes plan using relations in database as input, and produces output



Query execution engine

- What is it?

- Collection of query execution operators and mechanisms for operator communication and synchronization
- Query execution engine defines the space of possible plans that can be chosen by query optimizer.



Some of the techniques discussed

- Algorithms and their execution costs
- Sorting versus hashing
- Parallelism
- Resource allocation
- Scheduling issues
- Performance-enhancement techniques
- And more ...



Some notes

■ On the context

- While many of the techniques were developed for relational database systems most are applicable to any data mode that allows queries over sets and lists.


■ Type of queries

- Discusses only read-only queries but mostly applicable to updates.



Discussion

- This paper was written some time ago. Do you think that in the time since then the issues would have gotten better or worse. Why?

- 
- Hardware changes: More in the way of multi-core and multiprocessing, and SSD
 - Changes in how the hardware is working, faster to be able to move data into caching for processing
 - Challenges: Data has gotten much larger, more complex? More joins, more things, different server, different locations?
 - People are willing to put more data in, because of better hardware, cycle.
 - Basic things are pushed to hardware, that can only happen when the field matures



Architecture of query execution engines

- Focus on useful mechanisms for processing sets of items
 - Records
 - Tuples
 - Entities
 - Objects



Physical Algebra

- Taken as a whole, the query processing algorithms form an algebra which we call physical algebra of a database system



Physical vs. Logical Algebra

- Equivalent but different
- Logical algebra: related to data model and defines what queries can be expressed in data model
- Physical algebra: system specific
 - Different systems may implement the same data model and the same logical algebra but may use different physical algebras



Physical vs. Logical Algebra

- Specific algorithms and therefore cost functions are associated only with physical operators not logical algebra operators
- Mapping logical to physical non-trivial:
 - It involves algorithm choices
 - Logical and physical operators not directly mapped
 - Some operators in physical algebra may implement multiple logical operators
 - etc



Iterators

- Two important features of operators
 - Can be combined into arbitrarily complex evaluation plans
 - Any number of operators can schedule and execute each other in a single process without assistance from underlying OS



Implementation issues

- Prepare an operator for producing data
 - Open
- Produce an item
 - next
- Perform final housekeeping
 - close



Observations

- Entire query plan executed within a single process
- Operators produce an item at a time on request
- Items never wait in a temporary file or buffer (pipelining)
- Efficient in time-space-product memory cost
- Iterators can schedule any type of trees including bushy trees
- No operator affected by the complexity of the whole plan



Sorting & Hashing

- The purpose of many query-processing algorithms is to perform some kind of matching,
 - i.e., bringing items that are “alike” together and performing some operation on them.
- There are two basic approaches used for this purpose:
 - sorting
 - and hashing.
- These are the basis for many join algorithms



Design Issues

- Sorting should be implemented as an **iterator**
 - In order to ensure that **sort module** interfaces well with the other operators, (e.g., file scan or merge-join).
- Input to the sort module must be an iterator, and sort uses open, next, and close procedures to request its input
 - therefore, sort input can come from a scan or a complex query plan, and sort operator can be inserted into a query plan at any place or at several places.

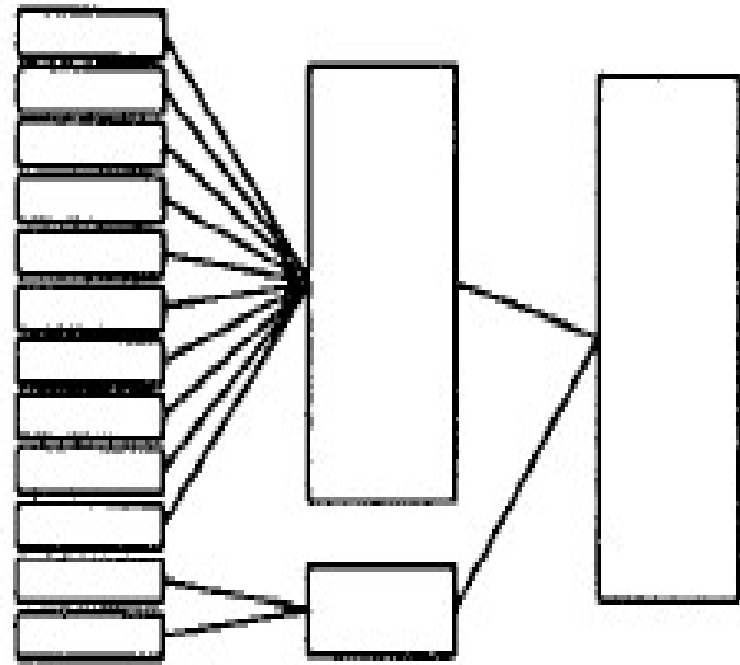


More on Sorting

- For sorting large data sets there are two distinct sub-algorithms :
 - One for sorting within main memory
 - One for managing subsets of the data set on the disk.
- For practical reasons, e.g., ensuring that a run fits into main memory, the disk management algorithm typically uses physical dividing and logical combining (merging).
- A point of practical importance is the fan-in or degree of merging, but this is a parameter rather than a defining algorithm property.

Level 0 run

- There are two alternative methods for creating initial runs
 - In-memory sort algorithm (usually quick sort)
 - Replacement Selection





Quick Sort vs.

Replacement Selection

- Run files in RS are typically larger than memory ,as oppose to QS where they are the size of the memory
- Qs results in burst of reads and writes for entire memory loads from the input file to initial run files while RS alternates between individual read and write
- In RS memory management is more complex
- The advantage of having fewer runs must be balanced with the different I/O pattern and the disadvantage of more complex memory management.



Hashing

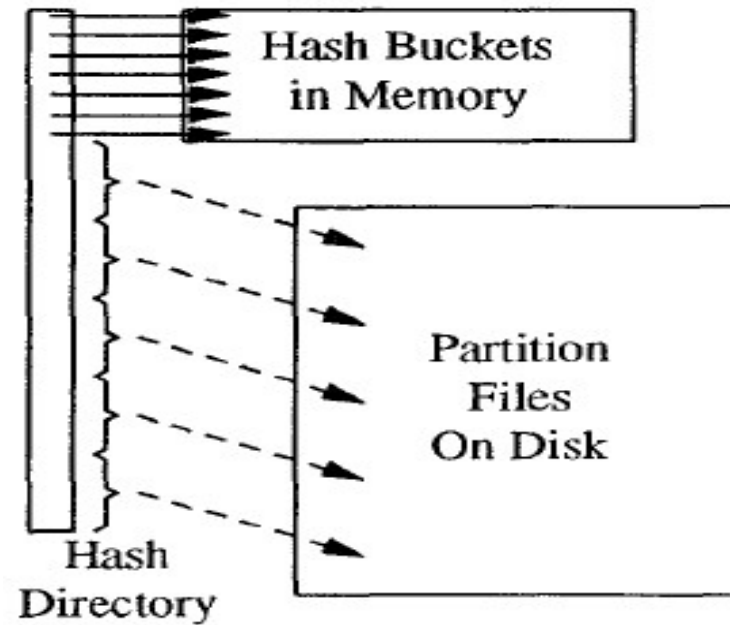
- Alternative to sorting
- Expected complexity of hashing algorithms is $O(N)$ rather than $O(N \log N)$ as for sorting.
- Hash-based query processing algorithms use an in-memory hash table of database objects to perform their matching task.



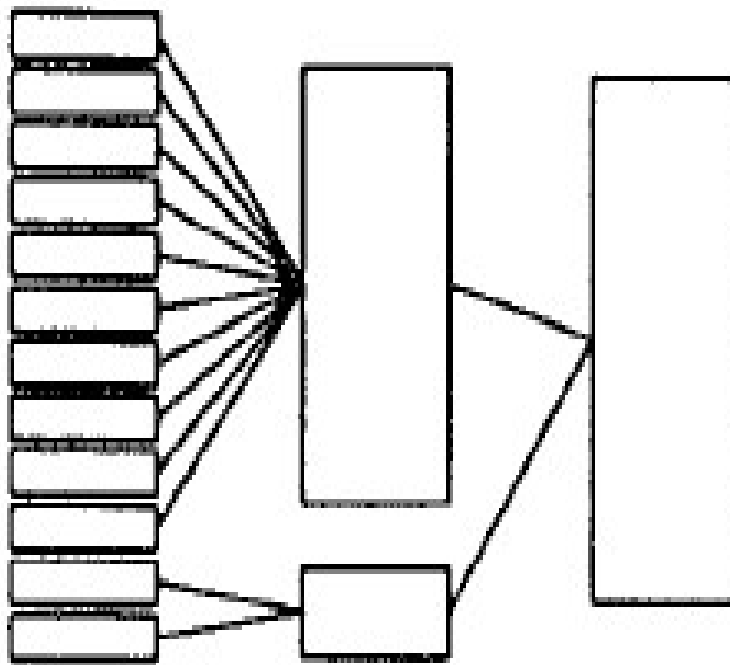
Hashing Overflow

- When hash table is larger than memory, hash table overflow occurs and must be dealt with.
- Input divided into multiple partition files such that partitions can be processed independently from one another,
- Concatenation of results of all partitions is the result of the entire operation.

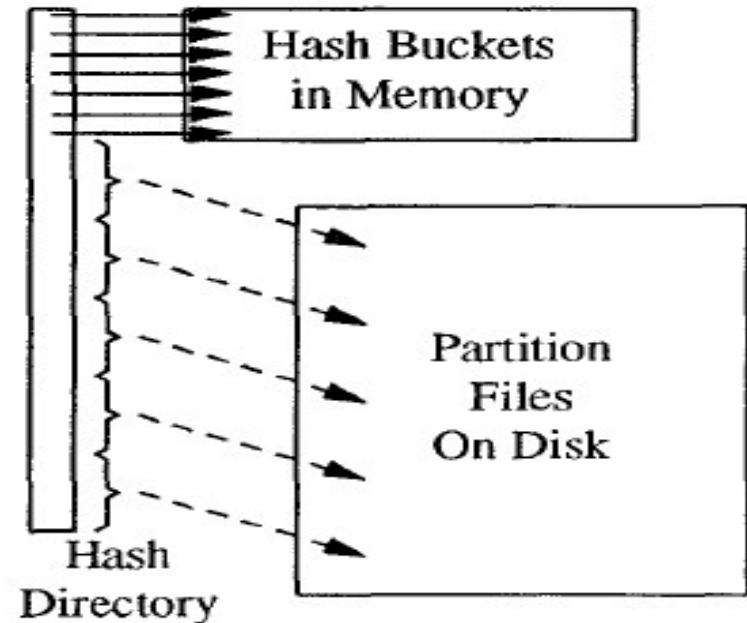
Hash overflow



Hashing & Sorting: similar issues on opposite ends



Sorting



Hashing



Indices/indexes

- Goal:

- To reduce the number of accesses to secondary storage

- How?

- By employing search techniques in the form of indices (sometimes, also materialized views, but not in this paper)
- Indices map key or attribute values to locator information with which database objects can be retrieved.



Some Index Structures:

- **Clustered & Un-clustered**

- Clustered: order or organization of index entries determines order of items on disk.

- **Sparse & Dense**

- Sparse: Indices do not contain an entry for each data item in the primary file, but only one entry for each page of the primary file;
- Dense: there are same number of entries in index as there are items in primary file.
- Non-clustering indices must always be dense




Buffer Management

- Goal: reduce I/O cost by caching data in an I/O buffer.
- Design Issues
 - Recovery
 - Replacement policy
 - performance effect of buffer allocation
 - Interactions of index retrieval and buffer management
- Implementation Issues
 - Interface provided : fixing –unfixing
 - Intermediate results kept in a separate buffer



Discussion (pairs)

- There are many issues that could be covered by either the OS or the database. Break into groups and discuss some of these issues. For each issue, what are the pros and the cons of handling it in the database?

- 
- GPUs
 - TPUs - Tensor matrix processing, hardware specific, ML
 - DOJ: combining databases and OS, create a monopoly.
 - OS as manager for resource allocations, works in general, but not great for databases,
 - in general, LRU (least recently used) works great, but not for DB
 - Database people always want more control. Turf war.
 - Different for different scenarios



BINARY MATCHING OPERATIONS

- Relational join most prominent binary matching operation (others: intersection, union, etc)
- Set operations such as intersection and difference needed for any data model
- Most commercial db systems as of 1993 used only nested loops and merge-join. As per research done for SystemR, these two were supposed to be most efficient.
- SystemR researchers did not consider Hash join algorithms, which are today considered even better in performance.



NESTED-LOOPS JOIN ALGORITHMS: simple elegance

- For each item in one input, scan entire other input to find matches.
- Performance is really poor, because inner input is scanned often. (paper points this out)
- Tricks to improve performance include:
 - larger input should be the outer one.
 - if possible, use an index on the attribute to be matched in the inner input.
 - Inner input can be scanned once for each 'page' of outer input.



MERGE-JOIN ALGORITHMS

- Requires both inputs sorted on the join attribute
- Requires keeping track of interesting orderings
- Hybrid join (used by IBM for DB2), uses elements from index nested-loop joins and merge join, and techniques joining sorted lists on index leaf entries.



HASH JOIN ALGORITHMS

- Based on in-memory hash table on one input (smaller one, called 'build input'), and probing this table using items from the other input (called 'probe input').
- Very fast if build input fits into memory, regardless of size of probe input.
- overflow avoidance methods needed for larger build inputs.
- both inputs partitioned using same partitioning function. Final join result formed by concatenating join results of pairs of partitioning files.
- Recursive partitioning may be used for both inputs
- More effective when the two input sizes are very different (smaller being the build input).



CONCLUSION:

The choice of Hash based or Sort based should be based on relative sizes of inputs and the danger of performance loss due to skewed data or hash value distribution.




- Does the number of generally used joins seem large or small to you? Why?
- Are you surprised by any of the joins that are used?



Discussion (adapted from Sid's comments)

- Who should write survey papers? Grad students? Senior researchers? Why? What are the benefits and disadvantages to having people from different groups write them?

- 
- More important to have a diverse selection of authors. These are usually single author papers.
 - Supervisor could have a view at both sides, working with grad students, but also know the research. Academia vs. Industry? interns?