Introduction to Data Management

Query Cost Estimation

Alyssa Pittman
Based on slides by Jonathan Leang, Dan Suciu, et al

Paul G. Allen School of Computer Science and Engineering
University of Washington, Seattle

October 25, 2019
Announcements

- Reminder:
  - Midterm review in class on Monday
    - Add questions to the Ed thread!
    - Midterm in class on Wednesday
- HW4 due today 11 pm
  - HW5 out next Friday, happy studying
Goals for Today

- Move to a short unit on RDBMS optimization
- Learn how an RDMS translates a logical query plan to a physical query plan and executes it
Outline

- Query execution
- Cost estimation ideas and assumptions
- Join algorithm analyses
So you wrote a SQL query...
  • SQL only tells the computer *what* you want
  • RDBMS needs to find a good way to actually do it
Logical vs Physical Plans

- SQL is translated into RA
- RA (logical plan) does not fully describe execution
- RA with algorithms (physical plan) is needed
Logical vs Physical Plans

- SQL is translated into RA
- RA (logical plan) does not fully describe execution
- RA with algorithms (physical plan) is needed

![Diagram showing logical vs physical plans]
Cost estimation is an active research topic
Equations and methods discussed in this class form a foundation of concepts, but usually cannot compare to a commercialized solution
Plan Enumeration

RDBMs optimize by selecting the **least cost plan**

- SQL $\implies$ RA
- RA $\implies$ Set of eq. RA
- Set of eq. RA $\implies$ Set of physical plans
- Set of physical plans $\implies$ The least cost plan

...Execute!
Plan Enumeration

RDBMS

SQL

SELECT *

FROM T, R, S

WHERE ...

October 25, 2019
Plan Enumeration

RDBMS

SQL

```
SELECT * 
FROM T, R, S 
WHERE ...
```
Plan Enumeration

SQL
SELECT * FROM T, R, S WHERE ...

Logical Plan

Equivalent Logical Plans

RDBMS
Plan Enumeration

SQL
SELECT * FROM T, R, S WHERE ...

Logical Plan

Equivalent Logical Plans

Physical Plans

RDBMS
Plan Enumeration

SQL
SELECT * FROM T, R, S WHERE ...

Logical Plan

Equivalent Logical Plans

RDBMS

Least Cost Plan

Physical Plans
Plan Enumeration

SQL
SELECT * FROM T, R, S WHERE ...

Logical Plan

Equivalent Logical Plans

Execution

Least Cost Plan

Physical Plans

RDBMS
Assumptions

For this class we make a lot of assumptions

- **Disk-based storage**
  - HDD not SDD

- **Row-based storage**
  - Tuples are stored contiguously

- **IO cost** (reading from disk) only considered
  - Comprehensive cost estimation involves many factors
    - Network, disk, and CPU cost
    - Cache (main mem., L1 cache, L2 cache, disk cache, ...)
    - Reading from disk is usually the biggest component
      - One IO access is ~100000x more expensive than one main memory access

- **Cold cache** (no data preloaded)
Disk Storage

- Mechanical hard drive
- Smallest unit of memory that can be read at once is a **block**
  - Usually 512B to 4kB
- DBMS will attempt to store table files in **contiguous chunks of memory** on disk
- Sequential disk reads are faster than random ones
Disk Storage

- Tables are stored as files
  - **Heap file** — Unsorted tuples (this lecture)
  - **Sequential file** — Sorted tuples (next lecture)
    - Attribute(s) sorted on is called a **key** (because that term isn’t overloaded...)

October 25, 2019
Making Cost Estimations

- RDBMS keeps statistics about our tables
  - $B(R) = \# \text{ of blocks}$ in relation $R$
  - $T(R) = \# \text{ of tuples}$ in relation $R$
  - $V(\text{attr}, R) = \# \text{ of distinct values}$ of attr in $R$
- We only discuss **join algorithms** because they are usually the most expensive part of a query.
- We only discuss **nested-loop** and **single-pass** join algorithms because cost equations get complex.
Join Algorithm Summary

- Nested-Loop Join
  - Versatile

- Hash Join (single pass)
  - Fast
  - Needs at least one input to be small

- Sort-Merge Join (single pass)
  - Fast
  - Sorts data at the same time!
  - Needs both inputs to be small
Join Algorithm Summary

- **Nested-Loop Join**
  - Versatile

- **Hash Join (single pass)**
  - Fast
  - Needs at least one input to be small

- **Sort-Merge Join (single pass)**
  - Fast
  - Sorts data at the same time!
  - Needs both inputs to be small
Nested Loop Join Algorithm

- Similar execution logic as nested-loop semantics

```python
for each tuple t1 in R:
    for each tuple t2 in S:
        if t1 and t2 can join:
            output (t1, t2)
```
Nested Loop Join Algorithm

- Similar execution logic as nested-loop semantics

For each tuple t1 in R:
  
  - For each tuple t2 in S:
    
    If t1 and t2 can join:
    
    Output (t1, t2)

To save time, we’ll read tuples from disk to memory in blocks. For fixed-size tuples, each block will have the same number of tuples.
Nested Loop Join Algorithm

Example equijoin

```sql
SELECT * 
FROM R, S 
WHERE R.attr = S.attr
```

(block-at-a-time nested loop join)

**Block-at-a-time nested loop join:**

for each block \( b_R \) in \( R \):
  for each block \( b_S \) in \( S \):
    for each tuple \( t_R \) in \( b_R \):
      for each tuple \( t_S \) in \( b_S \):
        if \( t_R \) and \( t_S \) can join:
          output \( (t_R, t_S) \)
Nested Loop Join Algorithm

Example equijoin

```sql
SELECT * 
FROM R, S 
WHERE R.attr = S.attr
```

(block-at-a-time nested loop join)

For each block `bR` in `R`:
  For each block `bS` in `S`:
    For each tuple `tR` in `bR`:
      For each tuple `tS` in `bS`:
        If `tR` and `tS` can join:
          Output `(tR, tS)`
Nested Loop Join Algorithm

Example equijoin

```
SELECT *  
FROM R, S  
WHERE R.attr = S.attr
```

(block-at-a-time nested loop join)

```
Block-at-a-time nested loop join:

for each block \( b_R \) in R:
  for each block \( b_S \) in S:
    for each tuple \( t_R \) in \( b_R \):
      for each tuple \( t_S \) in \( b_S \):
        if \( t_R \) and \( t_S \) can join:
          output \( (t_R, t_S) \)
```

Blocks are joined in memory
Nested Loop Join Algorithm

Example equijoin

```sql
SELECT * 
FROM R, S 
WHERE R.attr = S.attr
```

A tuple where x is the join attribute value

Assume block size = 2 tuples
Nested Loop Join Algorithm

Example equijoin

```sql
SELECT * 
FROM R, S 
WHERE R.attr = S.attr
```

Assume block size = 2 tuples

Main Memory

Disk

A tuple where x is the join attribute value
Example equijoin

```sql
SELECT * 
FROM R, S 
WHERE R.attr = S.attr
```

A tuple where x is the join attribute value

Assume block size = 2 tuples
Example equijoin

\[
\text{SELECT} \quad * \\
\text{FROM} \quad R, S \\
\text{WHERE} \quad R.\text{attr} = S.\text{attr}
\]

A tuple where x is the join attribute value

Assume block size = 2 tuples
Nested Loop Join Algorithm

Example equijoin

```
SELECT * 
FROM R, S 
WHERE R.attr = S.attr
```

- A tuple where x is the join attribute value

Assume block size = 2 tuples
Nested Loop Join Algorithm

Example equijoin

```
SELECT *
FROM R, S
WHERE R.attr = S.attr
```

(x) A tuple where x is the join attribute value

Assume block size = 2 tuples

(block-at-a-time nested loop join)
Nested Loop Join Algorithm

Example equijoin

\[
\text{SELECT } * \quad \text{FROM } R, S \quad \text{WHERE } R.\text{attr} = S.\text{attr}
\]

\[x\] A tuple where \(x\) is the join attribute value

Assume block size = 2 tuples
Nested Loop Join Algorithm

Example equijoin

```
SELECT * 
FROM R, S 
WHERE R.attr = S.attr
```

A tuple where x is the join attribute value

Assume block size = 2 tuples
Example equijoin

```
SELECT * 
FROM R, S 
WHERE R.attr = S.attr
```

A tuple where x is the join attribute value

Assume block size = 2 tuples
Nested Loop Join Algorithm

Block-at-a-time nested loop join

$$\text{Cost} = B(R) + B(R) \times B(S)$$

Reading all of R...

... for each block of R read all of S
Nested Loop Join Algorithm

Example equijoin

\[
\text{SELECT} \quad * \\
\text{FROM} \quad R, S \\
\text{WHERE} \quad R.\text{attr} = S.\text{attr}
\]

Can I do it faster?
Nested Loop Join Algorithm

Example equijoin

```
SELECT * 
FROM R, S 
WHERE R.attr = S.attr
```

Can I do it faster?
Yeah... if you’re willing to use more memory

Algorithms 101:
Time complexity vs space complexity tradeoff
Nested Loop Join Algorithm

Example equijoin

```
SELECT  *
FROM    R, S
WHERE   R.attr = S.attr
```

Optimized block-nested-loop join:

for each group of N blocks \(bR\) in R:
for each block \(bS\) in S:
    for each tuple \(tR\) in \(bR\):
        for each tuple \(tS\) in \(bS\):
            if \(tR\) and \(tS\) can join:
                output \((tR,tS)\)
Nested Loop Join Algorithm

Example equijoin

```sql
SELECT *
FROM R, S
WHERE R.attr = S.attr
```

N = 2 blocks

Assume block size = 2 tuples
Nested Loop Join Algorithm

Example equijoin

```
SELECT * 
FROM R, S 
WHERE R.attr = S.attr
```
Nested Loop Join Algorithm

Example equijoin

```sql
SELECT *
FROM R, S
WHERE R.attr = S.attr
```

N = 2 blocks

Assume block size = 2 tuples
Nested Loop Join Algorithm

Example equijoin

\[
\text{SELECT } * \quad \text{FROM } R, S \quad \text{WHERE } R.\text{attr} = S.\text{attr}
\]

\(N = 2\) blocks

Assume block size = 2 tuples
Nested Loop Join Algorithm

Example equijoin

```
SELECT * 
FROM R, S 
WHERE R.attr = S.attr
```

N = 2 blocks

Assume block size = 2 tuples
Nested Loop Join Algorithm

Block-nested-loop join
Cost = \( B(R) + \frac{B(R)}{N} \times B(S) \)

- Reading all of \( R \)…
- … for each group of \( N \) blocks of \( R \) read all of \( S \)
Join Algorithm Summary

- Nested-Loop Join
  - Versatile

- **Hash Join** (single pass)
  - Fast
  - Needs at least one input to be small

- Sort-Merge Join (single pass)
  - Fast
  - Sorts data at the same time!
  - Needs both inputs to be small
Hash Join

- Make a lookup/hash table from the smaller table
  - Smaller table has to be smaller than total main memory available ($B(R) < M$ or $B(S) < M$)
- For each block of the larger table, join using the lookup/hash table
A naive hash function:

\[ h(x) = x \mod 10 \]

Operations:

find(103) = ??
insert(488) = ??

Separate chaining:
Hash Tables 101

- \( \text{insert}(k, v) \) inserts key \( k \) with value \( v \)
- Many values for one key
  - Duplicates are ok for our bag semantics
- \( \text{find}(k) \) returns a list of all values associated with the key
Hash Join

Example equijoin

\[
\begin{align*}
\text{SELECT} & \quad * \\
\text{FROM} & \quad R, S \\
\text{WHERE} & \quad R.\text{attr} = S.\text{attr}
\end{align*}
\]

- Hash Join

\[
\text{(hash join)}
\]

\[
R.\text{attr} = S.\text{attr}
\]

\[
R \quad S
\]

Assume block size = 2 tuples

Disk

Main Memory

\[
\begin{array}{c}
R \\
1 & 7 & 3 & 5 \\
S \\
3 & 8 & 1
\end{array}
\]

M = 10 blocks, hash(x) = x mod 5
Hash Join

Example equijoin

```
SELECT * 
FROM R, S 
WHERE R.attr = S.attr
```

M = 10 blocks, hash(x) = x mod 5

Assume block size = 2 tuples
Hash Join

Example equijoin

```sql
SELECT *
FROM R, S
WHERE R.attr = S.attr
```

\[ M = 10 \text{ blocks, } \text{hash}(x) = x \mod 5 \]

Assume block size = 2 tuples
Hash Join

Example equijoin

\[ R \bowtie S \]

Assume block size = 2 tuples

\[ M = 10 \text{ blocks, } \text{hash}(x) = x \mod 5 \]
Hash Join

Hash join

Cost = B(R) + B(S)

Assuming B(R) < M
Read all of R into a hash table...

...and join with all of S
Hash Join

Hash join
Cost = \( B(R) + B(S) \)

Isn’t this the same as block-nested-loop join where \( B(R) = N \)?
\[
\text{Cost} = B(R) + \frac{B(R)}{N} \cdot B(S)
\]
Hash Join

Hash join

Cost = B(R) + B(S)

Isn’t this the same as block-nested-loop join where B(R) = N?

Cost = B(R) + B(R)/N*B(S)

Yes! It’s the optimal “one-pass” join!
Join Algorithm Summary

- Nested-Loop Join
  - Versatile

- Hash Join (single pass)
  - Fast
  - Needs at least one input to be small

- Sort-Merge Join (single pass)
  - Fast
  - Sorts data at the same time!
  - Needs both inputs to be small
Sort-Merge Join

- Sort both tables into lists in memory
  - Since the sorted lists must contain all tuples, both tables together must fit in memory \((B(R)+B(S) < M)\)

- Merge the lists in memory to join
  - Preserves order!
Sort-Merge Join

Example equijoin

```sql
SELECT * 
FROM R, S 
WHERE R.attr = S.attr
```

M = 10 blocks
**Sort-Merge Join**

**Example equijoin**

```sql
SELECT * 
FROM R, S 
WHERE R.attr = S.attr
```

\[ M = 10 \text{ blocks} \]
Sort-Merge Join

Example equijoin

```sql
SELECT *
FROM R, S
WHERE R.attr = S.attr
```

$M = 10$ blocks

Main Memory

```
R: 1 3 5 7
S: 1 3 8
```

Disk

```
R: 1 7 3 5
S: 3 8 1
```

We don’t care about exact implementation after disk read since it’s small compared to IO.
Takeaways

- Nested-Loop Joins
  - Block-at-a-time \( \square B(R) + B(R) \times B(S) \)
  - Nested-block-loop \( \square B(R) + B(R)/N \times B(S) \)

- Hash Join and Sort-Merge Join \( \square B(R) + B(S) \)
Post-midterm

- Comparing join algorithms to algorithms with index structures to help