CS 457: Database Management Systems

Lecture 15
Execution - Operator Algorithms
Outline

• Join operator algorithms
  – One-pass algorithms
  – Index-based algorithms
  – Two-pass algorithms
Two-Pass Algorithms

• What if data does not fit in memory?
• Need to process it in multiple passes

• Two key techniques
  – Sorting
  – Hashing
Basic Terminology

• A run in a sequence is an increasing subsequence

• What are the runs?

2, 4, 99, 103, 88, 77, 3, 79, 100, 2, 50
Basic Terminology

• A run in a sequence is an increasing subsequence

• What are the runs?

2, 4, 99, 103, 88, 77, 3, 79, 100, 2, 50
External Merge-Sort: Step 1

Phase one: load M blocks in memory, sort, sent to disk, repeat
External Merge-Sort: Step 1

Phase one: load M blocks in memory, sort, sent to disk, repeat

Q: How long are the runs?

CS 457 - Fall 2018
External Merge-Sort: Step 1

Phase one: load M blocks in memory, sort, sent to disk, repeat

Q: How long are the runs?

A: Length = M blocks
External Merge-Sort: Step 2

**Phase two:** merge $M$ (or, $M-1$) runs into a bigger run

- Merge $M - 1$ runs into a new run
- Result: runs of length $M$ ($M - 1$) $\approx M^2$

If $B \leq M^2$ then we are done
Example

- Merging three runs to produce a longer run:

  0, 14, 33, 88, 92, 192, 322  
  2, 4, 7, 43, 78, 103, 523  
  1, 6, 9, 12, 33, 52, 88, 320

Output:

0
Example

- Merging three runs to produce a longer run:

  0, 14, 33, 88, 92, 192, 322
  2, 4, 7, 43, 78, 103, 523
  1, 6, 9, 12, 33, 52, 88, 320

Output:
0, ?
Example

• Merging three runs to produce a longer run:

0, 14, 33, 88, 92, 192, 322
2, 4, 7, 43, 78, 103, 523
1, 6, 9, 12, 33, 52, 88, 320

Output:
0, 1, ?
Example

• Merging three runs to produce a longer run:

0, 14, 33, 88, 92, 192, 322
2, 4, 7, 43, 78, 103, 523
1, 6, 9, 12, 33, 52, 88, 320

Output:
0, 1, 2, 4, 6, 7, ?
Cost of External Merge Sort

• Read+write+read = 3B(R)

• Assumption: B(R) <= M^2
Discussion

• What does $B(R) \leq M^2$ mean?
• How large can $R$ be?
Discussion

• What does $B(R) \leq M^2$ mean?
• How large can $R$ be?

• Example:
  – Page size = 32KB
  – Memory size 32GB: $M = 10^6$-pages
Discussion

• What does \( B(R) \leq M^2 \) mean?
• How large can \( R \) be?

• Example:
  – Page size = 32KB
  – Memory size 32GB: \( M = 10^6 \)-pages

• \( R \) can be as large as \( 10^{12} \)-pages
  – \( 32 \times 10^{15} \) Bytes = 32 PB
Merge-Join

Join $R \bowtie S$

• How?....
Merge-Join

Join $R \Join S$

• Step 1a: initial runs for $R$
• Step 1b: initial runs for $S$
• Step 2: merge and join
Merge-Join

\[ M_1 = \frac{B(R)}{M} \text{ runs for } R \]
\[ M_2 = \frac{B(S)}{M} \text{ runs for } S \]

Merge-join requires \( M_1 + M_2 \) runs; need \( M_1 + M_2 < M \)
External Merge Join Example
Step 1: Read $M$ pages of $R$ and sort in memory
**Step 1:** Read $M$ pages of $R$ and sort in memory

<table>
<thead>
<tr>
<th>Disk</th>
<th>Memory $M = 5$ pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R$</td>
<td>$S$</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>12</td>
<td>14</td>
</tr>
<tr>
<td>11</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Step 1: Read $M$ pages of $R$ and sort in memory, then write to disk.
Step 2: Repeat for next $M$ pages until all $R$ is processed

Disk

$R$

$S$

Memory $M = 5$ pages

Run 1 of $R$

Run 2 of $R$
Step 3: Do the same with S

Disk

R
4 1 1 7
5 2 11 9
3 4
8 6
7 9
12 14
5 11
2 3

S
3 0
1 7
4 3
2 5
9 8
11 9
12 1
5 7

Memory M = 5 pages

Run 1 of S
0 1
2 3
3 4
5 7
8 9

Run 2 of S
1 5
7 9
11 12
Step 4: Join while merging sorted runs

Total cost: $3B(R) + 3B(S)$

Memory $M = 5$ pages

Run 1 of $R$ | Run 2 of $R$
---|---
1 | 2
3 | 4
4 | 5
6 | 7
8 | 9

Run 1 of $S$ | Run 2 of $S$
---|---
0 | 1
2 | 3
3 | 4
5 | 7
8 | 9

Output buffer

Input buffers

Run 1

Run 2
Partitioned Hash Algorithms

- Partition R it into k buckets:
  \( R_1, R_2, R_3, \ldots, R_k \)
Partitioned Hash Algorithms

- Partition $R$ into $k$ buckets: $R_1, R_2, R_3, \ldots, R_k$

- Assuming $B(R_1) = B(R_2) = \ldots = B(R_k)$, we have $B(R_i) = B(R)/k$, for all $i$
Partitioned Hash Algorithms

- Partition R it into k buckets: $R_1, R_2, R_3, \ldots, R_k$

- Assuming $B(R_1) = B(R_2) = \ldots = B(R_k)$, we have $B(R_i) = B(R)/k$, for all $i$

- Goal: each $R_i$ should fit in main memory: $B(R_i) \leq M$
Partitioned Hash Algorithms

- Partition R into k buckets: 
  \( R_1, R_2, R_3, \ldots, R_k \)

- Assuming \( B(R_1) = B(R_2) = \ldots = B(R_k) \), we have 
  \( B(R_i) = B(R)/k \), for all \( i \)

- Goal: each \( R_i \) should fit in main memory: 
  \( B(R_i) \leq M \)

How do we choose k?
Partitioned Hash Algorithms

- We choose $k = M - 1$. Each bucket has size approx.
  $B(R)/(M - 1) \approx B(R)/M$

Assumption: $B(R)/M \leq M$, i.e. $B(R) \leq M^2$
Grace-Join

\[ R \bowtie S \]

Note: grace-join is also called *partitioned hash-join*
Grace-Join

$R \bowtie S$

• Step 1:
  – Hash $S$ into $M$ buckets
  – Send all buckets to disk

• Step 2
  – Hash $R$ into $M$ buckets
  – Send all buckets to disk

• Step 3
  – Join every pair of buckets

Note: grace-join is also called *partitioned hash-join*
Grace-Join

- Partition both relations using hash function $h$: $R$ tuples in partition $i$ will only match $S$ tuples in partition $i$. 
Grace-Join

- Partition both relations using hash fn $h$: R tuples in partition $i$ will only match S tuples in partition $i$.

Read in a partition of R, hash it using $h2 (\neq h!)$. Scan matching partition of S, search for matches.
Grace Join

• Cost: $3B(R) + 3B(S)$
• Assumption: $\min(B(R), B(S)) \leq M^2$
Grace Join
Also called
Partitioned Hash-Join Example
• Relation R: 10 pages Each page holds 2 tuples
• Relation S: 8 pages We only show value of join attribute for each tuple
• Memory: 5 pages

<table>
<thead>
<tr>
<th>Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
</tr>
<tr>
<td>S</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>12</td>
<td>11</td>
</tr>
<tr>
<td>14</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
</tr>
</tbody>
</table>

Memory M = 5 pages
Step 1: Read relation $S$ one page at a time and hash into $M-1$ (=4 buckets)
Step 1: Read relation S one page at a time and hash into the 4 buckets.
Step 1: Read relation S one page at a time and hash into the 4 buckets

Disk

R

<table>
<thead>
<tr>
<th>4</th>
<th>1</th>
<th>1</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>2</td>
<td>11</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

S

<table>
<thead>
<tr>
<th>1</th>
<th>7</th>
<th>1</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>11</td>
<td>9</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>

Memory M = 5 pages

Hash h: value % 4

Input buffer

<table>
<thead>
<tr>
<th>0</th>
<th></th>
<th>1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Step 1: Read relation $S$ one page at a time and hash into the 4 buckets

Disk

Memory $M = 5$ pages

Hash $h$: value $\% 4$

Input buffer
Step 1: Read relation S one page at a time and hash into the 4 buckets
Step 1: Read relation S one page at a time and hash into the 4 buckets
When a bucket fills up, flush it to disk
Step 1: Read relation S one page at a time and hash into the 4 buckets. When a bucket fills up, flush it to disk.
Step 1: Read relation $S$ one page at a time and hash into the 4 buckets. When a bucket fills up, flush it to disk.
Step 1: Read relation S one page at a time and hash into the 4 buckets.
At the end, we get relation S back on disk split into 4 buckets.
Step 2: Read relation R one page at a time and hash into same 4 buckets

Memory M = 5 pages

Hash h: value % 4

Input buffer
**Step 3:** Read one partition of R and create hash table in memory using a different hash function

Memory $M = 5$ pages

Hash $h2$: value $\%$ 3

Input buffer

Output buffer
Step 4: Scan matching partition of S and probe the hash table

Step 5: Repeat for all the buckets

• **Total cost**: $3B(R) + 3B(S)$
Hybrid Hash Join Algorithm

- Partition S into k buckets
  - t buckets $S_1, \ldots, S_t$ stay in memory
  - k-t buckets $S_{t+1}, \ldots, S_k$ to disk

- Partition R into k buckets
  - First t buckets join immediately with S
  - Rest k-t buckets go to disk

- Finally, join k-t pairs of buckets:
  $(R_{t+1}, S_{t+1}), (R_{t+2}, S_{t+2}), \ldots, (R_k, S_k)$
Hybrid Hash Join Algorithm
Hybrid Join Algorithm

• How to choose k and t?
Hybrid Join Algorithm

• How to choose k and t?
  
  – Choose k large but s.t. $k \leq M$
Hybrid Join Algorithm

• How to choose k and t?
  – Choose k large but s.t. $k \leq M$
Hybrid Join Algorithm

• How to choose k and t?
  – Choose k large but s.t. \( k \leq M \)
  – Choose \( t/k \) large but s.t. \( t/k \times B(S) \leq M \)
Hybrid Join Algorithm

• How to choose k and t?
  – Choose k large but s.t.
    \[ k \leq M \]
  – Choose t/k large but s.t.
    \[ \frac{t}{k} \cdot B(S) \leq M \]
Hybrid Join Algorithm

- How to choose k and t?
  - Choose k large but s.t. \( k \leq M \)
  - Choose \( t/k \) large but s.t. \( t/k \times B(S) \leq M \)
  - Together: \( t/k \times B(S) + k-t \leq M \)
Hybrid Join Algorithm

• How to choose k and t?
  – Choose k large but s.t.
    
    \[ k \leq M \]

  – Choose t/k large but s.t.
    
    \[ \frac{t}{k} \cdot B(S) \leq M \]

  – Together:
    
    \[ \frac{t}{k} \cdot B(S) + k - t \leq M \]

• Assuming \( \frac{t}{k} \cdot B(S) \gg k-t \):
  
  \[ \frac{t}{k} = \frac{M}{B(S)} \]
Hybrid Join Algorithm

• How to choose k and t?
  – Choose k large but s.t. 
  \[ k \leq M \]
  – Choose \( t/k \) large but s.t. 
  \[ \frac{t}{k} \cdot B(S) \leq M \]
  – Together: 
  \[ \frac{t}{k} \cdot B(S) + k-t \leq M \]

• Assuming \( \frac{t}{k} \cdot B(S) \gg k-t: \) 
  \[ \frac{t}{k} = \frac{M}{B(S)} \]
Hybrid Join Algorithm

• How to choose k and t?
  – Choose k large but s.t. \( k \leq M \)
  – Choose \( t/k \) large but s.t. \( t/k \times B(S) \leq M \)
  – Together: \( t/k \times B(S) + k-t \leq M \) (why?)

• Assuming \( t/k \times B(S) >> k-t: \) \( t/k = M/B(S) \)

One block/bucket in memory
First t buckets in memory
Total size of first t buckets
Number of remaining buckets
Hybrid Join Algorithm

Even better: adjust $t$ dynamically

- Start with $t = k$: all buckets are in main memory
- Read blocks from $S$, insert tuples into buckets
- When out of memory:
  - Send one bucket to disk
  - $t := t - 1$
- Worst case:
  - All buckets are sent to disk ($t=0$)
  - Hybrid join becomes grace join
Hybrid Join Algorithm

Cost of Hybrid Join:

- **Grace join**: \(3B(R) + 3B(S)\)
- **Hybrid join**:
  - Saves 2 I/Os for \(t/k\) fraction of buckets
  - Saves \(2t/k(B(R) + B(S))\) I/Os
  - Cost:
    \[
    (3-2t/k)(B(R) + B(S)) = (3-2M/B(S))(B(R) + B(S))
    \]
Hybrid Join Algorithm

• What is the advantage of the hybrid algorithm?
Hybrid Join Algorithm

• What is the advantage of the hybrid algorithm?

It degrades gracefully when $S$ larger than $M$:
• When $B(S) \leq M$
  – Main memory hash-join has cost $B(R) + B(S)$
• When $B(S) > M$
  – Grace-join has cost $3B(R) + 3B(S)$
  – Hybrid join has cost $(3-2t/k)(B(R) + B(S))$
Summary of External Join Algorithms

- Block Nested Loop: \( B(S) + B(R) \times B(S)/M \)

- Index Join: \( B(R) + T(R)B(S)/V(S,a) \)

- Partitioned Hash: \( 3B(R) + 3B(S); \)
  - \( \min(B(R),B(S)) \leq M^2 \)

- Merge Join: \( 3B(R) + 3B(S) \)
  - \( B(R) + B(S) \leq M^2 \)
Summary of Query Execution

• For each logical query plan
  – There exist many physical query plans
  – Each plan has a different cost
  – Cost depends on the data

• Additionally, for each query
  – There exist several logical plans

• Next lecture: query optimization
  – How to compute the cost of a complete plan?
  – How to pick a good query plan for a query?