Overview

- Memory hierarchy
- RAID (briefly)
- Disk space management
- Buffer management
- Files of records
- Page Formats
- Record Formats

Leverage OS for disk/file management?

- Layers of abstraction are good … but:
Leverage OS for disk/file management?

• Layers of abstraction are good … but:
  – Unfortunately, OS often gets in the way of DBMS

DBMS wants/needs to do things “its own way”

– Specialized prefetching
– Control over buffer replacement policy
  • LRU not always best (sometimes worst!!)
– Control over thread/process scheduling
  • “Convoy problem”
    – Arises when OS scheduling conflicts with DBMS locking
– Control over flushing data to disk
  • WAL protocol requires flushing log entries to disk

Disks and Files

• DBMS stores information on disks.
  – but: disks are (relatively) VERY slow!
• Major implications for DBMS design!

Major implications for DBMS design:

– READ: disk -> main memory (RAM).
– WRITE: reverse
  – Both are high-cost operations, relative to in-memory operations, so must be planned carefully!
Why Not Store It All in Main Memory?

- **Costs too much.**
  - disk: ~$0.1/Gb; memory: ~$10/Gb
  - High-end Databases today in the 10-100 TB range.
  - Approx 60% of the cost of a production system is in the disks.
- **Main memory is volatile.**

Note: some specialized systems do store entire database in main memory.

The Storage Hierarchy

- Main memory (RAM) for currently used data.
- Disk for the main database (secondary storage).
- Tapes for archiving older versions of the data (tertiary storage).
The Storage Hierarchy

- Main memory (RAM) for currently used data.
- Disk for the main database (secondary storage).
- Tapes for archiving older versions of the data (tertiary storage).

Jim Gray’s Storage Latency Analogy: How Far Away is the Data?

10**9 tape Andromeda 2,000yr
10**6 disk Pluto 2yr
100 Memory Pittsburgh 1.5h
10 On board cache This building 10min
2 on chip cache This room 1min
1 registers In my head 1min

Disks
- Secondary storage device of choice.
- Main advantage over tapes: random access vs. sequential.
- Data is stored and retrieved in units called disk blocks or pages.
- Unlike RAM, time to retrieve a disk page varies depending upon location on disk.
  - relative placement of pages on disk is important!

Anatomy of a Disk
- Sector
- Track
- Cylinder
- Platter
- Block size = multiple of sector size (which is fixed)
Accessing a Disk Page

• Time to access (read/write) a disk block:
  – seek time: moving arms to position disk head on track
  – rotational delay: waiting for block to rotate under head
  – transfer time: actually moving data to/from disk surface
Rotational Delay

Accessing a Disk Page

- Relative times?
  - seek time: about 1 to 20msec
  - rotational delay: 0 to 10msec
  - transfer time: < 1msec per 4KB page

Seek time & rotational delay dominate

- Key to lower I/O cost: reduce seek/rotation delays!
- Also note: For shared disks, much time spent waiting in queue for access to arm/controller
Arranging Pages on Disk

- “Next” block concept:
  - blocks on same track, followed by
  - blocks on same cylinder, followed by
  - blocks on adjacent cylinder
- Accessing ‘next’ block is cheap
- An important optimization: pre-fetching
  - See R&G page 323

Q1: Why not equal?
Q2: Why?

Rules of thumb…

1. Memory access much faster than disk I/O (~ 1000x)
- “Sequential” I/O faster than “random” I/O (~ 10x)

Overview

- Memory hierarchy
- RAID (briefly)
- Disk space management
- Buffer management
- Files of records
- Page Formats
- Record Formats

Disk Arrays: RAID

- Benefits:
  - Higher throughput (via data “striping”)
  - Longer MTTF
  (Why?)
Disk Arrays: RAID

- Benefits:
  - Higher throughput (via data “striping”)
  - Longer MTTF (via redundancy)

Overview

- Memory hierarchy
- RAID (briefly)
- Disk space management
- Buffer management
- Files of records
- Page Formats
- Record Formats

Disk Space Management

- Lowest layer of DBMS software manages space on disk
- Higher levels call upon this layer to:
  - allocate/de-allocate a page
  - read/write a page
- Best if requested pages are stored sequentially on disk! Higher levels don’t need to know if/how this is done, nor how free space is managed.
Recall: DBMS Layers

Buffer Management in a DBMS

Buffer Management in a DBMS

When a Page is Requested ...

• Data must be in RAM for DBMS to operate on it!
• Buffer Mgr hides the fact that not all data is in RAM

Buffer pool information table contains:

• If requested page is not in pool:
  – Choose an (un-pinned) frame for replacement
    • If frame is "dirty", write it to disk
  – Read requested page into chosen frame
• Pin the page and return its address
When a Page is Requested ...

Buffer pool information table contains:
<frame#, pageid, pin_count, dirty-bit>

• If requested page is not in pool:
  – Choose an (un-pinned) frame for replacement
    • If frame is "dirty", write it to disk
  – Read requested page into chosen frame
• Pin the page and return its address

When a Page is Requested ...

• If requests can be predicted (e.g., sequential scans)
  • then pages can be pre-fetched several pages at a time!

More on Buffer Management

• When done, requestor of page must
  – unpin it, and
  – indicate whether page has been modified: dirty bit
• Page in pool may be requested many times:
  – pin count
• if pin count = 0 ("unpinned"), page is candidate for replacement

More on Buffer Management

• CC & recovery may entail additional I/O when a frame is chosen for replacement.
  (Write-Ahead Log protocol; more later.)
Buffer Replacement Policy

• Frame is chosen for replacement by a replacement policy:
  – Least-recently-used (LRU), MRU, Clock, etc.
• Policy -> big impact on # of I/O’s; depends on the access pattern.

LRU Replacement Policy

• Least Recently Used (LRU)
  – for each page in buffer pool, keep track of time last unpinned
  – replace the frame which has the oldest (earliest) time
  – very common policy: intuitive and simple

• Problems?

LRU Replacement Policy

• Problem: Sequential flooding
  – LRU + repeated sequential scans.
  – # buffer frames < # pages in file means each page request causes an I/O. MRU much better in this situation (but not in all situations, of course).

Sequential Flooding – Illustration

-Repeated scan of file …
Sequential Flooding – Illustration

Repeated scan of file ...

Repeated scan of file ...

Repeated scan of file ...

Repeated scan of file ...

BUFFER POOL

LRU: 1 116 242 105

MRU: 102 116 242 105

BUFFER POOL

LRU: 1 2 242 105

MRU: 102 116 242 105

BUFFER POOL

LRU: 1 2 3 105

MRU: 102 116 242 105

BUFFER POOL

LRU: 1 2 3 4

MRU: 102 116 242 105
How will MRU work?

Sequential Flooding – Illustration

LRU: 1 2 3 4 will not re-use these pages;

MRU: 102 116 242 105

Repeated scan of file ...

Repeated scan of file ...
Sequential Flooding – Illustration

Repeated scan of file ...

Other policies?

- LRU is often good - but needs timestamps and sorting on them
- something easier to maintain?

“Clock” Replacement Policy

Main ideas:
- Approximation of LRU.
- Instead of maintaining & sorting time-stamps, find a ‘reasonably old’ frame to evict.
- How? by round-robin, and marking each frame - frames are evicted the second time they are visited.
- Specifically:
“Clock” Replacement Policy

- Arrange frames into a cycle, store one “reference bit” per frame
- When pin count goes to 0, reference bit set on (= ‘one life left’ - not ready for eviction yet)
- When replacement necessary, get the next frame that has reference-bit = 0

```
do {
    if (pincount == 0 && ref bit is off)
        choose current page for replacement;
    else if (pincount == 0 && ref bit is on)
        turn off ref bit;
        advance current frame;
} until a page is chosen for replacement;
```
“Clock” Replacement Policy

Summary

- Buffer manager brings pages into RAM.
- Very important for performance
  - Page stays in RAM until released by requestor.
  - Written to disk when frame chosen for replacement (which is sometime after requestor releases the page).
  - Choice of frame to replace based on replacement policy.
  - Good to pre-fetch several pages at a time.

Overview

- Memory hierarchy
- RAID (briefly)
- Disk space management
- Buffer management
- Files of records
- Page Formats
- Record Formats

Files

- **FILE**: A collection of pages, each containing a collection of records.
- Must support:
  - insert/delete/modify record
  - read a particular record (specified using record id)
  - scan all records (possibly with some conditions on the records to be retrieved)
Alternative File Organizations

Several alternatives (w/ trade-offs):

- Heap files: Suitable when typical access is a file scan retrieving all records.
- Sorted Files:
- Index File Organizations: } later

Files of records

- Heap of pages
  - as linked list or
  - directory of pages

Heap File Using Lists

- The header page id and Heap file name must be stored somewhere.
- Each page contains 2 ‘pointers’ plus data.

Heap File Using Lists

- Any problems?
Heap File Using a Page Directory

• The entry for a page can include the number of free bytes on the page.
• The directory is a collection of pages; linked list implementation is just one alternative.
  – Much smaller than linked list of all HF pages!

Overview

• Memory hierarchy
• RAID (briefly)
• Disk space management
• Buffer management
• Files of records
• Page Formats
• Record Formats

Page Formats

• fixed length records
• variable length records
Problem definition

Q: How would you store records on a page/file, such that
1. you can point to them
2. you can insert/delete records with few disk accesses

Page Formats

Important concept: \textit{rid} == record id
Q0: why do we need it?
A0: eg., for indexing
Q1: How to mark the location of a record?
A1:
\Rightarrow Q2: Why not its byte offset in the file?
A2:

Page Formats

Important concept: \textit{rid} == record id
Q0: why do we need it?
A0: eg., for indexing
Q1: How to mark the location of a record?
A1:
Q2: Why not its byte offset in the file?
A2: too much re-organization on ins/del.
Page Formats

Important concept: \( rid \) == record id

Q0: why do we need it?
A0: eg., for indexing

Q1: How to mark the location of a record?
A1: \( rid = \text{record id} = \text{page-id} \& \text{slot-id} \)

Q2: Why not its byte offset in the file?
A2: too much re-organization on ins/del.

Fixed length records

• Q: How would you store them on a page/file?

\[
\begin{array}{c}
\text{4kb page} \\
\end{array}
\]

Fixed length records

• OK – how about insertion?

\[
\begin{array}{c}
\text{'Packed'} \\
\end{array}
\]
Fixed length records

- OK – how about insertion?

`Packed`

- Number of full slots

Free space

Fixed length records

- How about deletion?

`Packed`

- Number of full slots

Free space

Fixed length records

- How about deletion?
  - Bad - we have too much to reorganize/update

B-tree

Fixed length records

- What would you do?
Fixed length records

- Q: How would you store them on a page/file?
- A2: Bitmaps

Variable length records

- Q: How would you store them on a page/file?
- A2: Pack them
  - keep ptrs to them
Variable length records

- Q: How would you store them on a page/file?

  • pack them
  • keep ptrs to them
  • mark start of free space

Page header

occupied records

Slot directory

other info (# slots etc)

SLOTTED PAGE

Variable length records

- Q: How would you store them on a page/file?

  • how many disk accesses to insert a record?
  • to delete one?

Page header

occupied records

SLOTTED PAGE

SLOTTED PAGE organization - popular.

SLOTTED PAGE

Overview

- Memory hierarchy
- RAID (briefly)
- Disk space management
- Buffer management
- Files of records
- Page Formats
- Record Formats
Formats of records

- Fixed length records
  - How would you store them?
- Variable length records

Record Formats: Fixed Length

\[
\text{Address} = B + L1 + L2
\]

- Information about field types same for all records in a file; stored in system catalogs.
- Finding \(i^{th}\) field done via arithmetic.

- Fixed length records: straightforward - store info in catalog
- Variable length records: encode the length of each field
  - ?
  - ?
Variable Length records

- Two alternative formats (# fields is fixed):
  
<table>
<thead>
<tr>
<th>$</th>
<th>$</th>
<th>$</th>
<th>$</th>
</tr>
</thead>
</table>
  
  Fields Delimited by Special Symbols

- Array of Field Offsets

Pros and cons?

Offset approach: usually superior (direct access to i-th field)

Conclusions

- Memory hierarchy
- Disks: (>1000x slower) – thus
  - pack info in blocks
  - try to fetch nearby blocks (sequentially)
- Buffer management: very important
  - LRU, MRU, Clock, etc
- Record organization: Slotted page