6.830 Lecture 5
3/3/21

Project partners due next Wednesday. Final project details posted.

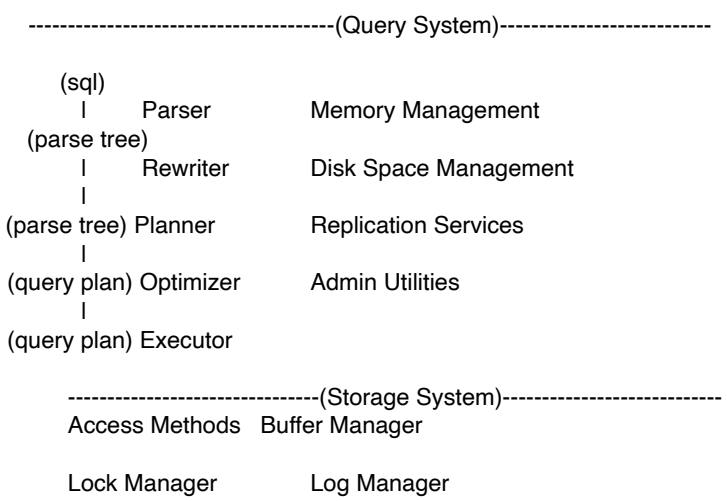
Lab 1 due next Tuesday — start now!!!

Recap Anatomy of a database system

Major Components:

Admission Control

Connection Management



SimpleDB Overview

Today:

cost estimation basics
buffer pool

What's the "cost" of a particular plan?

CPU cost (# of instructions)
- 1 ghz == 1 billions instrs / sec, 1 nsec / instr

Spinning Disk
I/O cost (# of pages read, # of seeks)
- 100 MB / sec = 10 nsec / byte
- 10 msec / io operation = 100 io ops / sec

Random I/O disk = page read + seek

Flash disk
- 300-500 MB/sec seq rd/write
- 10K + random iops / sec
  (100 usec / io operation)
  4K pages ~= 40 MB/sec

Random I/O can be a real killer (10 million instrs/seek) . When does a disk need to seek?

Which do you think dominates in most database systems?

(Depends. Not always disk I/O. Typically vendors try to configure machines so they are 'balanced'. Can vary from query to query. Because of sequential access methods, buffer pool, we can often end up CPU bound.)

For example, fastest TPC-H benchmark result (data warehousing benchmark), on 10 TB of data, uses 1296 74 GB disks, which is 100 TB of storage. Add'l storage is partly for indices, but partly just because they needed add'l disk arms. 72 processors, 144 cores -- ~10 disks / processor!

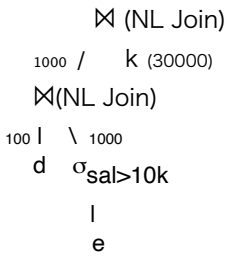
But, if we do a bad job, random I/O can kill us!

100 tuples/page
10 pages RAM
10 KB/page

10 ms seek time
100 MB/sec I/O

```
select * from
emp, dept., kids
where e.sal > 10k
emp.dno = dept.dno
e.eid = kids.eid
```

|dept| = 100 records = 1 page = 10 KB
|emp| = 10K = 100 pages = 1 MB
|kids| = 30K = 300 pages = 3 MB



1st Nested loops join -- 100,000 predicate ops; 2nd nested loops join -- 30,000,000 predicate ops

Let's look at # disk I/Os assuming LRU and no indices

Join 1
if d is outer:
- 1 scan of d
- 100 sec scans of e
- (100 x 100 pg. reads) -- cache doesn't benefit since e doesn't fit

- 1 scan of e: 1 seek + read in 1MB
  - 10 ms + 1 MB / 100 MB/sec = 20 msec
- 20 ms x 100 depts = 2 sec

- 10 msec seek to start of d and read into memory

2.1 secs

if d is inner
- read page of e -- 10 msec
- read all of d into RAM -- 10 msec
- seek back to e -- 10 sec
- scan rest of e -- 10 msec, joining with d in memory

Because d fits into memory, total cost is just 40 msec
NL # predicates evaluated doesn't depend on inner vs outer, but I/O cost does (due to caching effect!)

Join 2

k inner:

1000 scans of 300 pages
3 / 100 = 30 msec + 10 msec seek = 40 x 1000 = 40 sec

if plan is pipelined, k must be inner

So what will be cached?

That's the job of the buffer pool.

Buffer pool is a cache for memory access. Typically caches pages of files / indices.

Convenient "bottleneck" through which references to underlying pages go. When page is in buffer pool, don't need to read from disk. Updates can also be cached.

What is optimal buffer pool caching strategy? Always LRU?
No.
What if some relation doesn't fit into memory? (MRU preferred)
2 pages RAM, 3 pages of a relation R -- a, b c, accessed sequentially in a loop

	Access			
RAM	1	2	3	4
1	a	a	c	c
2		b	b	a

==> always evicting page we are about to access! Note that MRU would hit on 2/3

What if I know I will not be accessing a relation again in a query?
Are some records higher value than others?

Postgres examples