CS186 Discussion 1

TA: Victor Zhu
Sections 102, 109

Today

- Intro
- Buffer Management
- Sequential vs. Random IO
- SQL Join Algorithm
- Worksheet

About Me

- EECS, 4th year...last semester :'(
- Running, watching movies, traveling
- AI -> ML -> CV
- Took 186 Fall 2012
- victor.zhu@berkeley.edu
- OH, Friday 2-3pm, Soda 283E (Alcove)

Course Advice

- Around CS188 difficulty
- Keep up with readings
- Attend lecture
- Come to discussion! :D :D
- Start projects early (first one is on Wed!)
- Questions?

The Page

- Can’t fit all the database in memory
- Tables split up into pages
- Size varies, but usually 4KB, 8KB, etc.
- If tuple is 2.1KB, probably don’t want 4KB/page (low tuple density, wasted space)
- If you want to look at Tuple A which is on Page A, you have to load the entire page into memory

The Buffer Pool

- Set of pages reserved for memory management
- Buffer = Frame = Page
- Can’t fit all of database in memory
- Buffer Pool is fast (1000x)
- Data must be in memory for operation!
- Fit some relevant subset
- What’s “relevant”?
  - Stuff we need to operate on right now
  - Stuff we may need to operate on in the near future

Managing the Buffer Pool

Access pages C,B,A

BufferPool (3 pages) initially full with E,A,H

Evict E, H

What was our hit rate?

- (# cache hits) / (# reads)
- 1 / 3.
- Would be 0 if we had evicted A.

In reality, would we have known not to evict A?

Eviction Policies

Big effect on #IOs

Assuming buffer pool is full. If not, populate.

LRU (Least Recently Used)

- Evict the page that was last used/accessed the farthest in the past.
- Good, but need timestamps + sorting
- Better solution?

MRU (Most Recently Used)

- Just the opposite!
- Only good for sequential flooding

Sequential Flooding

- Nasty situation caused by LRU and repeated sequential scans
- # buffer frames < # pages in file

Example: Access Pattern: ABCDEABCDE, BufferPool is 4 pages, initially empty

Try it out yourself!

- With LRU - 10 cache misses
- With MRU - 4 misses, 6 hits

Sequential vs. Random IOs

- IO = Getting stuff from/to disk
- If access order can be predicted (sequential), we can prefetch many pages, and next page to fetched will be close at hand
- Else, the next page could be anywhere on disk = slowwwww
- Rule of thumb: Random IO takes 10x more time due to disk arm seeking

Naive Nested Loop Join

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

```
For each Rtuple in R do
    For each Stuple in S do
        If Rtuple.id = Stuple.id
            Output tuple <Rtuple, Stuple>
```

Here, table R is the outer, S is the inner.

For every tuple in R, scan all of S.

Worksheet time!
Groups of 3-4
10 minutes

Consider two tables:
Students(sid, name, year, department), 200 pages, 1,000 tuples
Enrolled(sid, course, grade), 500 pages, 6,000 tuples

Query: for each student, list all his/her class grades:
SELECT name, course, grade
FROM Students, Enrolled
WHERE Students.sid = Enrolled.sid

Assume that we only have 1 disk, and that we do **not** have to write the resultant tuples back to disk. Consider the join of Student and Enrolled in a nested loop (the naïve nested loops algorithm) with Student as the outer. Also assume that we **don't** cache any pages in our buffer pool.

1. What is the total number of IOs this join will require?

2. Of the total number of IOs, how many are **sequential** IOs? (Assume that the data for each relation is located in a continuous clump, but the two relations are located in different places.)
   Of the total number of IOs, how many are **random** IOs?

Can think of this in terms of...

for each student page in Students: (executes 200 times)

    Load student page (1 IO cost)

    for each tuple in student page: (this for loop executes a total of 1000 times)

        for each enrolled page in Enrolled: (executes 500 times for each student)

            Load enrolled page (1 IO cost)

Consider two tables:

Students(sid, name, year, department), 200 pages, 1,000 tuples

Enrolled(sid, course, grade), 500 pages, 6,000 tuples

Query: for each student, list all his/her class grades:

SELECT name, course, grade

FROM Students, Enrolled

WHERE Students.sid = Enrolled.sid

1. What is the total number of IOs this join will require?
   200 + 1000 * 500 = 500,200 IOs
2. Of the total number of IOs, how many are **sequential** IOs? (Assume that the data for each relation is located in a continuous clump, but the two relations are located in different places.)
   Outer: Every access will require a random IO, from the end of the inner.
   Inner: 1000 * 499 = 499,000
   Total: 499,000 sequential IOs
   Of the total number of IOs, how many are **random** IOs?
   Outer: 200
   Inner: 1000
   Total: 1200 random IOs