class 11

b-trees

prof. Stratos Idreos

HTTP://DASLAB.SEAS.HARVARD.EDU/CLASSES/CS165/
Midway check-in:
Two design docs tmr (Canvas) & tests on Sunday

Next weekend: Lab marathon for midway check-in & tests will run every 1 hour
(similar idea to OH for midterm preparation last weekend)

Remember: Midway check-in goal is not to test you but to “force” you to see the scope and size of the project if you are behind and get you going
**Midway check-in:**
Two design docs tmr (Canvas) & tests on Sunday

Next weekend: Lab marathon for midway check-in & tests will run every 1 hour (similar idea to OH for midterm preparation last weekend)

Remember: Midway check-in goal is not to test you but to “force” you to see the scope and size of the project if you are behind and get you going

**Midterm 1:**
Grading out in about 10 days - come to OH for 1-1 feedback + 1-2 sections

Remember: Midterm goal is not to test you but to *force* you see the concepts and way of thinking if you have fallen behind.
**Midway check-in:**
Two design docs tmr (Canvas) & tests on Sunday

Next weekend: Lab marathon for midway check-in & tests will run every 1 hour (similar idea to OH for midterm preparation last weekend)

Remember: Midway check-in goal is not to test you but to “force” you to see the scope and size of the project if you are behind and get you going

**Midterm 1:**
Grading out in about 10 days - come to OH for 1-1 feedback + 1-2 sections

Remember: Midterm goal is not to test you but to *force* you see the concepts and way of thinking if you have fallen behind.

**Research Meeting 2:**
Friday 5pm, New research on tree-based concurrency control
Scaling to many many cores. My office and Zoom. No recording.
Email me if you want to participate by Thursday.
index knows order about the data

filtering data: point/range queries/locate data to update
A B C

initial state
columns in
insertion order

sorted A B C

propagate
order of A

sorted A B C
select max(D), min(E) from R where (A>10 and A<40) and (B>20 and B<60)

avoid scan of A
avoid TR on B
work on a restricted area
across all columns
good for memory hierarchy

binary search for 10 & 40
for all B values between pos1 & 2: if B>20 and B<60
mark bit vector at pos i
for each marked position
max(D)
today+1: more about indexing

tree indexes in data systems
primary/clustered (all columns)

secondary/unclustered indexes
subset of columns

you are doing both in the project
Btree on A, A is sorted, order is propagated to the rest of the columns

every table can/should have one (be a) clustered index

clustered index on A (no need for mappings)
Btree on C, copy of C is sorted, we keep a copy of the positions that map on the clustered index.

secondary index on any column(s) needs positions
original order

A  B  C

clustered index/base data

A  B  C

secondary index

pos C

7  8  10  1
5

classified index case plan vs secondary index plan

select max(B)  
from R  
where A<20

select max(B)  
from R  
where C<20

CS165, Fall 2017
Stratos Idreos
clustered index case plan vs secondary index plan

original order

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
</table>

clustered index/base data

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
</table>

secondary index

pos C

7 8 10 11 5

same discussion as with sorting
“It could be said that the world’s information is at our fingertips because of B-trees”

**Goetz Graefe**
Microsoft, HP Fellow, now at Google
ACM Software System Award
ACM SIGMOD Innovations Award
b-tree = a “fat” tree, each node holds $>>1$ keys

Why is this good?
ok and how do we build, search, update a tree efficiently?

structure = complexity = if statements, random access, instruction misses, etc. 
= no free lunch

node size, data organization fanout ...
Page size: 64K - holds 16K 4 byte ints
N elements, P pages

sorted array
info to navigate lower level value-pointer

12,20  35,...  50,...

<12  >=12

1,2,3...  12,15,17  20,...

page size: 64K - holds 16K 4 byte ints
N elements, P pages
info to navigate lower level value-pointer

12,20  35,...  50,...

<12  >=12

1,2,3...  12,15,17  20,...

4+4 bytes for each page (value+pointer)
64K/8 = index 8K pages

page size: 64K - holds 16K 4 byte ints
N elements, P pages

sorted array
info to navigate lower level value-pointer

35,50
can **index 8K pages**
of the next level

12,20  35,...  50,...

<12  >=12

1,2,3,...  12,15,17  20,...  ...

page size: 64K - holds 16K 4 byte ints
N elements, P pages

4+4 bytes for each page (value+pointer)
64K/8= **index 8K pages**
The diagram illustrates a tree structure with a root node containing the value 35,50. The tree has internal nodes that are labeled with values 12,20, 35,..., 50,..., and leaves labeled with values 1,2,3,..., 12,15,17, 20,..., etc. The height of the tree is given by $\log_{\text{fanout}} N$. The fanout of the tree is indicated by the number of children each internal node has. The leaves of the tree correspond to the values 1,2,3,..., 12,15,17, 20,..., etc.
random accesses

height \log_{\text{fanout}} N

root

fanout

internal nodes

leaves

1,2,3...

12,15,17

20,...

35,50

50,....

...
how do we search the leaves? a) sorted and b) unsorted leaves

get 15
get 15-55
should we store leaves as independent nodes or as a single contiguous column
should we store leaves as independent nodes or as a single contiguous column

diff in structure of clustered vs secondary index
for a point query (looking a single key)

1) compare tree search to binary search cost
   assume a clustered index

assume: input 1 Billion integers array, page size 64Kb, system 64bit

2) design a cache conscious tree
   (a tree that minimizes cache misses)
**thinking process: (keep it simple and focus only on what matters for the question)**

We know: 1B ints 8 bytes each, 64 bit system: our pointers are 8 bytes - page size 64Kb

**How many pages hold our data?**

total data size / page size: 1B*8bytes/64Kb=122*10^3 pages

**Binary search cost: log2(N)**

Since we measure data movement = page access cost log2(pages)

So log2(122*10^3)=log2(1.22*10^5) =~17 pages

(For every factor of 10 we add ~3.3)

**B-tree:** We assume a design where each b-tree node size is exactly one page

# of keys each internal node holds: page size/(key size+pointer size)=64Kb/(8+8)bytes=4K

--> Fanout is 4K

**Search cost:**

1) With each internal level we filter through xFanout pages. So from root we can search 4K pages; at the next level we filter through 4K*4K=16M pages. 16M >122*10^3 (total data pages) which means that two internal levels are enough to index our data pages. Our total cost is two pages (one for each internal level) + one more to access the correct data page. So 3 pages in total

2) logFanout(total pages)

**What if we have more data?** Say 100 billion data entries. -~122*10^5 data pages

Binary search = log2(1.22*10^7) = ~23 pages

B-tree: 16M >122*10^5 so we can still search by accessing just 3 pages
search for key X:
search (Node, Key)
find where Key falls in the keys of Node
binary search local keys
then follow the respective pointer and repeat

we move the whole node but we really need just one data pointer...
-内部节点

指针 - 关键字 - 指针 - 关键字 - 指针 - ... - 指针

n 个关键字

n+1 个指针

child 1  child 2  child 3  child n+1

- 内部节点

关键字 - 关键字 - 关键字 - 关键字 - 关键字 - 关键字 - 关键字 - ... - 指针

2n 个关键字

1 个指针

child 1  child 2  child 3  ...  child 2n+1

单个连续内存区域
Browse: **textbook**: Chapters 8, 9 (general indexing & H/W)

Read: **textbook**: Chapter 10 (b-trees)

Read: **Modern B-Tree Techniques**
by Goetz Graefe
Foundations and Trends in Databases, 2011
Sections: 1,2,3,5

Read: **Making B+trees Cache Conscious in Main Memory**
Jun Rao and Ken Ross
ACM **SIGMOD** International Conference on Management of Data, 2000
b-trees

DATA SYSTEMS

prof. Stratos Idreos

HARVARD
School of Engineering and Applied Sciences