b-trees 2.0

prof. Stratos Idreos

HTTP://DASLAB.SEAS.HARVARD.EDU/CLASSES/CS165/
Laura Haas

Data Systems Researcher

Director of IBM Research’s Accelerated Discovery Lab & Harvard alumna

The Power Behind the Throne: Information Integration in the Age of Data-Driven Discovery

big data V’s (it is not about size only)

volume velocity variety veracity

ACM SIGMOD Codd Innovation Award, 2015
Jignesh Patel

Data Systems Researcher

Professor at Wisconsin University

October 8

part of the CS colloquium (4pm)

hardware/software co-design
next research meeting - Tuesday Oct 20, 6:30, Pierce Pierce 213

MaSM: Efficient Online Updates in Data Warehouses
M. Athanassoulis, S. Chen, A. Ailamaki, P. Gibbons and R. I. Stoica.,
ACM SIGMOD International Conference on Management of Data, 2011

Designing Access Methods: The RUM Conjecture
Manos Athanassoulis, Michael S. Kester, Lukas Maas, Radu Stoica, Stratos Idreos, Anastasia Ailamaki, Mark Callaghan

updates, flash disks and how to design new data structures
**midterms**
how to prepare

- open book, notes, no laptop/discussion
- material from lectures only
- check all quizzes and questions

**quiz-like questions** - no exact answer
explain all steps and tradeoffs

**expectations:** describe the design space - chose what you think is the best approach (>1 if we ask for it) and then analyze in detail all requests - if you made the wrong choice in the begging it is ok - but say so if you find out in the end and explain as much as possible

**next Wednesday during class:** TFs will go over past quizzes
**Sunday before midterm:** Stratos office hours 2-6pm
clustered (all columns)

index

data

secondary indexes
subset of columns
Btree on A, A is sorted, order is propagated to the rest of the columns

clustered index on A
(no need for positions)

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

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

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

sorted array

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

<12

12,20

35,...

50,...

>=12
Page size: 64K - holds 16K 4 byte ints
N elements, P pages

info to navigate lower level value-pointer

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

sorted array

<12 =>12
Page size: 64K - holds 16K 4 byte ints

N elements, P pages

info to navigate lower level value-pointer

4+4 bytes for each page (value+pointer)

64K/8 = index 8K pages

1, 2, 3…
12, 15, 17
20, …
30, 50
35, …
50, …

>=12
<12

can index 8K pages
of the next level

sorted array
The image illustrates a B-tree data structure. The nodes are labeled with numbers, and the relationships between the root, internal nodes, and leaves are shown. The height of the tree is given by $\log_{\text{fanout}} N$, where $\text{fanout}$ is the fanout of the tree. The random accesses are indicated, and the diagram shows the structure of the tree with internal nodes and leaves.
**b-tree** - dynamic tree - always balanced

Every node/page is at least 50% full (except root)
leaves

get 15
get 15-25

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

<12  >=12

1,2,3,...  12,15,17  20,...  ...
searching internal node

(v1,p1) (v2,p2) (v3,p3) (v4,p4) (v5,p5) (v6,p6)…
what does a leaf contain

searching leaf nodes
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
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
how big should nodes be
updates

30,50

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

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

2,5,6,3,2  22,25,24

7,8,3,5,4

good
new value

leaf
leaves

node capacity = 4
leaves

node capacity = 4
insert “6”
node capacity = 4
delete "5"
node capacity = 4
how to load a b-tree

for(i=0; i<totalValues;i++)
    insertToBtree(tree,value[i])

?
buffers for loading

root level buffer

k level buffer

leaf level buffer

any problems
bulk loading
1. sort all
2. build tree
bulk loading
1. sort all
2. build tree

```
3.5
/
/   |
/    |
1,2   3,4 5,6
```

7,8  9,10
bulk loading
1. sort all
2. build tree
the more data we can fit in a node (without changing its size) the faster our index becomes

correct - wrong - why
affects the way we design algorithms and data structures
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
internal node

pointer - key - pointer - key - pointer - … - pointer

n keys
n+1 pointers

child 1  child 2  child 3  child n+1

internal node

key - key - key - key - key - key - key - … - pointer

2n keys
1 pointer

child 1  child 2  child 3  …  child 2n+1

single contiguous memory area
project milestone 2
(reasonably) cache conscious b-tree
use as both clustered and secondary index
(start with a linked list)
Employee(Id, name, address, office, salary, year hired, …)

We have a B-tree on table Employee which uses salary as the key and also contains attributes “name” and “year hired”.

We want to give a 5% raise to all employees that work for more than 10 years in the company and have a salary lower than 100K.

1) Write the SQL query
2) How to update the B-tree?
3) What is the query plan?
b-tree on employee.salary

get all qualifying IDs first
then update in one go
or
maintain an extra structure
e.g., a bit vector
or hash on tuple ID
to remember
the updated tuples

for each tuple
if it qualifies (check years hired)
update

we are going to keep updating until everybody is at least at 100K
Modern B-Tree Techniques
by Goetz Graefe
Foundations and Trends in Databases, 2011
Sections: 1,2,3,5

textbook: Chapter 10 (b-trees)

Making B+trees Cache Conscious in Main Memory
Jun Rao and Ken Ross
ACM SIGMOD International Conference on Management of Data, 2000
b-trees 2.0
DATA SYSTEMS
prof. Stratos Idreos