CS 265
Stratos Idreos
BIG DATA SYSTEMS

NoSQL | Neural Networks | SQL | Graph | Data Science
Logistics:

All details on class website
Slides always uploaded a few hours after class
Recordings available in canvas
Systems vs Research project: based on background
Required for research project: CS165||161||Systems PhD

http://daslab.seas.harvard.edu/classes/cs265/
review and slides should focus on

what is the problem
why is it important
why is it hard
why existing solutions do not work
what is the core intuition for the solution
solution step by step
does the paper prove its claims
exact setup of analysis/experiments
are there any gaps in the logic/proof
possible next steps

* follow a few citations to gain more background
how to prepare slides

no bullets  2 colors  big text  images  animation for examples
how to prepare slides

- no bullets
- 2 colors
- big text
- images
- animation for examples

story

- one message per slide
- connection from slide to slide
how to prepare slides

- no bullets
- 2 colors
- big text
- images
- animation for examples

story

- one message per slide
- connection from slide to slide

1) prepare slides, 2) meet with Subarna and Stratos the week before
how to prepare slides

- no bullets
- 2 colors
- big text
- images
- animation for examples

story

- one message per slide
- connection from slide to slide

1) prepare slides, 2) meet with Subarna and Stratos the week before

We will have a schedule to sign up by Week 4
what should you be doing?

READING
Get familiar with the very basics of traditional database architectures:

Get familiar with very basics of modern database architectures:

Get familiar with the very basics of modern large scale systems:

The Periodic Table of Data Structures.
Stratos Idreos, Kostas Zoumpatianos, Manos Athanassoulis, Niv Dayan, Brian Hentschel, Michael S. Kester, Demi Guo, Lukas Maas, Wilson Qin, Abdul Wasay, Yiyou Sun. IEEE Data Engineering Bull. Sep, 2018
Readings for this week (and systems project)


review a few past slides
declarative interface
ask “what” you want

data* system

the system decides
“how” to best store
and access data
DESIGN SPACE  COST SYNTHESIS  WHAT-IF
CEREAL MILK PANNA COTTA

non obvious valid combinations

milk + cream + sugar + vanilla/lemon
fundamental building blocks properties when combined
fundamental building blocks

properties when combined
How many and which structures are possible?

Can we compute performance w/o coding?
Today

start in depth discussion on NoSQL storage
(systems project and related research projects)

first steps in constructing a design space
FIRST PRINCIPLE: design concept that does not break further

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

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

1,2,3… 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

sorted pages

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

64K/8 = index 8K pages

info to navigate lower level
value-pointer

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

<12  >=12

1,2,3...
12,15,17
20,....
info to navigate lower level value-pointer

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

1,2,3...

<12 >=12

12,20

12,15,17

20,...

35,50

35,...

50,...

can index 8K pages of the next level

4+4 bytes for each page (value+pointer)
64K/8 = index 8K pages
height $\log_{\text{fanout}} P$

internal nodes

fanout

root

35,50

12,20

35,...

50,...

leaves

1,2,3...

12,15,17

20,...

...
The diagram illustrates a tree-like data structure with the following key elements:

- **Root**: The topmost node labeled with the values 35 and 50.
- **Internal Nodes**: Include nodes with values like 12, 20, and 35, among others, connected to the root and other nodes.
- **Leaves**: The bottom nodes without any further connections, such as 1, 2, 3, etc.

**Height of the Tree**:

The height of the tree is denoted by $\log_{\text{fanout}} P$, indicating the depth of the tree.

**Random Accesses**:

The diagram shows the traversal of the tree, emphasizing random accesses to different nodes.

**Fanout**:

The fanout of the tree refers to the number of children each node can have.

**fanout**:

Nodes are connected to the root, illustrating the fanout of the tree.

**Internal Nodes**:

These are nodes that are not leaves and are connected to the root and other internal nodes.

**Leaves**:

These are the terminal nodes at the bottom of the tree, labeled with values like 1, 2, 3, etc.

**Fanout**

The fanout of the tree is indicated by the number of connections each node has.

**Random Accesses**

The diagram highlights the random access nature of the tree traversal, showing how different nodes can be accessed randomly.

**Height**

The height of the tree is logarithmic with respect to the fanout and probability $P$. This indicates how the depth of the tree grows as the fanout increases.

**Logarithmic Height**

The height $h$ is given by $\log_{\text{fanout}} P$, suggesting a logarithmic scale for the depth of the tree.

**Accesses**

Random accesses are shown to traverse the tree, indicating the efficiency and structure of the data access pattern.
leaves

get 15
get 15-55
how do we search the leaves?

a) sorted and b) unsorted leaves
NoSQL Key-value Stores

machine learning  social media  
smart homes  web browsers  
phones  web-based apps  
security  health devices  
graphs  analytics  

log+index  b-tree  lsm-tree
insert (key-value)
insert (key-value)
insert (key-value)

buffer

Level 1

MEMORY

DISK
MEMORY

DISK

Level 1

Level 2
insert (key-value)

- Buffer
- Level 1
- Level 2
- Level 3
- ... (omitted)
- Level N
insert (key-value)

buffer

Level 1

Level 2

Level 3

... 

Level N

MEMORY

DISK
insert (key-value)

buffer

Level 1

Level 2

Level 3
...

Level N

MEMORY
DISK

pages

SSTables

tiered

leveled

sorted
disks

...
get (key)

buffer

MEMORY
DISK

pages

SSTables
tiered
leveled
sorted

Level 1
Level 2
Level 3
Level N

[1,0,0,1,1,1] hash fun.
[min-max] /page
bloom filters
fence pointers

hash fun. /page

/min-max]
get (key)

buffer

Level 1

Level 2

Level 3

Level N

bloom filters

fence pointers

[1,0,0,1,1,1]
hash fun.

[min-max]
/page

MEMORY

DISK

pages

SSTables

[1,0,0,1,1,1]
hash fun.

[min-max]
/page

Level 1

Level 2

Level 3

Level N

bloom filters

fence pointers
Level 1

Level 2

Level 3

...  

Level N

Memory

Disk

SSTables

Pages

Levelled

Tiered

Sorted

bloom filters

fence pointers

[1,0,0,1,1,1] hash fun.

[min-max] /page

hash fun.

fence pointers

[1,0,0,1,1,1] hash fun.

[min-max] /page

hash fun.
get (key)

buffer

Level 1

Level 2

Level 3

Level N

[1, 0, 0, 1, 1, 1] hash fun.

bloom filters

[fence point] /page

MEMORY

DISK

pages

SSTables

 leveled

tiered

sorted
get (key)

buffer

Level 1

Level 2

Level 3

... 

Level N

DISK

MEMORY

pages

SSTables

leveled

tiered

sorted

bloom filters

fence pointers

[1,0,0,1,1,1] hash fun.

[min-max] /page

hash fun.

fence pointers

[1,0,0,1,1,1] hash fun.

[min-max] /page

hash fun.

fence pointers

[1,0,0,1,1,1] hash fun.

[min-max] /page

hash fun.

fence pointers

[1,0,0,1,1,1] hash fun.

[min-max] /page

hash fun.

fence pointers

[1,0,0,1,1,1] hash fun.

[min-max] /page

hash fun.

fence pointers

[1,0,0,1,1,1] hash fun.

[min-max] /page

hash fun.

fence pointers

[1,0,0,1,1,1] hash fun.

[min-max] /page

hash fun.

fence pointers

[1,0,0,1,1,1] hash fun.

[min-max] /page

hash fun.

fence pointers

[1,0,0,1,1,1] hash fun.

[min-max] /page

hash fun.

fence pointers

[1,0,0,1,1,1] hash fun.

[min-max] /page

hash fun.

fence pointers

[1,0,0,1,1,1] hash fun.

[min-max] /page

hash fun.

fence pointers

[1,0,0,1,1,1] hash fun.

[min-max] /page

hash fun.

fence pointers

[1,0,0,1,1,1] hash fun.

[min-max] /page

hash fun.

fence pointers

[1,0,0,1,1,1] hash fun.
bloom filters
fence pointers
[1,0,0,1,1,1]
hash fun.
[min-max]
/pages
get (key)

buffer

Level 1
Level 2
Level 3
Level N

MEMORY
DISK
pages
SSTables
tiered
leveled
sorted