data systems

>$200B by 2020, growing at 11.7% every year

[The Forbes, 2016]

- complex analytics
- simple queries
- access data
- store, maintain, update
data systems

>$200B by 2020, growing at 11.7% every year

[The Forbes, 2016]

access methods*

*algorithms and data structures for organizing and accessing data

complex analytics

simple queries

access data

store, maintain, update
data systems core: storage engines

main decisions

how to *store* data?  
how to *access* data?

how to *update* data?
let’s simplify: **key-value** storage engines

collection of keys-value pairs

query on the key, return both key and value

*remember*

amazon DynamoDB  Apache HBase  Google BigTable  Asterix DB  Cassandra

Facebook RocksDB  Couchbase  mongoDB

*state-of-the-art* design
how general is a key value store?

can we store relational data?

yes!  

example:  

\{ student_id, \{ name, login, yob, gpa \} \}

what is the caveat?

how to index these attributes?

index:  

\{ name, \{ student_id \} \}

index:  

\{ yob, \{ student_id_1, student_id_2, \ldots \} \}

other problems?
how general is a key value store?

can we store relational data?

yes! \{<primary_key>,<rest_of_the_row>\}

how to index these attributes?

index: \{name, student_id\}

index: \{yob, \{student_id1, student_id2, \ldots\}\}

how to efficiently code if we do not know the structure of the “value”

index: \{yob, \{student_id1, student_id2, \ldots\}\\}
how to use a key-value store?

**Basic Interface**

- `put(k,v)`
- `{v} = get(k)  \quad \{v_1, v_2, \ldots\} = get(k)`
- `{v_1, v_2, \ldots} = get\_range(k_{\text{min}}, k_{\text{max}})`  \quad  \{v_1, v_2, \ldots\} = full\_scan() \quad c = count(k_{\text{min}}, k_{\text{max}})`

**Deletes:** `delete(k)`

**Updates:** `update(k,v)`  \quad  \text{is it different than put?}

**Get Set:** `{v_1, v_2, \ldots} = get\_set(k_1, k_2, \ldots)`
how to build a key-value store?

if we have only *put* operations

if we mostly have *get* operations

what about full scan?

range queries?

and then?
can we separate keys and values?

at what price?

locality?  code?
read queries
  (point or range)

inserts
  (or updates)

sort data

simply append

amortize sorting cost

avoid resorting after every update

how to bridge?
LSM-tree
Key-Value Stores

What are they really?
updates

buffer

memory

storage

sort & flush

runs
buffer

memory        storage

exponentially increasing sizes

$O(\log(N))$ levels
lookup X

buffer

memory

storage

fence

pointers

one I/O per run

X
lookup X

buffer

memory

storage

fence

pointers

one I/O per run

X
lookup X

buffer

Bloom filters

true negative
false positive
true positive

memory

fence pointers

storage

one I/O per run

X
performance & cost trade-offs

lookup X

buffer

Bloom filters
true negative
false positive
ture positive

memory

fence pointers

storage

more merging → fewer runs
read cost vs. update cost
one I/O per run

bigger filters → fewer false positives
memory space vs. read cost
other operations

lookup X

buffer

Bloom filters
true negative
false positive
true positive

memory

fence pointers

storage

one I/O per run

range scans?
deletes?
remember merging?

what strategies?

- sort & flush
- runs
- sort-merge
Merge Policies

Tiering
write-optimized

Leveling
read-optimized
Tiering
write-optimized

$T$ runs per level

Leveling
read-optimized
Tiering
write-optimized

$T$ runs per level

merge & flush

Leveling
read-optimized
Tiering
write-optimized

$T$ runs per level

Leveling
read-optimized

merge
Tiering
write-optimized

$T$ runs per level

Leveling
read-optimized

merge
Tiering
write-optimized

$T$ runs per level

Leveling
read-optimized

$T$ times bigger

flush
Systems Project: LSM-Trees

lookup X

buffer

Bloom filters

true
negative

false
positive

ture
negative

true
positive

memory

fence pointers

storage

one I/O per run

X

tuning knobs

merge policy

size ratio
more on LSM-Tree performance
Tiering
write-optimized

Leveling
read-optimized

lookup cost:

\[ O(T \cdot \log T(N) \cdot e^{-M/N}) \]

- \( T \) runs per level
- levels
- false positive rate

\[ O(\log T(N) \cdot e^{-M/N}) \]

- levels
- false positive rate

31
**Tiering**
write-optimized

$T$ runs per level

**Leveling**
read-optimized

1 run per level

**lookup cost:**

$O\left( T \cdot \log_T(N) \cdot e^{-M/N} \right)$

**update cost:**

$O\left( \log_T(N) \right)$

levels

$O\left( \log_T(N) \cdot e^{-M/N} \right)$

merges per level

levels
Tiering
write-optimized

Leveling
read-optimized

lookup cost: \( O(T \cdot \log_T(N) \cdot e^{-M/N}) \)
update cost: \( O(\log_T(N)) \)

lookup cost: \( O(\log_T(N) \cdot e^{-M/N}) \)
update cost: \( O(T \cdot \log_T(N)) \)

for size ratio \( T \) \( \Rightarrow \)
Tiering write-optimized

Leveling read-optimized

lookup cost: \( O(\log_T(N) \cdot e^{-M/N}) = O(\log_T(N) \cdot e^{-M/N}) \)

update cost: \( O(\log_T(N)) = O(\log_T(N)) \)

for size ratio \( T \)  

1 run per level
Tiering
write-optimized

Leveling
read-optimized

lookup cost: \( O(T \cdot \log_T(N) \cdot e^{-M/N}) \)

update cost: \( O(\log_T(N)) \)

\[ \text{for size ratio } T \uparrow \]

lookup cost: \( O(\log_T(N) \cdot e^{-M/N}) \)

update cost: \( O(T \cdot \log_T(N)) \)
**Tiering** write-optimized

$O(N)$ runs per level

$\log$ sorted array

**Leveling** read-optimized

1 run per level

lookup cost: $O(T \cdot \log_T(N) \cdot e^{-M/N})$

update cost: $O(\log_N(N)) = O(1)$

$O(N \cdot \log_N(N)) = O(N)$ for size ratio $T$
Tiering

Leveling

T : size ratio

read cost

update cost

log

T=2

sorted array
Research Question on LSM-Trees

how to do range scans?
how to delete? how to delete quickly?
how to allocate memory between buffer/Bloom filters/fence pointers?

what if data items come ordered?
what if data items come almost ordered?

study these questions and navigate LSM design space using Facebook’s RocksDB
What “almost ordered” even mean?

Research question on **sortedness**

How to quantify it? Need a metric!

How does the sortedness of the data affect the behavior of LSM-Trees, B-Trees, Zonemaps?

similar question to:
how does the order of the values in an array affect a sorting algorithm
How to tune our system?

if we know the workload ...

LSM-Trees: memory (Buffer/BF/FP) – what about caching?

Back to column-stores: do we need to sort? 

*partition* the data?

add *empty slots* in the column for future inserts?
Workload-based tuning

\[
\text{find } \text{Tuning}, \text{ s.t.} \\
\min \text{cost}(\text{Workload, Data, Tuning}) \\
given \text{Workload and Data}
\]

what if workload information is a bit wrong?

robust optimization (come and find me)
what is an index?

sorted data

1 1 1 2 3 5 10 11 12 13 18 19 20 50 54 58 62 98 101 102

\[ \text{position}(val) = \text{CDF}(val) \cdot \text{array\_size} \]

can you learn the CDF?
what is the best way to do so?
how to update that?
what to do now?

systems project
form groups of 2
(speak to me in OH if you want to work on your own)

research project
form groups of 3-4
pick one of the subjects & read background material
define the behavior you will study and address
sketch approach and success metric
(if LSM-related get familiar with RocksDB)
what to do now?

**systems project**
form groups of 2
(speak to me in OH if you want to work on your own)

**research project**
form groups of 3-4
pick one of the subjects & read background material
define the behavior you will study and address
sketch approach and success metric

**come to OH**
finalize your project in 2-3 weeks (by Feb 22nd)
submit proposal on March 8th
Systems & Research Project

Prof. Manos Athanassoulis

http://manos.athanassoulis.net/classes/CS591