Systems & Research Project

Prof. Manos Athanassoulis

https://midas.bu.edu/classes/CS591A1
Data systems

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

[The Forbes, 2016]
data systems

>$200B by 2020, growing at 11.7% every year
[The Forbes, 2016]

**access methods**

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

*algorithms and data structures for organizing and accessing data*
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*

![Logos of various key-value storage engines](image)

**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 } }  

different attributes:

index:  

{ yob, { student_id_1, student_id_2, ... } }  

other problems?
how general is a key value store?

can we store relational data?  

yes!  

\{<primary_key>,<rest_of_the_row>\}

what is the caveat?

how to index these attributes?

index:  

\{<primary_key>, <rest_of_the_row>\}

\{name, student_id\}

\{yob, student_id_1, student_id_2, ...\}

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

index:  

\{yob, \{student_id_1, student_id_2, ...\}\}
how to use a key-value store?

**basic interface**

```plaintext
put(k,v)

{v} = get(k)    \{v_1, v_2, ...\} = get(k)

{v_1, v_2, ...} = get_range(k_{min}, k_{max}) \quad \{v_1, v_2, ...\} = full_scan()

\(c = count(k_{min}, k_{max})\)
```

**deletes: delete(k)**

**updates: update(k,v)**  is it different than put?

get set: \{v_1, v_2, ...\} = get_set(k_1, k_2, ...)

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)

sort data

inserts
(or updates)

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
buffer

memory

storage

exponentially increasing sizes

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

buffer

one I/O per run

memory

storage

fence pointers

X
lookup X

buffer

memory

fence

pointers

one I/O per run

storage

X
lookup X

buffer

Bloom filters

ture negative
false positive
true positive

memory

fence pointers

storage

one I/O per run

X

true negative
false positive
true positive
performance & cost trade-offs

- lookup X
  - buffer
  - Bloom filters
    - true negative
    - false positive
  - true positive
  - fence pointers
  - memory
  - 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
remember merging?

what strategies?
- sort & flush
- runs
- sort-merge
Merge Policies

Tiering
write-optimized

Leveling
read-optimized
Tiering
write-optimized

Leveling
read-optimized

$T$ runs per level
Tiering
write-optimized

Leveling
read-optimized

$T$ runs per level

merge & flush ↓
Tiering
write-optimized

Leveling
read-optimized

$T$ runs per level

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 positive
- true negative
- false positive

memory

fence pointers

storage

one I/O per run

X

tuning knobs

merge policy

size ratio

Systems Project: LSM-Trees

buffer

Bloom filters

- true positive
- true negative
- false 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

\[ T \text{ runs per level} \]

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

lookup cost:

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

runs per level
levels
false positive rate

levels
false positive rate
Tiering
write-optimized

Leveling
read-optimized

lookup cost:

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

update cost:

\[ O(\log_T(N)) \]

\[ \uparrow \]
levels

\[ \downarrow \]
levels

merges per level

\[ O(\log_T(N) \cdot e^{-M/N}) \]
Tiering
write-optimized

Leveling
read-optimized

$T$ runs per level

1 run per level

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

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

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

$O(T \cdot \log_T(N))$

for size ratio $T$
Tiering
write-optimized

Leveling
read-optimized

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

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

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

Leveling
read-optimized

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

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

for size ratio $T$　∪

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

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

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

Leveling
read-optimized

1 run per level

sorted array

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

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

for size ratio $T \gtrless N$
Tiering

Leveling

log

read cost

update cost

T = size ratio

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 is the CPU overhead of Bloom filters?
- 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

find $Tuning$, s.t.

$\min \text{cost}(Workload, Data, Tuning)$

given $Workload$ and $Data$

what if workload information is a bit wrong?

robust optimization (come and find me)
Asynchronous Bufferpool

**what is the bufferpool?**

- Manages available memory
- Reads/writes from/to disk

**what happens when full?**

- Writes one page back and reads on page

**is this optimal?**
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

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

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
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
pick one of the subjects & read background material
define the behavior you will study and address sketch approach and success metric

come to OH
finalizie your project in 1-2 weeks (by Feb 14th)
submit proposal on February 21st
CS 591: Data Systems Architectures

Systems & Research Project

Prof. Manos Athanassouli

https://midas.bu.edu/classes/CS591A1