Monkey: Optimal Navigable Key-Value Store

Niv Dayan, Manos Athanassoulis, Stratos Idreos
price per GB

storage is cheaper

time

inserts & updates

workload
need for write-optimized database structures
need for write-optimized database structures
need for write-optimized database structures
LSM-tree
Key-Value Stores

What are they really?
updates → buffer → memory → storage

sort & flush runs
updates

buffer

memory storage

sort & flush

runs

sort-merge
buffer

memory  storage

exponentially increasing capacities

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

buffer

memory

storage

fence

pointers

one I/O per run

$X$
lookup key $X$

buffer

memory

storage

fence pointers

one I/O per run

X
lookup key $X$

- buffer
- Bloom filters
  - true
  - negative

memory
- fence pointers

storage

$X$
lookup key $X$ → buffer → Bloom filters

- true negative
- false positive

memory

存储

fence pointers

true
false
positive
negative

Bloom filters

false positive
true negative
lookup key $X$

buffer

Bloom filters

memory

fence pointers

storage

false positive

true positive

true negative

false positive

X
Performance & Cost Tradeoffs

lookup key $X$

buffer

Bloom filters

true negative

false positive

true positive

memory

fence pointers

storage

false positive

true negative

true positive

$X$
Performance & Cost Tradeoffs

lookup key $X$

buffer

Bloom filters

true negative

false positive

true positive

fence pointers

memory

storage

X

bigger filters $\rightarrow$ fewer false positives
Performance & Cost Tradeoffs

lookup key $X$

```
buffer
```

Bloom filters

- true negative
- false positive
- true positive

fence pointers

memory vs. lookups

- bigger filters $\rightarrow$ fewer false positives

false positive

true negative

true positive

storage

memory

X
Performance & Cost Tradeoffs

memory vs. lookups

bigger filters   ➔  fewer false positives

more merging    ➔  fewer runs

lookup key \( X \)

buffer

Bloom filters

true negative

false positive

true positive

fence pointers

storage

memory
Performance & Cost Tradeoffs

- **lookup key** $X$
- **buffer**
- **Bloom filters**:
  - true negative
  - false positive
  - true positive
- **fence pointers**
- **memory vs. lookups**
  - bigger filters $\rightarrow$ fewer false positives
- **storage**
- **lookups vs. updates**
  - more merging $\rightarrow$ fewer runs
- **memory vs. lookups**
main memory

lookup cost

update cost
lookup cost

update cost

main memory

lookup cost

update cost

fixed memory

existing systems

lookup cost

update cost
lookup cost

update cost

main memory

fixed memory

merge

less

existing systems

merge

more

update cost

lookup cost
lookup cost

update cost

main memory

lookup cost

update cost

less memory

more memory

update cost
Problem 1: 
Problem 2:
Problem 1: **suboptimal filters allocation**

Problem 2:
Problem 1: **suboptimal filters allocation**

Problem 2:
Problem 1: suboptimal filters allocation

Problem 2: hard to tune
Problem 1: suboptimal filters allocation

Problem 2: **hard to tune**
Problem 1: suboptimal filters allocation
Problem 2: **hard to tune**
Monkey: Optimal Navigable Key-Value Store
<table>
<thead>
<tr>
<th>Monkey:</th>
<th>Optimal</th>
<th>Navigable</th>
<th>Key-Value Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>observations:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>insights:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>steps:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Monkeys:</td>
<td>Optimal</td>
<td>Navigable</td>
<td>Key-Value Store</td>
</tr>
<tr>
<td>-------------</td>
<td>---------</td>
<td>-----------</td>
<td>-----------------</td>
</tr>
<tr>
<td>observations:</td>
<td>filters</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>fixed false positive rates</td>
<td></td>
<td></td>
</tr>
<tr>
<td>insights:</td>
<td>lookup cost = $\sum p_i$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>suboptimal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>steps:</td>
<td>optimize allocation asymptotically better memory vs. lookups</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Monkey:</strong></td>
<td><strong>Optimal</strong></td>
<td><strong>Navigable</strong></td>
<td><strong>Key-Value Store</strong></td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
<td>---------------</td>
<td>---------------------</td>
</tr>
<tr>
<td><strong>observations:</strong></td>
<td>filters</td>
<td>LSM-tree</td>
<td>merge policy</td>
</tr>
<tr>
<td></td>
<td>fixed false</td>
<td></td>
<td>↓?</td>
</tr>
<tr>
<td>positive rates</td>
<td></td>
<td></td>
<td>performance</td>
</tr>
<tr>
<td><strong>insights:</strong></td>
<td></td>
<td>log</td>
<td></td>
</tr>
<tr>
<td></td>
<td>lookup cost = $\sum p_i$</td>
<td>sorted array</td>
<td></td>
</tr>
<tr>
<td></td>
<td>suboptimal</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>steps:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>optimize allocation</td>
<td>updates vs. lookups</td>
<td></td>
</tr>
<tr>
<td></td>
<td>asymptotically better</td>
<td>navigate</td>
<td></td>
</tr>
<tr>
<td></td>
<td>memory vs. lookups</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Monkey:</td>
<td>Optimal</td>
<td>Navigable</td>
<td>Key-Value Store</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td>-----------</td>
<td>-----------------</td>
</tr>
<tr>
<td>filters</td>
<td>LSM-tree</td>
<td>merge policy</td>
<td>ad-hoc trade-offs</td>
</tr>
<tr>
<td>fixed false positive rates</td>
<td>log</td>
<td></td>
<td>memory</td>
</tr>
<tr>
<td>lookup cost = (\sum p_i)</td>
<td>sorted array</td>
<td></td>
<td>lookups</td>
</tr>
<tr>
<td>suboptimal</td>
<td></td>
<td></td>
<td>updates</td>
</tr>
<tr>
<td>optimize allocation</td>
<td>updates vs. lookups</td>
<td></td>
<td>answer what-if design questions</td>
</tr>
<tr>
<td>asymptotically better memory vs. lookups</td>
<td>navigate</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**insights:**
- lookup cost = \(\sum p_i\)
- suboptimal

**steps:**
- optimize allocation
- asymptotically better memory vs. lookups
- update cost vs. lookups
- navigate

**observations:**
- fixed false positive rates
- filters

**Monkey:**
- answer what-if design questions
- navigate
The diagram illustrates the trade-off between update cost and lookup cost for fixed memory. The Pareto frontier represents the set of points where a cost cannot be improved without degrading the other cost. Key databases mentioned include WiredTiger, Cassandra, HBase, Monkey, and RocksDB, LevelDB. The diagram suggests that WiredTiger, Cassandra, and HBase are Pareto efficient for this scenario, while Monkey and RocksDB, LevelDB are not as efficient.
A graph showing the Pareto frontier for fixed memory, comparing lookup cost and update cost. Notable points include:

- **Max Throughput**
- **WiredTiger**
- **Cassandra, HBase**
- **RocksDB, LevelDB**
- **Monkey**

The graph illustrates the trade-off between lookup cost and update cost, with the Pareto frontier representing the optimal points for fixed memory.
<table>
<thead>
<tr>
<th><strong>Monkey:</strong></th>
<th><strong>Optimal</strong></th>
<th><strong>Navigable</strong></th>
<th><strong>Key-Value Store</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>observations:</td>
<td>filters</td>
<td>LSM-tree</td>
<td>ad-hoc trade-offs</td>
</tr>
<tr>
<td></td>
<td>fixed false positive rates</td>
<td>merge policy</td>
<td>memory</td>
</tr>
<tr>
<td>insights:</td>
<td>lookup cost = $\sum p_i$</td>
<td>log</td>
<td>lookups</td>
</tr>
<tr>
<td></td>
<td>suboptimal</td>
<td>sorted array</td>
<td>updates</td>
</tr>
<tr>
<td>steps:</td>
<td>optimize allocation</td>
<td>updates vs. lookups</td>
<td>answer what-if design questions</td>
</tr>
<tr>
<td></td>
<td>asymptotically better</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>memory vs. lookups</td>
<td>navigate</td>
<td></td>
</tr>
</tbody>
</table>

**LSM-tree**
- Merge policy trade-offs
- Navigate
- Log sorted array
- Memory
- Lookup
- Updates vs. lookups

**Insights**
- Lookup cost = $\sum p_i$
- Suboptimal

**Steps**
- Optimize allocation
- Asymptotically better
- Memory vs. lookups

**Monkeys**
- Answer what-if design questions
**Monkey:**

<table>
<thead>
<tr>
<th><strong>Optimal</strong></th>
<th><strong>Navigable</strong></th>
<th><strong>Key-Value Store</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>filters</td>
<td>LSM-tree</td>
<td>ad-hoc trade-offs</td>
</tr>
<tr>
<td>fixed false positive rates</td>
<td>merge policy</td>
<td>memory</td>
</tr>
<tr>
<td>suboptimal</td>
<td>log</td>
<td>lookups</td>
</tr>
<tr>
<td></td>
<td>sorted array</td>
<td>updates</td>
</tr>
</tbody>
</table>

**insights:**

- lookup cost = $\sum p_i$
- suboptimal

**steps:**

- optimize allocation
- asymptotically better
- memory vs. lookups

**observations:**

- fixed false positive rates
buffer

Bloom filters

fence pointers

data

memory

storage
Bloom filters

memory

buffer

fence pointers

storage

data
buffer < Bloom filters > fence pointers

memory

storage

data
Bloom filters

memory

X bits per entry

storage

data
Bloom filters

memory

X bits per entry

storage

data
Bloom filters

memory

\[ X \text{ bits per entry} \]

\[ \text{false positive rate } p = e^{-\frac{\text{bits } M}{\text{entries } N} \ln(2)^2} \]
Bloom filters

memory

$X$ bits per entry

Bloom filters

$\frac{\text{bits } M}{\text{entries } N} \ln(2)^2$

false positive rate $p = e^{\frac{\text{bits } M}{\text{entries } N} \ln(2)^2}$
Bloom filters

memory

$X$ bits per entry

Bloom filters

$p$

$p$

$p$

worst-case I/O overhead:

$$\text{false positive rate } p = e^{-\frac{\text{bits } M}{\text{entries } N} \ln(2)^2}$$
worst-case I/O overhead:

\[ O( \sum p ) \]

false positive rate \( p \)

\[\begin{align*}
\text{false positive rate } p &= e \\
\text{- } \frac{\text{bits } M}{\text{entries } N} \ln(2)^2
\end{align*}\]
Bloom filters memory

X bits per entry

Bloom filters

$O(\sum p)$

false positive rate $p = e^{\frac{-\text{bits } M}{\text{entries } N} \ln(2)^2}$
Bloom filters

worst-case I/O overhead:

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

false positive rate \( p = e^{-\frac{\text{bits } M}{\text{entries } N} \ln(2)^2} \)
worst-case I/O overhead:

\[ O\left( \sum e^{-M/N} \right) \]
Bloom filters

memory

$X$ bits per entry

worst-case I/O overhead:

$O(\log(N) \cdot e^{-M/N})$
Can we do better?

worst-case I/O overhead:

$$O(\log(N) \cdot e^{-M/N})$$
lookup key X

Bloom filters

fence pointers

data runs

...
lookup key $X$

Bloom filters

false positive

false positive

false positive

false positive

false positive

most memory

fence pointers

I/O

I/O

I/O

I/O

I/O

data runs

...
most memory saves at most 1 I/O
Bloom filters

false positive rates

reallocate some

most memory
same memory, fewer lookup I/Os

false positive rates

reallocate some

most memory
relax

false positive rates

\[ p_0 \]
\[ p_1 \]
\[ p_2 \]
relax

false positive rates

\[ 0 < p_2 < 1 \]
\[ 0 < p_1 < 1 \]
\[ 0 < p_0 < 1 \]
false positive rates

\[ 0 < p_2 < 1 \]
\[ 0 < p_1 < 1 \]
\[ 0 < p_0 < 1 \]

model

lookup cost
\[ = f(p_0, p_1 \ldots) \]

memory footprint
\[ = f(p_0, p_1 \ldots) \]
relax

false positive rates

\[ 0 < p_0 < 1 \]
\[ 0 < p_1 < 1 \]
\[ 0 < p_2 < 1 \]

model

lookup cost

\[ = f(p_0, p_1 \ldots) \]

memory footprint

\[ = f(p_0, p_1 \ldots) \]

optimize

in terms of \( p_0, p_1 \)
Bloom filters

\[ \text{lookup cost} = \sum p_i \]

false positive rates

\[ p_0 \]
\[ p_1 \]
\[ p_2 \]
Bloom filters

false positive rates

\[ p_0, p_1, p_2 \]

lookup cost = \( \sum p_i \)

memory footprint

false positive rate

\[ = e - \frac{\text{bits}}{\text{entries}} \ln(2)^2 \]
Bloom filters

\[
\begin{align*}
\text{false positive rates} & : \\
p_2 & \\
p_1 & \\
p_0 & \\
\text{lookup cost} & = \sum p_i
\end{align*}
\]

memory footprint

\[
\text{bits} = - \ln\left( \frac{\text{false positive rate}}{\ln(2)^2} \right) \text{ entries}
\]
Bloom filters

false positive rates

lookup cost \( = \sum p_i \)

memory footprint

\( \text{bits}(p_2, \frac{N}{T^2}) \)

\( \text{bits}(p_1, \frac{N}{T}) \)

\( \text{bits}(p_0, N) \)
Bloom filters

\[ : \]

false positive rates

\[ p_2 \]

\[ p_1 \]

\[ p_0 \]

lookup cost

\[ = \sum p_i \]

memory footprint

\[ : \]

\[ \text{bits}(p_2, N/T^2) \]

\[ \text{bits}(p_1, N/T) \]

\[ \text{bits}(p_0, N) \]

false positive rates

memory

\[ = - c \cdot N \cdot \sum \frac{\ln(p_i)}{T^i} \]

size ratio

constant

entries

\[ \frac{\ln(p_i)}{T^i} \]
Bloom filters

false positive rates

\[ p_2 \]

\[ p_1 \]

\[ p_0 \]

optimize

lookup cost

\[ \sum p_i \]

memory

\[ -c \cdot N \cdot \sum \frac{\ln(p_i)}{T_i} \]
false positive rates

\[
p_0/T^2
\]

\[
p_0/T
\]

\[
p_0
\]
false positive rates
\[ \frac{p_0}{T^2} \]
\[ \frac{p_0}{T} \]
\[ p_0 \]

exponential decrease

State-of-the-Art Bloom filters

\[ p \]
\[ p \]
\[ p \]
Monkey Bloom filters

State-of-the-Art Bloom filters

false positive rates

lookup cost $= \sum p_i < \sum p$

$p_0 / T^2 < p$

$p_0 / T < p$

$p_0 > p$

$p_0$
Monkey Bloom filters

State-of-the-Art Bloom filters

false positive rates

lookup cost

\[ \sum p_i \leq \sum p \leq O(\ e^{-M/N}) \]

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

\( N \) | number of entries

\( M \) | overall memory for Bloom filters
State-of-the-Art
Bloom filters

... < ...

$\frac{p_0}{T^2} < p$

$\frac{p_0}{T} < p$

$p_0 > p$

lookup cost

$= \sum p_i < \sum p$

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

asymptotic win
lookup cost increases at slower rate as data grows
false positive rates

\[ p_0/T \]

\[ p_0/T^2 \]

\[ p_0 \]

convergent geometric series

\[ p_0/T \]

\[ p_0/T^2 \]
false positive rates

\[ \frac{p_0}{T^2} \]
\[ \frac{p_0}{T} \]
\[ p_0 \]

\[ \text{memory} = c \cdot \text{entries} \cdot \sum - \frac{\ln(p_i)}{T^i} \]
false positive rates

\[ p_0 / T^2 \]

\[ p_0 / T \]

\[ p_0 \]

\[ \text{memory} = c \cdot \text{entries} \cdot -\ln(\text{lookup cost}) \]
false positive rates

\[
p_0/T^2
\]

\[
p_0/T
\]

\[
p_0
\]

\[
\text{memory} = c \cdot \text{entries} \cdot -\ln(\text{lookup cost})
\]
Problem 1: suboptimal filters allocation

Problem 2: hard to tune
Problem 1: **suboptimal filters allocation**

Problem 2: hard to tune
Problem 1: suboptimal filters allocation

Problem 2: **hard to tune**
Problem 1: suboptimal filters allocation

Problem 2: **hard to tune**
<table>
<thead>
<tr>
<th>Monkey:</th>
<th>Optimal</th>
<th>Navigable</th>
<th>Key-Value Store</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>observations:</strong></td>
<td>filters</td>
<td>LSM-tree</td>
<td>merge policy</td>
</tr>
<tr>
<td>fixed false positive rates</td>
<td></td>
<td>merge policy</td>
<td>performance</td>
</tr>
<tr>
<td><strong>insights:</strong></td>
<td>lookup cost = $\sum p_i$</td>
<td>log</td>
<td>lookup cost</td>
</tr>
<tr>
<td>suboptimal</td>
<td>sorted array</td>
<td></td>
<td>log cost</td>
</tr>
<tr>
<td><strong>steps:</strong></td>
<td>optimize allocation</td>
<td>updates vs. lookups</td>
<td>answer what-if design questions</td>
</tr>
<tr>
<td>asymptotically better</td>
<td>memory vs. lookups</td>
<td>navigate</td>
<td></td>
</tr>
</tbody>
</table>

- Monkey
- Optimal
- Navigable
- Key-Value Store

- observations:
  - filters
  - fixed false positive rates

- insights:
  - lookup cost = $\sum p_i$
  - suboptimal

- steps:
  - optimize allocation
  - asymptotically better
  - memory vs. lookups

- merge policy
- performance
- ad-hoc trade-offs
- memory
- lookups
- updates

- fixed false positive rates
- LSMTree
- merge policy
- memory
- lookups
- updates

- existing
- Monkey
- update cost

- update cost
- insights
- steps
- observations
<table>
<thead>
<tr>
<th>Monkey:</th>
<th>Optimal</th>
<th>Navigable</th>
<th>Key-Value Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>observations:</td>
<td>filters</td>
<td>LSM-tree</td>
<td>ad-hoc trade-offs</td>
</tr>
<tr>
<td></td>
<td>fixed false positive rates</td>
<td>merge policy</td>
<td>memory</td>
</tr>
<tr>
<td></td>
<td></td>
<td>↓ ??</td>
<td>lookups</td>
</tr>
<tr>
<td></td>
<td></td>
<td>performance</td>
<td>updates</td>
</tr>
<tr>
<td>insights:</td>
<td>lookup cost = $\sum p_i$</td>
<td>log</td>
<td></td>
</tr>
<tr>
<td></td>
<td>suboptimal</td>
<td>sorted array</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>steps:</td>
<td>optimize allocation</td>
<td>updates vs. lookups</td>
<td>answer what-if</td>
</tr>
<tr>
<td></td>
<td>asymptotically better</td>
<td></td>
<td>design questions</td>
</tr>
<tr>
<td></td>
<td>memory vs. lookups</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Monkey:** optimal and navigable key-value stores are compared against a fixed false positive rate filters. The optimal solution involves an LSM-tree merge policy, while the navigable solution includes a trade-off between memory and update cost. The insights show that the lookup cost can be optimized through allocation and asymptotically better memory vs. lookups. The steps include optimizing allocation and answering what-if design questions.
Identify

merge policy

size ratio
Identify

merge policy

size ratio

Map

lookups

log

sorted array

updates

Navigate

workload

hardware

maximum throughout
Merge Policies

**Tiering**
write-optimized

**Leveling**
read-optimized
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

Leveling
read-optimized

$T$ runs per level
Tiering
write-optimized

Leveling
read-optimized

$T$ runs per level

merge
Tiering
write-optimized

$T$ runs per level

Leveling
read-optimized

$T$ times bigger

flush
Tiering
write-optimized

$T$ runs per level

Leveling
read-optimized

$T$ times bigger
Tiering
write-optimized

Leveling
read-optimized

$T$ runs per level

1 run per level
Tiering
write-optimized

Leveling
read-optimized

lookup cost:

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

runs per level
levels
false positive rate

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

levels
false positive rate
Tiering
write-optimized

Leveling
read-optimized

lookup cost:

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

\( T \) runs per level

false positive rate

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

1 run per level

levels
Tiering
write-optimized

Leveling
read-optimized

$T$ runs per level

lookup cost:

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

runs per level

false positive rate

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

false positive rate
Tiering
write-optimized

lookup cost:

update cost:

$O(T \cdot \text{log}_T(N))$

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

Leveling
read-optimized

$O(T \cdot \text{log}_T(N))$

$O(e^{-M/N})$
Tiering
write-optimized

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

lookup cost:

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

update cost:

\[ O(\log T(N)) \]

Leveling
read-optimized

\[ 1 \text{ run per level} \]

lookup cost:

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

update cost:

\[ O(T \cdot \log T(N)) \]

size ratio \( T \downarrow \)
Tiering
write-optimized

Leveling
read-optimized

lookup cost:

update cost:

O(e^{-M/N}) = O(e^{-M/N})

O(log(N)) = O(log(N))

size ratio $T \downarrow$
Tiering
write-optimized

Leveling
read-optimized

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

update cost: $O(\log_\tau(N))$

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

update cost: $O(T \cdot \log_\tau(N))$

size ratio $T \uparrow$
Tiering
write-optimized

\( O(N) \) runs per level

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

update cost:
\( O(1) \)

Leveling
read-optimized

1 run per level

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

size ratio \( T \uparrow \)
Tiering
write-optimized

O(\(N\)) runs per level

log

lookup cost:

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

update cost:

O(1)

Leveling
read-optimized

1 run per level

sorted array

O(e^{-M/N})

size ratio \(T\) \(\uparrow\)
lookup cost

update cost

log

Tiering

Leveling

sorted array

$T = 2$

$T \mid $ size ratio
lookup cost -> log

Tiering

log <-> LSM-tree <-> sorted array

T | size ratio
lookup cost

T | size ratio

workload

hardware

maximum throughout
Problem 1: suboptimal filters allocation
Problem 2: **hard to tune**
better asymptotic scalability

lookup latency (ms)

number of entries (log scale)

LevelDB

Monkey

levelDB
better asymptotic scalability

workload adaptability

lookup latency (ms)

number of entries (log scale)

lookup latency (ms)

% lookups in workload
CrimsonDB

self-designs

navigates

what-if?

http://daslab.seas.harvard.edu/crimsondb/
<table>
<thead>
<tr>
<th>Monkey:</th>
<th>Optimal</th>
<th>Navigable</th>
<th>Key-Value Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>observations:</td>
<td>filters</td>
<td>LSM-tree</td>
<td>merge policy</td>
</tr>
<tr>
<td></td>
<td>fixed false positive rates</td>
<td></td>
<td>memory</td>
</tr>
<tr>
<td>insights:</td>
<td>lookup cost = $\sum p_i$</td>
<td>log sorted array</td>
<td>lookup cost</td>
</tr>
<tr>
<td></td>
<td>suboptimal</td>
<td></td>
<td>existing</td>
</tr>
<tr>
<td>steps:</td>
<td>optimize allocation</td>
<td>updates vs. lookups</td>
<td>answer what-if design questions</td>
</tr>
<tr>
<td></td>
<td>asymptotically better</td>
<td>navigate</td>
<td></td>
</tr>
<tr>
<td></td>
<td>memory vs. lookups</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Monkey: Optimal Navigable Key-Value Store

more in paper:

0 < memory < ∞

filters  buffer  cache

skewed & range lookups
Monkey: Optimal Navigable Key-Value Store

more in paper:

\[ 0 < \text{memory} < \infty \]

filters \quad \downarrow \quad \downarrow \quad \downarrow \quad \text{skewed & range lookups}

buffer \quad \text{cache}

http://daslab.seas.harvard.edu/monkey/

http://daslab.seas.harvard.edu/monkey/
Monkey: **Optimal** Navigable **Key**-Value Store

more in paper:

\[ 0 < \text{memory} < \infty \]

filters \hspace{1cm} buffer \hspace{1cm} cache

\[ \text{skewed & range lookups} \]

http://daslab.seas.harvard.edu/monkey/

Thanks!