Scaling Log-Structured KV-Stores

featuring

Monkey, Dostoevsky and Wacky

SIGMOD 17/18/19

DASlab
@ Harvard SEAS

Niv Dayan
Log-Structured KV-Stores
Log-Structured KV-Stores
Why Log-Structured KV-Stores?
Why Log-Structured KV-Stores?

fast writes
Why Log-Structured KV-Stores?

memory

storage
Why Log-Structured KV-Stores?
Why Log-Structured KV-Stores?
Why Log-Structured KV-Stores?

byte-addressable

block-addressable

SSD

DASlab © Harvard SEAS
write data
write data
write data
In-Place Writes

write data
In-Place Writes

write data

SSD

B-trees
Log-Structured Writes

buffer writes

SSD
Log-Structured Writes

buffer writes
Log-Structured Writes

buffer writes

SSD
Log-Structured Writes

buffer writes
Log-Structured KV-Stores

fast writes
Log-Structured KV-Stores

- fast writes
- fast reads
- massive data
Background
The Log-Structured Merge-Tree
Background

buffer

LSM-tree
buffer
writes

→

buffer
buffer gets full
buffer

exponentially increasing capacities

level 1

level 2

level 3
where’s Waldo

buffer

Bloom filters
true negative

pointers

SSD

X
where's Waldo

buffer

Bloom filters

true negative

false positive

true positive

pointers

SSD

X
merging

writes

reads
merging

writes ↑

↑

reads ↓
merging

Tiering
write-optimized
cassandra

Leveling
read-optimized
RocksDB
Tiering
write-optimized

Leveling
read-optimized
Tiering
write-optimized

gather

Leveling
read-optimized
Tiering
write-optimized

gather

Leveling
read-optimized
Tiering
write-optimized
gather

Leveling
read-optimized
merge
Tiering
write-optimized

gather

Leveling
read-optimized

merge
$\log_R(N)$

Tiering
write-optimized

Leveling
read-optimized
Tiering write-optimized

Leveling read-optimized

$\log_R(N)$ size ratio
Tiering
write-optimized

Leveling
read-optimized

$\log_R(N)$

$R$ runs per level

1 run per level

size ratio
Tiering
write-optimized

R runs per level

Leveling
read-optimized

1 run per level

.setSize ratio R
Tiering
write-optimized

Leveling
read-optimized

$T$ runs per level

$1$ run per level

$\text{size ratio } R$
Monkey: Optimal Navigable Key-Value Store

Niv Dayan
Manos Athanassoulis
Stratos Idreos
Monkey: Optimal Navigable Key-Value Store

SIGMOD17

data

Bloom filters
Bloom filters

false positive rate

\[ O(e^{-x}) \]

\[ O(e^{-x}) \]

\[ O(e^{-x}) \]

\[ = O(e^{-x} \cdot \log_R(N)) \]
Bloom filters

false positive rate

\[ O(e^{-x}) \]
\[ O(e^{-x}) \]
\[ O(e^{-x}) \]

\[ \{ \} = O(e^{-x} \cdot \log_R(N)) \]

\[ \text{dislike} \]
Bloom filters

false positive rate

$O(e^{-x})$

$O(e^{-x})$

$O(e^{-x})$

most memory
Bloom filters 

most memory

saves at most 1 I/O!

false positive rate

$O(e^{-x})$

$O(e^{-x})$

$O(e^{-x})$
reallocate
same memory - fewer false positives

reallocate
relax

false positive rates

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

false positive rates

0 \leq p_0 \leq 1
0 \leq p_1 \leq 1
0 \leq p_2 \leq 1

model

read cost = f(p_0, p_1, ...)

memory footprint = f(p_0, p_1, ...)
false positive rates

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

model

read cost

\[ \sum_{i=1}^{L} p_i \]

memory footprint

\[ -\sum_{i=1}^{L} \frac{N}{T^{L-i}} \cdot \frac{\ln(p_i)}{\ln(2)^2} \]
relax

false positive rates

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

model

read cost

\[ \sum_{i=1}^{L} p_i \]

memory footprint

\[ -\sum_{i}^{L} \frac{N}{T^{L-i}} \cdot \frac{\ln(p_i)}{\ln(2)^2} \]

optimize

in terms of \( p_0, p_1 \ldots \)
false positive rate

\[ p_0 \approx O\left(\frac{e^{-x}}{R^2}\right) \]
\[ p_1 \approx O\left(\frac{e^{-x}}{R^1}\right) \]
\[ p_2 \approx O\left(\frac{e^{-x}}{R^0}\right) \]
false positive rate

\[ O\left(\frac{e^{-x}}{R^2}\right) \]
\[ O\left(\frac{e^{-x}}{R^1}\right) \]
\[ O\left(\frac{e^{-x}}{R^0}\right) \]

\[ \begin{aligned} \text{geometric progression} \end{aligned} \]
\[ = \quad O\left(e^{-x}\right) \]
\( O(e^{-x} \cdot \log_R(N)) > O(e^{-x}) \)
$O(e^{-x} \cdot \log_R(N))$
read latency (ms)

number of entries (log scale)

RocksDB

Monkey

$O(e^{-x} \cdot \log_R(N))$

$O(e^{-x})$
I/O overheads with leveling

- point
- long range
- short range
- writes
false positive rates

exponentially decreasing

\[ O\left(\frac{e^{-x}}{R^2}\right) \]

\[ O\left(\frac{e^{-x}}{R}\right) \]

\[ O\left(e^{-x}\right) \]
false positive rates

$O(e^{-x}/R^2)$

$O(e^{-x}/R)$

point $\rightarrow O(e^{-x})$ largest level
point
largest level
$O(e^{-x})$

long range

short range

writes
long range

target range

$O(s/R^2)$

$O(s/R)$

$O(s)$

target key range
short range

target range

1
1
1

all levels
O(\log{\text{M}})
point
largest level
$O(e^{-x})$

long range
largest level
$O(s)$

short range
all levels
$O(\log_R(N))$

writes
writes

exponentially more work
writes

exponentially more work

exponentially less frequent
write-amplification

$O(R)$

$O(R)$

$O(R)$

writes
largest level

point

long range

all levels

O(\(e^{-x}\))

O(s)

O(R)

O(R)

O(R)
for point lookups and long range lookups merging at smaller levels is superfluous
worse as data grows!
poor performance
poor performance
lower device lifetime (on SSD)
Dostoevsky: Space-Time Optimized Evolvable Scalable Key-Value Store
very write-optimized

Dostoevsky: Space-Time Optimized Evolvable Scalable Key-Value Store
Tiering
write-optimized

Leveling
read-optimized
Tiering
data-optimized

Lazy Leveling
mixed-optimized

Leveling
read-optimized
Lazy Leveling

Tiering

Leveling
Lazy Leveling

merge when level fills

merge to have at most 1 run
point  →  long range  →  short range  →  writes
false positive rates

$\mathcal{O}(e^{-x}/R^3)$

$\mathcal{O}(e^{-x}/R^2)$

$\mathcal{O}(e^{-x})$
false positive rates

\[ O\left(\frac{e^{-x}}{R^3}\right) \]

exponentially decreasing

\[ O\left(\frac{e^{-x}}{R^2}\right) \]

\[ O(e^{-x}) \]
false positive rates

$O\left(\frac{e^{-x}}{R^3}\right)$

$O\left(\frac{e^{-x}}{R^2}\right)$

point $\rightarrow O( e^{-x})$

largest level
\[ O(e^{-x}) \]

with uniform FPRs

\[ O(\log_R(N) \cdot R \cdot e^{-x}) \]

\[ O(e^{-x}) \]

\[ O(e^{-x}) \]

\[ O(e^{-x}) \]
point: \( O(e^{-x}) \)

long range: \( O(s) \)

short range

writes
short range

O(R)

O(1 + R \cdot (\log_R(N) - 1))

target key range
point \( O(e^{-x}) \)

long range \( O(s) \)

short range \( O(1+R \cdot (\log R(N) - 1)) \)

writes
write-amplification

O(1)
O(1)
O(R)
write-amplification

\[ O(1) \]
\[ O(1) \]
\[ O(R) \]

\[ \text{writes} \]

\[ O(R + \log_R(N)) \]
- point: $O(e^{-x})$
- long range: $O(s)$
- short range: $O(1 + R \cdot (\log_R(N) - 1))$
- writes: $O(R + \log_R(N))$
<table>
<thead>
<tr>
<th></th>
<th>point</th>
<th>long range</th>
<th>short range</th>
<th>writes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tiering</strong></td>
<td>$O(R \cdot e^{-x})$</td>
<td>$O(R \cdot s)$</td>
<td>$O(R \cdot \log_R(N))$</td>
<td>$O(\log_R(N))$</td>
</tr>
<tr>
<td>V</td>
<td>V</td>
<td>V</td>
<td>V</td>
<td>V</td>
</tr>
<tr>
<td><strong>Lazy Leveling</strong></td>
<td>$O(e^{-x})$</td>
<td>$O(s)$</td>
<td>$O(1 + R \cdot (\log_R(N) - 1))$</td>
<td>$O(R + \log_R(N))$</td>
</tr>
<tr>
<td><strong>Leveling</strong></td>
<td>$O(e^{-x})$</td>
<td>$O(s)$</td>
<td>$O(\log_R(N))$</td>
<td>$O(R \cdot \log_R(N))$</td>
</tr>
</tbody>
</table>
Tiering  Lazy Leveling  Leveling
Tiering writes 🌟 Lazy Leveling Leveling
Tiering writes

Lazy Leveling

Leveling short range
Tiering writes ✌️

Lazy Leveling writes & point ✌️

Leveling short range ✌️
Lazy Leveling

Tiering

Fluid

Leveling
Fluid LSM-Tree

\[ \{ K \text{ runs} \} \quad \{ Z \text{ runs} \} \]
Fluid LSM-Tree

Lazy Leveling

\{ \}

R runs

\{ \}

1 runs
point  long range  short range  writes

Lazy Leveling

\{\}
\{\}

$R$ runs
$1$ runs
optimize

short range

writes

point

long range

Leveling

1 runs

1 runs
point  long range  short range  optimize  writes

Lazy Leveling

\{ R \text{ runs} \}
\{ 1 \text{ runs} \}
optimize

point

long range  short range  writes

Tiering

\{ R \text{ runs} \}
\{ R \text{ runs} \}
Lazy Leveling

- Optimize point
- Long range
- Short range
- Writes
optimize point

long range
short range
writes

Lazy Leveling

R size ratio

\{\}

R runs

1 runs
optimize point
long range
short range
writes

Lazy Leveling

$R$ size ratio

$R$ runs
$1$ runs
Fluid LSM-Tree

- $R$ size ratio
- $K$ runs at smaller levels
- $Z$ runs at largest level
Lazy Leveling

Fluid LSM-Tree

Tiering

Leveling
New Work
Lazy Leveling

\[ O( R + \log_R(N) ) \]
Lazy Leveling

writes

$O(1)$

$O(1)$

$O(R)$
<table>
<thead>
<tr>
<th>memory</th>
<th>point reads</th>
<th>writes</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(X/R^4)$</td>
<td>$O(e^{-X}/R^5)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>$O(X/R^3)$</td>
<td>$O(e^{-X}/R^4)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>$O(X/R^2)$</td>
<td>$O(e^{-X}/R^3)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>$O(X/R^1)$</td>
<td>$O(e^{-X}/R^2)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>$O(X)$</td>
<td>$O(e^{-X})$</td>
<td>$O(R)$</td>
</tr>
</tbody>
</table>

Growing as $O(\log_R(N))$
<table>
<thead>
<tr>
<th>memory</th>
<th>point reads</th>
<th>writes</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(X/R^4)$</td>
<td>$O(e^{-X}/R^5)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>$O(X/R^3)$</td>
<td>$O(e^{-X}/R^4)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>$O(X/R^2)$</td>
<td>$O(e^{-X}/R^3)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>$O(X/R)$</td>
<td>$O(e^{-X}/R^2)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>

exponentially diminishing returns

Growing as $O(\log_R(N))$
Wacky SIGMOD19
Wacky: Amorphous Calculable Key-Value Store
Wacky: Amorphous Calculable Key-Value Store

introducing the Log-Structured Merge-Bush
Wacky: Amorphous Calculable Key-Value Store

LSM-bush
runs

\[ O(R^{2^2}) \]
\[ O(R^{2^1}) \]
\[ O(R^{2^0}) \]
\[ O(1) \]
\[ O(R^{2^2}) \quad O(R^{2^1}) \quad O(R^{2^0}) \quad O(1) \]

\[
\text{writes}\quad O(\log_2 \log_R(N))
\]
memory

point reads

$O(X)$

$O(e^{-x})$

writes

$O(\log_2 \log_R(N))$
Wacky generalizes

Tiering  Lazy Leveling  LSM-bush  Leveling
Conclusion
Conclusion

Bloom filters  

LSM-tree
Conclusion

- Bloom filters
- LSM-tree

optimizes memory allocation
Conclusion

- Bloom filters
- LSM-tree
- lazy smaller levels

optimizes memory allocation
Conclusion

Bloom filters

optimizes memory allocation

LSM-tree

lazy smaller levels

increasing ratios
Conclusion

tiering (1997)
leveling (1996)
Conclusion

- tiering (1997)
- leveling (1996)

- little memory
Conclusion

tiering (1997)

leveling (1996)

ample memory
Conclusion

- tiering (1997)
- leveling (1996)
- ample memory
Conclusion

tiering (1997)

leveling (1996)

ample memory
Conclusion

- Tiering (1997)
- Leveling (1996)

Ample memory
Conclusion
Conclusion
Conclusion
Thanks!