New Topic: External Memory Hash Tables

- now hash tables
- first packed and compressed hash table
- presented in January ’23 at ALENEX
Motivation

Setting
- static hash table for objects of variable size
- storing objects in external memory
- ideally retrieve objects in single I/O
- very small internal memory data structure

Objects of Variable Size

External Memory
- only blocks of size $B$ bits can be transferred
- one I/O per block transfer
Space-Efficient Object Stores from Literature

- objects of size 256 bytes
- blocks of size 4096 bytes
- internal space $l_b$ (bits/block)
- (*) consecutive I/O
## Space-Efficient Object Stores from Literature

- objects of size 256 bytes
- blocks of size 4096 bytes
- internal space $I_b$ (bits/block)
- (*) consecutive I/O

<table>
<thead>
<tr>
<th>Method</th>
<th>$I_b$</th>
<th>Load Factor</th>
<th>I/Os</th>
</tr>
</thead>
<tbody>
<tr>
<td>fixed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Larson et al. [LR85]</td>
<td>96</td>
<td>&lt;96%</td>
<td>1</td>
</tr>
<tr>
<td>SILT SortedStore [Lim+11]</td>
<td>51</td>
<td>100%</td>
<td>1</td>
</tr>
<tr>
<td>Linear Separator [Lar88]</td>
<td>8</td>
<td>85%</td>
<td>1</td>
</tr>
<tr>
<td>Separator [GL88; LK84]</td>
<td>6</td>
<td>98%</td>
<td>1</td>
</tr>
<tr>
<td>Robin Hood [Cel88]</td>
<td>3</td>
<td>99%</td>
<td>1.3</td>
</tr>
<tr>
<td>Ramakrishna et al. [RT89]</td>
<td>4</td>
<td>80%</td>
<td>1</td>
</tr>
<tr>
<td>Jensen, Pagh [JP08]</td>
<td>0</td>
<td>80%</td>
<td>1.25</td>
</tr>
<tr>
<td>Cuckoo [Aza+94; Pag03]</td>
<td>0</td>
<td>&lt;100%</td>
<td>2</td>
</tr>
<tr>
<td><strong>PaCHash</strong>, $a = 1$</td>
<td>2</td>
<td>100%</td>
<td>2*</td>
</tr>
<tr>
<td><strong>PaCHash</strong>, $a = 8$</td>
<td>5</td>
<td>100%</td>
<td>1.13*</td>
</tr>
<tr>
<td>variable</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SILT LogStore [Lim+11]</td>
<td>832</td>
<td>100%</td>
<td>1</td>
</tr>
<tr>
<td>SkimpyStash [DSL11]</td>
<td>32</td>
<td>≤98%</td>
<td>8</td>
</tr>
<tr>
<td><strong>PaCHash</strong>, $a = 1$</td>
<td>2</td>
<td>99.95%</td>
<td>2.06*</td>
</tr>
<tr>
<td><strong>PaCHash</strong>, $a = 8$</td>
<td>5</td>
<td>99.95%</td>
<td>1.19*</td>
</tr>
</tbody>
</table>
PaCHash Overview

A hash function $h : K \to 1 \ldots a m$.

A sorted objects in EM in bins first bin (partially) in block.

Store offset in EM as a tuning parameter.

EM blocks have $\bar{B}$ bits remaining per EM block.

Objects of variable size.
PaCHash Overview

- objects of variable size
- hash function $h: K \rightarrow 1..am$
- EM

EM blocks with tuning parameter $p = \langle p_1, \ldots, p_m \rangle$

Bar space with remaining $\bar{B}$ bits per EM block.
PaCHash Overview

- Objects of variable size
- Hash function $h : K \rightarrow 1 \ldots am$
- Tuning parameter $p = \langle p_1, \ldots, p_m \rangle$
- $\bar{B}$ bits remaining per EM block
- No fragmentation
- Sorted objects in EM in bins
- First bin (partially) in block
- Store offset in EM

EM
PaCHash Overview

objects of variable size

hash function $h: K \rightarrow 1..am$

EM

# EM blocks

tuning parameter

$p = \langle p_1, \ldots, p_m \rangle$

$\bar{B}$ bits remaining per EM block
PaCHash Overview

- Objects of variable size
- Hash function $h : K \rightarrow 1..am$
- Tuning parameter
- Sorted objects in EM in bins
- No fragmentation
- Store offset in EM
- $\bar{B}$ bits remaining per EM block
- $\mathbf{p} = \langle p_1, \ldots, p_m \rangle$
PaCHash Overview

Objects of variable size

Hash function \( h: K \rightarrow 1..am \)

Tuning parameter

No fragmentation

Sorted objects in EM in bins

Number of EM blocks

Bar offset in EM
PaCHash Overview

- Objects of variable size
- Hash function $h : K \rightarrow 1..am$
- Sorted objects in EM in bins
- No fragmentation
- First bin (partially) in block

Tuning parameter:
- $p = \langle p_1, \ldots, p_m \rangle$
- $\bar{B}$ bits remaining per EM block

\# EM blocks
PaCHash Overview

- Objects of variable size
- Hash function $h: K \rightarrow 1..am$
- Tuning parameter
- # EM blocks
- Sorted objects in EM in bins
- No fragmentation
- First bin (partially) in block
- $p = \langle p_1, \ldots, p_m \rangle$
PaCHash Overview

- Objects of variable size
- Hash function $h: K \rightarrow 1..am$
- Sorted objects in EM in bins
- First bin (partially) in block
- No fragmentation
- Tuning parameter
- $p = \langle p_1, \ldots, p_m \rangle$
- $\bar{B}$ bits remaining per EM block
- Number of EM blocks

PaCHash Institute of Theoretical Informatics, Algorithm Engineering
PaCHash Overview

objects of variable size

hash function $h: K \rightarrow 1..am$

sorted objects in EM in bins

store offset in EM

tuning parameter

# EM blocks

no fragmentation

first bin (partially) in block

$p = \langle p_1, \ldots, p_m \rangle$
PaCHash Overview

Objects of variable size

Hash function \( h: K \rightarrow 1..am \)

Sorted objects in EM in bins

First bin (partially) in block

Store offset in EM

\( p = \langle p_1, \ldots, p_m \rangle \)

\( \bar{B} \text{ bits remaining per EM block} \)

Tuning parameter

No fragmentation

\# EM blocks
Finding Blocks

Query Algorithm

- \( b_x = h(x) \)
- find first \( i \) with \( p_i \leq b_x \)
- if \( p_i = b_x \) let \( i = i - 1 \)
- find first \( j \) with \( p_j > b_x \)
- return \( i..(j - 1) \)
**Query Algorithm**

- \( b_x = h(x) \)
- find first \( i \) with \( p_i \leq b_x \)
- if \( p_i = b_x \) let \( i = i - 1 \)
- find first \( j \) with \( p_j > b_x \)
- return \( i..(j-1) \)

**Elias-Fano Coding**

- given \( k \) monotonic increasing integers in \( 1..u \)
  - store \( \log k \) MSBs encoded in bit vector
  - store \( \log(u/k) \) LSBs plain
  - \( k(2 + \log(u/k)) + 1 + o(k) \) bits in total
- predecessor in \( O(k) \) time
**Query Algorithm**

1. $b_x = h(x)$
2. find first $i$ with $p_i \leq b_x$
3. if $p_i = b_x$ let $i = i - 1$
4. find first $j$ with $p_j > b_x$
5. return $i..(j - 1)$

**Elias-Fano Coding**

- given $k$ monotonic increasing integers in $1..u$
- store log $k$ MSBs encoded in bit vector
- store log($u/k$) LSBs plain
- $k(2 + \log(u/k)) + 1 + o(k)$ bits in total
- predecessor in $O(k)$ time

**Lemma: Space with Elias-Fano Coding**

When using Elias-Fano coding [Eli74; Fan71] to store $p$, the index needs $2 + \log a + o(1)$ bits of internal memory per block.
**Query Algorithm**

- $b_x = h(x)$
- find first $i$ with $p_i \leq b_x$
- if $p_i = b_x$ let $i = i - 1$
- find first $j$ with $p_j > b_x$
- return $i..(j-1)$

**Elias-Fano Coding**

- given $k$ monotonic increasing integers in $1..u$
- store $\log k$ MSBs encoded in bit vector
- store $\log(u/k)$ LSBs plain
- $k(2 + \log(u/k)) + 1 + o(k)$ bits in total
- predecessor in $O(k)$ time

**Lemma: Space with Elias-Fano Coding**

When using Elias-Fano coding [Eli74; Fan71] to store $p$, the index needs $2 + \log a + o(1)$ bits of internal memory per block.
Lemma: Expected Predecessor Time

When using Elias-Fano coding to store \( p \), the range of blocks containing the bin of an object \( x \) can be found in expected constant time.
Lemma: Expected Predecessor Time

When using Elias-Fano coding to store $p$, the range of blocks containing the bin of an object $x$ can be found in expected constant time.

Proof (Sketch)

- consider $\lceil \log m \rceil$ MSB
- let bin $b_x$ have MSBs equal to $u$
- expected size $\mathbb{E}(Y_u)$ of all bins with MSB $u$ that are $< b_x$ is

$$\sum_{y \in S} |y| \cdot \mathbb{P}(h(y) \text{ w/ MSB } = u; h(y) < h(x))$$

$$\leq \sum_{y \in S} |y| \cdot \mathbb{P}(h(y) \text{ w/ MSB } = u)$$

$$= \frac{1}{m} \sum_{y \in S} |y| = \frac{m\bar{B}}{m} = \bar{B}$$

- number of entries to scan is $\mathbb{E}(Y_u) / \bar{B} = 1$
Loading Blocks from External Memory

Lemma: Additional Blocks Loaded

Retrieving an object $x$ of size $|x|$ from a PaCHash data structure loads $\leq 1 + |x|/\bar{B} + 1/a$ consecutive blocks from the external memory in expectation.
Lemma: Additional Blocks Loaded

Retrieving an object $x$ of size $|x|$ from a PaCHash data structure loads $\leq 1 + |x|/\bar{B} + 1/a$ consecutive blocks from the external memory in expectation.

Proof (Sketch)

- expected size of bin $b_x = h(x)$

$$E(|b_x|) = |x| + \sum_{y \in S, y \neq x} |y| \cdot P(y \in b_x)$$

$$\leq |x| + \sum_{y \in S} |y| \cdot P(y \in b_x)$$

$$= |x| + \sum_{y \in S} |y| \cdot \frac{1}{am} = |x| + \frac{\bar{B}}{a}$$
Lemma: Additional Blocks Loaded

Retrieving an object $x$ of size $|x|$ from a PaCHash data structure loads $\leq 1 + |x|/\bar{B} + 1/a$ consecutive blocks from the external memory in expectation.

Proof (Sketch)

- expected size of bin $b_x = h(x)$

\[
E(|b_x|) = |x| + \sum_{y \in S, y \neq x} |y| P(y \in b_x) \\
\leq |x| + \sum_{y \in S} |y| P(y \in b_x) \\
= |x| + \sum_{y \in S} |y| \cdot \frac{1}{am} = |x| + \frac{\bar{B}}{a}
\]

Proof (Sketch, cnt.)

- expected number of blocks overlapped by $b_x$

\[
E(X) = 1 + \frac{E(|b_x|) - 1}{\bar{B}} \\
= 1 + \frac{|x|}{B} + \frac{1}{a} - 1/\bar{B}
\]

- $P($bin and block border align$) = 1/\bar{B}$
## Experimental Evaluation

### Hardware and Software

- Intel i7 11700 (base clock speed: 2.5 GHz)
- 1 TB Samsung 980 Pro NVMe SSD
- Ubuntu 21.10 (Kernel 5.13.0)
- `io_uring` for I/O operations
- GCC 11.2.0 (`-O3 -march=native`)
- \( B = 4096 \) bytes

---

<table>
<thead>
<tr>
<th>Competitors</th>
</tr>
</thead>
<tbody>
<tr>
<td>LevelDB [Goo21]</td>
</tr>
<tr>
<td>RocksDB [Fac21]</td>
</tr>
<tr>
<td>SILT [Lim+11]</td>
</tr>
<tr>
<td>std::unordered_map</td>
</tr>
<tr>
<td>RecSplit [EGV20]</td>
</tr>
<tr>
<td>CHD [BBD09; CR+12]</td>
</tr>
<tr>
<td>PTHash [PT21]</td>
</tr>
</tbody>
</table>
## Experimental Evaluation

### Hardware and Software
- Intel i7 11700 (base clock speed: 2.5 GHz)
- 1 TB Samsung 980 Pro NVMe SSD
- Ubuntu 21.10 (Kernel 5.13.0)
- io_uring for I/O operations
- GCC 11.2.0 (-O3 -march=native)
- B = 4096 bytes

### Objects
- here only **fixed size**
- more in the paper (very similar results)
Experimental Evaluation

Hardware and Software
- Intel i7 11700 (base clock speed: 2.5 GHz)
- 1 TB Samsung 980 Pro NVMe SSD
- Ubuntu 21.10 (Kernel 5.13.0)
- io_uring for I/O operations
- GCC 11.2.0 (-O3 -march=native)
- \( b = 4096 \) bytes

Objects
- here only **fixed size**
- more in the paper (very similar results)

Competitors
- LevelDB [Goo21]
- RocksDB [Fac21]
- SILT [Lim+11]
- `std::unordered_map`
- RecSplit [EGV20]
- CHD [BBD09; CR+12]
- PTHash [PT21]
Maximum Load Factor of Competitors

Cuckoo Hashing

Separator Hashing

Identical size  Normal distribution  Uniform distribution

Average object size [B]
Lemma: Space with Succincter

When using Succincter [Pat08] to store $p$, the index needs $1.44 + \log(a + 1) + o(1)$ bits of internal memory per block.
Lemma: Space with Succincter

When using Succincter [Pat08] to store \( p \), the index needs \( 1.44 + \log(a + 1) + o(1) \) bits of internal memory per block.

Structure of Bit Vector
- runs of 0s and 10s
- sometimes additional 1s
Lemma: Space with Succincter

When using Succincter [Pat08] to store \( p \), the index needs
\[ 1.44 + \log(a + 1) + o(1) \]
bits of internal memory per block.

Structure of Bit Vector

- runs of 0s and 10s
- sometimes additional 1s

Entropy Encoding

- encode positions directly
- compress bit vector using Huffman codes
- encode blocks of size 8, 16, 32, or 64
Lemma: Space with Succincter

When using Succincter [Pat08] to store $p$, the index needs $1.44 + \log(a + 1) + o(1)$ bits of internal memory per block.

Structure of Bit Vector
- runs of 0s and 10s
- sometimes additional 1s

Entropy Encoding
- encode positions directly
- compress bit vector using Huffman codes
- encode blocks of size 8, 16, 32, or 64

Query Throughput direct I/O [kQueries/s]

<table>
<thead>
<tr>
<th>Parameter a</th>
<th>Huffman, Twitter</th>
<th>Huffman, UniRef</th>
<th>Huffman, Wikipedia</th>
<th>Elias-Fano, Twitter</th>
<th>Elias-Fano, UniRef</th>
<th>Elias-Fano, Wikipedia</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>400</td>
<td>200</td>
<td>400</td>
<td>200</td>
<td>400</td>
</tr>
<tr>
<td>8</td>
<td>600</td>
<td>600</td>
<td>600</td>
<td>600</td>
<td>600</td>
<td>600</td>
</tr>
<tr>
<td>64</td>
<td>600</td>
<td>600</td>
<td>600</td>
<td>600</td>
<td>600</td>
<td>600</td>
</tr>
</tbody>
</table>

Space internal [B/block]
Conclusion and Outlook

This Lecture
- PaCHash

Advanced Data Structures

- PaCHash
- Successor
- RMQ
  - Kd- & Range Tree
  - static BV
  - static succ. trees
  - range min-max tree
  - succ. graphs
Conclusion and Outlook

This Lecture
- PaCHash

Next Lecture
- more on hashing

Advanced Data Structures

PaCHash

Successor

RMQ

Kd- & Range Tree

static BV

range min-max tree

static succ. trees

succ. graphs
Bibliography I


Bibliography II


Bibliography III


Bibliography IV


