

# PaCHash: Packed and Compressed Hash Tables

**First ACDA Workshop in Aussois**

Florian Kurpicz (joint work with Hans-Peter Lehmann and Peter Sanders)

The slides are licensed under a Creative Commons Attribution-ShareAlike 4.0 International License © ⓘ ⓘ: [www.creativecommons.org/licenses/by-sa/4.0](http://www.creativecommons.org/licenses/by-sa/4.0) | commit b18d7d0 compiled at 2022-09-06-10:38

# Motivation

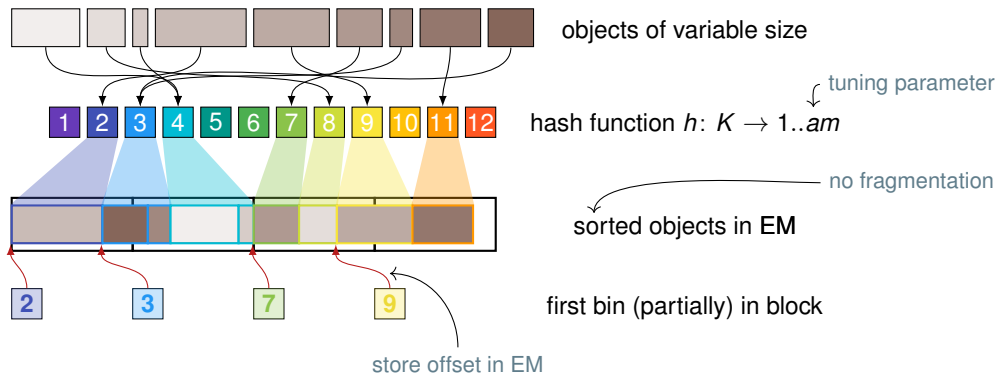
- hash table for objects of **variable** size
- storing objects in external memory
- very small internal memory data structure

- retrieve objects in 1 I/O
- consecutive I/O (\*)

- objects of size 256 bytes
- in blocks of size 4096 bytes
- internal space for one block

	Method	$l_b$	$\alpha$	I/Os
fixed	Larson et al. [LR85]	96	$<96$	1
	SILT SortedStore [Lim+11]	51	100	1
	Linear Separator [Lar88]	8	85	1
	Separator [GL88; LK84]	6	98	1
	Robin Hood [Cel88]	3	99	1.3
	Ramakrishna et al. [RT89]	4	80	1
	Jensen, Pagh [JP08]	0	80	1.25
	Cuckoo [Aza+94; Pag03]	0	$<100$	2
	<b>PaCHash</b> , $a = 1$	2	100	2*
<b>PaCHash</b> , $a = 8$	5	100	1.13*	
variable	SILT LogStore [Lim+11]	832	100	1
	SkimpyStash [DSL11]	32	$\leq 98$	8
	<b>PaCHash</b> , $a = 1$	2	99.95	2.06*
	<b>PaCHash</b> , $a = 8$	5	99.95	1.19*

# PaCHash Overview



# Setting

- external memory blocks of size  $B$



- bins containing (self-delimiting) objects



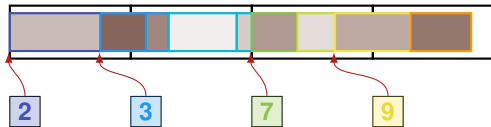
- offset requiring  $d = 0.. \lceil \log B \rceil$  bits
- $\bar{B} = B - d$  remaining per EM block
- $n$  objects of total size  $N$
- $m = N/\bar{B}$  blocks in EM
- $p = \langle p_1, \dots, p_m \rangle$  internal memory

- alternatives for different object sizes

$d$	Case Description
0	Identical object sizes, zero terminated strings and analogous cases
$\lceil \log(w + 1) \rceil$	Objects that use variable bit-length encoding with $\leq w \leq B$ bits
$\lceil \log(W/w + 1) \rceil$	Objects of size divisible by $w$ with $W = \min(B, \max \text{ object size})$
$\lceil \log(B) \rceil$	Explicit storage of a starting position of a bin

# Finding Blocks

## Query Algorithm



- $b = h(x)$
- find predecessor  $p_i$  of  $b$
- if  $p_i = b$  let  $i = i - 1$
- find first  $j$  with  $p_j > b$

## Uncompressed Bit Vector

- finding predecessor in bit vector in  $O(1)$  time
- requires  $m(a + 1) + o(m(a + 1))$  bits of space (uncompressed)

## Elias-Fano Coding

- using Elias-Fano coding saves space
  - given  $k$  monotonic increasing integers in  $1..u$
  - store  $\log k$  MSBs as bit vector
  - store  $\log(u/k)$  LSBs plain
  - $k(2 + \log(u/k)) + 1 + o(k)$  bits in total
- predecessor in  $O(\log(u/k))$  time
- finding range of ones in bit vector

# Internal Memory Data Structure

## 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.

### Proof

- $m$  blocks containing bins  $1..am$
- requires  $m(2 + \log a) + 1 + o(m)$  bits

## 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

- consider  $\log m$  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

$$\begin{aligned}
 & \sum_{y \in S} |y| \cdot \mathbb{P}(h(y) \text{ w/ MSB} = u; h(y) < h(x)) \\
 & \leq \sum_{y \in S} |y| \cdot \mathbb{P}(h(y) \text{ w/ MSB} = u) \\
 & = \frac{1}{m} \sum_{y \in S} |y| = \frac{m\bar{B}}{m} = \bar{B}
 \end{aligned}$$

# 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 (setting  $|x| = 0$  if  $x$  is not in the table).

## Proof (cnt.)

- expected number of blocks overlapped by bin

$$\begin{aligned} \mathbb{E}(X) &= 1 + (\mathbb{E}(|b_x|) - 1)/\bar{B} \\ &= 1 + |x|/\bar{B} + 1/a - 1/\bar{B} \end{aligned}$$

- $\mathbb{P}(\text{bin and block border align}) = 1/\bar{B}$

## Proof

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

$$\begin{aligned} \mathbb{E}(|b_x|) &= |x| + \sum_{y \in \mathcal{S}, y \neq x} |y| \mathbb{P}(y \in b_x) \\ &\leq |x| + \sum_{y \in \mathcal{S}} |y| \mathbb{P}(y \in b_x) \\ &= |x| + \sum_{y \in \mathcal{S}} |y| \cdot \frac{1}{am} = |x| + \frac{\bar{B}}{a} \end{aligned}$$

# Experimental Evaluation

## Hardware and Software

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

## Objects

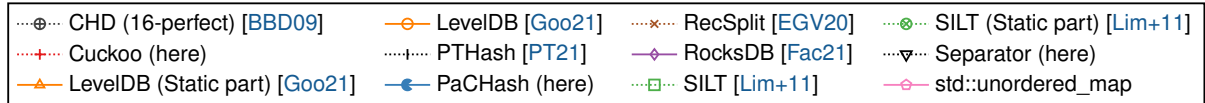
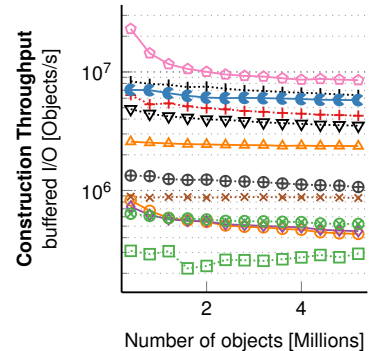
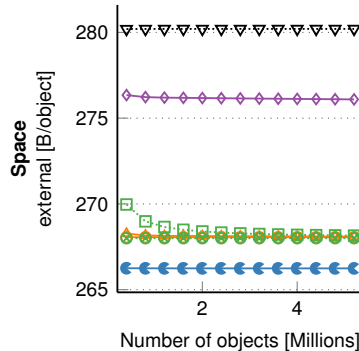
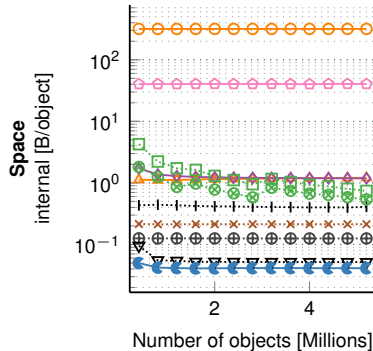
- here only fixed size
- more in the paper

## Competitors

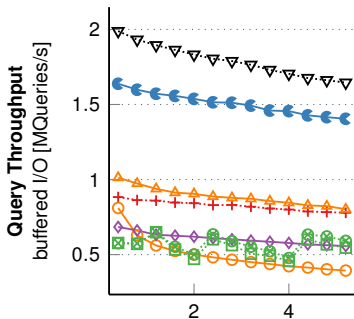
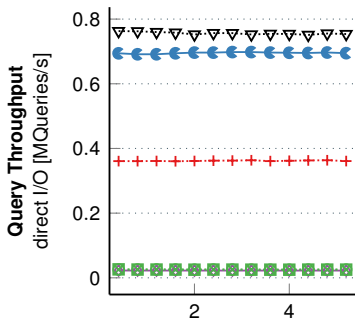
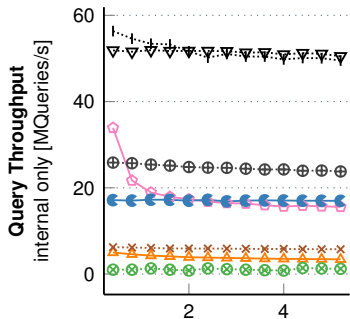
- LevelDB [[Goo21](#)]
- RocksDB [[Fac21](#)]
- SILT [[Lim+11](#)].
- `std::unordered_map`
- RecSplit [[EGV20](#)]
- CHD [[BBD09](#); [CR+12](#)]
- PTHash [[PT21](#)]



# Construction

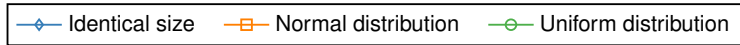
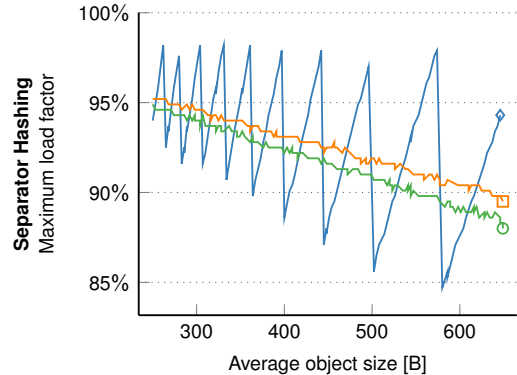
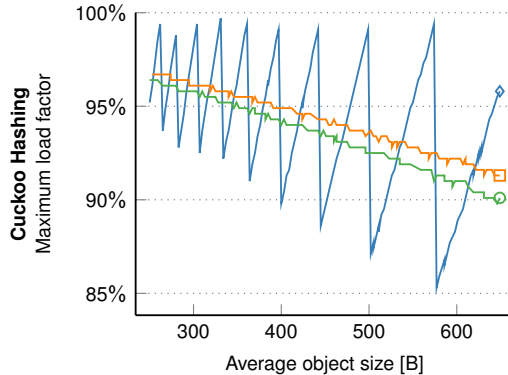


# Queries



⋯⊕⋯ CHD (16-perfect) [BBD09]	—○— LevelDB [Goo21]	⋯×⋯ RecSplit [EGV20]	⋯⊗⋯ SILT (Static part) [Lim+11]
⋯+⋯ Cuckoo (here)	⋯+⋯ PTHash [PT21]	—◇— RocksDB [Fac21]	⋯▽⋯ Separator (here)
—△— LevelDB (Static part) [Goo21]	—●— PaCHash (here)	⋯□⋯ SILT [Lim+11]	—◇— std::unordered_map

# Maximum Load Factor of Competitors



# Alternative Internal Memory Data Structure

## Lemma: Space with Succincter

When using Succincter [Pat08] to store  $p$ , the index needs  $1.4427 + \log(a + 1) + o(1)$  bits of internal memory per block.

## Entropy Encoding

- encode positions directly
- compress bit vector using entropy coding
- currently very naive Huffman codes on blocks of size 8, 16, 32, or 64

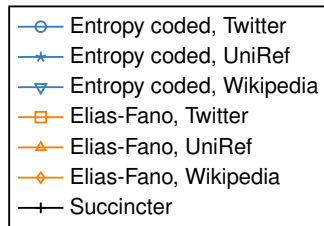
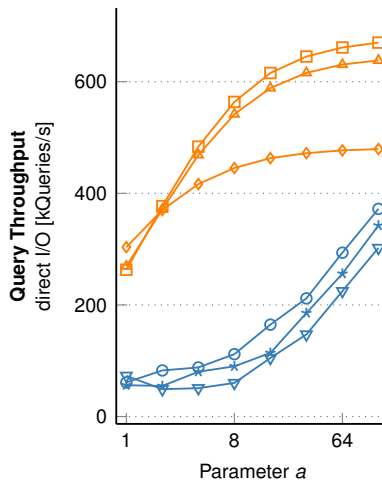
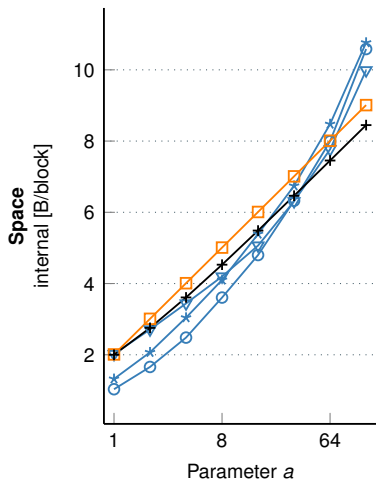
## Structure of Bit Vector

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

## Open Question

- better compression using the structure
- better as in faster to decompress
- at most the size of the entropy encoding

# Compressing Internal Memory Data Structure



# Conclusion

- static hash table for objects of variable size
- constant number of bits per EM block
- outperforming competitors (variable size)
- matching and outperforming competitors (fixed size)

- code available under GPLv3 license
- <https://github.com/ByteHamster/PaCHash>
- preprint available in arxiv



European Research Council  
Established by the European Commission

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 882500).

# Bibliography I

- [Aza+94] Yossi Azar, Andrei Z. Broder, Anna R. Karlin, and Eli Upfal. “Balanced allocations (extended abstract)”. In: *STOC*. ACM, 1994, pages 593–602. DOI: [10.1145/195058.195412](https://doi.org/10.1145/195058.195412).
- [BBD09] Djamel Belazzougui, Fabiano C. Botelho, and Martin Dietzfelbinger. “Hash, Displace, and Compress”. In: *ESA*. Volume 5757. Lecture Notes in Computer Science. Springer, 2009, pages 682–693. DOI: [10.1007/978-3-642-04128-0\\_61](https://doi.org/10.1007/978-3-642-04128-0_61).
- [Cel88] Pedro Celia. *External Robin Hood Hashing*. Technical report. Computer Science Department, Indiana University. TR246, 1988.
- [CR+12] Davi de Castro Reis, Djamel Belazzougui, Fabiano Cupertino Botelho, and Nivio Ziviani. *CMPH - C Minimal Perfect Hashing Library*. <http://cmph.sourceforge.net/>. 2012.
- [DSL11] Biplob K. Debnath, Sudipta Sengupta, and Jin Li. “SkimpyStash: RAM space skimpy key-value store on flash-based storage”. In: *SIGMOD Conference*. ACM, 2011, pages 25–36. DOI: [10.1145/1989323.1989327](https://doi.org/10.1145/1989323.1989327).

## Bibliography II

- [EGV20] Emmanuel Esposito, Thomas Mueller Graf, and Sebastiano Vigna. “RecSplit: Minimal Perfect Hashing via Recursive Splitting”. In: *ALLENEX*. SIAM, 2020, pages 175–185. DOI: [10.1137/1.9781611976007.14](https://doi.org/10.1137/1.9781611976007.14).
- [Eli74] Peter Elias. “Efficient Storage and Retrieval by Content and Address of Static Files”. In: *J. ACM* 21.2 (1974), pages 246–260. DOI: [10.1145/321812.321820](https://doi.org/10.1145/321812.321820).
- [Fac21] Facebook. *RocksDB. A Persistent Key-Value Store for Fast Storage Environments*. <https://rocksdb.org>. 2021.
- [Fan71] Robert Mario Fano. *On the number of bits required to implement an associative memory*. Technical report. Project MAC, Memorandum 61". MIT, Computer Structures Group, 1971.
- [GL88] Gaston H. Gonnet and Per-Åke Larson. “External hashing with limited internal storage”. In: *J. ACM* 35.1 (1988), pages 161–184. DOI: [10.1145/42267.42274](https://doi.org/10.1145/42267.42274).
- [Goo21] Google. *LevelDB is a Fast Key-Value Storage Library Written at Google*. <https://github.com/google/leveldb>. 2021.



## Bibliography III

- [JP08] Morten Skaarup Jensen and Rasmus Pagh. “Optimality in External Memory Hashing”. In: *Algorithmica* 52.3 (2008), pages 403–411. DOI: [10.1007/s00453-007-9155-x](https://doi.org/10.1007/s00453-007-9155-x).
- [Lar88] Per-Åke Larson. “Linear Hashing with Separators - A Dynamic Hashing Scheme Achieving One-Access Retrieval”. In: *ACM Trans. Database Syst.* 13.3 (1988), pages 366–388. DOI: [10.1145/44498.44500](https://doi.org/10.1145/44498.44500).
- [Lim+11] Hyeontaek Lim, Bin Fan, David G. Andersen, and Michael Kaminsky. “SILT: a memory-efficient, high-performance key-value store”. In: *SOSP*. ACM, 2011, pages 1–13. DOI: [10.1145/2043556.2043558](https://doi.org/10.1145/2043556.2043558).
- [LK84] Per-Åke Larson and Ajay Kajla. “File Organization: Implementation of a Method Guaranteeing Retrieval in One Access”. In: *Commun. ACM* 27.7 (1984), pages 670–677. DOI: [10.1145/358105.358193](https://doi.org/10.1145/358105.358193).
- [LR85] Per-Åke Larson and M. V. Ramakrishna. “External Perfect Hashing”. In: *SIGMOD Conference*. ACM Press, 1985, pages 190–200. DOI: [10.1145/318898.318916](https://doi.org/10.1145/318898.318916).

## Bibliography IV

- [Pag03] Rasmus Pagh. “Basic External Memory Data Structures”. In: *Algorithms for Memory Hierarchies*. Volume 2625. Lecture Notes in Computer Science. Springer, 2003, pages 14–35. DOI: [10.1007/3-540-36574-5\\_2](https://doi.org/10.1007/3-540-36574-5_2).
- [Pat08] Mihai Patrascu. “Succincter”. In: *FOCS*. IEEE Computer Society, 2008, pages 305–313. DOI: [10.1109/FOCS.2008.83](https://doi.org/10.1109/FOCS.2008.83).
- [PT21] Giulio Ermanno Pibiri and Roberto Trani. “PHash: Revisiting FCH Minimal Perfect Hashing”. In: *SIGIR*. ACM, 2021, pages 1339–1348. DOI: [10.1145/3404835.3462849](https://doi.org/10.1145/3404835.3462849).
- [RT89] M. V. Ramakrishna and Walid R. Tout. “Dynamic External Hashing with Guaranteed Single Access Retrieval”. In: *FODO*. Volume 367. Lecture Notes in Computer Science. Springer, 1989, pages 187–201. DOI: [10.1007/3-540-51295-0\\_127](https://doi.org/10.1007/3-540-51295-0_127).