Hecura: Streaming B-trees for File Systems and Databases

Michael A. Bender
Stony Brook

Martin Farach-Colton
Rutgers

Problem

Traditional B-tree operations (update, range query, search) are slow.

- inserts / deletes / searches are 0.02% to 0.05% of disk bandwidth.

Improving this performance is an algorithmic problem.

Problem

Traditional B-tree operations (update, range query, search) are slow.

- inserts / deletes / searches are 0.02% to 0.05% of disk bandwidth.

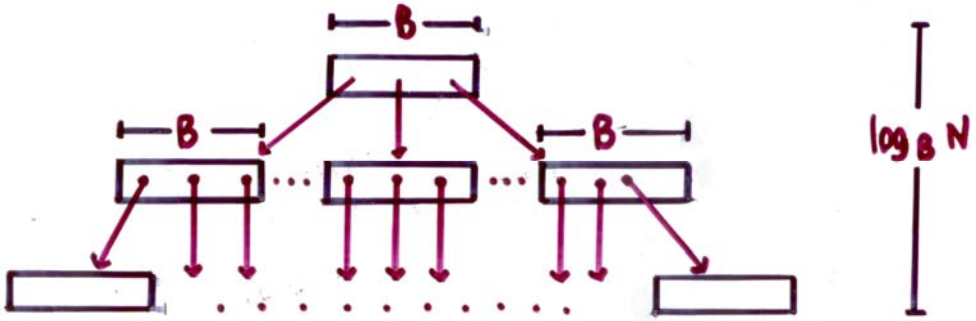
Improving this performance is an algorithmic problem.

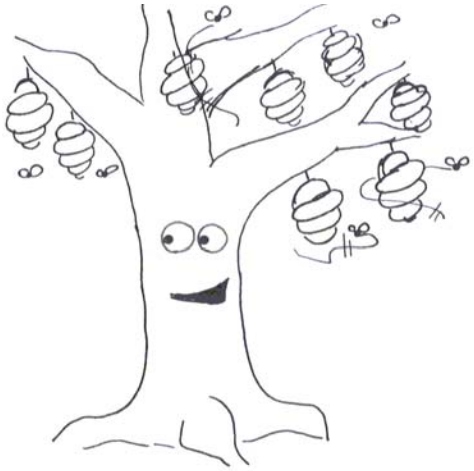
(We believe a better algorithm can replace 10-100 disks.)

This research: Build prototype streaming B-trees [Bender, Farach-Colton, Kuszmaul 06]
for indexing high bandwidth data.
"data ingest problem"

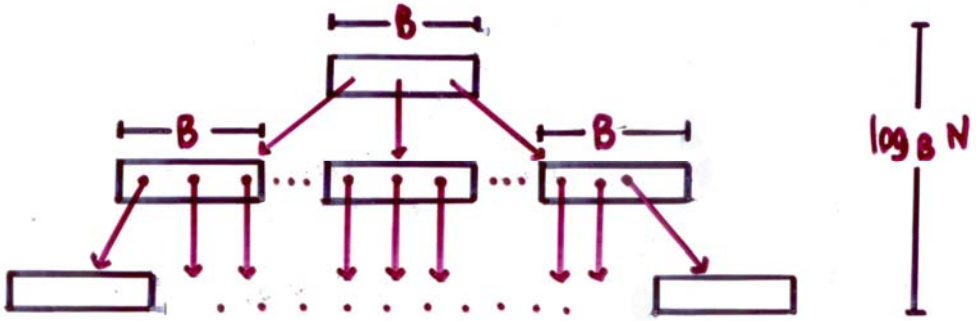
Conventional wisdom: searches more common than inserts.
Often holds, but in many critical cases it doesn't...

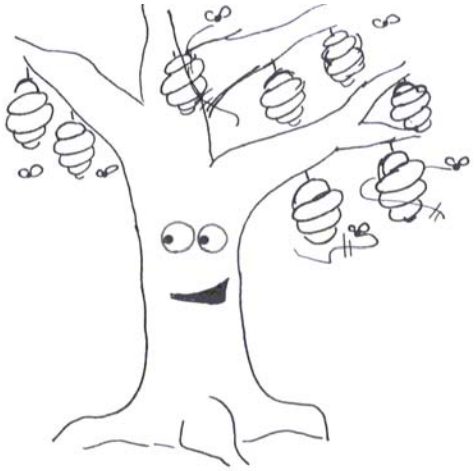
Traditional B-tree [Bayer, McCreight 72][Comer 79]





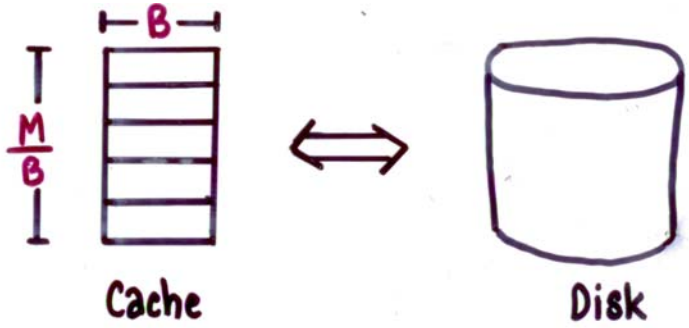
Traditional B-tree [Bayer, McCreight 72][Comer 79]





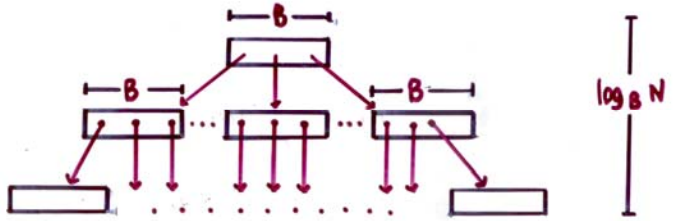
Designed for Disk Access Model (DAM) [Aggarwal, Vitter 88]

- block size B, memory size M
- count number of block transfers



⇒ B-tree operations take $O(\log_B N)$ memory transfers.

B-tree Updates are slow

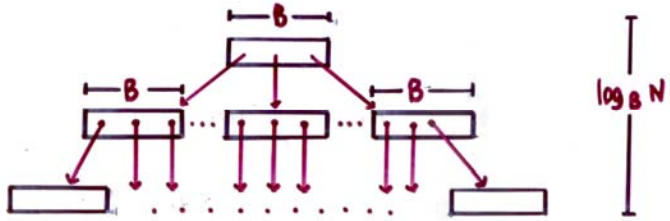


- $O(\log_B N)$ memory transfers is provably optimal for searches, but we can do better for inserts.

Streaming B-tree - Tradeoff between search and insert.

Small sacrifice in search gives great advantage in insert.

B-tree Updates are Slow



- $O(\log_B N)$ memory transfers is provably optimal for searches, but we can do better for inserts.

Streaming B-tree – Tradeoff between search and insert.

Small sacrifice in search gives great advantage in insert.

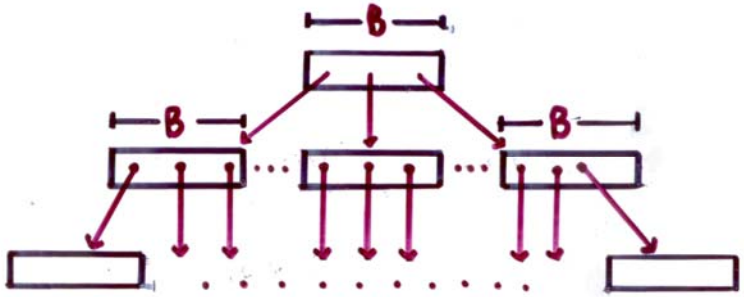
	Search	Insert
B-tree	$\log_B N$	$\log_B N$
Streaming B-trees	$2 \log_B N (= \log_{\sqrt{B}} N)$	$2 \log_B N / \sqrt{B}$
Streaming B-trees	$3 \log_B N (= \log_{B^{1/3}} N)$	$3 \log_B N / B^{2/3}$
Streaming B-trees	$\lg N$	$\lg N / B$

[Buchsbaum, Goldwasser, Kenkatasubramanian, Westbrook 00]

B-tree Range Queries are Slow

Range Query: scan of elements in chosen range.

- leaf blocks scattered throughout disk
- random block transfers are 1-2 orders of magnitude slower than sequential block transfers.



Streaming B-trees

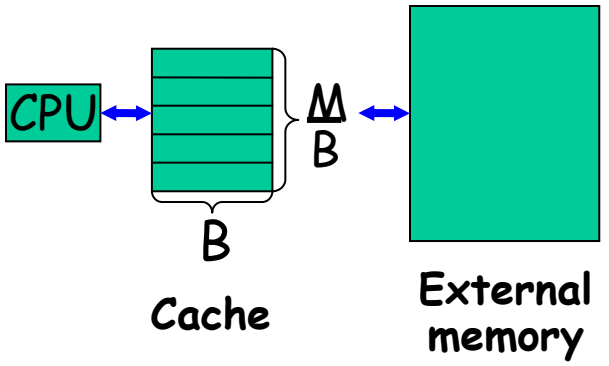
- one version keeps dynamic data physically in order on disk

  ⇒ very fast range queries

Research

- Build prototype streaming B-trees [Bender, Farach-Colton, Kuszmaul 06]
  - fast indexing: 1-2 orders of magnitude faster than B-trees
  - fast range queries: 1-2 orders of magnitude faster than B-trees
  - slower searches: 1-3 times slower
  - cache-oblivious techniques
- Study related issues:
  - different-length keys
  - transactions
  - parallel disks
  - o/s support of cache-obliviousness

Cache-Oblivious Model

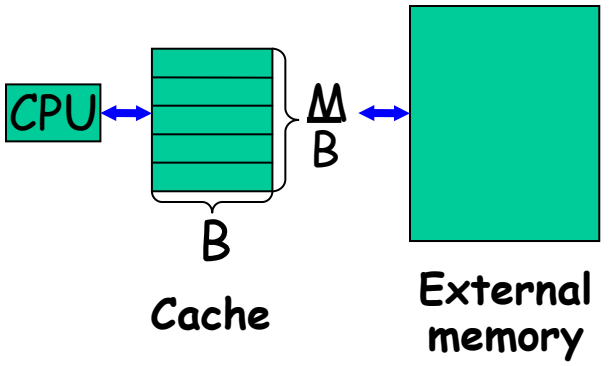


- Like DAM model, except
  B & M unknown to programmer/algorithm.

⇒ Reason about B, M, but prove results about unknown multilevel memory hierarchy.

Attractive for disks with no "correct" blocksize.

Cache-Oblivious Model



- Like DAM model, except
  B & M unknown to programmer/algorithm.

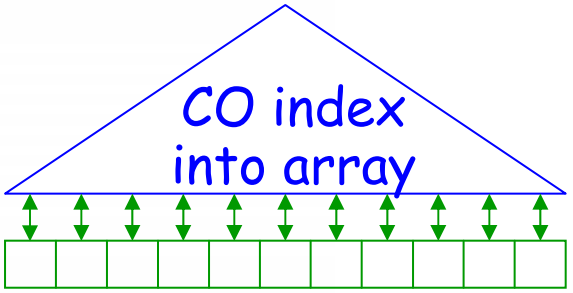
⇒ Reason about B, M, but prove results about unknown multilevel memory hierarchy.

Attractive for disks with no "correct" blocksize.

- Manual Tuning - The programmer writes block size into code (B-trees).
- Active Automatic Tuning - The program measures the cache parameters. (FFTW [Frigo, Johnson 97]
- Passive Automatic Tuning - In CO model, the program passively employs whatever cache exists. Surprisingly, optimal data structures exist.

Historical Context

- Cache-oblivious model [Frigo, Leiserson, Prokop, Ramachandran 99]
- Cache-oblivious B-tree [Bender, Demaine, Farach-Colton 00]
- Cache-oblivious B-tree simplifications
  [Rahman, Cole, Raman 01] [Brodal, Fagerberg, Jacob 02]
  [Bender, Duan, Iacono, Wu 02]



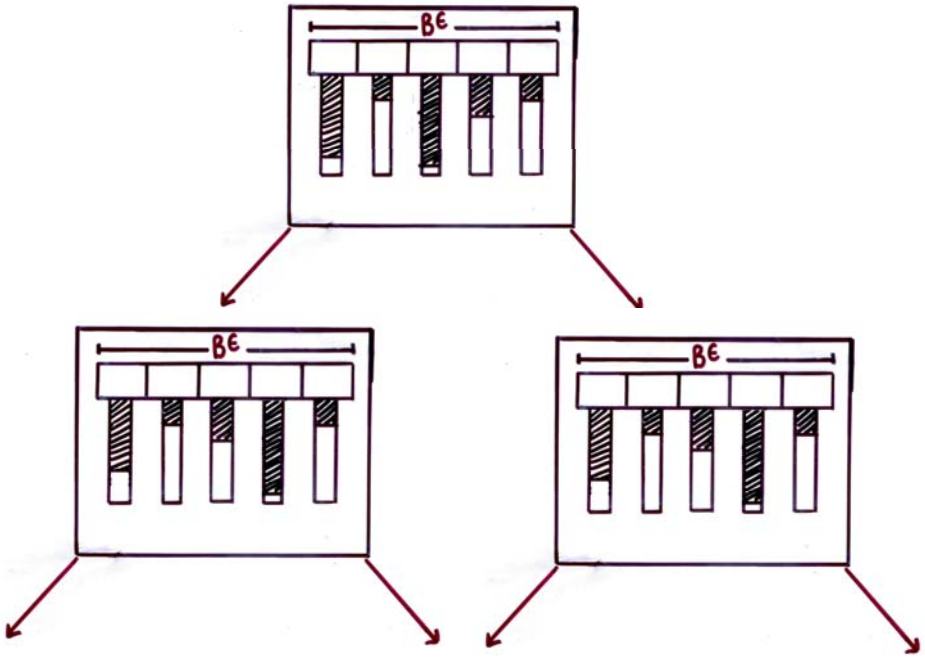
cache-oblivious B-tree

- Many other data structures and algorithms (2001-2006).
- Cache-oblivious B-tree beats traditional B-tree [Bender, Farach-Colton, Kuszmaul 06].
  Idea: no "right" value for B on disks.

This Research: use cache-oblivious technology to build Streaming B-tree.

Idea of Streaming B-trees [Bender, Farach-Colton, Kuszmaul 06]

Use part of the streaming B-tree node to buffer elements as they are inserted.



Insert: $O(\log_B N/\epsilon B^{1-\epsilon})$

Search: $O(\log_B N/\epsilon)$

But no better at range queries than B-trees…

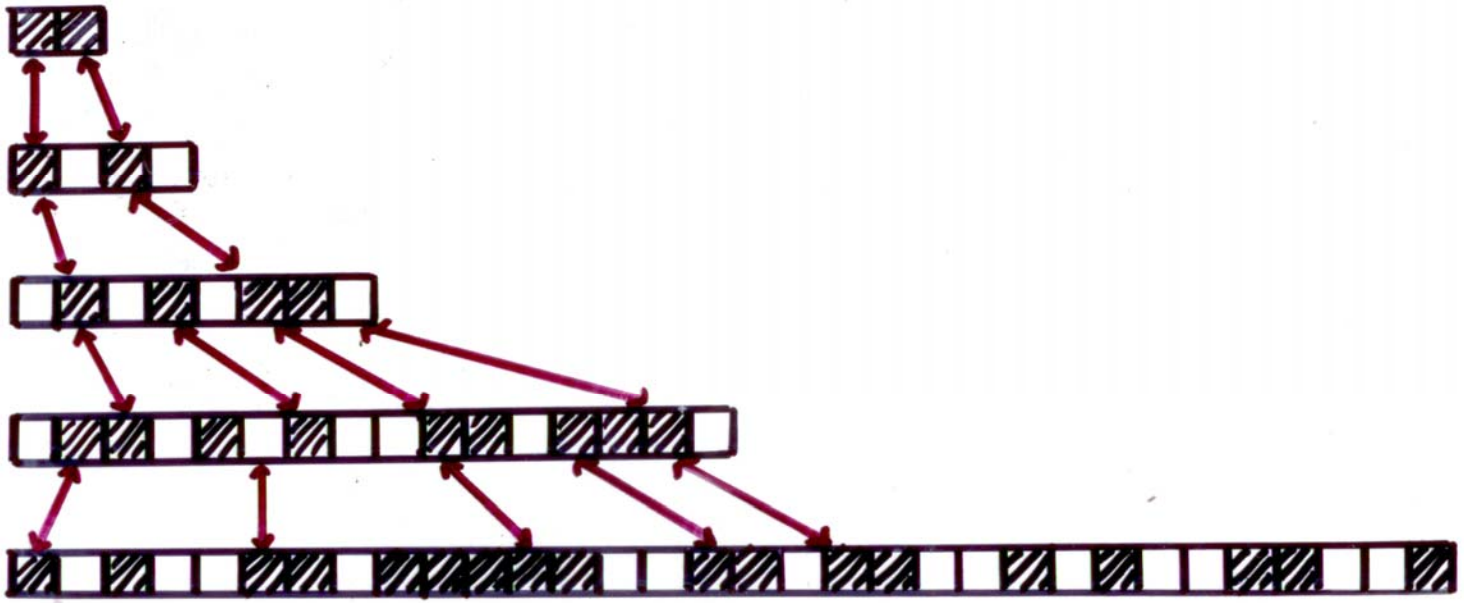
Packed Memory Array (PMA) [Bender, Demaine, Farach-Colton 00]

Maintain dynamic data physically in order on disk with $O(1)$ gaps (e.g., 30% extra space).

Like leaving gaps on a bookshelf for insertions:

- $O(1+\log^2 N/B)$ amortized moves memory transfers. [Bender, Demaine, Farach-Colton 00]
- $O(1+\log N/B)$ amortized moves memory transfers for common case [Bender, Hu 00]

Using PMA to improve Streaming B-tree



$O(\log_B N)$ *packed-memory arrays*

Summary

- Build Prototype Streaming B-trees
  - Updates and Range Queries: 1-2 orders of magnitude speedup
  - Searches: 1-3 times slower
  - Cache-oblivious techniques
- Other Directions
  - Different-length keys, Transactions, Parallel Disks, O/S support for CO programming

- Interaction with National Labs
  - We appreciate interaction with national labs on streaming problems (e.g., "data ingest problem"). Do national labs have relevant streaming applications?
  - Past interaction, with Sandia National Labs, was on locality in processor allocation.
  - We won an **R&D 100 Award** in 2006 for the **Compute Process Allocator**.

Summary

- Build Prototype Streaming B-trees
  - Updates and Range Queries: 1-2 orders of magnitude speedup
  - Searches: 1-3 times slower
  - Cache-oblivious techniques
- Other Directions
  - Different-length keys, Transactions, Parallel Disks, O/S support for CO programming

- Interaction with National Labs
  - We appreciate interaction with national labs on streaming problems (e.g., "data ingest problem"). Do national labs have relevant streaming applications?
  - Past interaction, with Sandia National Labs, was on locality in processor allocation.
  - We won an **R&D 100 Award** in 2006 for the **Compute Process Allocator**.

Summary

- Build Prototype Streaming B-trees
  - Updates and Range Queries: 1-2 orders of magnitude speedup
  - Searches: 1-3 times slower
  - Cache-oblivious techniques
- Other Directions
  - Different-length keys, Transactions, Parallel Disks, O/S support for CO programming

- Interaction with National Labs
  - We appreciate interaction with national labs on streaming problems (e.g., "data ingest problem"). Do national labs have relevant streaming applications?
  - Past interaction, with Sandia National Labs, was on locality in processor allocation.
  - We won an **R&D 100 Award** in 2006 for the **Compute Process Allocator**.

Static B-tree Performance

[Bender, Farach-Colton, Kuszmaul '06]

Data structure	Average time per search	
	small-machine	big-machine
CO B-tree	12.3ms	13.8ms
Btree: 4KB Blocks:	17.2ms	22.4ms
16KB blocks:	13.9ms	22.1ms
32KB blocks:	11.9ms	17.4ms
64KB blocks:	12.9ms	17.6ms
128KB blocks:	13.2ms	16.5ms
256KB blocks:	18.5ms	14.4ms
512KB blocks:		16.7ms

- Static CO B-tree comparable with optimized static traditional B-trees
  - optimizes for right "effective block size"

Dynamic B-trees

[Bender, Farach-Colton, Kuszmaul '05]

	Block Size	insert 440,000 random values	insert 450,000 random values	range query of all data	1000 random searches
	CO B-tree	15.8s		4.6s	5.9s
	CO B-tree		54.8s	9.3s	7.1s
Sequential block allocation:	2K		19.2s	24.8s	12.6s
	4K		19.1s	23.1s	10.5s
	8K		26.4s	22.3s	8.4s
	16K		41.5s	22.2s	7.7s
	32K		71.5s	21.4s	7.3s
	64K		128.0s	11.5s	6.5s
	128K		234.8s	7.3s	6.2s
	256K		444.5s	6.5s	5.3s
Random block allocation:	2K		3928.0s	460.3s	24.3s
Berkeley DB:			1201.1s		
Berkeley DB (64 MB pool):			76.6s		

- CO B-trees fantastic for range queries
- CO B-tree always near best parameter choices in traditional B-trees for inserts, range, queries, searches

Dynamic B-trees-Seq inserts

[Bender, Farach-Colton, Kuszmaul '05]

Time to insert a sorted sequence of 450,000 keys	
Dynamic CO B-tree	61.2s
4KB Btree	17.1s
Berkeley DB (64MB)	37.4s

- CO B-tree worst for seq inserts, Berkeley DB optimized for seq inserts

- Goal: PMA good but should be even better…