Small Refinements to the DAM Can Have Big Consequences for Data-Structure Design

Michael A. Bender, Alex Conway, Martín Farach-Colton, William Jannen, Yizheng Jiao, Rob Johnson, Eric Knorr, Sara McAllister, Nirjhar Mukherjee, Prashant Pandey, Donald E. Porter, Jun Yuan, Yang Zhan
I/O models → predict performance

I/Os are slow! Even fast I/Os!

HDD → Sequential I/Os

SSD → Concurrent I/Os

Common to all storage: block I/Os
The DAM model: de facto for external memory
Aggarwal+Vitter ’88

- **How computations work:**
  - Data is transferred in blocks between RAM and disk.
  - The number of block transfers dominate the running time.

- **Goal: Minimize number of block transfers**
  - Performance bounds are parameterized by block size $B$, memory size $M$, data size $N$. 
B-tree: a classic external memory data structure

... ≈ B children ...

... ≈ N / B leaves ...

O (log₂ N)
B-trees in the DAM model

<table>
<thead>
<tr>
<th></th>
<th>Insert</th>
<th>Point query</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-tree (DAM)</td>
<td>$O(\log_B N)$</td>
<td>$O(\log_B N)$</td>
</tr>
</tbody>
</table>
B-trees in the DAM model

<table>
<thead>
<tr>
<th></th>
<th>Insert</th>
<th>Point query</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-tree (DAM)</td>
<td>$O \left( \log_B \frac{N}{M} \right)$</td>
<td>$O \left( \log_B \frac{N}{M} \right)$</td>
</tr>
</tbody>
</table>
## B-trees in the DAM model

<table>
<thead>
<tr>
<th></th>
<th>Insert</th>
<th>Point query</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-tree (DAM)</td>
<td>$O\left(\frac{1}{\log B} \log \frac{N}{M}\right)$</td>
<td>$O\left(\frac{1}{\log B} \log \frac{N}{M}\right)$</td>
</tr>
</tbody>
</table>

### DAM analysis

![Graph showing the relationship between node size B and I/Os.](image)

- The graph illustrates the decrease in I/Os as the node size increases.
- The red line indicates the trend observed in the DAM analysis.
DAM × Hardware = Profit

DAM analysis

Hardware measurements

- #/I/Os vs. Node size B (decreasing)
- Time of I/Os vs. Node size B (increasing)
- Total time vs. Node size B (minimum at a specific node size B)
Why is one $B$ better than another $B$?

- Latency $\rightarrow$ seek time
- Bandwidth $\rightarrow$ transfer cost
- **Half-bandwidth point**, $B$ where latency = bandwidth

When $B =$ half-bandwidth point, the DAM 2-approximates the I/O cost
DAM ✗ Hardware = Profit

DAM analysis

Hardware measurements
An example of the half-bandwidth point

- HDD: Bandwidth = 300MB/sec and Seek time = 5 millisec
  - Half-bandwidth point = 1.5MB
An example of the half-bandwidth point

- HDD: Bandwidth = 300MB/sec and Seek time = 5 millisec
  - Half-bandwidth point = 1.5MB
- In practice,
  - B-trees node sizes = 4KB-64KB
  - $B^\epsilon$-trees (explained later) node sizes = 4MB
An example of the half-bandwidth point

- HDD: Bandwidth = 300\text{MB/sec} and Seek time = 5 \text{ millisec}
  - Half-bandwidth point = 1.5\text{MB}
- In practice,
  - B-trees node sizes = 4\text{KB-64\text{KB}}
  - $B^c$-trees (explained later) node sizes = 4\text{MB}
- Why deviate from the half-bandwidth point?
An example of the half-bandwidth point

- HDD: Bandwidth = \textbf{300MB/sec} and Seek time = 5 \text{ millisec}
  - Half-bandwidth point = \textbf{1.5MB}
- In practice,
  - B-trees node sizes = \textbf{4KB-64KB}
  - B^c-trees (explained later) node sizes = \textbf{4MB}
- \textbf{Why deviate from the half-bandwidth point?}

The DAM model cannot answer these questions because \( B \) is a parameter of the model.
This paper: using refined models for HDDs and SSDs

- **Affine model**\(^{[1, 2]}\): for HDDs
  - How to optimize \(B\) for \(B\)-trees and \(B^\varepsilon\)-trees on hard drives.
  - Explains node-size variability in \(B\)-trees and \(B^\varepsilon\)-trees.
  - Reveals new intra-node optimizations for \(B^\varepsilon\)-trees.

- **PDAM model**\(^{[3]}\): for SSDs
  - Allows parallelism-oblivious optimizations.
This paper: using refined models for HDDs and SSDs

- **Affine model**\(^1, 2\): for HDDs
  - How to optimize \( B \) for B-trees and \( B^\epsilon \)-trees on hard drives.
  - Explains node-size variability in B-trees and \( B^\epsilon \)-trees.
  - Reveals new intra-node optimizations for \( B^\epsilon \)-trees.

- **PDAM model**\(^3\): for SSDs
  - Allows parallelism-oblivious optimizations.

---

\(^3\) Alok Aggarwal and Jeffrey Scott Vitter. 1988. The Input/Output Complexity of Sorting and Related Problems. Commun. ACM
The affine model: for hard drives

- An I/O of size $x$ words costs $1 + \alpha x$,
  - 1 is the normalized setup cost
  - $\alpha \leq 1$ is the normalized bandwidth cost

  for spinning disks, $\alpha = \text{transfer time}/\text{seek time}$

Half-bandwidth point $= \frac{1}{\alpha}$
Experimental validation of the affine model

Empirical I/O cost is almost exactly same as the cost predicted by the affine model.
B-trees in the affine model

<table>
<thead>
<tr>
<th></th>
<th>Insert</th>
<th>Point query</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-tree (DAM)</td>
<td>$O \left( \frac{1}{\log B} \log \frac{N}{M} \right)$</td>
<td>$O \left( \frac{1}{\log B} \log \frac{N}{M} \right)$</td>
</tr>
<tr>
<td>B-tree (Affine)</td>
<td>$O \left( \frac{1+\alpha B}{\log B} \log \frac{N}{M} \right)$</td>
<td>$O \left( \frac{1+\alpha B}{\log B} \log \frac{N}{M} \right)$</td>
</tr>
</tbody>
</table>

Point queries and inserts are optimized when the node size is:

$$B = \Theta\left( \frac{1}{\alpha} \frac{1}{\ln(1/\alpha)} \right)$$

B-trees are optimized by making nodes much smaller than the half-bandwidth point.
Example: optimal B-tree node size in the affine model

<table>
<thead>
<tr>
<th>Seagate 2TB Hard Drive</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Seek time</strong></td>
</tr>
<tr>
<td><strong>bandwidth</strong></td>
</tr>
<tr>
<td><strong>Key-value pair size</strong></td>
</tr>
<tr>
<td><strong>ln (1/α)</strong></td>
</tr>
<tr>
<td><strong>Optimal node size B</strong></td>
</tr>
</tbody>
</table>

The affine model explains most of the discrepancy between the node sizes used in practice and those that optimize the DAM model.
$B^{1/2}$-tree: using nodes as buffers

Use the node space $(B - \sqrt{B})$ as buffer for inserts and delete messages.
B^{1/2}-tree: buffering reduces the fanout

Reduce the fanout in the B-tree to \( \sqrt{B} \).
Inject insert and delete messages at the root.
When buffer fills up, flush to child nodes.
$B^{1/2}$-tree: move all messages destined for a child

$\approx \frac{N}{B}$ leaves

$O (\log_{\sqrt{B}} N)$

$\approx \sqrt{B}$ elements move down the tree in 1 I/O.
**$B^{1/2}$-tree in the DAM model**

<table>
<thead>
<tr>
<th></th>
<th>Insert</th>
<th>Point query</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B^{1/2}$-tree</td>
<td>$O \left( \frac{1}{\sqrt{B}} \log \sqrt{B} \frac{N}{M} \right)$</td>
<td></td>
</tr>
</tbody>
</table>
\( B^{1/2} \)-tree: searches cost the same as the B-tree

Examine each buffer on root-to-leaf path. 1 I/O per node.
$B^{1/2}$-tree in the DAM model

<table>
<thead>
<tr>
<th></th>
<th>Insert</th>
<th>Point query</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B^{1/2}$-tree</td>
<td>$O\left(\frac{1}{\sqrt{B}} \log\sqrt{B} \frac{N}{M}\right)$</td>
<td>$O\left(\log\sqrt{B} \frac{N}{M}\right)$</td>
</tr>
</tbody>
</table>
**B^{1/2}-tree in the DAM model**

<table>
<thead>
<tr>
<th></th>
<th>Insert</th>
<th>Point query</th>
</tr>
</thead>
<tbody>
<tr>
<td>B^{1/2}-tree</td>
<td>$O \left( \frac{1}{\sqrt{B \log B}} \log \frac{N}{M} \right)$</td>
<td>$O \left( \frac{1}{\log B} \log \frac{N}{M} \right)$</td>
</tr>
</tbody>
</table>
### B^{1/2}-tree in the affine model

<table>
<thead>
<tr>
<th></th>
<th>Insert</th>
<th>Point query</th>
</tr>
</thead>
<tbody>
<tr>
<td>B^{1/2}-tree</td>
<td>$\mathcal{O} \left( \frac{1+\alpha B}{\sqrt{B \log B}} \log \frac{N}{M} \right)$</td>
<td>$\mathcal{O} \left( \frac{1+\alpha B}{\log B} \log \frac{N}{M} \right)$</td>
</tr>
</tbody>
</table>
**B^{1/2}-tree in the affine model**

<table>
<thead>
<tr>
<th>B^{1/2}-tree</th>
<th>Insert</th>
<th>Point query</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O \left( \frac{1+\alpha B}{\sqrt{B \log B}} \log \frac{N}{M} \right)$</td>
<td>$O \left( \frac{1+\alpha B}{\log B} \log \frac{N}{M} \right)$</td>
<td></td>
</tr>
</tbody>
</table>

Insert cost increases more slowly in B^{1/2}-trees than in B-trees as the node size increases.
**B^{1/2}**-tree in the affine model

<table>
<thead>
<tr>
<th></th>
<th>Insert</th>
<th>Point query</th>
</tr>
</thead>
<tbody>
<tr>
<td>B^{1/2}\text{-tree}</td>
<td>$O \left( \frac{1+\alpha B}{\sqrt{B \log B}} \log \frac{N}{M} \right)$</td>
<td>$O \left( \frac{1+\alpha B}{\log B} \log \frac{N}{M} \right)$</td>
</tr>
</tbody>
</table>

Insert cost increases more slowly in B^{1/2}-trees than in B-trees as the node size increases.

But the bandwidth cost of queries is still linear in $B$. 
Organizing messages $\rightarrow$ 2 I/Os per node

Elements destined for a particular node are stored together.
The cost to read all these elements is $1 + \alpha \sqrt{B}$.

Organizing messages → 2 I/Os per node

... ≈ $\frac{N}{B}$ leaves ...

$O \left( \log_{\sqrt{B}} N \right)$
The cost to read all these elements is $1 + \alpha \sqrt{B}$.

Organizing messages → 2 I/Os per node

The cost to read all these elements is $1 + \alpha \sqrt{B}$.
Store child’s pivot in parent $\rightarrow$ 1 I/O per node

Keeping pivots of a node in its parent avoids the extra I/O.
Store child’s pivot in parent → 1 I/O per node

If fanout is $\approx \sqrt{B}$ the node size increases by at most constant factor.
Optimized query costs in $B^{1/2}$-trees

<table>
<thead>
<tr>
<th>$B^{1/2}$-tree</th>
<th>Insert</th>
<th>Point query</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$O\left(\frac{1+\alpha B}{\sqrt{B \log B}} \log \frac{N}{M}\right)$</td>
<td>$O\left(\frac{1+\alpha \sqrt{B}}{\log B} \log \frac{N}{M}\right)$</td>
</tr>
</tbody>
</table>
Optimized query cost grows slowly

The query cost now grows more slowly with increasing node size.
Example: optimal $B^{1/2}$-tree node size in the affine model

<table>
<thead>
<tr>
<th>Seagate 2TB Hard Drive</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Seek time</td>
<td>5 msec</td>
</tr>
<tr>
<td>bandwidth</td>
<td>300 MB/sec</td>
</tr>
<tr>
<td>Key-value pair size</td>
<td>16 bytes</td>
</tr>
<tr>
<td>Optimal node size $B$</td>
<td>$\approx$1.8MB</td>
</tr>
</tbody>
</table>

The affine model explains the discrepancy between the node sizes used in practice and those that optimize the DAM model.
Conclusion: Design in the DAM, Refine in the Affine Hardware
Conclusion: Design in the DAM, Refine in the Affine

- Data structure performance
- Hardware
Conclusion: Design in the DAM, Refine in the Affine
Conclusion: Design in the DAM, Refine in the Affine

- Software optimizations
- Data structure performance
- Hardware

New data structure ideas