Write-Optimized Data Structures Revisited

HQN

September 22, 2017

Problem and Notations

I/O model (Aggrawal and Vitter – CACM 1988)

- M: number of words in main memory
- P: number of words in a disk page[1]
- Each page read/write costs one IO.
- Runtime measured in # IOs

[1] P is non-standard, people often use B

Problem and Notations

I/O model (Aggrawal and Vitter – CACM 1988)

- M: number of words in main memory
- P: number of words in a disk page[1]
- Each page read/write costs one IO.
- Runtime measured in # IOs

Problem (The problem)

data structure $\mathcal{D}$ to store N items on disk for efficient (bulk) reads and writes. Characterize the tradeoff point:

$$\langle \textit{point query IO-time}, \textit{update IO-time} \rangle = \langle t_q^{\mathcal{D}}, t_u^{\mathcal{D}} \rangle$$

[1] P is non-standard, people often use B

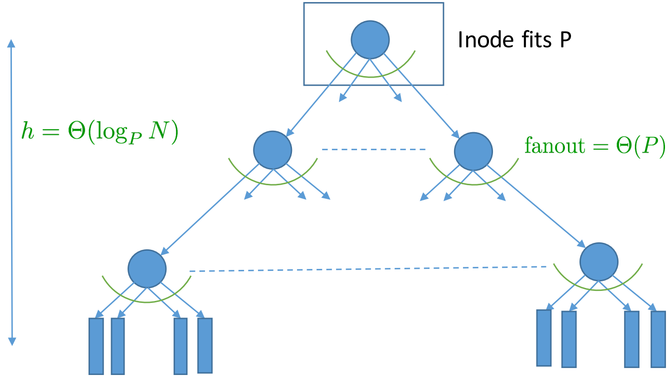
Some Well-Known Bounds

- For now, assume each item and each pointer costs one word.

Problem	Best IO time	Note	Internal memory model
Scanning	$\frac{N}{P}$	*Trivial*	N
Sorting	$\frac{N}{P} \log_{M/P} \frac{N}{P}$	*Merge sort*	$N \log_2 N$
Searching	$\log_P N$	*"Optimal"*	$\log_2 N$

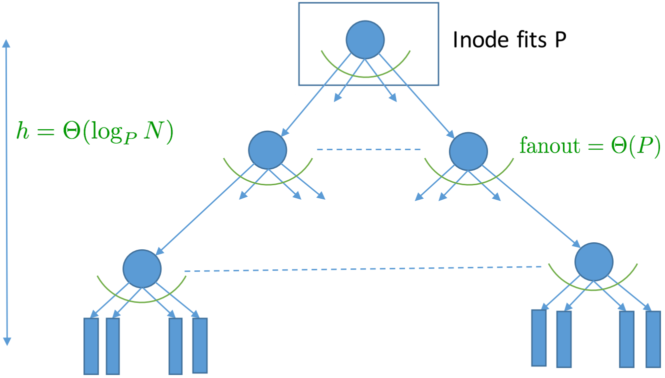
- Optimal IO-time sorting can even be made cache-oblivious (Frigo et al., FOCS 99)

B+Tree: Best Point Query IO-Time!



B+Tree: Best Point Query IO-Time!

Tradeoff point	$\langle t_q, t_u \rangle = \langle \log_P N, \log_P N \rangle$
Range search	$\log_P N + \dfrac{\#\text{outputs}}{P}$



B+Tree: Bad Update IO-Time

- Sorting using B+Tree:

B+Tree: Bad Update IO-Time

- Sorting using B+Tree:
  - Insert N items, $N \log_P N$

B+Tree: Bad Update IO-Time

- Sorting using B+Tree:
  - Insert N items, $N \log_P N$
  - range search $(-\infty, +\infty)$: $\log_P N + N/P$

B+Tree: Bad Update IO-Time

- Sorting using B+Tree:
  - Insert N items, $N \log_P N$
  - range search $(-\infty, +\infty)$: $\log_P N + N/P$
  - Overall sorting IO-time: $N \log_P N$

B+Tree: Bad Update IO-Time

- Sorting using B+Tree:
  - Insert N items, $N \log_P N$
  - range search $(-\infty, +\infty)$: $\log_P N + N/P$
  - Overall sorting IO-time: $N \log_P N$
- Off from optimal by a factor of

$$\frac{N \log_P N}{(N/P) \log_{M/P}(N/P)} \approx P \cdot \frac{\log(M/P)}{\log P}$$

B+Tree: Bad Update IO-Time

- Sorting using B+Tree:
  - Insert N items, $N \log_P N$
  - range search $(-\infty, +\infty)$: $\log_P N + N/P$
  - Overall sorting IO-time: $N \log_P N$
- Off from optimal by a factor of

$$\frac{N \log_P N}{(N/P) \log_{M/P}(N/P)} \approx P \cdot \frac{\log(M/P)}{\log P}$$

- The culprit:

$$\langle t_q, t_u \rangle = \boxed{\langle \log_P N, \log_P N \rangle}$$

Write-Optimized Data Structures

- “Write-Optimized B-Trees” (Goetz Graefe, 2004 – VLDB)

Write-Optimized Data Structures

- “Write-Optimized B-Trees” (Goetz Graefe, 2004 – VLDB)
  - Fence keys

Write-Optimized Data Structures

- “Write-Optimized B-Trees” (Goetz Graefe, 2004 – VLDB)
  - Fence keys
- Log structure merged (LSM) trees (O’Neil, 1996)

Write-Optimized Data Structures

- “Write-Optimized B-Trees” (Goetz Graefe, 2004 – VLDB)
  - Fence keys
- Log structure merged (LSM) trees (O’Neil, 1996)
  - Cascading trees (Veldhuizen, circa ???)

Write-Optimized Data Structures

- “Write-Optimized B-Trees” (Goetz Graefe, 2004 – VLDB)
  - Fence keys
- Log structure merged (LSM) trees (O’Neil, 1996)
  - Cascading trees (Veldhuizen, circa ???)
- Buffer trees

Write-Optimized Data Structures

- “Write-Optimized B-Trees” (Goetz Graefe, 2004 – VLDB)
  - Fence keys
- Log structure merged (LSM) trees (O’Neil, 1996)
  - Cascading trees (Veldhuizen, circa ???)
- Buffer trees
  - Lars Arge (1995)

Write-Optimized Data Structures

- "Write-Optimized B-Trees" (Goetz Graefe, 2004 – VLDB)
  - Fence keys
- Log structure merged (LSM) trees (O'Neil, 1996)
  - Cascading trees (Veldhuizen, circa ???)
- Buffer trees
  - Lars Arge (1995)
  - Brodal and Fagerberg (2003 – B^ϵ-tree)

Write-Optimized Data Structures

- "Write-Optimized B-Trees" (Goetz Graefe, 2004 – VLDB)
  - Fence keys
- Log structure merged (LSM) trees (O'Neil, 1996)
  - Cascading trees (Veldhuizen, circa ???)
- Buffer trees
  - Lars Arge (1995)
  - Brodal and Fagerberg (2003 – B^{ϵ}-tree)
  - ...

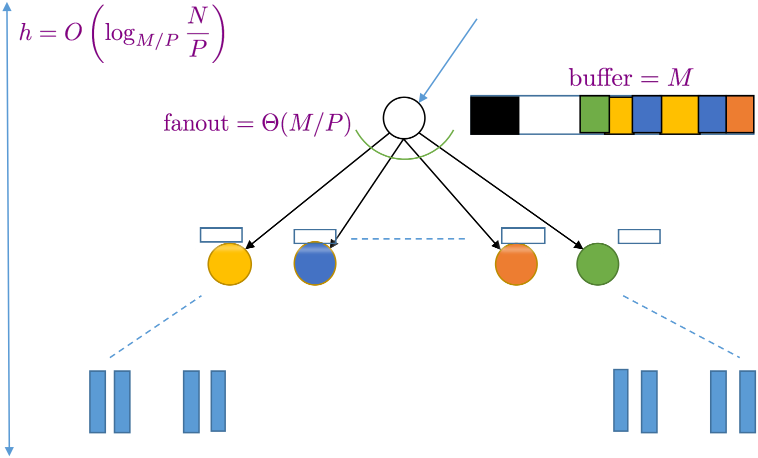
Write-Optimized Data Structures

- "Write-Optimized B-Trees" (Goetz Graefe, 2004 – VLDB)
  - Fence keys
- Log structure merged (LSM) trees (O'Neil, 1996)
  - Cascading trees (Veldhuizen, circa ???)
- Buffer trees
  - Lars Arge (1995)
  - Brodal and Fagerberg (2003 – B^ϵ-tree)
  - ...
- Tokutek guys (COLA, variations of Brodal-Fagerberg's idea and LSM and fractional cascading)

Write-Optimized Data Structures

- “Write-Optimized B-Trees” (Goetz Graefe, 2004 – VLDB)
  - Fence keys
- Log structure merged (LSM) trees (O’Neil, 1996)
  - Cascading trees (Veldhuizen, circa ???)
- Buffer trees
  - Lars Arge (1995)
  - Brodal and Fagerberg (2003 – B^ϵ-tree)
  - ...
- Tokutek guys (COLA, variations of Brodal-Fagerberg’s idea and LSM and fractional cascading)
- Idea: Turn multiple small writes into a single big write!

Lars Arge's Buffer Tree: Best Update IO-Time!



Amortized tradeoff: $\left\langle \frac{M}{P} \log_{M/P}(N/P), \frac{\log_{M/P}(N/P)}{P} \right\rangle$

Buffer Tree's Drawbacks

- Bad query time
- Buffer of size M is unrealistic
- Large buffers (even $\ll M$) is still unrealistic (reading entire buffer causes thrashing, e.g.)

Buffer Size Selection: Subtle

- Bender and Kuszmaul's XLDB'12 tutorial says

Buffer Size Selection: Subtle

- Bender and Kuszmaul's XLDB'12 tutorial says
  - Buffer stored on consecutive disk pages is *much* better

Buffer Size Selection: Subtle

- Bender and Kuszmaul's XLDB'12 tutorial says
  - Buffer stored on consecutive disk pages is *much* better
  - Sequential page accesses run $10\times$ faster than random page accesses (pay seektime once) Berkeley DB and InnoDB (2kB, 4kB) are two small

Buffer Size Selection: Subtle

- Bender and Kuszmaul's XLDB'12 tutorial says
  - Buffer stored on consecutive disk pages is *much* better
  - Sequential page accesses run $10\times$ faster than random page accesses (pay seektime once) Berkeley DB and InnoDB (2kB, 4kB) are two small
- In vanilla B+tree

Buffer Size Selection: Subtle

- Bender and Kuszmaul's XLDB'12 tutorial says
  - Buffer stored on consecutive disk pages is *much* better
  - Sequential page accesses run $10\times$ faster than random page accesses (pay seektime once) Berkeley DB and InnoDB (2kB, 4kB) are two small
- In vanilla B+tree
  - Small inode $\Longrightarrow$ less write amplification

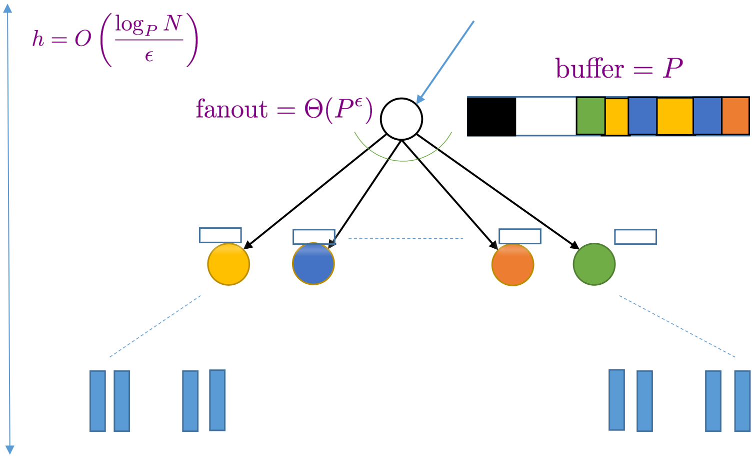
Buffer Size Selection: Subtle

- Bender and Kuszmaul's XLDB'12 tutorial says
  - Buffer stored on consecutive disk pages is *much* better
  - Sequential page accesses run $10\times$ faster than random page accesses (pay seektime once) Berkeley DB and InnoDB (2kB, 4kB) are two small
- In vanilla B+tree
  - Small inode $\Longrightarrow$ less write amplification
  - Large inode $\Longrightarrow$ faster point query

Buffer Size Selection: Subtle

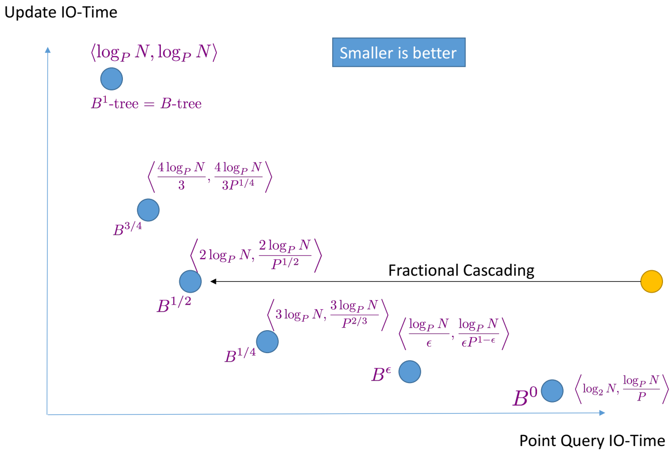
- Bender and Kuszmaul's XLDB'12 tutorial says
  - Buffer stored on consecutive disk pages is *much* better
  - Sequential page accesses run $10\times$ faster than random page accesses (pay seektime once) Berkeley DB and InnoDB (2kB, 4kB) are two small
- In vanilla B+tree
  - Small inode $\Longrightarrow$ less write amplification
  - Large inode $\Longrightarrow$ faster point query
- For now, we analyze buffer tree for buffer size P

B^ϵ-Tree



Amortized tradeoff: $\left\langle \frac{\log_P N}{\epsilon}, \frac{\log_P N}{\epsilon P^{1-\epsilon}} \right\rangle$

Summary



- LSM-Tree / Cascading tree: $\left\langle (\log_P N)^2, \frac{\log_P N}{P^{1/2}} \right\rangle$
- *Fractional cascading* removes $\log_P N$ factor

Introducing α

- B^1-tree: $\langle \log_P N, \log_P N \rangle = \langle \mathrm{opt}_q, P \cdot \mathrm{opt}_u \rangle$ (= B+tree)

Introducing α

- B^1-tree: $\langle \log_P N, \log_P N \rangle = \langle \text{opt}_q, P \cdot \text{opt}_u \rangle$ (= B+tree)
- B^0-tree: $\left\langle \log_2 N, \frac{\log_2 N}{P} \right\rangle = \langle \log_2 P \cdot \text{opt}_q, \text{opt}_u \rangle$

Introducing α

- B^1-tree: $\langle \log_P N, \log_P N \rangle = \langle \mathrm{opt}_q, P \cdot \mathrm{opt}_u \rangle$ (= B+tree)
- B^0-tree: $\left\langle \log_2 N, \frac{\log_2 N}{P} \right\rangle = \langle \log_2 P \cdot \mathrm{opt}_q, \mathrm{opt}_u \rangle$
- α-tree: $\langle (1+\alpha)\mathrm{opt}_q, \text{what?} \rangle = \langle (1+\alpha)\mathrm{opt}_q, f(\alpha) \cdot \mathrm{opt}_u \rangle$

Introducing α

- B^1-tree: $\langle \log_P N, \log_P N \rangle = \langle \text{opt}_q, P \cdot \text{opt}_u \rangle$ (= B+tree)
- B^0-tree: $\left\langle \log_2 N, \frac{\log_2 N}{P} \right\rangle = \langle \log_2 P \cdot \text{opt}_q, \text{opt}_u \rangle$
- α-tree: $\langle (1+\alpha)\text{opt}_q, \text{what?} \rangle = \langle (1+\alpha)\text{opt}_q, f(\alpha) \cdot \text{opt}_u \rangle$
  - More precise analysis

Introducing α

- B^1-tree: $\langle \log_P N, \log_P N \rangle = \langle \mathrm{opt}_q, P \cdot \mathrm{opt}_u \rangle$ (= B+tree)
- B^0-tree: $\left\langle \log_2 N, \frac{\log_2 N}{P} \right\rangle = \langle \log_2 P \cdot \mathrm{opt}_q, \mathrm{opt}_u \rangle$
- α-tree: $\langle (1+\alpha)\mathrm{opt}_q, \text{what?} \rangle = \langle (1+\alpha)\mathrm{opt}_q, f(\alpha) \cdot \mathrm{opt}_u \rangle$
  - More precise analysis
  - Key occupies k words (k is the key's arity)

Introducing α

- B^1-tree: $\langle \log_P N, \log_P N \rangle = \langle \mathrm{opt}_q, P \cdot \mathrm{opt}_u \rangle$ (= B+tree)
- B^0-tree: $\left\langle \log_2 N, \frac{\log_2 N}{P} \right\rangle = \langle \log_2 P \cdot \mathrm{opt}_q, \mathrm{opt}_u \rangle$
- α-tree: $\langle (1+\alpha)\mathrm{opt}_q, \text{what?} \rangle = \langle (1+\alpha)\mathrm{opt}_q, f(\alpha) \cdot \mathrm{opt}_u \rangle$
  - More precise analysis
  - Key occupies k words (k is the key's arity)
  - Each value occupies v words

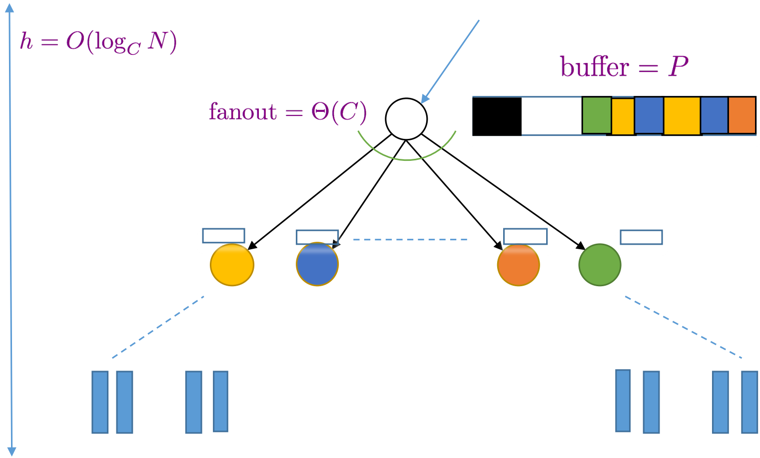
Introducing α

- B^1-tree: $\langle \log_P N, \log_P N \rangle = \langle \mathrm{opt}_q, P \cdot \mathrm{opt}_u \rangle$ (= B+tree)
- B^0-tree: $\left\langle \log_2 N, \dfrac{\log_2 N}{P} \right\rangle = \langle \log_2 P \cdot \mathrm{opt}_q, \mathrm{opt}_u \rangle$
- α-tree: $\langle (1+\alpha)\mathrm{opt}_q, \text{what?} \rangle = \langle (1+\alpha)\mathrm{opt}_q, f(\alpha) \cdot \mathrm{opt}_u \rangle$
  - More precise analysis
  - Key occupies k words (k is the key's arity)
  - Each value occupies v words
  - Child pointer occupies 1 word

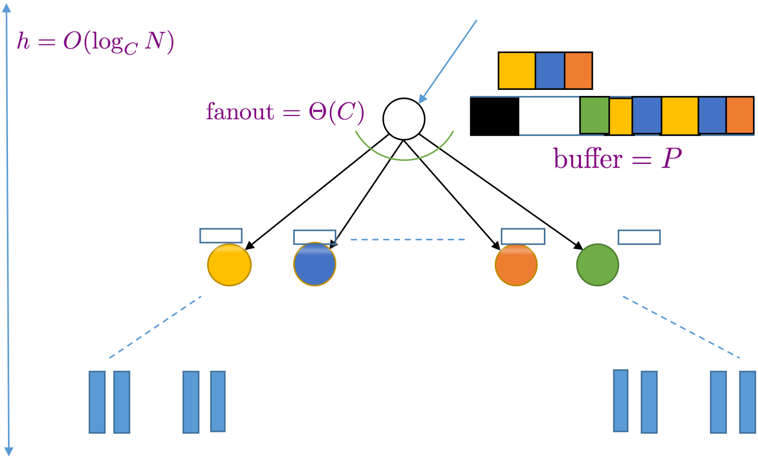
Introducing α

- B^1-tree: $\langle \log_P N, \log_P N \rangle = \langle \mathrm{opt}_q, P \cdot \mathrm{opt}_u \rangle$ (= B+tree)
- B^0-tree: $\left\langle \log_2 N, \dfrac{\log_2 N}{P} \right\rangle = \langle \log_2 P \cdot \mathrm{opt}_q, \mathrm{opt}_u \rangle$
- α-tree: $\langle (1+\alpha)\mathrm{opt}_q, \text{what?} \rangle = \langle (1+\alpha)\mathrm{opt}_q, f(\alpha) \cdot \mathrm{opt}_u \rangle$
  - More precise analysis
  - Key occupies k words (k is the key's arity)
  - Each value occupies v words
  - Child pointer occupies 1 word
  - Compression ratio (a nasty *random variable*)

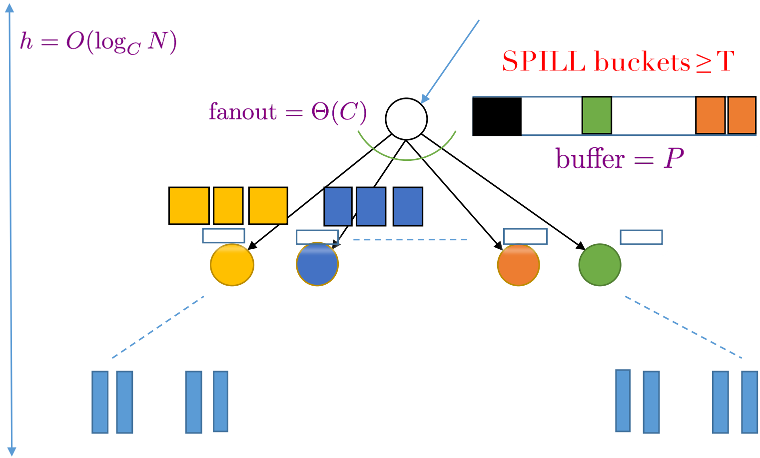
Amortized Analysis of Buffer Tree



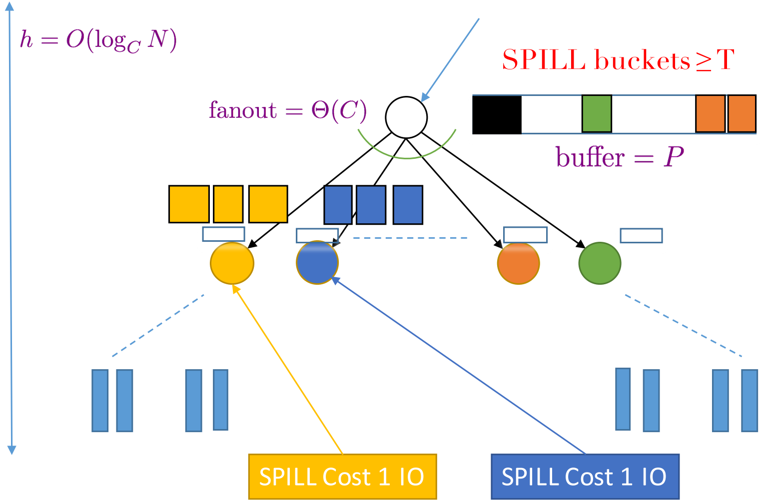
Have to SPILL



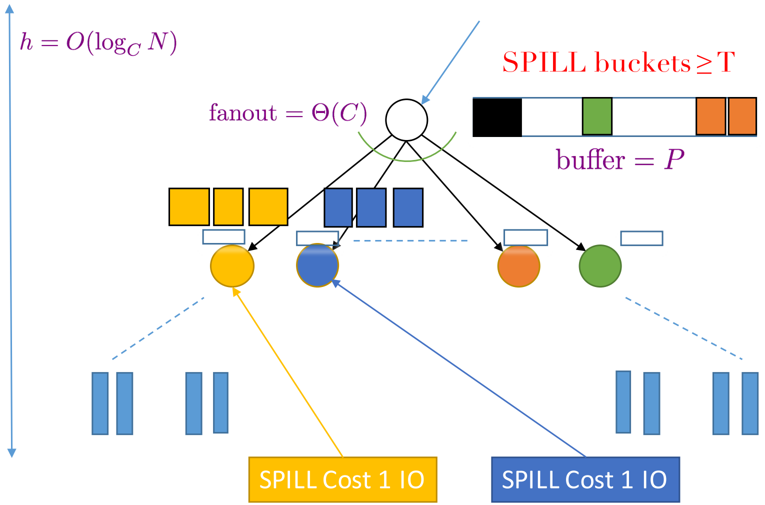
T = SPILL Threshold



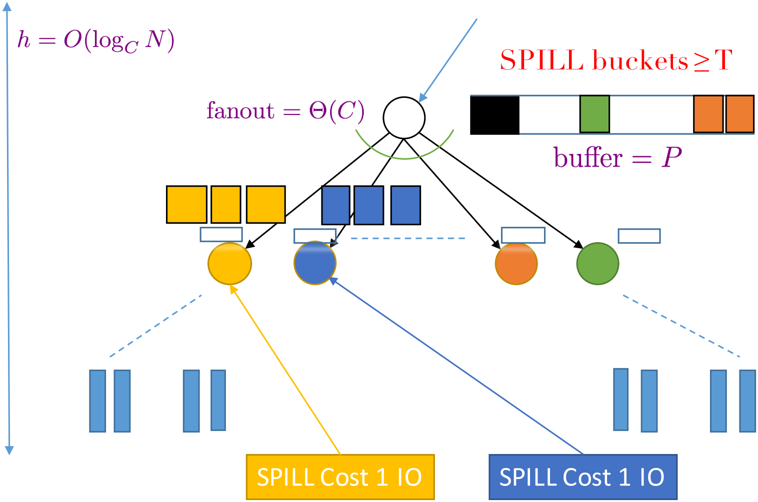
Each SPILL Bucket Imposes 1 IO



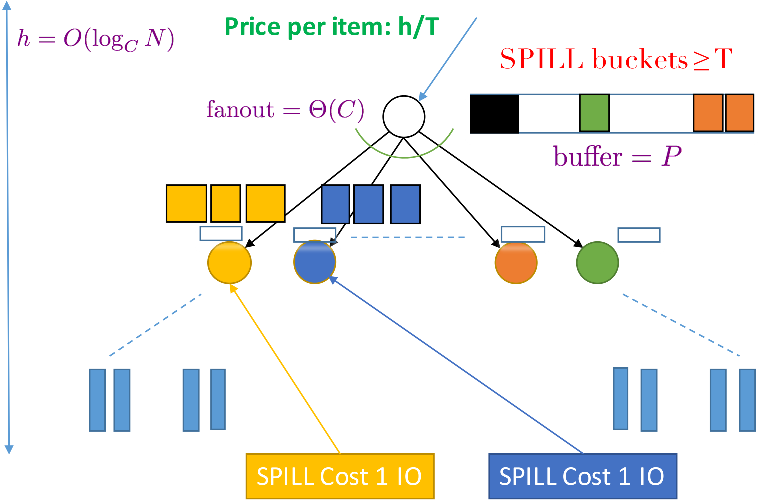
Who Pay for Those IOs?



Assign Each Item $\frac{1}{T}$ Credits

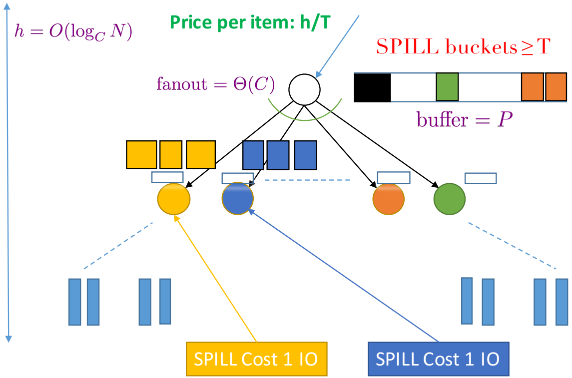


Tradeoff point: $\left\langle h, \frac{h}{T} \right\rangle = \boxed{\left\langle \log_C N, \frac{\log_C N}{T} \right\rangle}$



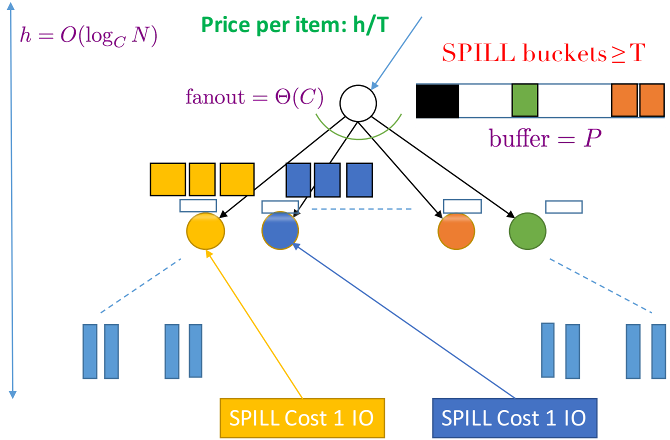
Tradeoff point: $\left\langle h, \frac{h}{T} \right\rangle = \boxed{\left\langle \log_C N, \frac{\log_C N}{T} \right\rangle}$

How Do We Set T?



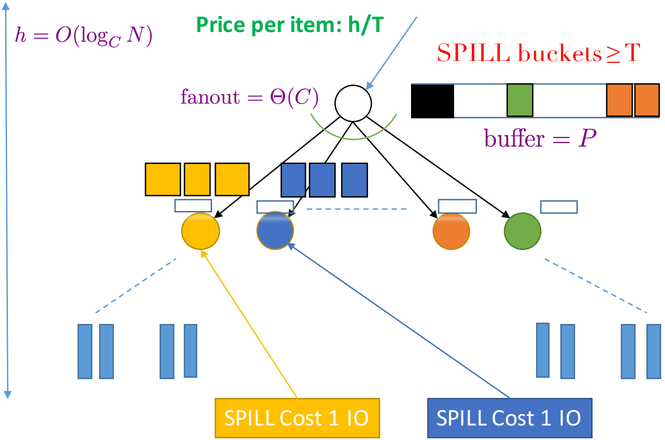
Tradeoff point: $\left\langle h, \frac{h}{T} \right\rangle = \boxed{\left\langle \log_C N, \frac{\log_C N}{T} \right\rangle}$

- At least one bucket spilled



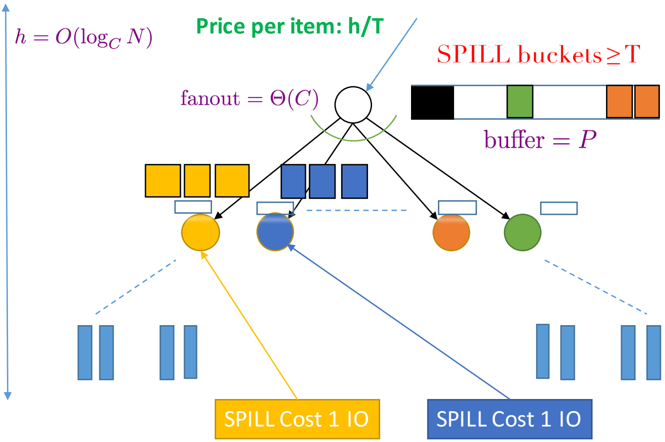
Tradeoff point: $\left\langle h, \frac{h}{T} \right\rangle = \left\langle \log_C N, \frac{\log_C N}{T} \right\rangle$

- At least one bucket spilled
- Buffer no longer overflown



Tradeoff point: $\left\langle h, \frac{h}{T} \right\rangle = \left\langle \log_C N, \frac{\log_C N}{T} \right\rangle$

E.g., in the simplest case $T = \frac{P - C}{C}$



Precision Complicates opt_q and opt_u

- α-tree: $\langle (1+\alpha)\mathrm{opt}_q, \text{what?} \rangle = \langle (1+\alpha)\mathrm{opt}_q, f(\alpha) \cdot \mathrm{opt}_u \rangle$

Precision Complicates opt_q and opt_u

- α-tree: $\langle (1+\alpha)\mathrm{opt}_q, \text{what?} \rangle = \langle (1+\alpha)\mathrm{opt}_q, f(\alpha)\cdot \mathrm{opt}_u \rangle$
- Let $r = k + 1$, number of words for child tuple

Precision Complicates opt_q and opt_u

- α-tree: $\langle (1+\alpha)\mathrm{opt}_q, \text{what?} \rangle = \langle (1+\alpha)\mathrm{opt}_q, f(\alpha)\cdot \mathrm{opt}_u \rangle$
- Let $r = k + 1$, number of words for child tuple
- Let $d = k + v(+1)$, number of words for item (delta)

Precision Complicates opt_q and opt_u

- α-tree: $\langle (1+\alpha)\text{opt}_q, \text{what?} \rangle = \langle (1+\alpha)\text{opt}_q, f(\alpha) \cdot \text{opt}_u \rangle$
- Let $r = k + 1$, number of words for child tuple
- Let $d = k + v(+1)$, number of words for item (delta)
- Space constraint $rC + dB \leq P$

Precision Complicates opt_q and opt_u

- α-tree: $\langle (1+\alpha)\mathrm{opt}_q, \text{what?} \rangle = \langle (1+\alpha)\mathrm{opt}_q, f(\alpha) \cdot \mathrm{opt}_u \rangle$
- Let $r = k + 1$, number of words for child tuple
- Let $d = k + v(+1)$, number of words for item (delta)
- Space constraint $rC + dB \leq P$
- B^1-tree sets $C = P/r$,

$$\mathrm{opt}_q = \log_{P/r} N$$

Precision Complicates opt_q and opt_u

- α-tree: $\langle (1+\alpha)\text{opt}_q, \text{what?} \rangle = \langle (1+\alpha)\text{opt}_q, f(\alpha) \cdot \text{opt}_u \rangle$
- Let $r = k + 1$, number of words for child tuple
- Let $d = k + v(+1)$, number of words for item (delta)
- Space constraint $rC + dB \leq P$
- B^1-tree sets $C = P/r$,

$$\text{opt}_q = \log_{P/r} N$$

- B^0-tree sets $C = 2$, $B = (P - 2r)/d \approx P/d$,

$$\text{opt}_u = 2d \cdot \frac{\log_2 N}{P}$$

(larger items increase update time!)

α-tree: The No-Compression Case

- $rC + dB \approx P$,
- $C = \left\lceil (P/r)^{1/(1+\alpha)} \right\rceil$
- $h \approx \log_C N = (1+\alpha)\mathrm{opt}_q$
- $T = \dfrac{B}{C} \geq \left\lfloor \dfrac{P - rC}{Cd} \right\rfloor \approx \dfrac{P}{dC} \approx \dfrac{1}{d} \cdot \left(\dfrac{r}{P}\right)^{1/(1+\alpha)} \cdot P$
- Tradeoff: $\left\langle (1+\alpha) \cdot t_q^{\text{B+tree}}, \dfrac{(1+\alpha)d}{(rP^\alpha)^{\frac{1}{1+\alpha}}} \cdot t_u^{\text{B+tree}} \right\rangle$
- Tradeoff: $\left\langle (1+\alpha) \cdot \mathrm{opt}_q, \dfrac{1+\alpha}{2\log_2(P/r)} \left(\dfrac{P}{r}\right)^{\frac{1}{1+\alpha}} \cdot \mathrm{opt}_u \right\rangle$

Minor Note

- Note that $\text{opt}_u = 2d \dfrac{\log_{P/r} N}{P/\log_2(P/r)} = \dfrac{2d \cdot \log_2(P/r)}{P} \cdot t_u^{\text{B+tree}}$
- So, we can also compare directly with B+tree:

$$\langle (1+\alpha)\text{opt}_q, \text{what?} \rangle = \left\langle (1+\alpha) t_q^{\text{B+tree}}, g(\alpha) \cdot t_u^{\text{B+tree}} \right\rangle$$

- noting that $f(\alpha) = g(\alpha) \cdot \dfrac{P}{2d \cdot \log_2(P/r)}$

Compression Complicates Matters Further

- Compression ratio (CR) is a random variable

Compression Complicates Matters Further

- Compression ratio (CR) is a random variable
  - data-dependent

Compression Complicates Matters Further

- Compression ratio (CR) is a random variable
  - data-dependent
  - does not scale linearly with the input

Compression Complicates Matters Further

- Compression ratio (CR) is a random variable
  - data-dependent
  - does not scale linearly with the input
  - not even monotonic in input size

Compression Complicates Matters Further

- Compression ratio (CR) is a random variable
  - data-dependent
  - does not scale linearly with the input
  - not even monotonic in input size
  - changes from inode to inode

Compression Complicates Matters Further

- Compression ratio (CR) is a random variable
  - data-dependent
  - does not scale linearly with the input
  - not even monotonic in input size
  - changes from inode to inode
  - changes within an inode over time

Compression Complicates Matters Further

- Compression ratio (CR) is a random variable
  - data-dependent
  - does not scale linearly with the input
  - not even monotonic in input size
  - changes from inode to inode
  - changes within an inode over time
- Key's CR different from value's CR

Compression Complicates Matters Further

- Compression ratio (CR) is a random variable
  - data-dependent
  - does not scale linearly with the input
  - not even monotonic in input size
  - changes from inode to inode
  - changes within an inode over time
- Key's CR different from value's CR
  - Conjecture: keys compress better than values. (Same keys follow same path from the root)

Compression Complicates Matters Further

- Compression ratio (CR) is a random variable
  - data-dependent
  - does not scale linearly with the input
  - not even monotonic in input size
  - changes from inode to inode
  - changes within an inode over time
- Key's CR different from value's CR
  - Conjecture: keys compress better than values. (Same keys follow same path from the root)
  - Hunch: children's keys and item' keys should be compressed together

Compression Complicates Matters Further

- Compression ratio (CR) is a random variable
  - data-dependent
  - does not scale linearly with the input
  - not even monotonic in input size
  - changes from inode to inode
  - changes within an inode over time
- Key's CR different from value's CR
  - Conjecture: keys compress better than values. (Same keys follow same path from the root)
  - Hunch: children's keys and item' keys should be compressed together
- $\mathrm{opt}_q, \mathrm{opt}_u$ are also random variables!

Our Approach

- Make over-simplifying assumptions for analysis

Our Approach

- Make over-simplifying assumptions for analysis
- Use analysis to drive an implementation heuristic

Our Approach

- Make over-simplifying assumptions for analysis
- Use analysis to drive an implementation heuristic
- Experiment with this heuristic, see if analysis holds up

Over-Simplifying Assumptions

- Suppose compression ratio is a constant x

Over-Simplifying Assumptions

- Suppose compression ratio is a constant x
- What would a tree with opt_q do?

Over-Simplifying Assumptions

- Suppose compression ratio is a constant x
- What would a tree with opt_q do?
  - Simple: set $C = x \cdot (P/r)$, no item buffer

$$\mathrm{opt}_q(x) = \log_{xP/r} P = \frac{\log_{P/r} N}{1 + \log_{P/r} x} = \frac{\mathrm{opt}_q(1)}{1 + \log_{P/r} x}$$

Over-Simplifying Assumptions

- Suppose compression ratio is a constant x
- What would a tree with opt_q do?
  - Simple: set $C = x \cdot (P/r)$, no item buffer

$$\mathrm{opt}_q(x) = \log_{xP/r} P = \frac{\log_{P/r} N}{1 + \log_{P/r} x} = \frac{\mathrm{opt}_q(1)}{1 + \log_{P/r} x}$$

  - $\mathrm{opt}_q(1)$ is opt_q *without* compression

Over-Simplifying Assumptions

- Suppose compression ratio is a constant x
- What would a tree with opt_q do?
  - Simple: set $C = x \cdot (P/r)$, no item buffer

$$\mathrm{opt}_q(x) = \log_{xP/r} P = \frac{\log_{P/r} N}{1 + \log_{P/r} x} = \frac{\mathrm{opt}_q(1)}{1 + \log_{P/r} x}$$

  - $\mathrm{opt}_q(1)$ is opt_q *without* compression
- What would a tree with opt_u do?

Over-Simplifying Assumptions

- Suppose compression ratio is a constant x
- What would a tree with opt_q do?
  - Simple: set $C = x \cdot (P/r)$, no item buffer

$$\mathrm{opt}_q(x) = \log_{xP/r} P = \frac{\log_{P/r} N}{1 + \log_{P/r} x} = \frac{\mathrm{opt}_q(1)}{1 + \log_{P/r} x}$$

  - $\mathrm{opt}_q(1)$ is opt_q *without* compression
- What would a tree with opt_u do?
  - Simple: set $C = 2$, and give the rest to item buffers

$$\mathrm{opt}_u(x) = 2dx \cdot \frac{\log_2 N}{P} = \frac{\mathrm{opt}_u(1)}{x}.$$

Over-Simplifying Assumptions

- Suppose compression ratio is a constant x
- What would a tree with opt_q do?
  - Simple: set $C = x \cdot (P/r)$, no item buffer

$$\mathrm{opt}_q(x) = \log_{xP/r} P = \frac{\log_{P/r} N}{1 + \log_{P/r} x} = \frac{\mathrm{opt}_q(1)}{1 + \log_{P/r} x}$$

  - $\mathrm{opt}_q(1)$ is opt_q *without* compression
- What would a tree with opt_u do?
  - Simple: set $C = 2$, and give the rest to item buffers

$$\mathrm{opt}_u(x) = 2dx \cdot \frac{\log_2 N}{P} = \frac{\mathrm{opt}_u(1)}{x}.$$

  - $\mathrm{opt}_u(1)$ is opt_u *without* compression

What We are Up Against?

- A non-existent enemy

What We are Up Against?

- A non-existent enemy
  - whose tradeoff point is

$$\langle \mathrm{opt}_q(x), \mathrm{opt}_u(x) \rangle$$

What We are Up Against?

- A non-existent enemy
  - whose tradeoff point is

$$\boxed{\langle \mathrm{opt}_q(x), \mathrm{opt}_u(x) \rangle}$$

  - and, whose parameter x is unknown

What We are Up Against?

- A non-existent enemy
  - whose tradeoff point is

$$\langle \mathrm{opt}_q(x), \mathrm{opt}_u(x) \rangle$$

  - and, whose parameter x is unknown
- To compete against it, we ...

What We are Up Against?

- A non-existent enemy
  - whose tradeoff point is

$$\boxed{\langle \mathrm{opt}_q(x), \mathrm{opt}_u(x) \rangle}$$

  - and, whose parameter x is unknown
- To compete against it, we ...
  - First assume that we know x

What We are Up Against?

- A non-existent enemy
  - whose tradeoff point is

$$\langle \mathrm{opt}_q(x), \mathrm{opt}_u(x) \rangle$$

  - and, whose parameter x is unknown
- To compete against it, we ...
  - First assume that we know x
  - Then, note that

$$C = C(x) = \left\lceil (xP/r)^{1/(1+\alpha)} \right\rceil \Longrightarrow h \leq (1+\alpha)\mathrm{opt}_q(x)$$

What We are Up Against?

- A non-existent enemy
  - whose tradeoff point is

$$\langle \mathrm{opt}_q(x), \mathrm{opt}_u(x) \rangle$$

  - and, whose parameter x is unknown
- To compete against it, we ...
  - First assume that we know x
  - Then, note that

$$C = C(x) = \left\lceil (xP/r)^{1/(1+\alpha)} \right\rceil \Longrightarrow h \leq (1+\alpha)\mathrm{opt}_q(x)$$

  - Try not to waste inode's buffer space

Heuristic

- (Conservatively) Estimate x (e.g. $\max\{1, \bar{x} - \bar{\sigma}_x\}$)
- Use it to compute max fanout $C = C(x)$
- Give free space in buffer to items
- When full, spill all buckets $\geq T$
  - $T = \frac{\text{\# items}}{\text{\#current children}}, \quad T = \frac{\text{\# items}}{C}$
  - $T = \frac{B(x)}{C(x)}, \qquad T = \frac{B(1)}{C(1)}$

Main Worries and Resolutions

Worries

- Items leave no space for fanout to grow: $\langle t_q \nearrow, t_u \searrow \rangle$

Solution?

Main Worries and Resolutions

Worries

- Items leave no space for fanout to grow: $\langle t_q \nearrow, t_u \searrow \rangle$
- Under-estimate x, fanout too low: $\langle t_q \nearrow, t_u \searrow \rangle$

Solution?

Main Worries and Resolutions

Worries

- Items leave no space for fanout to grow: $\langle t_q \nearrow, t_u \searrow \rangle$
- Under-estimate x, fanout too low: $\langle t_q \nearrow, t_u \searrow \rangle$
- Over-estimate x, fanout too high: $\langle t_q \searrow, t_u \nearrow \rangle$

Solution?

Main Worries and Resolutions

Worries

- Items leave no space for fanout to grow: $\langle t_q \nearrow, t_u \searrow \rangle$
- Under-estimate x, fanout too low: $\langle t_q \nearrow, t_u \searrow \rangle$
- Over-estimate x, fanout too high: $\langle t_q \searrow, t_u \nearrow \rangle$
- Loss of compression triggers multiple spill rounds

Solution?

Main Worries and Resolutions

Worries

- Items leave no space for fanout to grow: $\langle t_q \nearrow, t_u \searrow \rangle$
- Under-estimate x, fanout too low: $\langle t_q \nearrow, t_u \searrow \rangle$
- Over-estimate x, fanout too high: $\langle t_q \searrow, t_u \nearrow \rangle$
- Loss of compression triggers multiple spill rounds
  - Might be OK theoretically, but practice much more complicated

Solution?

Main Worries and Resolutions

Worries

- Items leave no space for fanout to grow: $\langle t_q \nearrow, t_u \searrow \rangle$
- Under-estimate x, fanout too low: $\langle t_q \nearrow, t_u \searrow \rangle$
- Over-estimate x, fanout too high: $\langle t_q \searrow, t_u \nearrow \rangle$
- Loss of compression triggers multiple spill rounds
  - Might be OK theoretically, but practice much more complicated

Solution?

- Do not allow C to get too low. Perhaps $C \geq \dfrac{C(1)}{4}$

Main Worries and Resolutions

Worries

- Items leave no space for fanout to grow: $\langle t_q \nearrow, t_u \searrow \rangle$
- Under-estimate x, fanout too low: $\langle t_q \nearrow, t_u \searrow \rangle$
- Over-estimate x, fanout too high: $\langle t_q \searrow, t_u \nearrow \rangle$
- Loss of compression triggers multiple spill rounds
  - Might be OK theoretically, but practice much more complicated

Solution?

- Do not allow C to get too low. Perhaps $C \geq \dfrac{C(1)}{4}$
- Do not allow T to get too low. Perhaps $T \geq \dfrac{B(1)}{4C(1)}$, requiring $C \leq 4C(1)$

Question?

Precision Changes opt_q and opt_u

- Let $r = k + 1$, number of words for child tuple
- Let $d = k + v(+1)$, number of words for item (delta)
- Space constraint $rC + dB = P$
- B+tree sets $d = 0$, so $C = P/r$

$$\text{opt}_q = \log_{P/r} N.$$

- B^0-tree sets $C = 2$, $B = (P - 2r)/d \approx P/d$.

$$\text{opt}_u \approx \frac{\text{tree height}}{\text{spill size}} = \frac{\log_2 N}{P/(2d)} = 2d \cdot \frac{\log_2 N}{P}$$

(*larger items increase update time!*)

α-tree: The No-Compression Case

- For a fixed C, B such that $rC + dB = P$,
- $h \approx \log_C N = (1+\alpha)\mathrm{opt}_q$ requires $C = \left\lceil (P/r)^{1/(1+\alpha)} \right\rceil$
- Each spill handles at least this many items

$$\frac{B}{C} \geq \left\lfloor \frac{P - rC}{Cd} \right\rfloor \approx \frac{P}{dC} \approx \frac{1}{d} \cdot \left(\frac{r}{P}\right)^{1/(1+\alpha)} \cdot P$$

- Relative to B+tree: $\left\langle (1+\alpha) \cdot t_q^{\text{B+tree}}, \frac{(1+\alpha)d}{(rP^\alpha)^{\frac{1}{1+\alpha}}} \cdot t_u^{\text{B+tree}} \right\rangle$
- In absolute terms:

$$\left\langle (1+\alpha) \cdot \mathrm{opt}_q, \frac{1+\alpha}{2\log_2(P/r)} \left(\frac{P}{r}\right)^{\frac{1}{1+\alpha}} \cdot \mathrm{opt}_u \right\rangle$$