Designing Access Methods: The RUM Conjecture

Manos Athanassoulis\textsuperscript{1}, Michael S. Kester\textsuperscript{1}, Lukas M. Maas\textsuperscript{1}, Radu Stoica\textsuperscript{2}, Stratos Idreos\textsuperscript{1}, Anastasia Ailamaki\textsuperscript{3}, Mark Callaghan\textsuperscript{4}

\textsuperscript{1}Harvard University \quad \textsuperscript{2}IBM Research \quad \textsuperscript{3}EPFL \quad \textsuperscript{4}Facebook Inc.
memory wall

storage wall
Memory Wall

Performance vs. Time

- CPU: ~20-25% perf. increase annually
- DRAM: ~2-11% perf. increase annually
Memory Wall

every byte counts
Storage Wall

HDD
- capacity ✓
- sequential access ✓
- random access ×
- latency plateaus ×

SSD (Single Level Cell)
- random reads ✓
- low latency ✓
- capacity ×
- endurance ×
- read/write asymmetry ×

SSD (Multi Level Cell)
- capacity ✓
- endurance (worse) ×

HDD (Shingled Magnetic Rec.)
- capacity ✓
- read/write asymmetry ×

every byte counts
memory wall

every byte we \textit{read} counts

every byte we \textit{allocate} counts

storage wall

every byte we \textit{read} counts

every byte we \textit{update} counts

every byte we \textit{allocate} counts
in this fluid environment ...
we build
access methods
... since a long time ago!

Can we stop worrying about building access methods?
why do we keep building?

every access method

... is optimizing for the tradeoff between

Reads

Updates

Memory

this balance forms a three-way tug of war
The RUM Conjecture

every access method has a (quantifiable)
  • read overhead
  • update overhead
  • memory overhead

the three of which form a competing triangle

we can optimize for two of the overheads at the expense of the third
The RUM Conjecture

every access method has a (quantifiable)
  • read overhead
  • update overhead
  • memory overhead

the three of which form a competing triangle

we can optimize for two of the overheads at the expense of the third
The RUM Conjecture

every access method has a (quantifiable)
  • read overhead
  • update overhead
  • memory overhead

the three of which form a competing triangle

we can optimize for two of
the overheads at the
expense of the third
The RUM Conjecture

every access method has a (quantifiable)
  • read overhead
  • update overhead
  • memory overhead

the three of which form a competing triangle

we can optimize for two of the overheads at the expense of the third
what would be an **optimal read** behavior?

"read(x)" accesses only the bytes of object X
what would be an optimal read behavior?

read(x) accesses only the bytes of object X

how free can an oracle be?
what would be an **optimal read** behavior?

unique, positive integers stored in the location equal to their value
what would be an optimal read behavior?

minimum read overhead
what would be an **optimal read** behavior?

1 2 3 5

insert 2  delete 8  update 4 -> 3

**minimum read overhead**

**bound update overhead**
what would be an **optimal read** behavior?

minimum read overhead

bound update overhead

unbounded memory overhead
what would be an optimal update behavior?

always append, and invalidate on update

update (X) changes the minimal number of bytes

what about reads? what about more data?
what would be an **optimal update** behavior?

always *append*, and *invalidate* on update

*update* \((X)\) changes the minimal number of bytes

higher read and memory overhead
what would be an **optimal memory** overhead?

*no metadata whatsoever,* would result in the smallest memory footprint

---

*scan and in-place updates*

---

**do we need to reach the optimal(s)?**
what would be an **optimal memory** overhead?

*no metadata whatsoever, would result in the smallest memory footprint*

*scan and in-place updates*

No!

do we need to reach the optimal(s)?
RUM Conjecture: an example

a tight column: 8 2 1 7 6 9 3

- reads have to scan
- no memory overhead
- in-place updates and efficient inserts
RUM Conjecture: an example

A tight column:

- reads have to scan
- no memory overhead
- in-place updates and efficient inserts

A tight sorted column:

- very efficient reads (logarithmic search)
- no memory overhead
- updates & inserts reorganization
RUM Conjecture: an example

a tight column: 8217693
• reads have to scan
• no memory overhead
• in-place updates and efficient inserts

adding clustering:
• efficient reads
• small memory overhead
• updates & inserts require reorganization

a tight sorted column: 1236789
• very efficient reads (logarithmic search)
• no memory overhead
• updates & inserts require reorganization
RUM Conjecture: an example

a tight column:

- reads have to scan
- no memory overhead
- in-place updates and efficient inserts

adding clustering:

- efficient reads
- small memory overhead
- updates & inserts reorganization

a tight sorted column:

- very efficient reads (logarithmic search)
- no memory overhead
- updates & inserts reorganization

... and ghost values:

- efficient reads
- small memory overhead (but increased)
- updates: reorganization (but inserts for free)
RUM-aware access methods

Can we

... add **flexibility** to existing access methods?

... have **arbitrary** RUM balance?

... have **dynamic** RUM balance?
some active research directions

✓ classify existing access methods [Tutorial - SIGMOD2016]

✓ add more metadata, to optimize for updates/read
  ○ Bitmap indexing [SIGMOD2016] and LSM-Trees in the works

✓ pre-partition data: minimize both read and update cost

✓ shape-shifting index: match workload without offline optimizations
Thanks!

http://daslab.seas.harvard.edu/rum/