Competitive Data-Structure Dynamization

— SODA 2021 —

(an 11-minute talk summarizing the conference paper)

Claire Mathieu

CNRS, Paris

Rajmohan Rajaraman

Northeastern University

Neal E. Young

University of California Riverside

Northeastern University

Arman Yousefi

Google

— research funded by NSF and Google

compaction policies for LSM (log-structured merge) systems
through the lens of competitive analysis

MOTIVATION

B-TREES VS. MOORE'S LAW

LSM-SYSTEM COMPACTION VIA DATA-STRUCTURE DYNAMIZATION

DEFINITIONS

PROBLEM 1 — *MIN-SUM DYNAMIZATION*

PROBLEM 2 — *K-COMPONENT DYNAMIZATION*

COMPETITIVE ANALYSIS

RESULTS

PROBLEM 1 ALGORITHM, $\Theta(\log^* n)$-COMPETITIVE

PROBLEM 2 LOWER BOUND

PROBLEM 2 ALGORITHMS, K-COMPETITIVE

external-memory ordered dictionaries: Better than B-trees?

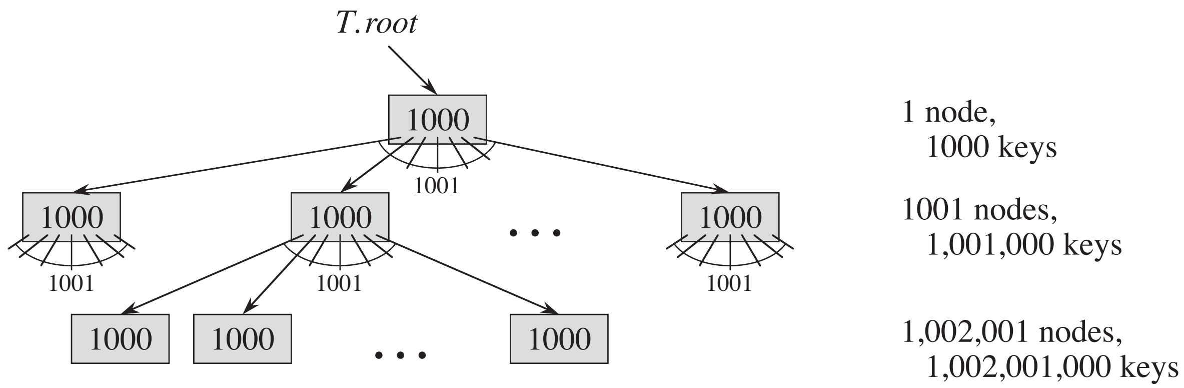
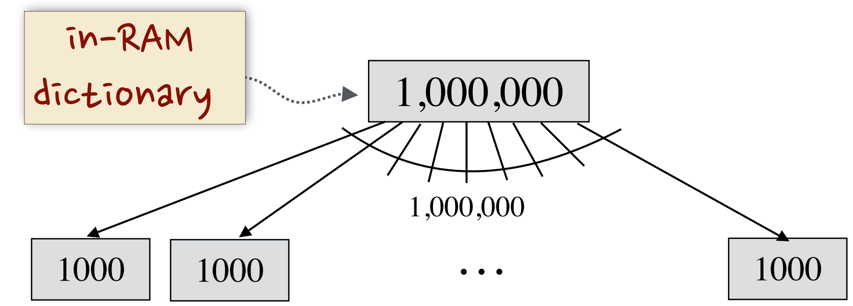


Figure 18.3 A B-tree of height 2 containing over one billion keys. Shown inside each node x is $x.n$, the number of keys in x. Each internal node and leaf contains 1000 keys. This B-tree has 1001 nodes at depth 1 and over one million leaves at depth 2.

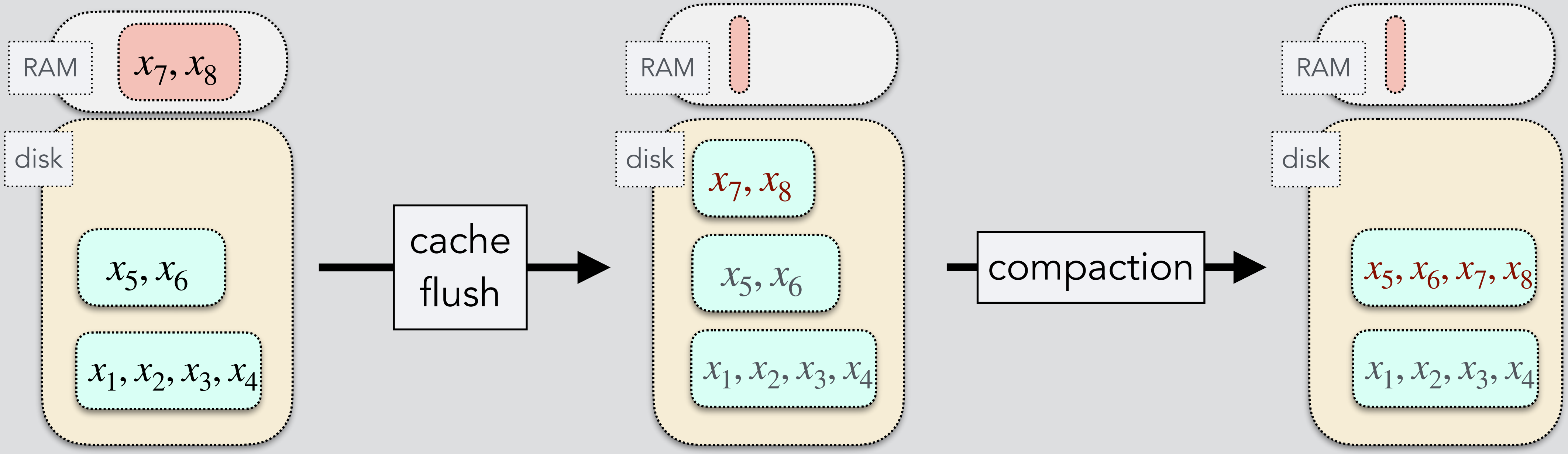


Contract the non-leaf nodes into a single mega-root, *held in RAM* and implemented as a RAM-based ordered dictionary.

As suggested in Figure 18.3 from *Introduction to Algorithms* by CLRS

But THIS way achieves about 1 disk access per insert or query.

1. B-tree node degree ≈ number of keys that can be fetched from disk in the twice the disk-access time
2. in 2020 → can fetch *thousands* of keys in twice the disk access time → node degree should be over 1000
3. **in 2020 → non-leaf nodes take up < 0.1% of the total space**
4. **Database servers are typically configured so that RAM size is 1–3% of disk size [31, p. 227] !**
5. Can easily hold all non-leaf nodes in (10% of) RAM, and replace them with in-RAM dictionary.
6. **Doing this achieves about 1 disk access per insertion or query.**
7. **Is it possible to get *less* than one disk access per insertion or query?**



external-memory dictionaries via *LSM = "log-structured merge"* [O'Neil et al 1996, and others]

1. INSERTs are cached in RAM, require no disk access
2. periodically flush RAM cache to disk in a single batch
3. maintain on-disk items in *immutable* sorted files called *components*
4. each QUERY checks the cache, then if necessary each on-disk component (one disk access per)
5. periodically *compact* — destroy some components and build new ones from scratch

   — use *data-structure dynamization algorithm* to choose which components to destroy and build

   — build cost vs query cost tradeoff

notes:

a. component builds use high-throughput sequential disk access, not slow random access

b. LSM systems are used today by most companies that need high-throughput big-data storage

c. most academic work assumes uniform batch sizes and uniform INSERT/QUERY rates,

   but these assumptions don't hold in production systems, e.g. Google Bigtable

compaction policies for LSM (log-structured merge) systems
through the lens of competitive analysis

Contents

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that
the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S\in\mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

MINIMIZE COST: $\sum_{t=1}^{n} \sum_{S\in\mathscr{C}_t\setminus\mathscr{C}_{t-1}} \mathrm{wt}(S) + \sum_{t=1}^{n} |\mathscr{C}_t|$ (build cost + query cost)

adding a new set S to the cover incurs build cost $\mathrm{wt}(S) = \sum_{x\in S} \mathrm{wt}(x)$

EXAMPLE

time	input batch	cover	query cost	build cost

we call the sets in each cover "components"

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S \in \mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

MINIMIZE COST: $\sum_{t=1}^{n} \sum_{S \in \mathscr{C}_t \setminus \mathscr{C}_{t-1}} \mathrm{wt}(S) + \sum_{t=1}^{n} |\mathscr{C}_t|$ (build cost + query cost)

adding a new set S to the cover incurs build cost $\mathrm{wt}(S) = \sum_{x \in S} \mathrm{wt}(x)$

EXAMPLE

time	input batch	cover	query cost	build cost
1				

we call the sets in each cover "components"

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S \in \mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

MINIMIZE COST: $\sum_{t=1}^{n} \sum_{S \in \mathscr{C}_t \setminus \mathscr{C}_{t-1}} \mathrm{wt}(S) + \sum_{t=1}^{n} |\mathscr{C}_t|$ (build cost + query cost)

adding a new set S to the cover incurs build cost $\mathrm{wt}(S) = \sum_{x \in S} \mathrm{wt}(x)$

EXAMPLE

time	input batch	cover	query cost	build cost
1	$\{a, b\}$			

we call the sets in each cover "components"

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that
the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S\in\mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

MINIMIZE COST: $\sum_{t=1}^{n} \sum_{S\in\mathscr{C}_t\setminus\mathscr{C}_{t-1}} \mathrm{wt}(S) + \sum_{t=1}^{n} |\mathscr{C}_t|$ (build cost + query cost)

adding a new set S to the cover incurs build cost $\mathrm{wt}(S) = \sum_{x\in S} \mathrm{wt}(x)$

EXAMPLE

time	input batch	cover	query cost	build cost
1	$\{a, b\}$	$\{a\}, \{b\}$		

we call the sets in each cover "components"

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S\in\mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

MINIMIZE COST: $\sum_{t=1}^{n} \sum_{S\in\mathscr{C}_t\setminus\mathscr{C}_{t-1}} \mathrm{wt}(S) + \sum_{t=1}^{n} |\mathscr{C}_t|$ (build cost + query cost)

adding a new set S to the cover incurs build cost $\mathrm{wt}(S) = \sum_{x\in S} \mathrm{wt}(x)$

EXAMPLE

time	input batch	cover	query cost	build cost
1	$\{a, b\}$	$\{a\}, \{b\}$	2	

we call the sets in each cover "components"

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S \in \mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

MINIMIZE COST: $\sum_{t=1}^{n} \sum_{S \in \mathscr{C}_t \setminus \mathscr{C}_{t-1}} \mathrm{wt}(S) + \sum_{t=1}^{n} |\mathscr{C}_t|$ (build cost + query cost)

adding a new set S to the cover incurs build cost $\mathrm{wt}(S) = \sum_{x \in S} \mathrm{wt}(x)$

EXAMPLE

time	input batch	cover	query cost	build cost
1	$\{a, b\}$	$\{a\}, \{b\}$	2	$\mathrm{wt}(a) + \mathrm{wt}(b)$

we call the sets in each cover "components"

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that
the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S\in\mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

MINIMIZE COST: $\sum_{t=1}^{n} \sum_{S\in\mathscr{C}_t\setminus\mathscr{C}_{t-1}} \mathrm{wt}(S) + \sum_{t=1}^{n} |\mathscr{C}_t|$ (build cost + query cost)

adding a new set S to the cover incurs build cost $\mathrm{wt}(S) = \sum_{x\in S} \mathrm{wt}(x)$

EXAMPLE

time	input batch	cover	query cost	build cost
1	$\{a, b\}$	$\{a\}, \{b\}$	2	$\mathrm{wt}(a) + \mathrm{wt}(b)$
2				

we call the sets in each cover "components"

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that
the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S\in\mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

MINIMIZE COST: $\sum_{t=1}^{n} \sum_{S\in\mathscr{C}_t\setminus\mathscr{C}_{t-1}} \mathrm{wt}(S) + \sum_{t=1}^{n} |\mathscr{C}_t|$ (build cost + query cost)

adding a new set S to the cover incurs build cost $\mathrm{wt}(S) = \sum_{x\in S} \mathrm{wt}(x)$

EXAMPLE

time	input batch	cover	query cost	build cost
1	$\{a, b\}$	$\{a\}, \{b\}$	2	$\mathrm{wt}(a) + \mathrm{wt}(b)$
2	$\{c\}$			

we call the sets in each cover "components"

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S\in\mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

MINIMIZE COST: $\sum_{t=1}^{n} \sum_{S\in\mathscr{C}_t\setminus\mathscr{C}_{t-1}} \mathrm{wt}(S) + \sum_{t=1}^{n} |\mathscr{C}_t|$ (build cost + query cost)

adding a new set S to the cover incurs build cost $\mathrm{wt}(S) = \sum_{x\in S} \mathrm{wt}(x)$

EXAMPLE

time	input batch	cover	query cost	build cost
1	$\{a, b\}$	$\{a\}, \{b\}$	2	$\mathrm{wt}(a) + \mathrm{wt}(b)$
2	$\{c\}$	$\{b\}, \{a, c\}$		

we call the sets in each cover "components"

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S \in \mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

MINIMIZE COST: $\sum_{t=1}^{n} \sum_{S \in \mathscr{C}_t \setminus \mathscr{C}_{t-1}} \mathrm{wt}(S) + \sum_{t=1}^{n} |\mathscr{C}_t|$ (build cost + query cost)

adding a new set S to the cover incurs build cost $\mathrm{wt}(S) = \sum_{x \in S} \mathrm{wt}(x)$

EXAMPLE

time	input batch	cover	query cost	build cost
1	$\{a, b\}$	$\{a\}, \{b\}$	2	$\mathrm{wt}(a) + \mathrm{wt}(b)$
2	$\{c\}$	$\{b\}, \{a, c\}$	2	

we call the sets in each cover "components"

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S\in\mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

MINIMIZE COST: $\sum_{t=1}^{n} \sum_{S\in\mathscr{C}_t\setminus\mathscr{C}_{t-1}} \mathrm{wt}(S) + \sum_{t=1}^{n} |\mathscr{C}_t|$ (build cost + query cost)

adding a new set S to the cover incurs build cost $\mathrm{wt}(S) = \sum_{x\in S} \mathrm{wt}(x)$

EXAMPLE

time	input batch	cover	query cost	build cost
1	$\{a, b\}$	$\{a\}, \{b\}$	2	$\mathrm{wt}(a) + \mathrm{wt}(b)$
2	$\{c\}$	$\{b\}, \{a, c\}$	2	$\mathrm{wt}(a) + \mathrm{wt}(c)$

we call the sets in each cover "components"

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that
the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S \in \mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

MINIMIZE COST: $\sum_{t=1}^{n} \sum_{S \in \mathscr{C}_t \setminus \mathscr{C}_{t-1}} \text{wt}(S) + \sum_{t=1}^{n} |\mathscr{C}_t|$ (build cost + query cost)

adding a new set S to the cover incurs build cost $\text{wt}(S) = \sum_{x \in S} \text{wt}(x)$

EXAMPLE

time	input batch	cover	query cost	build cost
1	$\{a, b\}$	$\{a\}, \{b\}$	2	$\text{wt}(a) + \text{wt}(b)$
2	$\{c\}$	$\{b\}, \{a, c\}$	2	$\text{wt}(a) + \text{wt}(c)$
3				

we call the sets in each cover "components"

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S \in \mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

MINIMIZE COST: $$\sum_{t=1}^{n} \sum_{S \in \mathscr{C}_t \setminus \mathscr{C}_{t-1}} \mathrm{wt}(S) + \sum_{t=1}^{n} |\mathscr{C}_t|$$ (build cost + query cost)

adding a new set S to the cover incurs build cost $\mathrm{wt}(S) = \sum_{x \in S} \mathrm{wt}(x)$

EXAMPLE

time	input batch	cover	query cost	build cost
1	$\{a, b\}$	$\{a\}, \{b\}$	2	$\mathrm{wt}(a) + \mathrm{wt}(b)$
2	$\{c\}$	$\{b\}, \{a, c\}$	2	$\mathrm{wt}(a) + \mathrm{wt}(c)$
3	$\{d, e\}$			

we call the sets in each cover "components"

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S \in \mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

MINIMIZE COST: $\sum_{t=1}^{n} \sum_{S \in \mathscr{C}_t \setminus \mathscr{C}_{t-1}} \mathrm{wt}(S) + \sum_{t=1}^{n} |\mathscr{C}_t|$ (build cost + query cost)

adding a new set S to the cover incurs build cost $\mathrm{wt}(S) = \sum_{x \in S} \mathrm{wt}(x)$

EXAMPLE

time	input batch	cover	query cost	build cost
1	$\{a, b\}$	$\{a\}, \{b\}$	2	$\mathrm{wt}(a) + \mathrm{wt}(b)$
2	$\{c\}$	$\{b\}, \{a, c\}$	2	$\mathrm{wt}(a) + \mathrm{wt}(c)$
3	$\{d, e\}$	$\{b\}, \{a, c\}, \{d, e\}$	3	$\mathrm{wt}(d) + \mathrm{wt}(e)$

we call the sets in each cover "components"

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S \in \mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

MINIMIZE COST: $\sum_{t=1}^{n} \sum_{S \in \mathscr{C}_t \setminus \mathscr{C}_{t-1}} \mathrm{wt}(S) + \sum_{t=1}^{n} |\mathscr{C}_t|$ (build cost + query cost)

adding a new set S to the cover incurs build cost $\mathrm{wt}(S) = \sum_{x \in S} \mathrm{wt}(x)$

EXAMPLE

time	input batch	cover	query cost	build cost
1	$\{a, b\}$	$\{a\}, \{b\}$	2	$\mathrm{wt}(a) + \mathrm{wt}(b)$
2	$\{c\}$	$\{b\}, \{a, c\}$	2	$\mathrm{wt}(a) + \mathrm{wt}(c)$
3	$\{d, e\}$	$\{b\}, \{a, c\}, \{d, e\}$	3	$\mathrm{wt}(d) + \mathrm{wt}(e)$

total cost: 7 + $2\,\mathrm{wt}(a)+\mathrm{wt}(b)+\mathrm{wt}(c)+\mathrm{wt}(d)+\mathrm{wt}(e)$

we call the sets in each cover "components"

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S\in\mathscr{C}_t} S = \bigcup_{i=1}^t I_i\right)$

MINIMIZE COST: $\sum_{t=1}^{n} \sum_{S\in\mathscr{C}_t\setminus\mathscr{C}_{t-1}} \mathrm{wt}(S) + \sum_{t=1}^{n} |\mathscr{C}_t|$ (build cost + query cost)

on uniform input ($\mathrm{wt}(I_t) = 1$)
pays query cost $\Theta(n^2)$
pays build cost $\Theta(n)$

TRIVIAL ALGORITHM 1: *insert batch as component*

time	input batch	cover	query cost	build cost
1	$\{a, b\}$	$\{a, b\}$	1	$\mathrm{wt}(a) + \mathrm{wt}(b)$
2	$\{c\}$	$\{a, b\}, \{c\}$	2	$\mathrm{wt}(c)$
3	$\{d, e\}$	$\{a, b\}, \{c\}, \{d, e\}$	3	$\mathrm{wt(d)} + \mathrm{wt}(e)$

total cost: $6 + \mathrm{wt}(a) + \mathrm{wt}(b) + \mathrm{wt}(c) + \mathrm{wt}(d) + \mathrm{wt}(e)$

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S\in\mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

MINIMIZE COST: $\sum_{t=1}^{n} \sum_{S\in\mathscr{C}_t\setminus\mathscr{C}_{t-1}} \mathrm{wt}(S) + \sum_{t=1}^{n} |\mathscr{C}_t|$ (build cost + query cost)

on uniform input ($\mathrm{wt}(I_t) = 1$)
pays query cost $\Theta(n)$
pays build cost $\Theta(n^2)$

TRIVIAL ALGORITHM 2: *use just one component*

time	input batch	cover	query cost	build cost
1	$\{a, b\}$	$\{a, b\}$	1	$\mathrm{wt}(a)+\mathrm{wt}(b)$
2	$\{c\}$	$\{a, b, c\}$	1	$\mathrm{wt}(a)+\mathrm{wt}(b)+\mathrm{wt}(c)$
3	$\{d, e\}$	$\{a, b, c, d, e\}$	1	$\mathrm{wt}(a)+\mathrm{wt}(b)+\mathrm{wt}(c)+\mathrm{wt}(d)+\mathrm{wt}(e)$

total cost: $4 + 4\mathrm{wt(a)}+4\mathrm{wt(b)}+3\mathrm{wt(c)}+2\mathrm{wt(d)}+2\mathrm{wt(e)}+\mathrm{wt(f)}$

BINARY TRANSFORM [Bentley, 1979] achieves cost $O(n \log n)$ on uniform inputs

time	in binary	input batch	cover
1	0001	I_1	I_1
2	0010	I_2	$I_1 \cup I_2$
3	0011	I_3	$I_1 \cup I_2,\ \ I_3$
4	0100	I_4	$I_1 \cup I_2 \cup I_3 \cup I_4$
5	0101	I_5	$I_1 \cup I_2 \cup I_3 \cup I_4,\ \ I_5$
6	0110	I_6	$I_1 \cup I_2 \cup I_3 \cup I_4,\ \ I_5 \cup I_6$
7	0111	I_7	$I_1 \cup I_2 \cup I_3 \cup I_4,\ \ I_5 \cup I_6,\ \ I_7$
8	0100	I_8	$I_1 \cup I_2 \cup I_3 \cup I_4 \cup I_5 \cup I_6 \cup I_7 \cup I_8$
9	0101	I_9	$I_1 \cup I_2 \cup I_3 \cup I_4 \cup I_5 \cup I_6 \cup I_7 \cup I_8,\ \ I_9$
10	0110	I_{10}	$I_1 \cup I_2 \cup I_3 \cup I_4 \cup I_5 \cup I_6 \cup I_7 \cup I_8,\ \ I_9 \cup I_{10}$
⋮		⋮	⋮

similar to classical Binomial Heap:

At each time t, there is one component for each 1 in the binary representation of t. Each step emulates an increment in binary.

⟶ on uniform input:

pays build cost $\Theta(n \log n)$

pays query cost $\Theta(n \log n)$

total cost $\Theta(n \log n)$

optimal for uniform input

Problem 2: K-COMPONENT DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of *batches* (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that
the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S \in \mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

same as before

CONSTRAINT: $|\mathscr{C}_t| \leq k$ — at each time t, cover size is at most k

MINIMIZE BUILD COST: $\displaystyle\sum_{t=1}^{n} \sum_{S \in \mathscr{C}_t \setminus \mathscr{C}_{t-1}} \text{wt}(S)$

Problem 2: K-COMPONENT DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of *batches* (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S\in\mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

same as before

CONSTRAINT: $|\mathscr{C}_t| \le k$ — at each time t, cover size is at most k

MINIMIZE BUILD COST: $\displaystyle\sum_{t=1}^{n} \sum_{S\in\mathscr{C}_t\setminus\mathscr{C}_{t-1}} \mathrm{wt}(S)$

K-BINOMIAL TRANSFORM **[Bentley & Saxe, 1980] achieves cost $\Theta(kn^{1+1/k})$ on uniform inputs**

optimal for uniform input

At each time t:

1. Let $i_1, \ldots, i_k$ be the k integers such that $0 \le i_1 < i_2 < i_3 < \cdots < i_k$ and $\sum_{j=1}^{k} \binom{i_j}{j} = t$.
2. Use the cover consisting of k sets, where
   - the first set contains the first $\binom{i_k}{k}$ batches,
   - the second set contains the next $\binom{i_{k-1}}{k-1}$ batches,
   - and so on.

MIN-SUM DYNAMIZATION, K-COMPONENT DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that
the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S \in \mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

...

Prior results for uniform input, but production LSM-system inputs are *online* and *non-uniform.*

Non-uniform inputs can be *easier* (less costly) than uniform inputs.

COMPETITIVE ANALYSIS

DEFN: An algorithm is *online* if its cover at each time t is independent of $I_{t+1}, I_{t+2}, \ldots, I_n$.

DEFN: An algorithm is *c-competitive* if, for every input, its solution costs at most c times the optimum for that input. The *competitive ratio* of the algorithm is the minimum such c.

} standard

GOAL: online algorithms with smallest possible competitive ratios…

PREVIOUS RESULTS

- For Min-Sum Dynamization, Bentley's Binary Transform yields competitive ratio $\Theta(\log n)$.
- For *k*-Component Dynamization, the *k*-Binomial Transform yields competitive ratio $\Theta(kn^{1/k})$.

compaction policies for LSM (log-structured merge) systems
through the lens of competitive analysis

MIN-SUM DYNAMIZATION, K-COMPONENT DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that
the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S \in \mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

...

PREVIOUS RESULTS

- For Min-Sum Dynamization, Bentley's Binary Transform yields competitive ratio $\Theta(\log n)$.
- For *k*-Component Dynamization, the *k*-Binomial Transform yields competitive ratio $\Theta(kn^{1/k})$.

MAIN RESULTS

- **THM 2.1.** *Min-Sum Dynamization has an online algorithm with competitive ratio* $\Theta(\log^* n)$.
- **THMS 3.1—3.4.** *For k-Component Dynamization, there are deterministic online algorithms with competitive ratio k, and this is best possible for deterministic algorithms.*
- **EXTENSIONS**: the *k*-Component Dynamization results extend to allow *lazy deletions, updates, item expiration* as they occur in LSM systems such as Bigtable (see the paper).

MIN-SUM DYNAMIZATION, K-COMPONENT DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that
the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S \in \mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

...

PREVIOUS RESULTS

- For Min-Sum Dynamization, Bentley's Binary Transform yields competitive ratio $\Theta(\log n)$.
- For *k*-Component Dynamization, the *k*-Binomial Transform yields competitive ratio $\Theta(kn^{1/k})$.

MAIN RESULTS

- **THM 2.1.** *Min-Sum Dynamization has an online algorithm with competitive ratio* $\Theta(\log^* n)$.

  OPEN: *constant* competitive ratio for Min-Sum Dynamization?

- **THMS 3.1—3.4.** *For k-Component Dynamization, there are deterministic online algorithms with competitive ratio k, and this is best possible for deterministic algorithms.*

  OPEN: randomized algorithms for k-Component Dynamization?

- **EXTENSIONS**: the results on *k*-Component Dynamization extend to allow *lazy deletions, updates, item expiration* as they occur in big-data storage systems (see the paper).

  OPEN: same extensions for Min-Sum Dynamization?

See the paper for many more open problems.

Problem 1: MIN-SUM DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of batches (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S\in\mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

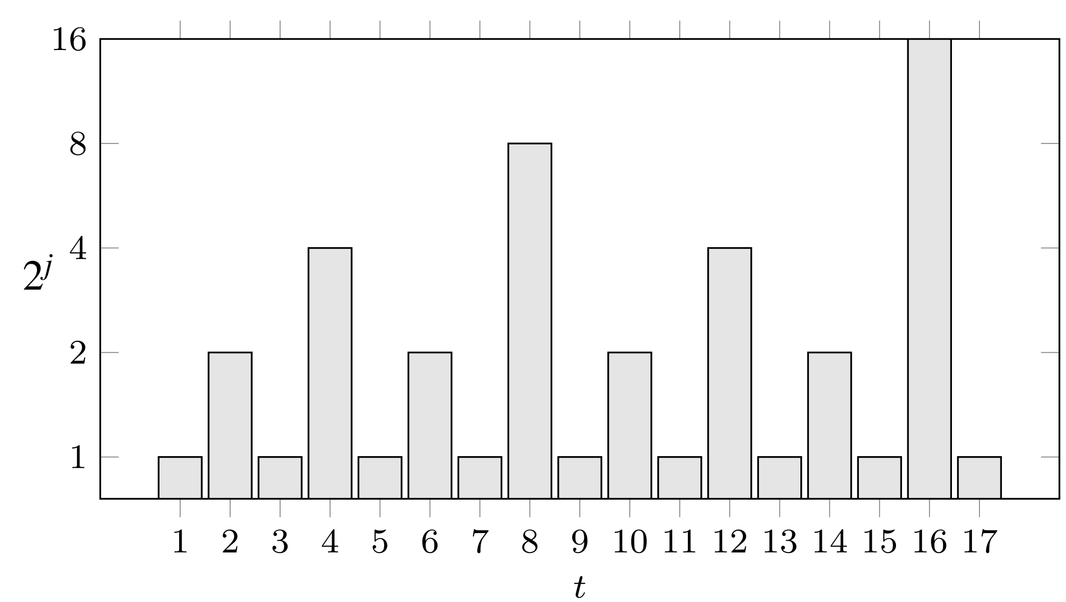
MINIMIZE COST: $\sum_{t=1}^{n} \sum_{S\in\mathscr{C}_t\setminus\mathscr{C}_{t-1}} \mathrm{wt}(S) + \sum_{t=1}^{n} |\mathscr{C}_t|$ (build cost + query cost)

THM 2.1. *The online algorithm below has competitive ratio* $\Theta(\log^* n)$.

at each time $t \leftarrow 1,2,\ldots,n$ do:

1. add current batch I_t to the current cover as a single new set
2. let 2^j be the largest power of 2 such that t is an integer multiple of 2^j
3. merge all sets S in the cover such that $\mathrm{wt}(S) \le 2^j$ into one new set

Roughly, every 2^j time steps it merges together all sets of weight 2^j or less.



compaction policies for LSM (log-structured merge) systems through the lens of competitive analysis

Problem 2: K-COMPONENT DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of *batches* (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S \in \mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

CONSTRAINT: $|\mathscr{C}_t| \le k$ — at each time t, cover size is at most k

MINIMIZE BUILD COST: $\sum_{t=1}^{n} \sum_{S \in \mathscr{C}_t \setminus \mathscr{C}_{t-1}} \mathrm{wt}(S)$

THM 3.1. *Any deterministic online algorithm has competitive ratio at least k.*

Here we give the idea for k=2.

time	input weight	alg cover	alg cost	OPT cost?	
1	1	$\{1\}$	1	1	1
2	ε	$\{1\}, \{\varepsilon\}$	ε	$1+\varepsilon$	ε
3	0	$\{1\}, \{\varepsilon, 0\}$	ε	0	ε
4	0	$\{1\}, \{\varepsilon, 0, 0\}$	ε	0	ε
⋮	⋮	⋮	⋮	⋮	⋮
m-1	0	$\{1\}, \{\varepsilon, 0, 0, \ldots, 0\}$	ε	0	ε
m	0	$\{1, \varepsilon, 0, 0, \ldots, 0, 0\}$	$1+\varepsilon$	0	ε
			total: $2 + (m\text{-}1)\varepsilon$	$\min(2+\varepsilon,$	$1+(m\text{-}1)\varepsilon)$

Alg chooses $m \approx 1/\epsilon$, so

$$\frac{\text{alg cost}}{\text{OPT cost}} \approx \frac{2+1}{2} = 3/2$$

If there were no "setup cost" of 1 at time 1, ratio would be

$$\frac{\text{alg cost}}{\text{OPT cost}} \approx \frac{1+1}{1} = 2$$

Problem 2: K-COMPONENT DYNAMIZATION

INPUT: $I_1, I_2, \ldots, I_n$ — a sequence of *batches* (sets of weighted items)

OUTPUT: $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ — a sequence of set covers such that the sets in $\mathscr{C}_t$ cover all items inserted up to time t $\left(\bigcup_{S\in\mathscr{C}_t} S = \bigcup_{i=1}^{t} I_i\right)$

CONSTRAINT: $|\mathscr{C}_t| \le k$ — at each time t, cover size is at most k

MINIMIZE BUILD COST: $\sum_{t=1}^{n} \sum_{S\in\mathscr{C}_t\setminus\mathscr{C}_{t-1}} \mathrm{wt}(S)$

THM 3.1. *Any deterministic online algorithm has competitive ratio at least k.*

Here we sketch a proof for k=2.

time	input weight	alg cover	alg cost	OPT cost?	
1	1	$\{1\}$	1	1	1
2	ε	$\{1\}, \{\varepsilon\}$	ε	$1+\varepsilon$	ε
3	0	$\{1\}, \{\varepsilon, 0\}$	ε	0	ε
4	0	$\{1\}, \{\varepsilon, 0, 0\}$	ε	0	ε
⋮	⋮	⋮	⋮	⋮	⋮
m-1	0	$\{1\}, \{\varepsilon, 0, 0, \ldots, 0\}$	ε	0	ε
m	0	$\{1, \varepsilon, 0, 0, \ldots, 0, 0\}$	$1+\varepsilon$	0	ε
m+1	$\sqrt{\epsilon}$	$\{1, \varepsilon, 0, \ldots, 0\}, \{\sqrt{\epsilon}\}$	$\sqrt{\epsilon}$	$1+\sqrt{\epsilon}+\epsilon$	$\sqrt{\epsilon}+\epsilon$
m+2	0	$\{1, \varepsilon, 0, \ldots, 0\}, \{\sqrt{\epsilon}, 0\}$	$\sqrt{\epsilon}$	0	$\sqrt{\epsilon}+\epsilon$
⋮	⋮	⋮	⋮	⋮	⋮
	0	$\{1, \varepsilon, 0, \ldots, 0\}, \{\sqrt{\epsilon}, 0, \ldots, 0\}$	$\sqrt{\epsilon}$	0	$\sqrt{\epsilon}+\epsilon$
	0	$\{1, \varepsilon, 0, \ldots, 0, \sqrt{\epsilon}, 0, \ldots, 0\}$	$1+\sqrt{\epsilon}+\epsilon$	0	$\sqrt{\epsilon}+\epsilon$

⋮

For this second round the ratio is $2 - O(\sqrt{\epsilon})$. Repeating drives the total ratio arbitrarily near 2.

for Problem 2, k-Component Dynamization:

THM 3.2. *The online algorithm below has competitive ratio k.*

at each time $t \leftarrow 1,2,\ldots,n$, in response to batch I_t do:

1. if there are k sets in the cover:
   a. increase all sets' credits continuously until a set S has credit$[S] \geq$ wt(S)
   b. let S_t be the oldest such set
   c. merge I_t , S_t , and all sets newer than S_t into one new set with credit 0
2. else: add I_t as a new set, with credit 0

we associate a "credit" with each set in the current cover

The paper also gives a second "recursive rent-or-buy" algorithm with a very different analysis.

for Problem 2, k-Component Dynamization:

THM 3.2. *The online algorithm below has competitive ratio k.*

at each time $t \leftarrow 1,2,\ldots,n$, in response to batch I_t do:

1. if there are k sets in the cover:
   a. increase all sets' credits continuously until a set S has credit$[S] \geq$ wt(S)
   b. let S_t be the oldest such set
   c. merge I_t , S_t , and all sets newer than S_t into one new set with credit 0
2. else: add I_t as a new set, with credit 0

PROOF OUTLINE:

1. let δ_t be the decrease in credit in iteration t
2. total credit given to sets is $k\sum_t \delta_t$
3. sets S_t contribute at most $k\sum_t \delta_t$ to algorithm's cost (as credit$[S_t] \geq$ wt(S_t) when merged)
4. remaining sets contribute at most $k\sum_t$ wt(I_t) to algorithm's cost (as items decrease in "rank")
5. so algorithm's cost is at most $k\sum_t$ wt$(I_t) + \delta_t$
6. charge credit to OPT (via implicit LP-dual soln) to show OPT cost is at least $\sum_t$ wt$(I_t) + \delta_t$

compaction policies for LSM (log-structured merge) systems
through the lens of competitive analysis

MOTIVATION

B-TREES VS. MOORE'S LAW

LSM-SYSTEM COMPACTION VIA DATA-STRUCTURE DYNAMIZATION

DEFINITIONS

PROBLEM 1 — *MIN-SUM DYNAMIZATION*

PROBLEM 2 — *K-COMPONENT DYNAMIZATION*

COMPETITIVE ANALYSIS

RESULTS

PROBLEM 1 ALGORITHM, $\Theta(\log^* n)$-COMPETITIVE

PROBLEM 2 LOWER BOUND

PROBLEM 2 ALGORITHMS, K-COMPETITIVE

QUESTIONS, IF TIME PERMITS