Indexing for Near-Sorted Data

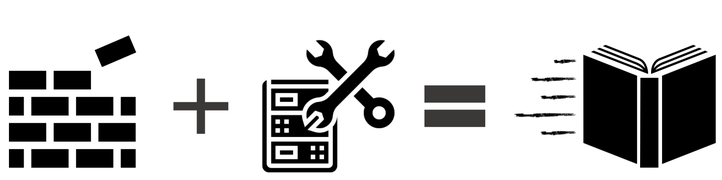
Aneesh Raman Subhadeep Sarkar Matthaios Olma Manos Athanassoulis

BOSTON UNIVERSITY

Microsoft Research

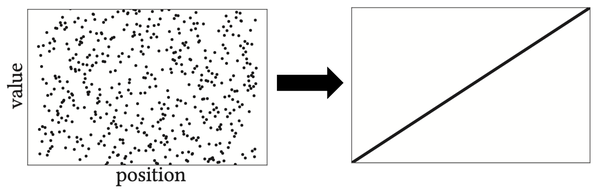
Indexes in Data Systems

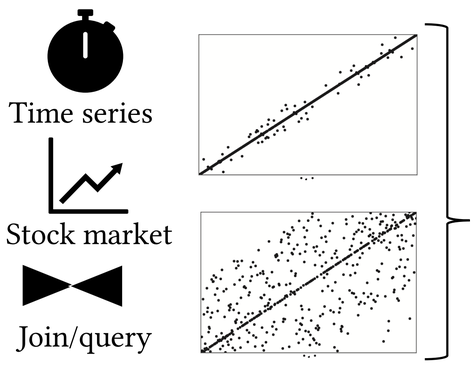
PostgreSQL MySQL ORACLE DATABASE SQLite IBM DB2 Microsoft SQL Server



Build & maintain index for fast queries

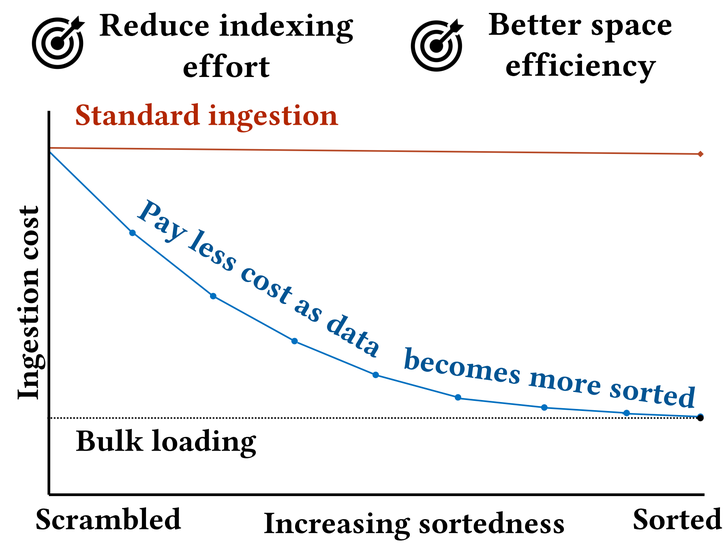
Indexing Adds Structure



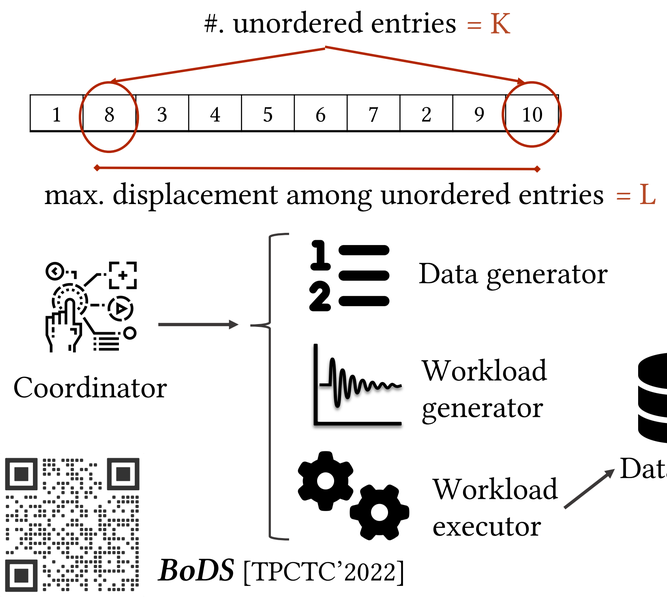


What if data is almost sorted?

Vision

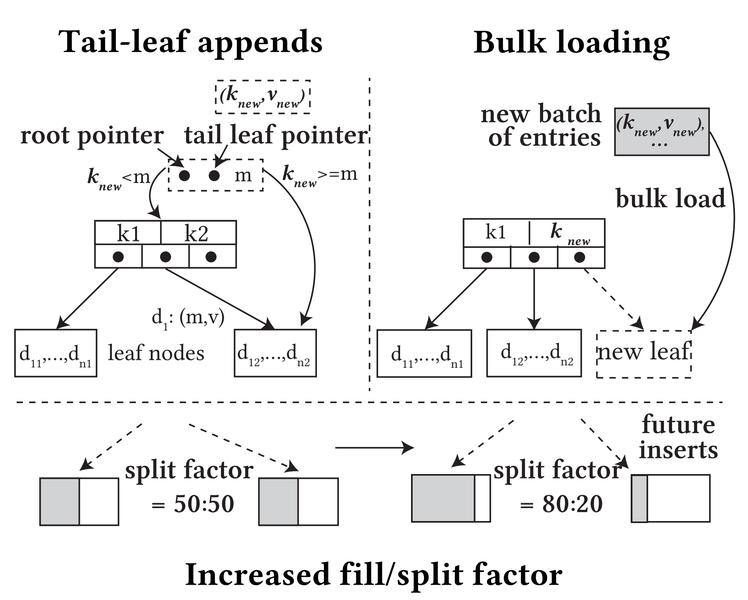


BoDS: A Benchmark on Data Sortedness

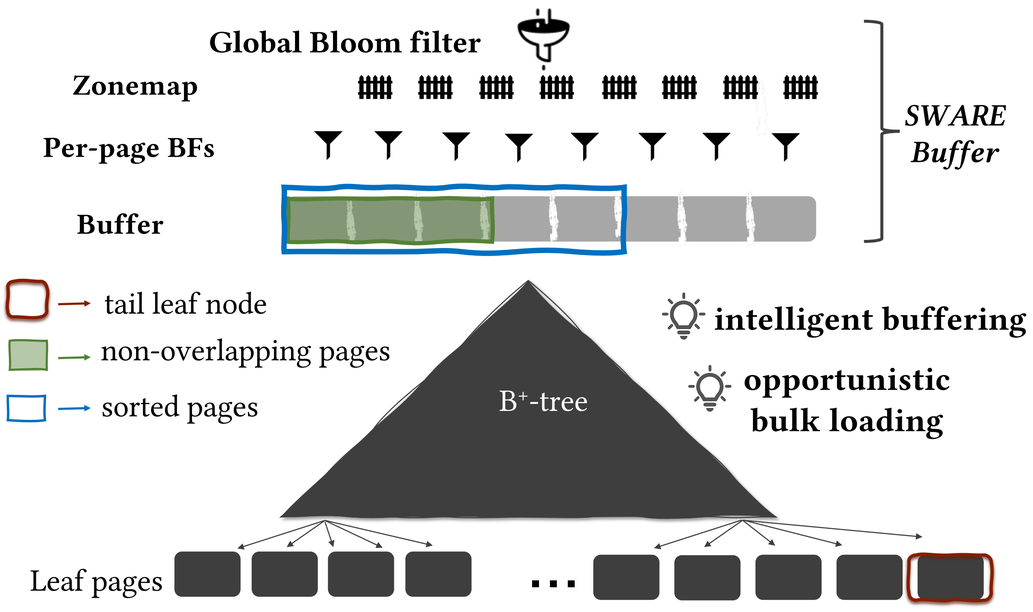


Type	Workload	Data Loading		Operations	
		Method	% data	R-W	% data
Insert only	A	Bulk loading	100%	-	-
	B	Individual inserts	100%	-	-
Mixed reads & writes	C	-	0%	17%-83%	100%
	D	Bulk loading	80%	50%-50%	20%
	E	Individual inserts	80%	50%-50%	20%

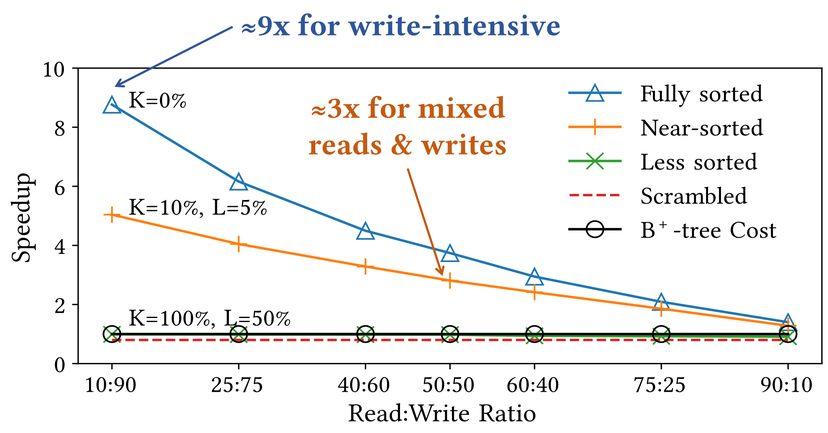
Sortedness-Aware Design Elements



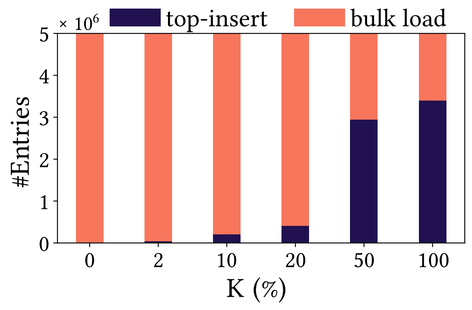
The Sortedness-Aware (SWARE) Paradigm



A *sortedness-aware* (SA) B⁺-tree outperforms its baseline for fully and near-sorted workloads



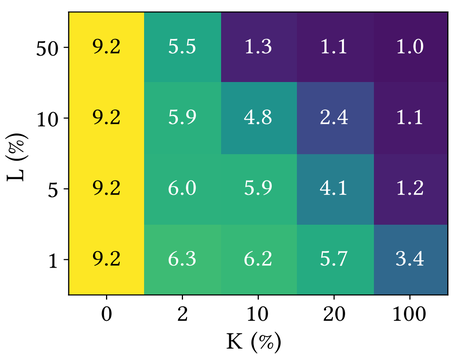
SWARE can work with any tree index!



SWARE maximizes opportunistic bulk loading with high data sortedness

Applicability of SWARE

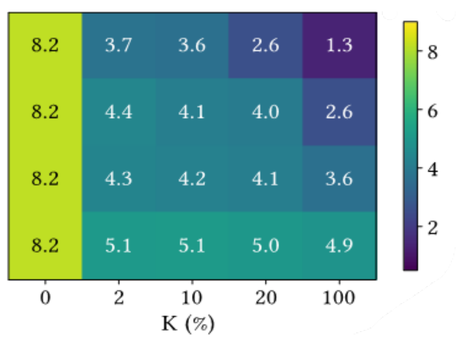
R/W= 10/90; Buffer = 1%



A larger buffer better captures sortedness at a higher read cost

SWARE benefits more write-heavy workloads

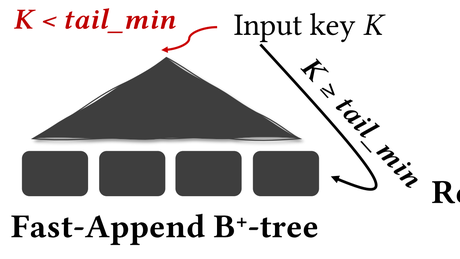
R/W= 50/50; Buffer = 5%



Degree of Sortedness	# Nodes	
	B⁺-tree	SA B⁺-tree
Fully sorted	2 M	0.52x
Near-sorted	1.85 M	0.6x
Less-sorted	1.88 M	1.01x

SWARE reduces memory footprint for fully-sorted and near-sorted data

Future Work



Fast-Append B⁺-tree

Reducing write amp. in LSM-trees

Near-sorted joins

? Can SMJ be optimized for near-sorted data?

Compression algorithms

? Can algorithms like RLE exploit near-sortedness?

BOSTON UNIVERSITY Department of Computer Science

NSF

disc.bu.edu DiSC lab