Redbench: Workload Synthesis From Cloud Traces

VLDB 2026

Johannes Wehrstein, Roman Heinrich, Mihail Stoian, Skander Krid, Martin Stemmer, Andreas Kipf, Carsten Binnig, Muhammad El-Hindi

TU Darmstadt, TU Nuremberg, TU Munich

Benchmarks Today

Today's **benchmarks** are designed to be **representative** for all WLs…
… make **simplifying assumptions:**

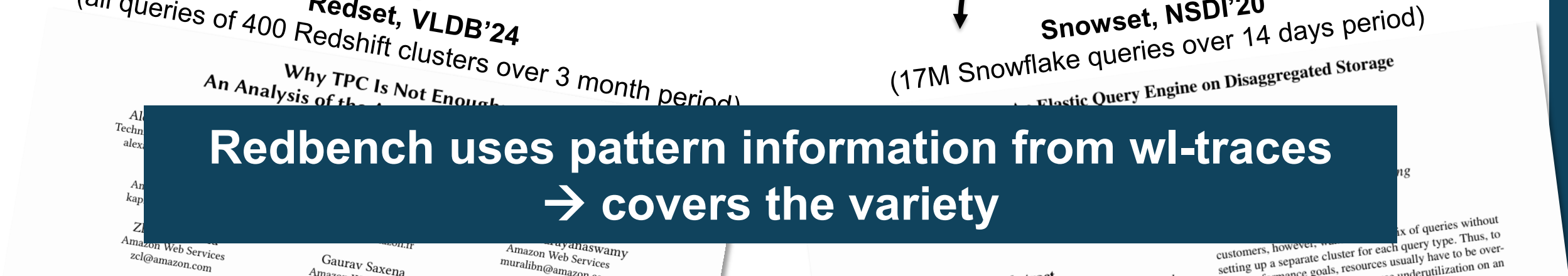
e.g. TPC-H:
- 22 queries
- fixed execution order
- fixed read/write load

➔ **BUT: different from how analytical systems are used!**

Huge variety of patterns across customers
Today's benchmarks:
→ only one pattern

Redset, *VLDB'24*
(all queries of 400 Redshift clusters over 3 month period)

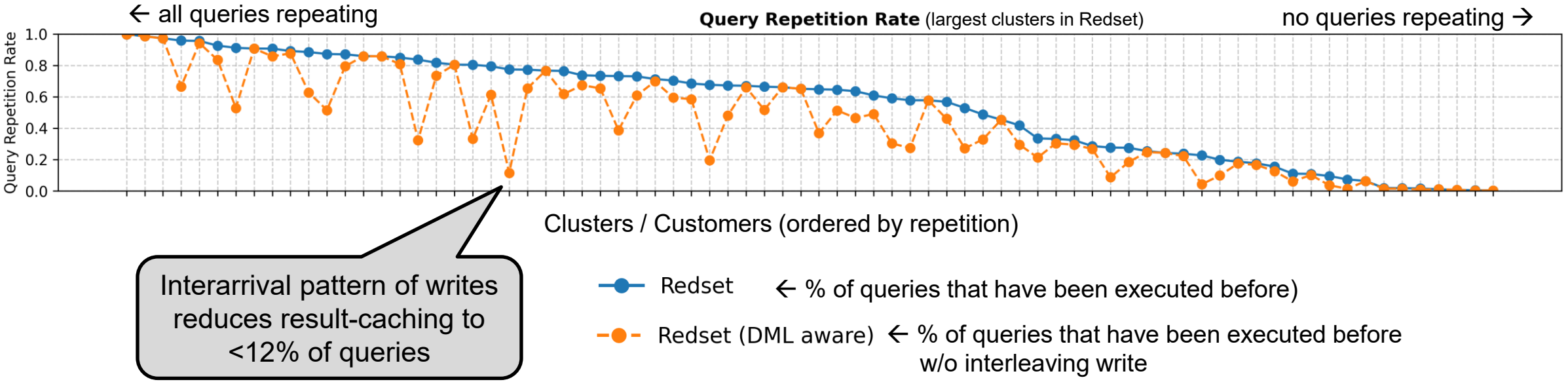
Snowset, NSDI'20
(17M Snowflake queries over 14 days period)



Redbench uses pattern information from wl-traces
→ covers the variety

Repetitions!

99% of SELECT could be answered from result cache!

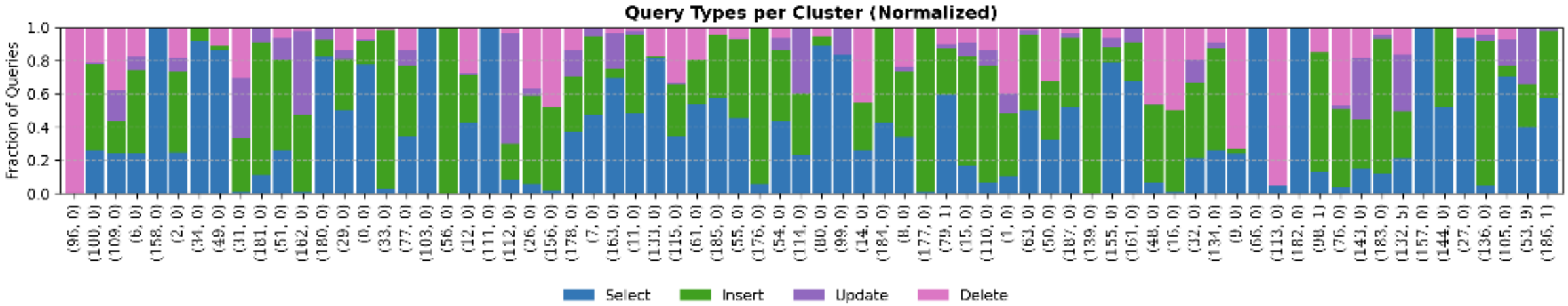


Interarrival pattern of writes reduces result-caching to <12% of queries

Wide spectrum of *repetitions*: up to 99% of SELECTs are repeating

Query Type Distribution

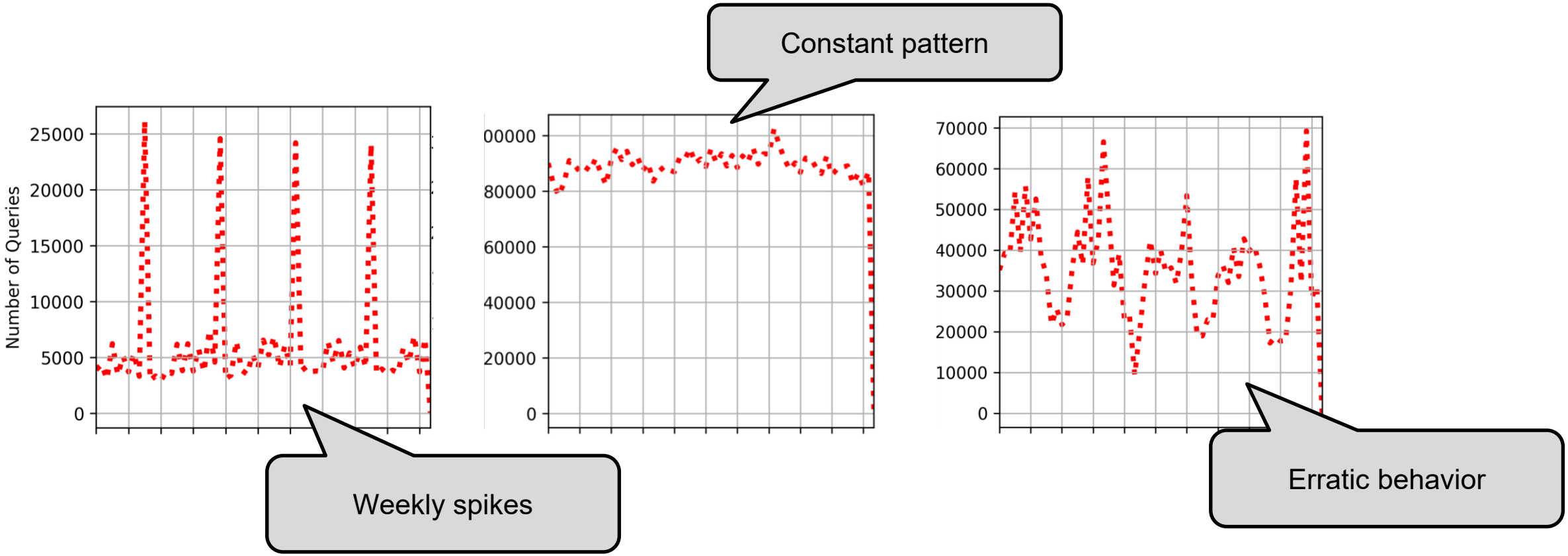
Distribution by #Queries



Query-Type Ratios of largest clusters from Redset

There is no default query-type mix!

Arrival Times

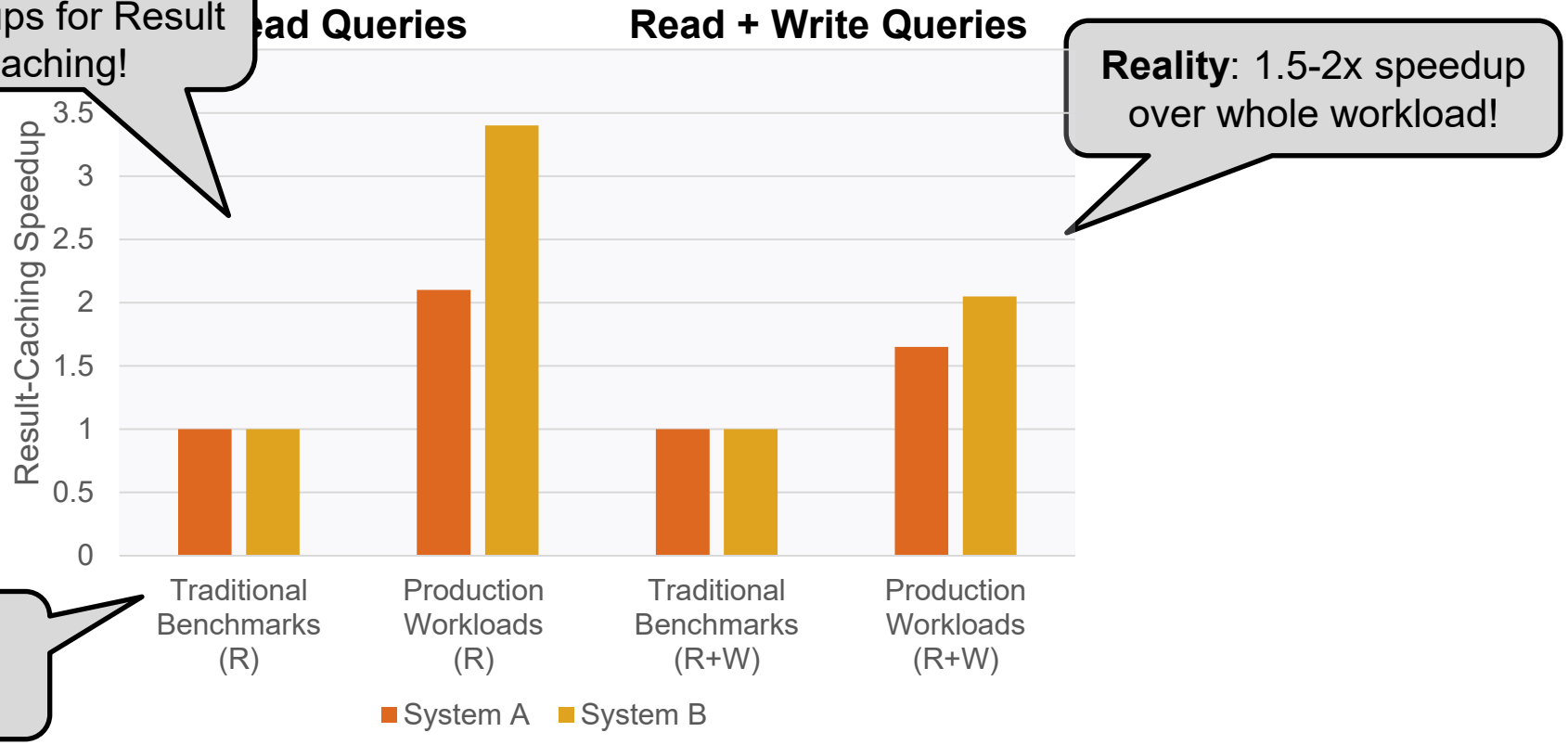


Arrival times of largest clusters from Redset (over a 4-week period)

Arrival Times are more complex than "more during the week, less on the weekend"

The Case for Result Caching

Traditional Benchmarks: No speedups for Result Caching!



Reality: 1.5-2x speedup over whole workload!

Trad. Benchmark = round-robin JOB with interleaving writes

Traditional benchmarks hide speedup potential

Redbench: Workload Statistics as the Basis

Input statistics:

- Arrival Times
- Query Type (`SELECT` / `INSERT` / `DELETE` / …)
- Query Complexity (`num_joins`, `bytes_read`, …)
- Read/Write Sets
- Repetition Information

→ **Published e.g. in Redset, or Snowset (partially)**

Redbench mimics these trace statistics!

From Redset to Redbench

The workload stats Redbench will mimic (e.g. specific cluster / database from the Redset)

The schema Redbench should target

Your Data & Schema
(e.g. IMDB)

Assigns queries with same:
- Read / write-Sets (tables)
- Repetitions
- Runtime/bytes-read distribution
- Query complexity

Timestamp	Query Type	# Joins	Bytes Read	Read-Tables	Write-Table	Hash
00:13:33	SELECT	2	3 GB	1,3,4		*0x3bc*
00:15:21	INSERT	1	1 GB	1,2	3	*0x113*
00:15:33	DELETE	0	30 MB	3	4	*0x83d*
00:15:45	SELECT	2	3 GB	1,3,4		*0x3bc (repetition)*

Redbench

SQL
SELECT * FROM title …
INSERT INTO title VALUES …
DELETE …
SELECT * FROM title … (rep #1)

Optional:

SELECT COUNT(*) FROM …

Reference Query Pool
(instead of generating arbitrary "new" queries)

Redbench supports sampling from **query pool** ("Matching") **or generates queries** from scratch ("Generation")

Matching vs. Generation

Systems-under-development

- that support only limited query set
- you want to bring in more realistic characteristics (repetitions, r/w arrival times)

→***Matching***
(map to closest existing queries)

More closer representation of workload you try to replicate

- You want real trace faithful benchmark
- read-volume, read/write-sets, query-complexity, …

→ ***Generation***
(generate new queries that exactly match)

Downside/Feature:
unseen queries → might break your system (e.g. large intermediate result)

Matching

Mapping Redset repetitions & arrival times to user supplied workload.

Timestamp	Query Type	# Joins	Bytes Read	Read-Tables	Write-Table	Hash
00:13:33	SELECT	2	3 GB	1,3,4		*0x3bc*
00:15:21	INSERT	1	1 GB	1,2	3	*0x113*
00:15:33	DELETE	0	30 MB	3	4	*0x83d*

Workload Data – Redset Cluster 42
(repetitions, scansets)

Similarity Search
Compare by query complexity e.g. #joins

Your Queries
(e.g. existing benchmarks: JOB)

```
SELECT * FROM title, movie_comp…
SELECT COUNT(*) FROM movie_...
  …
```

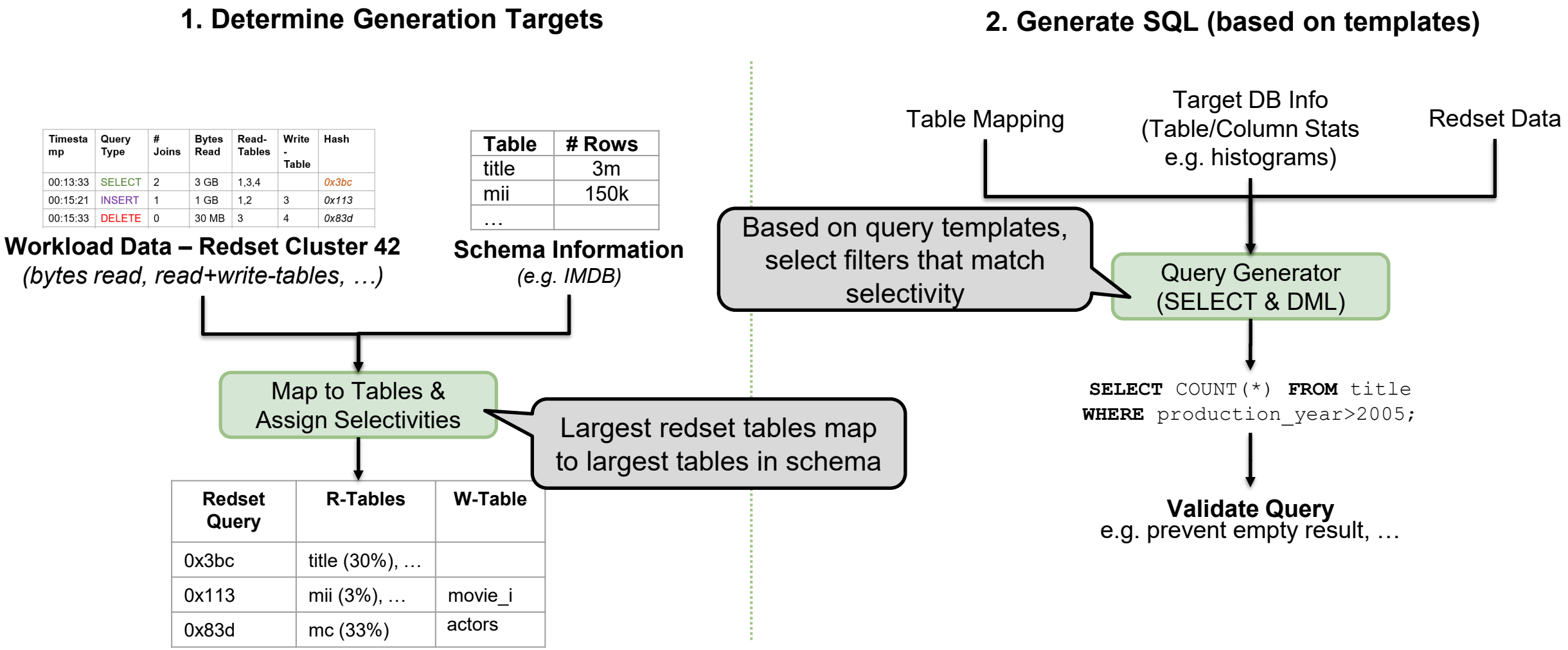
Query Mapping
Q1: JOB 7a
Q2: (INSERT)
Q3: JOB 3d

JOB with **realistic repetitions** and **r/w inter-arrival times**

DML Generator
(Simple delete/inserts)

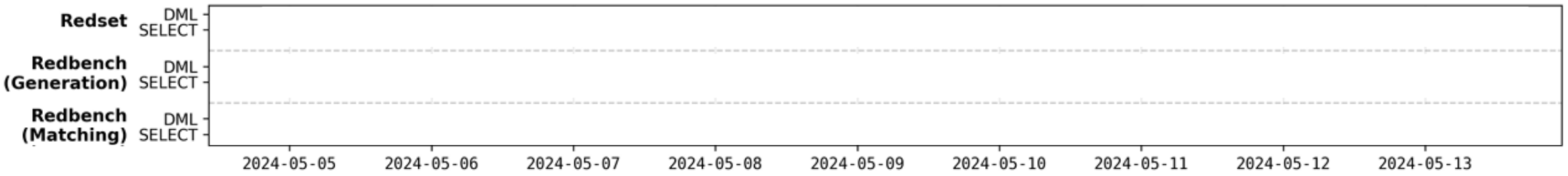
Generation

Generate queries that exactly mimic redset queries.



Trace Faithfulness: Read/Write Intervals

More Comparisons in the paper
(num joins, bytes-read, runtime, scan-set repetition rate, …)



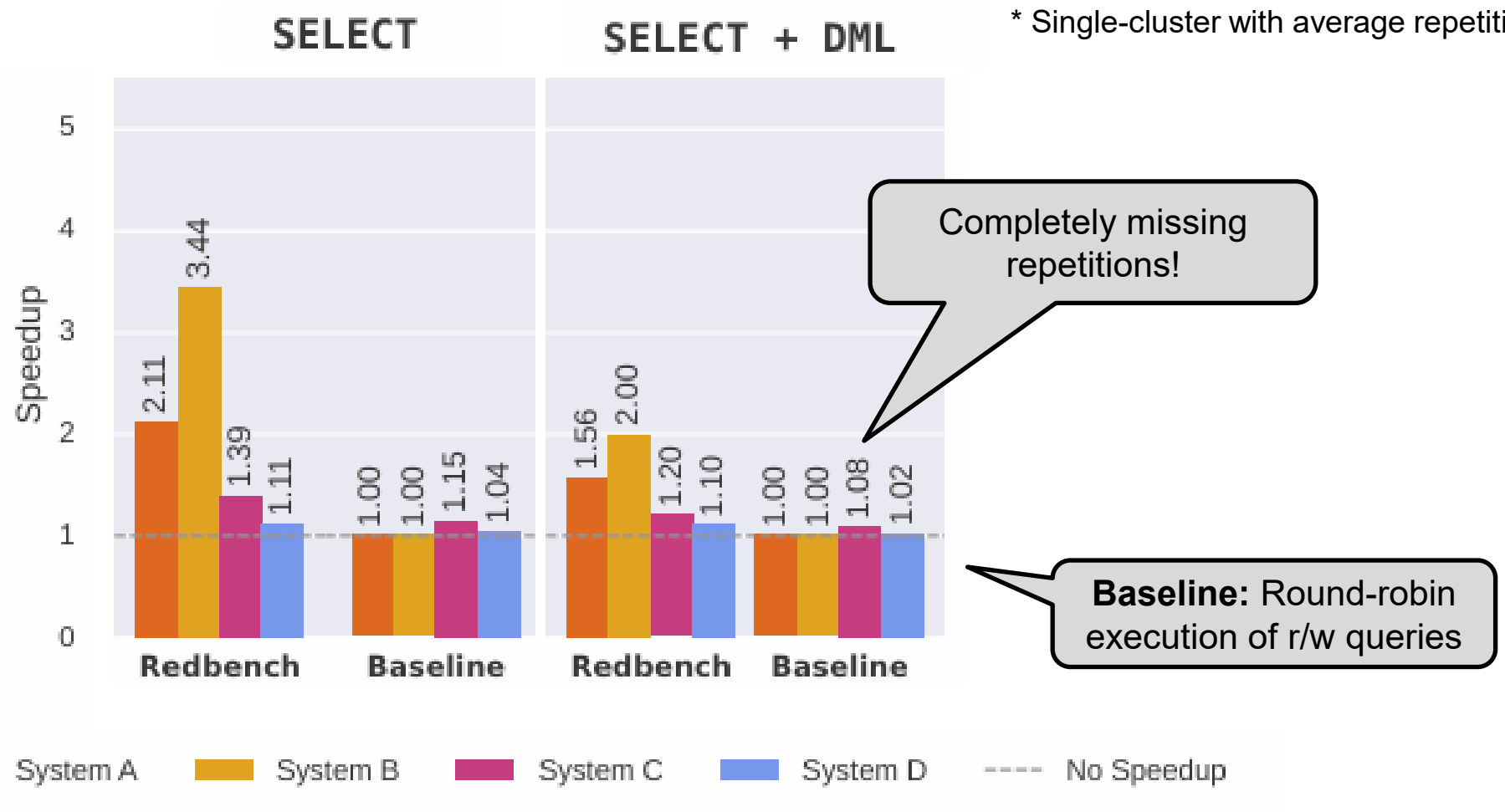
Redbench is maintaining Read/Write Intervals

Analyze System Behaviour with Redbench

Case-Study: Caching

Result Caching Evaluated in Different Systems

* Single-cluster with average repetitions

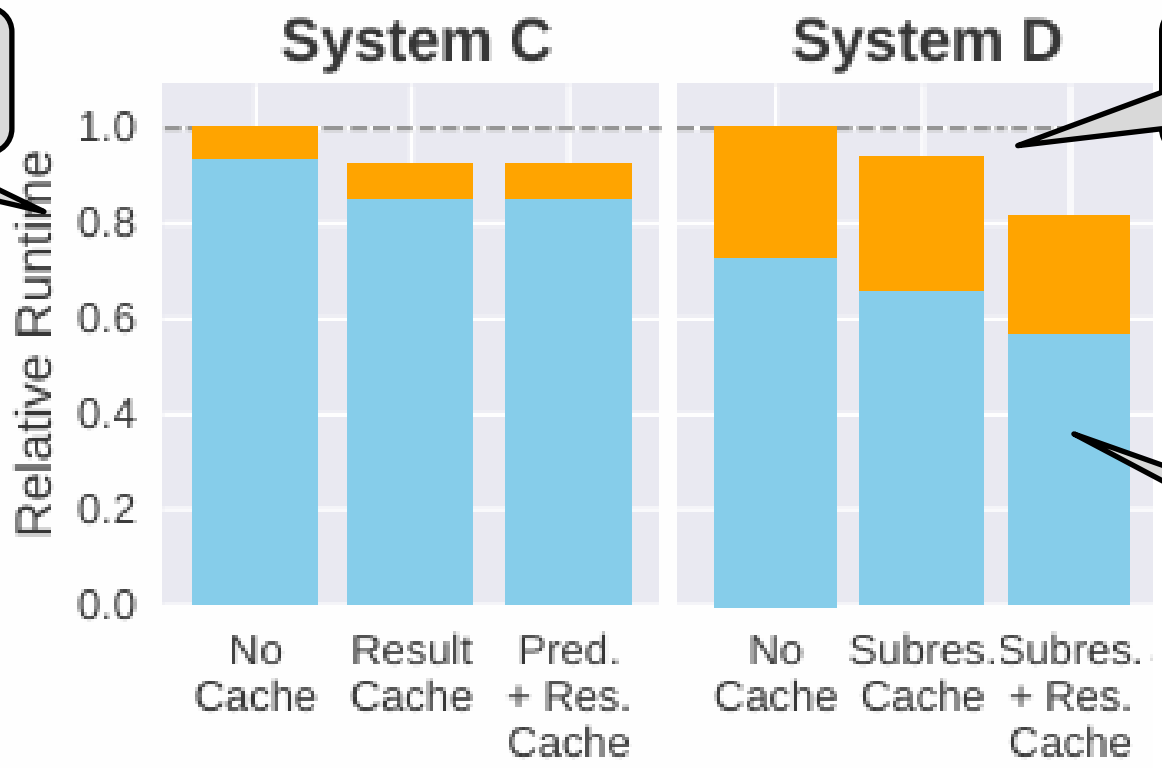


Systems vary heavily in result-caching speedups!

Fine-Grained Caching Strategies

System **C** is tuned for **fast writes**

System **D** shows **write overheads**



Fine-grained caching reduces runtime by 20%

Fine-grained caching brings benefits, but systems use heavily diverging strategies

Redbench

Try it out:

datamanagementlab.github.io/Redbench/

- ✅ Repetitions
- ✅ Non-trivial arrival times + interleaving r/w
- ✅ Query load & read-volume
- ✅ Realistic Scan-set faithfulness
- ✅ String- & DML-heavy

→ **Effective** benchmark for **evaluating** modern cloud **DW optimizations**

More experiments in the paper.