Predicate Caching: Query-Driven Secondary Indexing for Cloud Data Warehouses

Tobias Schmidt, Andreas Kipf, Dominik Horn, Gaurav Saxena, Tim Kraska

TUM, UTN, Amazon Webservices

13.06.2024

Motivation

Traditional indexes are not suited for the cloud:

- large data volumes
- high update costs
- slow lookup times

Cloud data warehouses rely on more lightweight caching techniques:

- result caching
- materialized views
- sorting

$\Rightarrow$ **Caches are query-driven and adapt to the workload.**

Workload Analysis

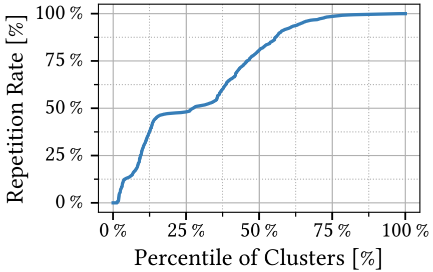
Query Repetitiveness

Caches require repetitive workloads to be effective.

Workload Analysis

Query Repetitiveness

Caches require repetitive workloads to be effective.



For more than half of the clusters, 75 % of the queries repeat.

Workload Analysis

Result Cache Hit Rate

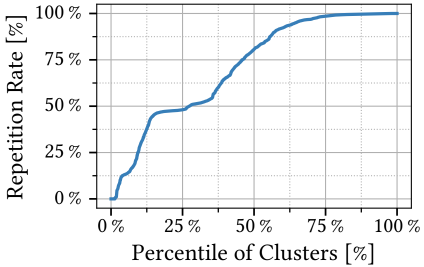


Figure: Query Repetitiveness

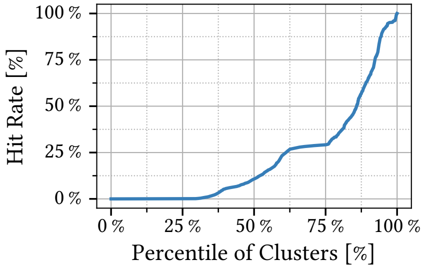


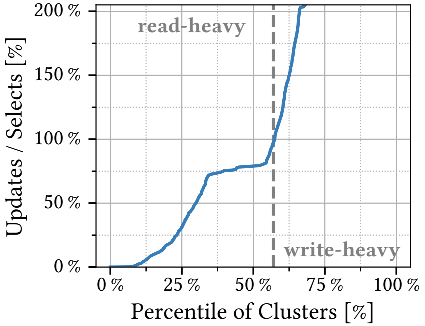
Figure: Result Cache Hit Rate

However, the result cache hit rate is relatively low.

Workload Analysis

Types of SQL Statements

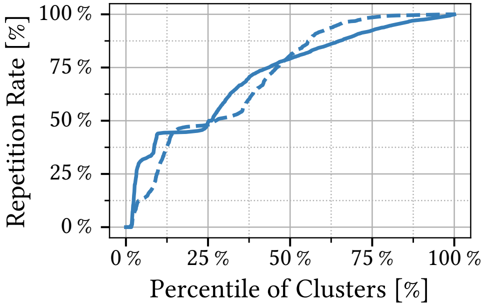
Type	Percentage
select	42.3 %
insert	17.8 %
copy	6.9 %
delete	6.3 %
update	3.6 %
other	23.3 %



60 % of the clusters execute more SELECT statements than updates.

Workload Analysis

Scan Repetitiveness



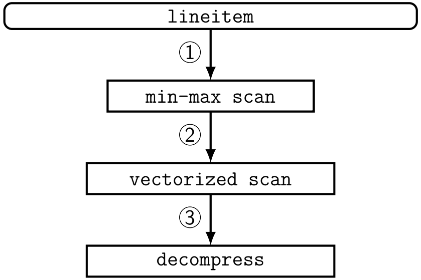
Scans and Queries are similarly repetitive.

Predicate Caching

High-Level Idea

Cache qualifying row ranges and inject them into subsequent scans.

```
select * from lineitem where l_discount = 0.1 and l_quantity >= 40
```



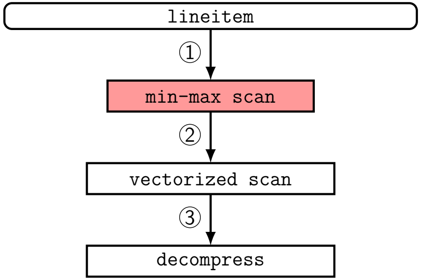
discount	quantity	
0.15	45	block 1
0.20	10	
0.10	30	
0.05	40	
0.10	40	block 2
0.05	20	
0.10	50	
0.15	40	
0.05	10	block 3
0.15	35	
0.00	20	
0.05	15	

Predicate Caching

High-Level Idea

Cache qualifying row ranges and inject them into subsequent scans.

```
select * from lineitem where l_discount = 0.1 and l_quantity >= 40
```



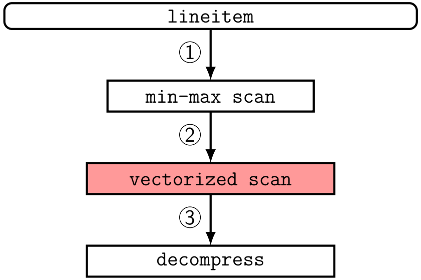
discount	quantity	
0.15	45	block 1
0.20	10	block 1
0.10	30	block 1
0.05	40	block 1
0.10	40	block 2
0.05	20	block 2
0.10	50	block 2
0.15	40	block 2
0.05	10	block 3
0.15	35	block 3
0.00	20	block 3
0.05	15	block 3

Predicate Caching

High-Level Idea

Cache qualifying row ranges and inject them into subsequent scans.

```
select * from lineitem where l_discount = 0.1 and l_quantity >= 40
```



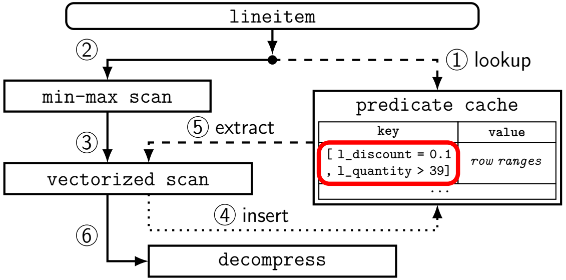
discount	quantity	
0.15	45	block 1
0.20	10	
0.10	30	
0.05	40	
0.10	40	block 2
0.05	20	
0.10	50	
0.15	40	
0.05	10	block 3
0.15	35	
0.00	20	
0.05	15	

Predicate Caching

High-Level Idea

Cache qualifying row ranges and inject them into subsequent scans.

```
select * from lineitem where l_discount = 0.1 and l_quantity >= 40
```



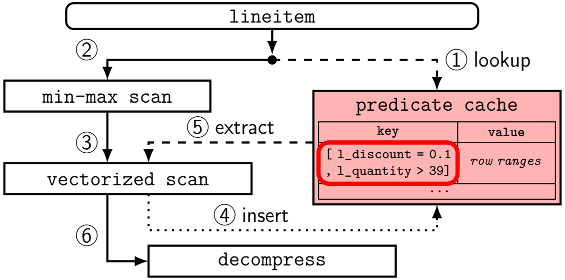
discount	quantity	
0.15	45	block 1
0.20	10	
0.10	30	
0.05	40	
0.10	40	block 2
0.05	20	
0.10	50	
0.15	40	
0.05	10	block 3
0.15	35	
0.00	20	
0.05	15	

Predicate Caching

High-Level Idea

Cache qualifying row ranges and inject them into subsequent scans.

```
select * from lineitem where l_discount = 0.1 and l_quantity >= 40
```



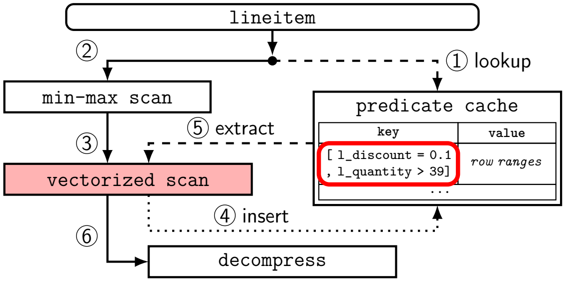
discount	quantity	
0.15	45	block 1
0.20	10	block 1
0.10	30	block 1
0.05	40	block 1
0.10	40	block 2
0.05	20	block 2
0.10	50	block 2
0.15	40	block 2
0.05	10	block 3
0.15	35	block 3
0.00	20	block 3
0.05	15	block 3

Predicate Caching

High-Level Idea

Cache qualifying row ranges and inject them into subsequent scans.

```
select * from lineitem where l_discount = 0.1 and l_quantity >= 40
```



discount	quantity	
0.15	45	block 1
0.20	10	block 1
0.10	30	block 1
0.05	40	block 1
0.10	40	block 2
0.05	20	block 2
0.10	50	block 2
0.15	40	block 2
0.05	10	block 3
0.15	35	block 3
0.00	20	block 3
0.05	15	block 3

Predicate Caching

Data Manipulation Operations

Inserts:	New tuples are appended to the end of the table; the new rows are scanned the next time.
Delete:	Rows are marked as deleted; the cached row ranges remain valid.
Update:	Combination of insert and delete.

Predicate Caching
Properties

On-the-Fly: The cache is populated as a by-product of query processing without additional build tasks.

Lightweight: Minimize resource usage, synchronization overhead, and impact on other operations

Online: Update, insert, or delete statements do not invalidate the caches' entries.

Predicate Caching

Properties

On-the-Fly: The cache is populated as a by-product of query processing without additional build tasks.

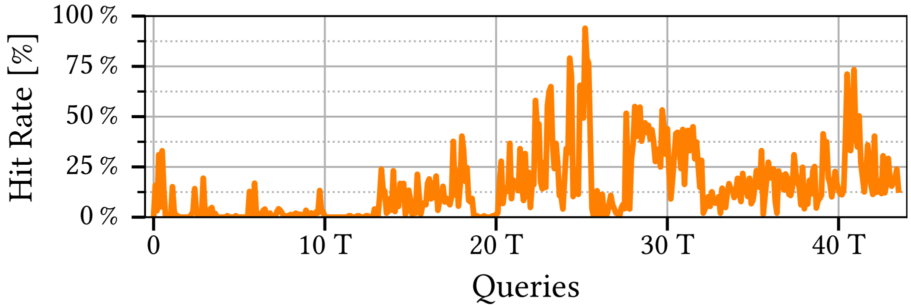
Lightweight: Minimize resource usage, synchronization overhead, and impact on other operations

Online: Update, insert, or delete statements do not invalidate the caches' entries.

$\Rightarrow$ **Predicate Caching has almost no overhead and exploits repetitive queries in cloud data warehouses.**

Results

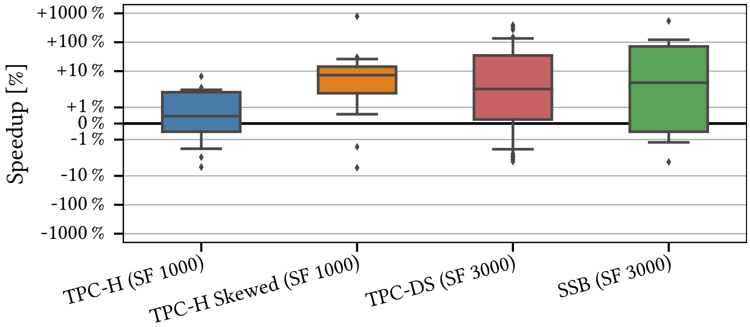
Hit Rate



⟹ **High hit rate on representative workloads.**

Results

Query Performance



Up to 10 % overall performance improvement and 10× speedup on selected queries.

Conclusion

Predicate Caching offers a lightweight, fast, and online query-driven index for Cloud Data Warehouses.

Conclusion

Predicate Caching offers a lightweight, fast, and online query-driven index for Cloud Data Warehouses.

- no build overhead, and space efficient.
- online, does not affect inserts, deletes, and updates
- significant performance improvements, in particular, on skewed data or selective queries

Conclusion

Predicate Caching offers a lightweight, fast, and online query-driven index for Cloud Data Warehouses.

Check out the full paper for more details!

https://www.amazon.science/publications/predicate-caching-query-driven-secondary-indexing-for-cloud-data-warehouses