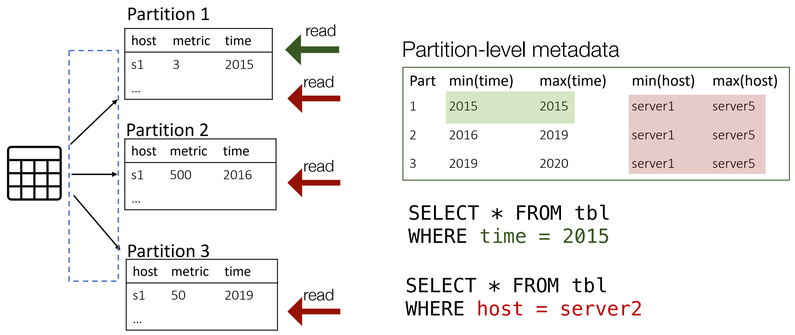
Dynamic Data Layout Optimization with Worst-case Guarantees

Kexin Rong[1,3], Paul Liu[2], Sarah Ashok Sonje[1], Moses Charikar[2]
Georgia Tech[1], Stanford University[2], VMware Research[3]

Background: Data Layout

Data layouts affect query performance

Data layout: $f(row\_id) \longrightarrow partition\_id$



Partition-level metadata

Part	min(time)	max(time)	min(host)	max(host)
1	2015	2015	server1	server5
2	2016	2019	server1	server5
3	2019	2020	server1	server5

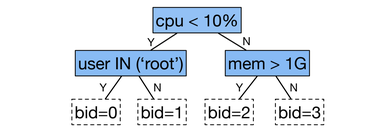
```
SELECT * FROM tbl
WHERE time = 2015

SELECT * FROM tbl
WHERE host = server2
```

Workload-aware data layouts

Example: Qd-tree [1]

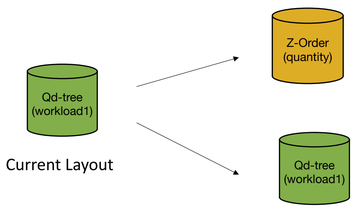
- Use workload predicates to partition data
- Efficient for target query workloads
- Performance degrades when workload changes



[1] Z. Yang, et al. Qd-tree: Learning Data Layouts for Big Data Analytics. In SIGMOD 2020.

Challenge: Adapting to Workload Changes

Trade off between query and reorganization costs



Option 1: Change layout
Reorganization cost +
Query cost -

Option 2: Do nothing
Reorganization cost
Query cost +

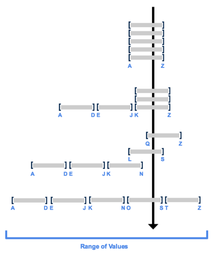
Current Practices

Heuristics-based

- Snowflake [2]: monitor "clustering depth" (#partitions that overlap at the same point)
- SAT [3]: monitor the ratio of actual query selectivity and data skipping rate
- Easy to implement and maintain
- No guarantee on performance

Based on future workload behaviors

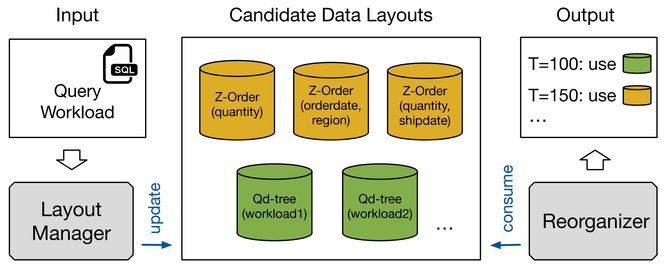
- MTO [4]: can run q more queries from the same distribution before the next workload shift
- Assumptions on workload distribution



[2] https://docs.snowflake.com/en/user-guide/tables-clustering-micropartitions
[3] X. Xie, et al. "Sat: sampling acceleration tree for adaptive database repartition," World Wide Web, vol. 26, no. 5, pp. 3503–3533, 2023.
[4] J. Ding, et al. "Instance-optimized data layouts for cloud analytics workloads,", SIGMOD 2021.

Problem Formulation

OREO Overview



Objective: minimize $\text{total cost} = \text{query} + \alpha \cdot \#\text{reorgs}$

system dependent parameter; assumed constant for simplicity

Benefits:

- Does NOT rely on assumptions of future workload
- Provide guarantees in the form of *competitive ratio*

$$\sup_I \frac{cost(online\ algorithm)}{cost(offline\ algorithm)}$$

e.g., a single data layout optimized for the entire workload

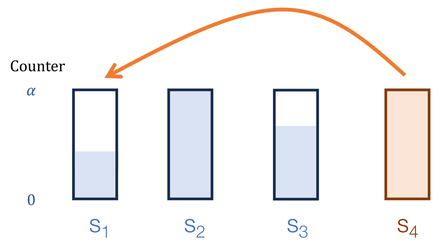
MTS with Dynamic State Space

Our contribution

- Introduce a dynamic variant of MTS (D-UMTS)
- An algorithm that solves it with a tight competitive ratio $\sim \log(|S_{max}|)$

The classic algorithm of Borodin, Linial, and Saks[1]

- For each query q: increase counter for S_i by $c(q, s_i)$
  - $c(q, s_i)$: % tables read
- When a counter is full (>= α), *randomly* switch to a state whose counter is not full



Handling dynamic state space:

- Insert: delay until next phase
- Delete: set counter to full

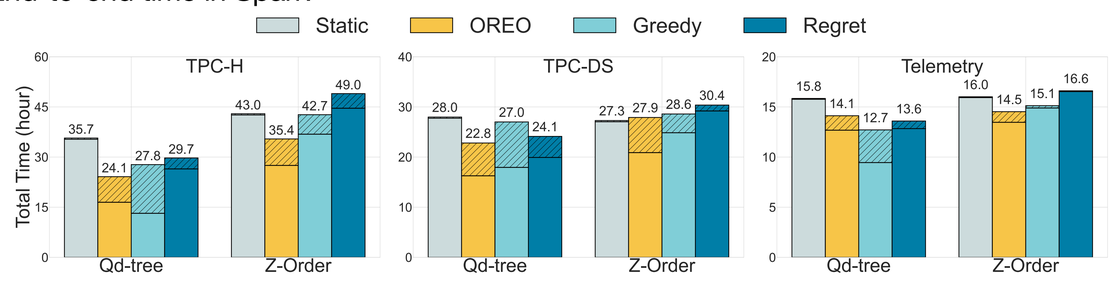
Theorem. The modified algorithm solves D-UMTS with competitive ratio $2H(|S_{max}|) \leq 2(1 + log\ |S_{max}|)$.

Extensions:

- Data-driven state switching
- Maintaining multiple query-able states

Evaluations

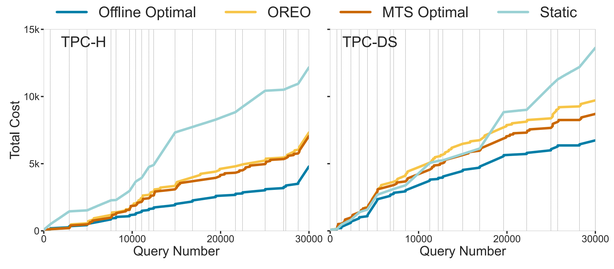
End-to-end time in Spark



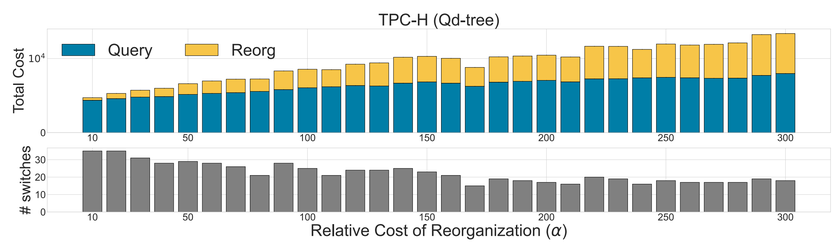
Baselines:

- **Static**: Best offline layout for the entire query workload. Minimal reorganization cost.
- **Greedy**: Switch to new layout if better. Minimal query cost.
- **Regret**: Switch to new layout if cumulative regret > reorganization cost[5]

Gap to Optimal



Effect of Reorganization Cost α



[1] Z. Yang, et al. Qd-tree: Learning Data Layouts for Big Data Analytics. In SIGMOD 2020.
[2] https://docs.snowflake.com/en/user-guide/tables-clustering-micropartitions
[3] X. Xie, et al. "Sat: sampling acceleration tree for adaptive database repartition," WWW 2023.
[4] J. Ding, et al. "Instance-optimized data layouts for cloud analytics workloads,", SIGMOD 2021.
[5] M. Daum et al., "Tasm: A tile-based storage manager for video analytics, ICDE 2021