

Rethink Query Optimization in HTAP Databases

Haoze Song, Wenchao Zhou, Feifei Li, Xiang Peng, Heming Cui

2023 SIGMOD

汇报人：王思宇 张霆轩

CONTENTS

目录

- 1 背景介绍 ↗
- 2 过往工作的不足 ↗
- 3 MetisDB 系统 ↗
- 4 Demain 成本模型 ↗
- 5 Visibility-aware 计划选择 ↗
- 6 Proactive 重优化 ↗
- 7 实验评估 ↗

背景介绍

HTAP Databases

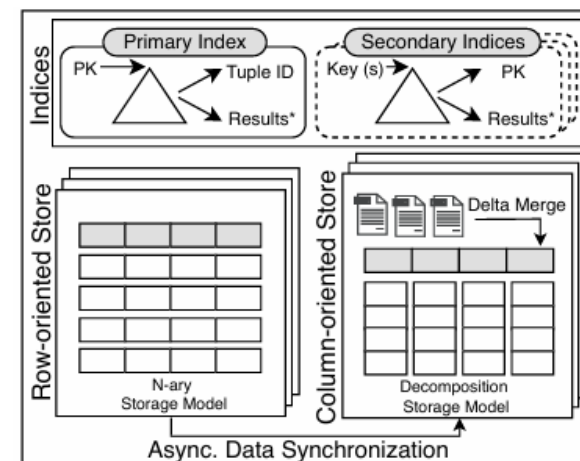
现代HTAP数据库通常维护两份数据副本来处理混合 workload:

- 按行存储 --> 适合OLTP
- 按列存储 --> 适合OLAP

基于这种混合存储架构, 查询计划可以有三种选择:

- 纯行存储计划
- 纯列存储计划
- 混合计划 (**hybrid plans**)

本文提出的工作即为对混合计划的全方位改进。



(a) Hybrid Physical Layout.

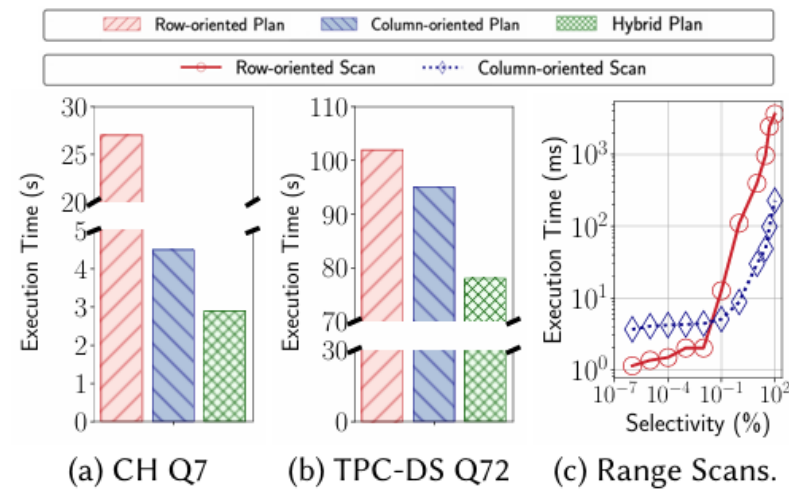
混合计划的性能优势

混合计划允许单个查询同时从行存储和列存储中获取数据，这样可以发挥两种存储格式的各自优势。比如在星型模式中，维度表适合用主键索引访问，而事实表更适合列扫描。

在不同的查询场景下，混合计划能够：

- 为维度表选择行存储索引访问
- 为事实表选择列存储扫描
- 根据查询特点动态组合最优访问路径

实验结果显示混合计划在多种查询中都能获得显著的性能提升。



HTAP 系统核心组件

系统的核心组件包括:

- **Row Store**: 存储完整的数据行, 处理所有写操作
- **Column Store**: 按列组织数据, 支持快速的列扫描
- **Delta Store**: 作为缓冲区吸收新的更新, 定期合并到列存储
- **异步数据同步机制**: 将行存储的更新传播到列存储

过往工作的不足

过往查询优化器的不足

传统的查询优化器在设计时并没有充分考虑HTAP数据库的**数据动态性**。

什么是数据动态性？

- OLTP引擎会持续执行写事务
- 不断地将新数据从行存储同步到列存储
- 这种持续的数据更新会影响查询计划的性能

本文提出的**MetisDB**将会解决这个问题：设计一个HTAP感知的查询优化器，能够捕获读操作和写操作之间的相互影响。

对HTAP的特性不敏感的Hybrid Plan的问题

传统的HTAP数据库在生成混合计划时存在一个核心问题：

基本上是把列扫描当作一个额外的数据访问路径加到现有的成本模型中，**进而忽略了HTAP环境的特殊性。**

这种做法会导致三个主要问题：

1. Data Synchronization的影响

问题核心： 在HTAP系统中，列存储需要通过delta store来吸收新的更新。

具体表现：

- 执行列扫描时，系统必须合并delta store和主列存储的数据
- 这个过程需要使用k-way merge算法，带来额外开销

实验发现：

- 随着OLTP并发量增加，列扫描的执行时间会**线性增长**
- 行扫描和列扫描性能相等的选择性阈值会随工作负载变化
- **没有固定的阈值**可以用来做访问路径选择

2. Data Freshness的影响

问题核心： 由于数据是异步从行存储同步到列存储的，列存储的数据新鲜度总是落后于行存储。

可见性延迟（visibility delay）问题：

- 新的更新从在行存储提交到能在列存储中读取之间有时间差
- 在高并发的OLTP工作负载下，延迟可能达到**10秒甚至更长**

性能矛盾：

- 即使列扫描在理论上比行扫描更快，但如果查询必须等待数据同步完成，总响应时间反而可能更长

3. Performance Isolation的影响

问题核心： 当使用混合计划时，OLAP查询会同时访问行存储和列存储，**打破了原本的性能隔离。**

具体影响：

- 在行存储负载较高时，混合计划会与OLTP事务竞争CPU、内存等资源
- 导致OLTP吞吐量下降多达**58%**
- 混合计划本身的性能也会因为资源竞争而下降

现有解决方案的局限性：

- 传统的配额限制方法（比如限制OLAP只能访问500MB以下的表）太过僵化
- 无法适应不同的工作负载变化

MetisDB的改进

MetisDB针对这些问题提出了三个关键技术：

- 1. **Delta-aware的成本模型 Demain** - 精确建模数据访问路径的代价
- 2. **Visibility-aware的计划选择算法** - 通过预执行掩盖可见性延迟
- 3. **Resource-aware的重优化算法** - 保证性能隔离的关键机制

这些技术共同构成了HTAP感知的查询优化器，能够在保持性能隔离的同时充分发挥混合计划的优势。

Desiderata	Roadmaps of METIS	Design Knobs	Proposed Solution
Low Execution Time	Hybrid Plans	Cost Model	Demain Model
High Data Freshness	Masking the Cost of Visibility Delay	Plan Selection Algo.	Visibility-aware Optimizations
Strong Performance Isolation	Restricting the Abuse of Hybrid Plans	Re-optimization Algo.	Proactive Re-optimization with Plan Stitch

Table 1. Desideratas, roadmaps, design knobs, and solutions of METIS for efficient query optimization in HTAP Databases.

MetisDB

MetisDB

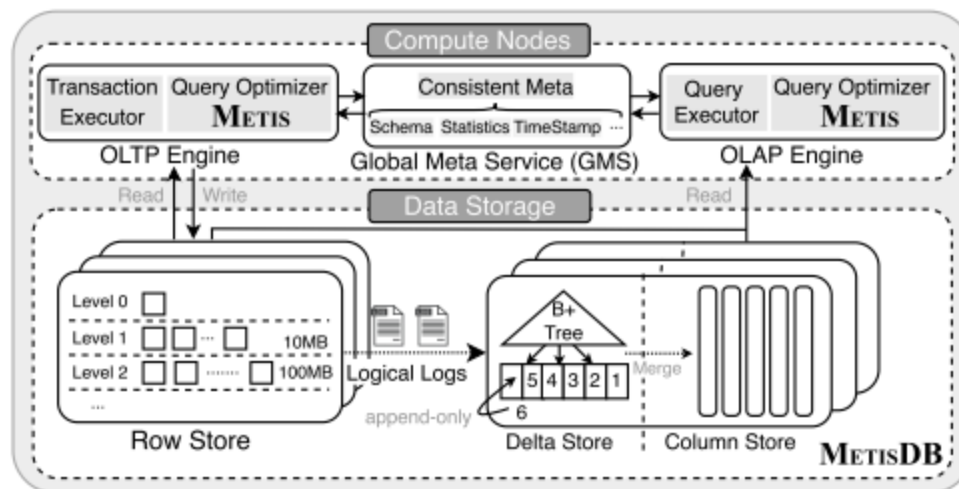


Fig. 6. An overview of METISDB.

Metis算法基于MetisDB，遵循存算分离架构：

- 行存储：LSM-TREE
- 列存储：基于append-only的B+树
- 计算单元：OLTP事务，OLAP事务，使用一个Consistent Meta进行内容统一控制（基于时间戳）

查询计划的优化使用**METIS**。

Metis

Metis算法分为代价评估，计划选择和执行两部分:

256:9

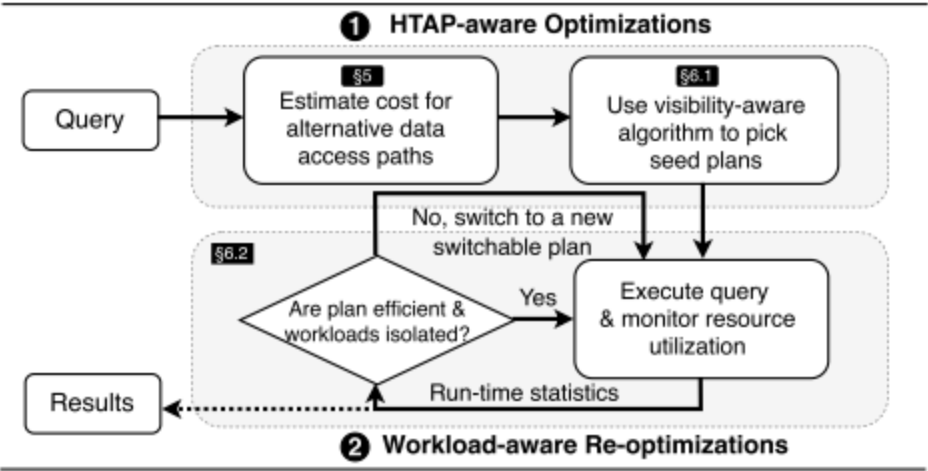


Fig. 7. METIS’s optimization workflow.

Domain

Demain

代价评估基于增量存储感知模型（ Demain ）和基数规模进行估计

代价评估部分：

以 *SELECT col1 FROM table WHERE col1 between \${a} and \${b}* 为例

分布式系统的网络延迟：

$$Cost_{net} = \frac{sel \cdot N \cdot w_{res}}{B_{net}} + L_{net}$$

行扫描：

$$Cost_{row} = T \cdot N \cdot \max \left\{ \frac{ts}{DB_{row}^{seq}}, \frac{ts}{MB_{row}^{seq}}, f_p \cdot p \right\} \approx \frac{T \cdot N \cdot ts}{DB_{row}^{seq}}$$

Query	<i>sel</i>	Selectivity of query q (%)
	<i>w_{res}</i>	Results width (bytes per output tuple)
Dataset	<i>N</i>	Number of tuples
	<i>ts</i>	Tuple size (bytes per tuple)
Hardware Resource	<i>B_{net}</i>	Network bandwidth (bytes/s)
	<i>L_{net}</i>	Latency of in-Network delay (s)
	<i>DB_{row}</i>	Disk bandwidth of read in row store (bytes/s)
	<i>DB_{col}</i>	Disk bandwidth of read in column store (bytes/s)
	<i>MB_{row}</i>	Memory bandwidth of read in row store (bytes/s)
	<i>MB_{col}</i>	Memory bandwidth of read in column store (bytes/s)
	<i>p</i>	The inverse of CPU frequency
	<i>f_p</i>	Factor accounting for instruction pipeline
	<i>f_{vec}</i>	Factor accounting for vectorized OLAP engine
Storage	<i>T</i>	Ratio of the size amplification in LSM tree
	<i>w_a</i>	Attribute width (bytes)
	<i>w_{id}</i>	RowID width (bytes)
	<i>b</i>	B+ tree fanout for delta store
	<i>w_k</i>	Key width of the index (bytes)
	<i>N_d</i>	Delta size (Unconsolidated tuples per column)

Demain Cost Model

行索引遍历:

$$Cost_{index} = Cost_{IndexTraversal} + Cost_{DataTraversal}$$

$$\approx \frac{T \cdot N \cdot (w_k + w_{id})}{DB_{row}^{seq}} + \frac{T \cdot N \cdot (w_{id} + sel \cdot ts)}{DB_{row}^{rand}}$$

列扫描:

1. 增量树查找:

$$Cost_{TreeTraversal} = (1 + \lceil \log_b(N_d) \rceil) \cdot \frac{b}{2} \cdot (f_p \cdot p + \frac{1}{MB_{col}^{rand}})$$

$$Cost_{DataTraversal} = N_d \cdot \frac{(w_{id} + sel \cdot ts)}{DB_{col}^{rand}}$$

$$Cost_{delta} = Cost_{TreeTraversal} + Cost_{DataTraversal}$$

Query	sel	Selectivity of query q (%)
	w_{res}	Results width (bytes per output tuple)
Dataset	N	Number of tuples
	ts	Tuple size (bytes per tuple)
Hardware Resource	B_{net}	Network bandwidth (bytes/s)
	L_{net}	Latency of in-Network delay (s)
	DB_{row}	Disk bandwidth of read in row store (bytes/s)
	DB_{col}	Disk bandwidth of read in column store (bytes/s)
	MB_{row}	Memory bandwidth of read in row store (bytes/s)
	MB_{col}	Memory bandwidth of read in column store (bytes/s)
	p	The inverse of CPU frequency
	f_p	Factor accounting for instruction pipeline
Storage	f_{vec}	Factor accounting for vectorized OLAP engine
	T	Ratio of the size amplification in LSM tree
	w_a	Attribute width (bytes)
	w_{id}	RowID width (bytes)
	b	B+ tree fanout for delta store
	w_k	Key width of the index (bytes)
	N_d	Delta size (Unconsolidated tuples per column)

Demain优化策略

2. 稳定列查找:

$$Cost_{col}^* = N \cdot \max \left\{ \frac{w_a}{DB_{col}^{seq}}, \frac{w_a}{MB_{col}^{seq}}, \frac{f_p \cdot p}{f_{vec}} \right\} \approx \frac{N \cdot w_a}{DB_{col}^{seq}}$$

总代价:

$$Cost_{col} = Cost_{col}^* + Cost_{delta}$$

与传统方式一样，Metis参考sel进行选取，如果sel较小会考虑直接进行行读取，避免过多的IO

相较于传统方法，Metis在计算列存储时会考虑增量B+树的遍历代价，使得其会在sel适中时更偏向行存储

Query	<i>sel</i>	Selectivity of query q (%)
	<i>w_{res}</i>	Results width (bytes per output tuple)
Dataset	<i>N</i>	Number of tuples
	<i>ts</i>	Tuple size (bytes per tuple)
Hardware Resource	<i>B_{net}</i>	Network bandwidth (bytes/s)
	<i>L_{net}</i>	Latency of in-Network delay (s)
	<i>DB_{row}</i>	Disk bandwidth of read in row store (bytes/s)
	<i>DB_{col}</i>	Disk bandwidth of read in column store (bytes/s)
	<i>MB_{row}</i>	Memory bandwidth of read in row store (bytes/s)
	<i>MB_{col}</i>	Memory bandwidth of read in column store (bytes/s)
	<i>p</i>	The inverse of CPU frequency
	<i>f_p</i>	Factor accounting for instruction pipeline
Storage	<i>f_{vec}</i>	Factor accounting for vectorized OLAP engine
	<i>T</i>	Ratio of the size amplification in LSM tree
	<i>w_a</i>	Attribute width (bytes)
	<i>w_{id}</i>	RowID width (bytes)
	<i>b</i>	B+ tree fanout for delta store
	<i>w_k</i>	Key width of the index (bytes)
	<i>N_d</i>	Delta size (Unconsolidated tuples per column)

Visibility-aware Plan Selection

Visibility-aware Plan Selection

进行代价估计之后，首先排除所有远非最优的计划，然后利用 Visibility-aware 算法生成一系列种子计划，用于后续的进一步优化和执行：

Visibility-aware Plan Selection 通过提前调度可用数据上面的预执行，从而掩盖 HTAP 数据库中的可见性延迟。

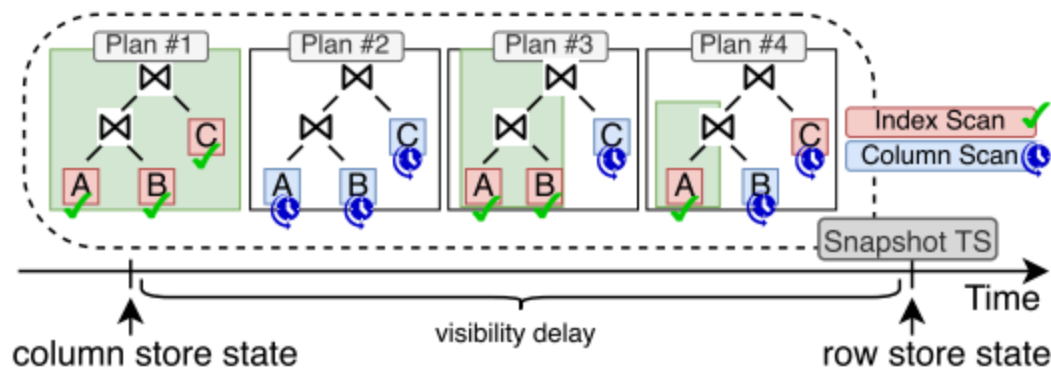


Fig. 8. An example of visibility-aware plan selection. The parts in green are ready to be executed.

Visibility-aware Plan Selection

可见性延迟的来源:

1. 新的写入被提交到数据传输管道中, 然后从行存储运送到列存储
2. 传输的逻辑日志由delta-store解析
3. 新的更新被附加到delta-store中并由B+树索引
4. 更新被提交到delta-store中并最终对查询可见

估计方法:

Metis使用历史数据估计可见性延迟, 并将OTLP的工作负载的并发映射到可见性延迟, 同时全局元服务持续收集新的测量值, 再运行时进行校准。计算每个并发的可见性延迟移动平均值(最后30个样本), 用多项式回归曲线求解

Visibility-aware Plan Selection

算法流程如图:

Algo 1: Visibility-aware Plan Selection (§6.1).

```
1 Para:  $G_p \leftarrow$  The DAG representation of a hybrid plan  $p$ 
2 Para:  $M \leftarrow$  The cost model that estimates cost in seconds
3 Para:  $\alpha \leftarrow$  The visibility delay in the HTAP database
4 Function planSelection(DAGs,  $\alpha$ ) do
5    $bestPlan \leftarrow \emptyset$ ;  $Cost_{best} \leftarrow 0$ ;
6   for each  $G_p$  in DAGs do
7      $Cost_p \leftarrow \text{calculateCost}(G_p, \alpha)$ ;
8     if  $bestPlan == \emptyset \vee Cost_p < Cost_{best}$  :
9        $bestPlan = p$ ;  $Cost_{best} \leftarrow Cost_p$ ;
10  return  $bestPlan$ ;
11 Function calculateCost( $G_p, \alpha$ ) do
12   $preCost \leftarrow 0$ ;  $exeCost \leftarrow \text{estimateCost}(G_p, M)$ ;
13   $invalidSet \leftarrow \text{findAllPendingTasks}(G_p)$ ;
14   $subs \leftarrow \text{removePendingTasks}(G_p, invalidSet)$ ;
15  /* removePendingTasks returns the strongly connected components after removing pending
16     tasks in  $G_p$ . */
17  foreach  $sub \in subs$  do
18     $Cost_{sub} \leftarrow \text{estimateCost}(sub, M)$ ;
19    if  $preCost < Cost_{sub}$  :  $preCost \leftarrow Cost_{sub}$ ;
20  return  $Cost \leftarrow exeCost + \alpha - \min\{preCost, \alpha\}$ ;
```


Visibility-aware Plan Selection

进行计划枚举时，将每一个物理计划表现为**有向无环图**。

1. 顶点：物理操作符。
2. 有两种边：**硬依赖**（后一个操作必须等待前一个操作的全部输出），**软依赖**（两者可以流水线执行）

执行算法时：

1. 首先通过删除图中unavailable pending tasks来计算每个计划的预执行（pre-execution）tasks。由于并行性，pre-execution的整体性能改进应该是最长任务的执行时间。
2. 随后将visibility delay的知识结合到第 18 行的计划成本中，该成本捕获了用户观察到的端到端查询延迟。
3. 最后使用修订后的成本来指导选择函数并生成一个具有最低查询延迟成本的visibility-aware物理计划。

Proactive Query Re-optimizations

Proactive Query Re-optimizations

在执行查询优化时，Metis 会生成一组种子计划（即面向行的计划、面向列的计划和 HTAP 感知计划）以节省重新优化的成本。

在执行查询时，Metis 从最便宜的计划开始，并根据运行时统计数据不断重新优化计划。

有三个优化点：

1. 种子计划
2. Runtime Statistics 考量
3. Plan stitch

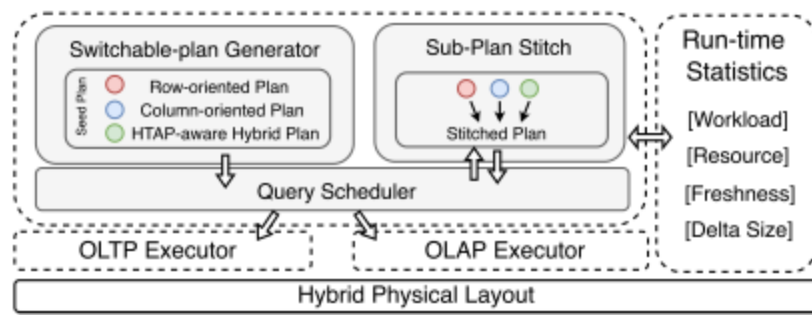
种子计划

由之前的算法给出。子计划都具有相同的逻辑结构，但可能采用不同的物理运算符。

通常，当存在优于row-oriented和column-oriented计划的混合计划时，种子计划由三个物理计划组成；否则，种子计划由两个计划组成。不会单独优化面向行和面向列的计划的逻辑结构。

Runtime Statistics

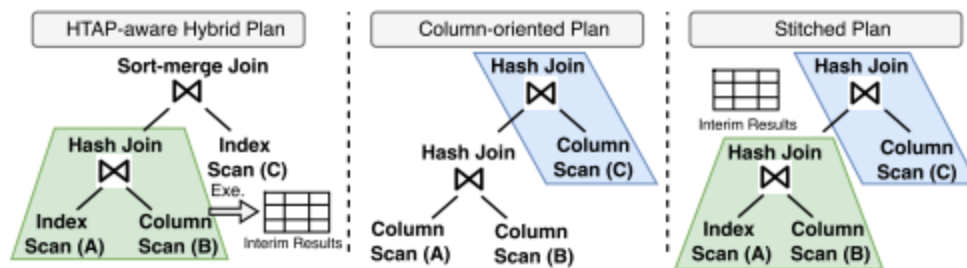
Metis 会根据运行时统计信息定量评估其决策的影响。为了高效执行，当估计基数与实际统计数据相差甚远（例如超过 20%）时，Metis 会重新优化物理运算符。



(a) Proactive plan re-optimization in METIS.

Plan Stitch

在重新优化计划时，Metis 会从种子计划中拼接子计划，而不会丢失之前的进度，重新优化本质在于对剩余未执行计划的物理运算符的选择进行重新优化。



(b) An example of combining sub-plans.

实验评估

实验设置与结果

通过实验验证了Metis的有效性:

实验环境

- **硬件:** 7台机器集群, 每台机器24核CPU、64GB内存
- **工作负载:** CH-benCHmark、TPC-DS、YCSB三种基准测试

核心实验结果

混合计划的有效性:

- CH-benCHmark: 40.9%的查询受益于混合计划, 平均获得1.68倍加速
- TPC-DS: 77.8%的查询受益, 平均3.06倍加速
- 最高加速: Q72查询达到了11倍加速

性能隔离的维护:

- Metis能够很好地维护OLTP和OLAP之间的性能隔离
- 即使在高负载情况下, OLTP吞吐量下降也控制在8%以内
- 而传统方法可能导致58%的性能下降

关键发现

三个核心技术的不同贡献:

1. **Demain成本模型** - 性能提升的关键
 - 能够纠正传统方法的次优计划选择
 - 次优的计划选择可能导致高达**16.4倍的性能下降**
2. **可见性感知的计划选择** - 进一步性能提升
 - 通过预执行技术掩盖可见性延迟
3. **资源感知的重优化** - 保证性能隔离的关键
 - 即使成本模型完全准确, 重优化机制仍然不可或缺
 - 能处理成本模型无法覆盖的资源冲突问题

实验结论

Metis成功实现了三个设计目标:

- 在不同HTAP场景下有效利用混合计划
- 保持OLTP和OLAP间强性能隔离
- 对工作负载变化具有鲁棒性和适应性