HTAP简述及
面向HTAP的存储数据格式

汇报人：任勇闯

DSC
分布式存储与计算实验室

2022.04.01

目录

01 ／ OLTP&OLAP概述

01 OLTP&OLAP概述

分布式存储与计算实验室

1.1 OLTP

什么是OLTP

- On-Line Transaction Processing
- 在线事务处理/在线事务交易
- 使得大量的用户通过 Internet 实时执行大量数据库事务
- OLTP最基本的要求：原子性




01 OLTP&OLAP概述

分布式存储与计算实验室

1.1 OLTP

- N-ary Storage Model
- DBMS将单个元组的所有属性存储在一段连续的内存中
- 数据加载非常快
- 插入、删除、修改的速度非常快

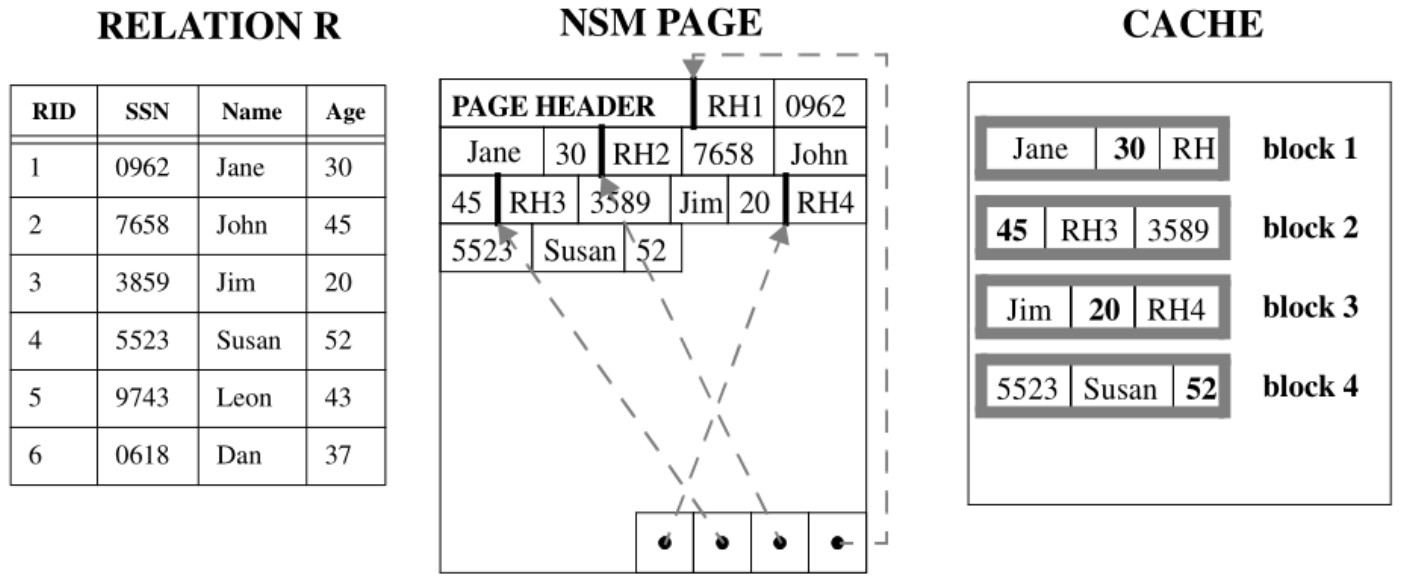


FIGURE 1: The N-ary Storage Model (NSM) and its cache behavior. *Records in R (left) are stored contiguously into disk pages (middle), with offsets to their starts stored in slots at the end of the page. While scanning* age, *NSM typically incurs one cache miss per record and brings useless data into the cache (right).*

01 OLTP&OLAP概述

分布式存储与计算实验室

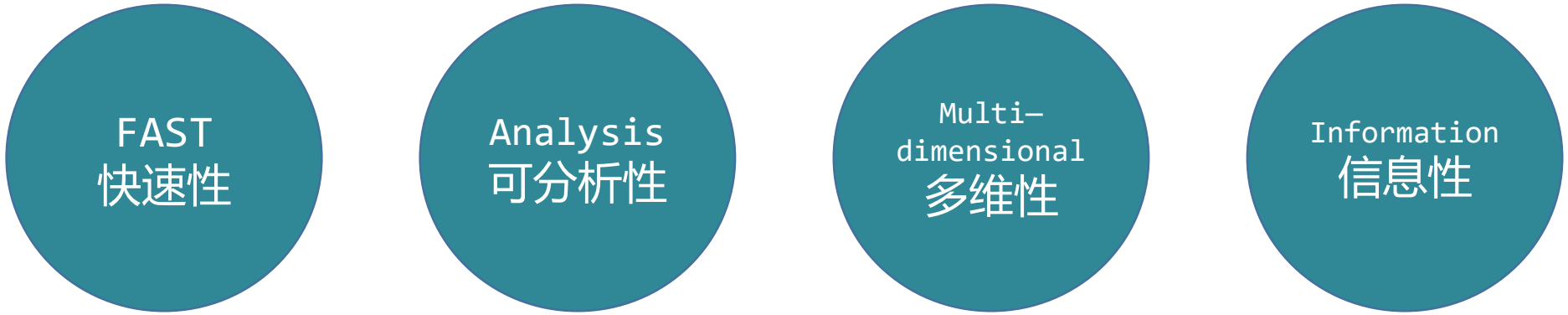
1.1 OLAP

On-Line Analytical Processing

单个数据没有价值，但聚在一起的数据就会蕴含巨大的财富

OLAP（在线分析处理）在 1993 年提出

主要能力就是计算分析决策



01 OLTP&OLAP概述

分布式存储与计算实验室

1.1 OLAP

Decomposition Storage Model

优点:
- 读取指定属性缓存利用率高
- 列存压缩率高
- 适合OLAP系统

缺点:
- 操作事务对元组的读写效率低
- 不适合OLTP事务

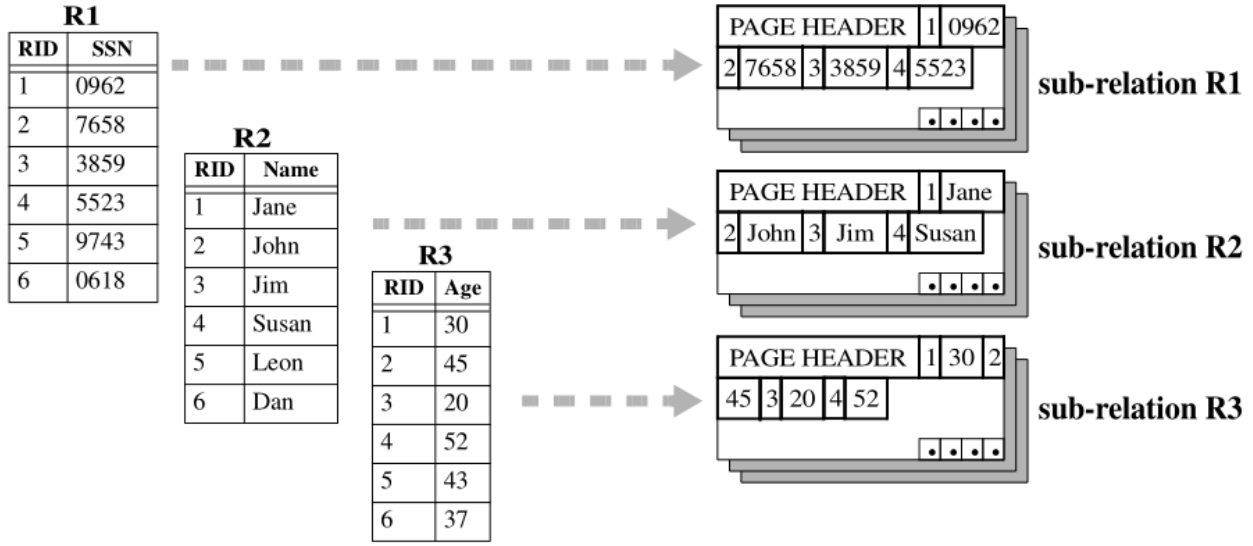


FIGURE 2: The Decomposition Storage Model (DSM). *The relation is partitioned vertically into one thin relation per attribute. Each sub-relation is then stored in the traditional fashion.*

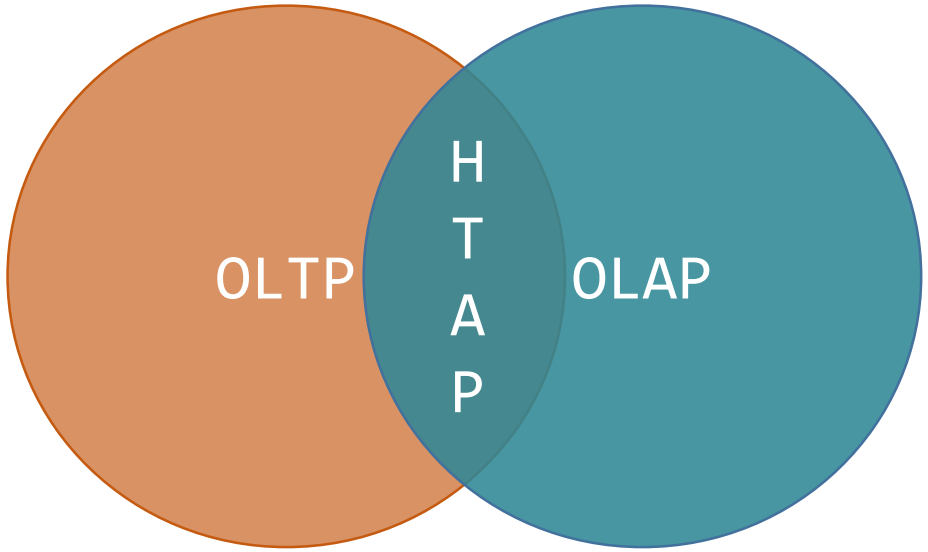
02 HTAP概述

02 HTAP概述

分布式存储与计算实验室

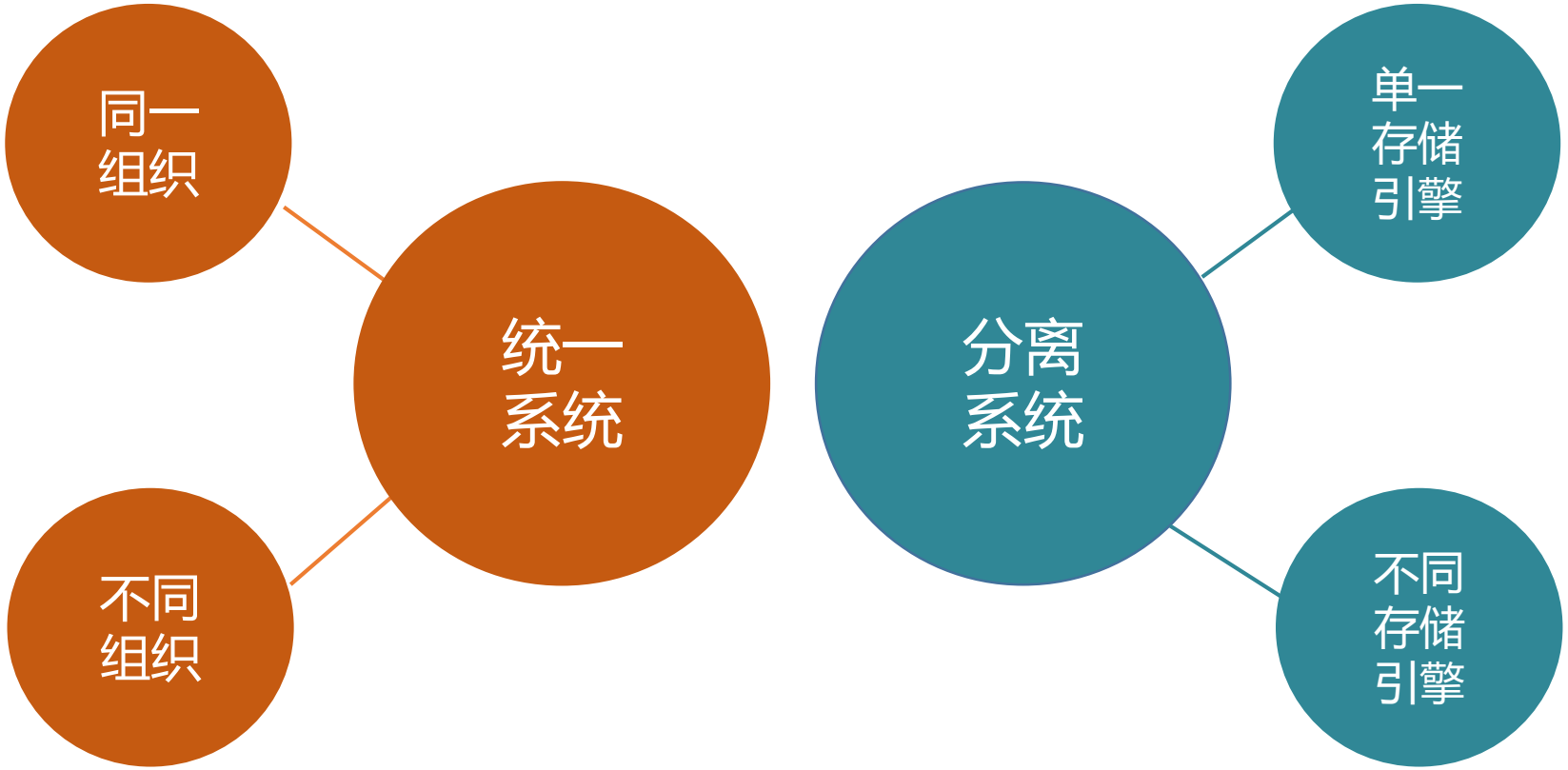
2.1 HTAP

- Hybrid transactional/analytical processing
- 实时分析、事务处理
- 简化架构
- 降低运维成本
- 使用便捷
- 应用场景
  - 实时推荐
  - 物流跟踪
  - 风控



HTAP概述

2.2 HTAP实现方案



参考资料：Hybrid Transactional/Analytical Processing: A Survey

HTAP概述

2.2.1 不同系统不同存储引擎

实现方式：CDC、分布式协议、RDMA

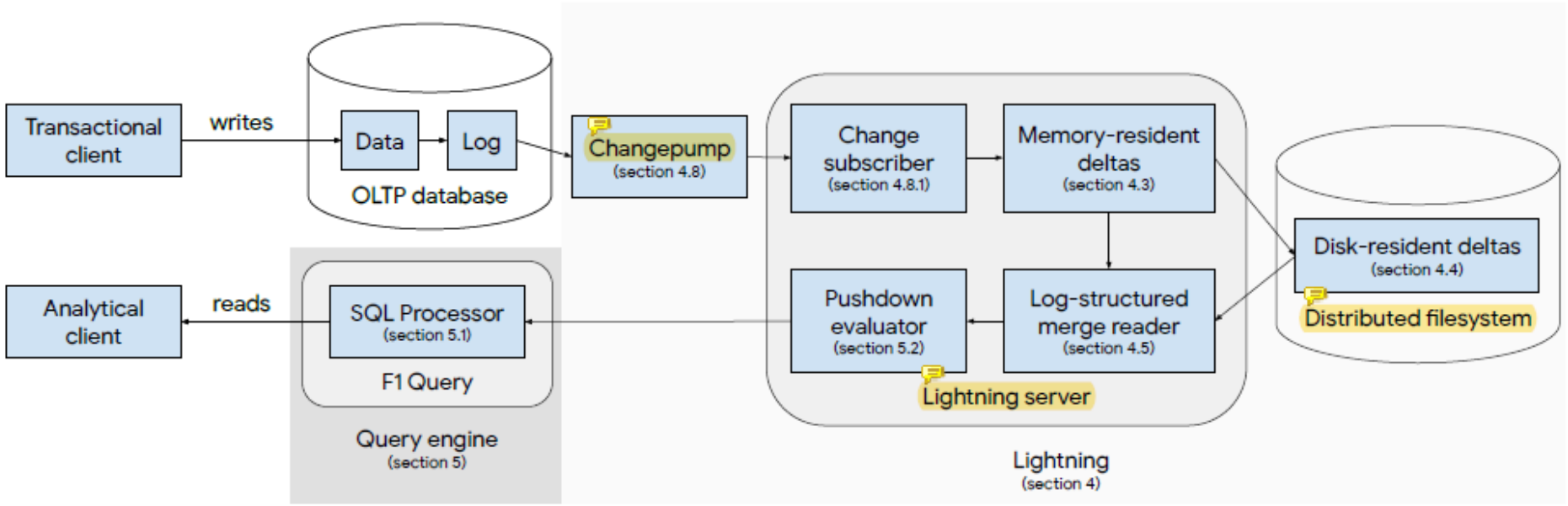


Figure 1: An overview of how data flows through an HTAP system with Lightning. Details on Changepump and the Lightning server architecture are described in section 4, and details on Lightning's integration with F1 Query are described in section 5.

02 HTAP概述

2.2.1 不同系统不同存储引擎

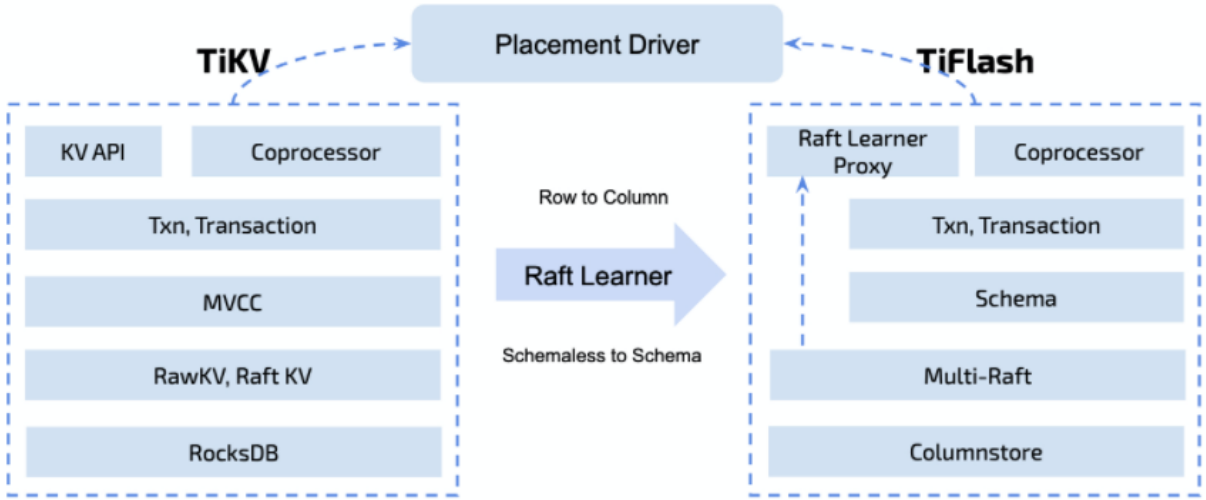
实现方式：CDC、分布式协议、RDMA

数据备份的问题：

1．实时数据

2．数据一致性

3．可维护性



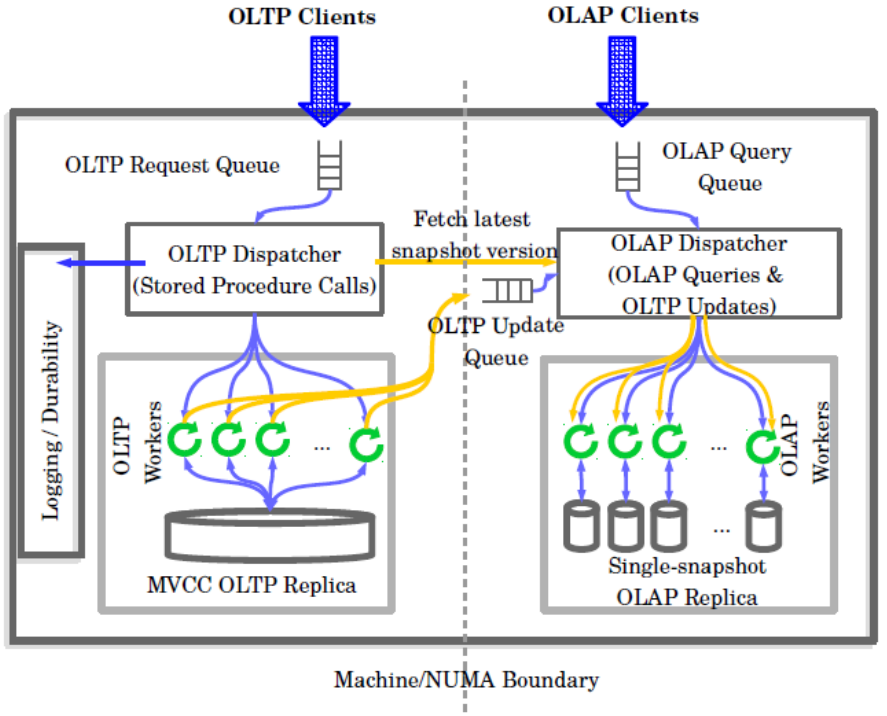
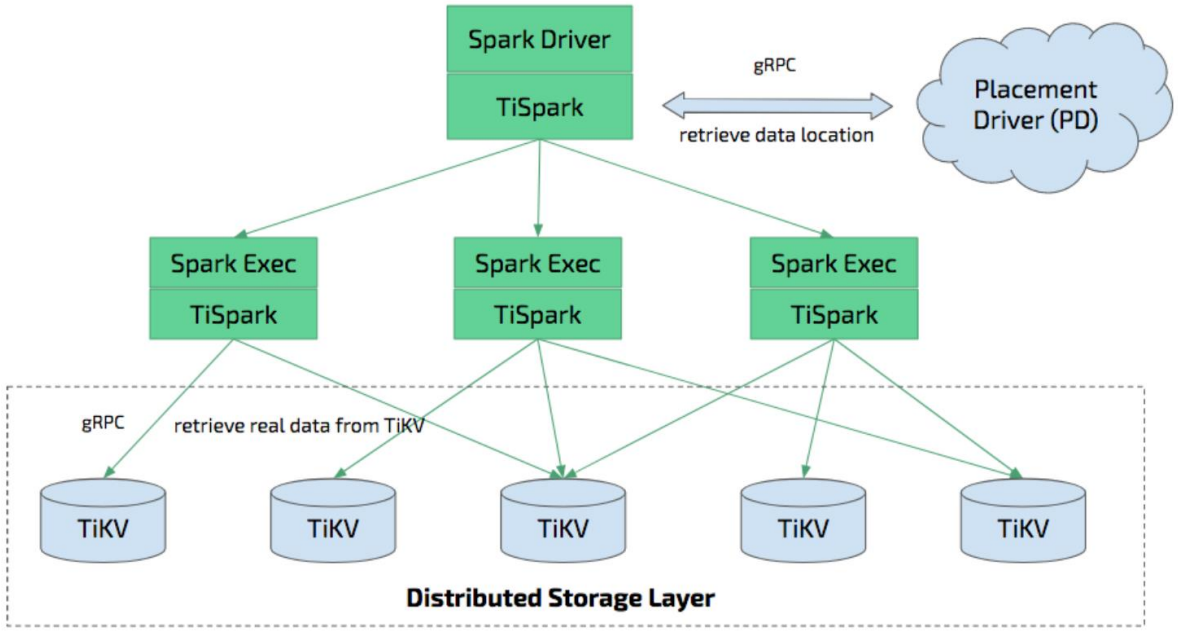


Figure 1: System architecture

02 HTAP概述

2.2.2 不同系统同一存储引擎

实现方案：TP扩展AP；AP扩展TP



Spark：MapReduce优化后的大规模处理通用计算引擎

MPP：大规模并行架构

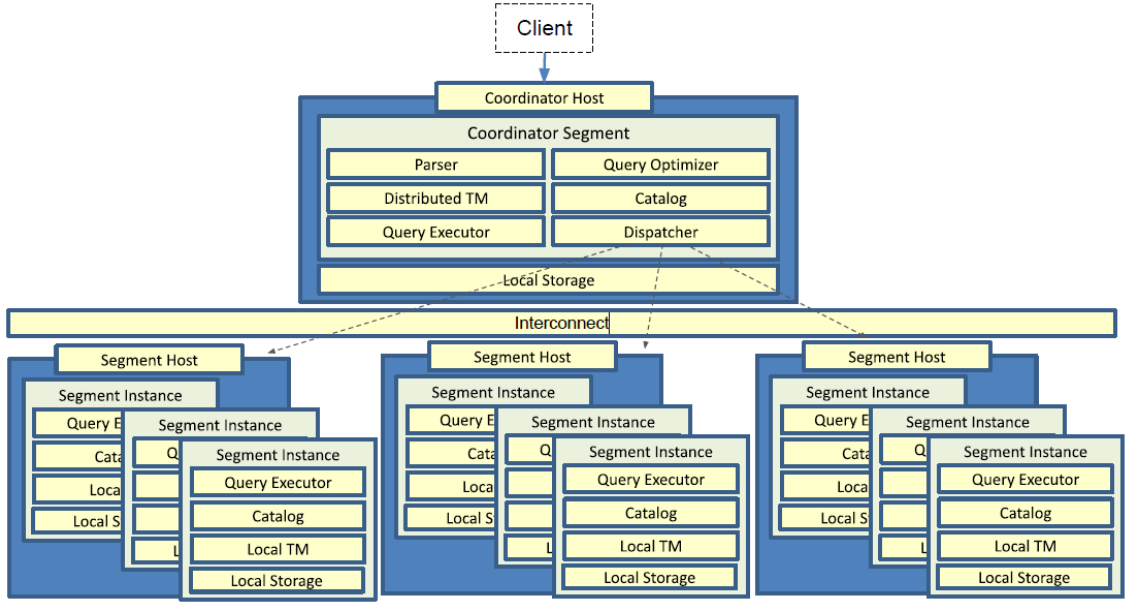


Fig. 3. Greenplum's Architecture

HTAP概述

2.2.3 统一系统相同存储格式

实现方案：SQL-On-Hadoop、HHTAP、混合存储格式

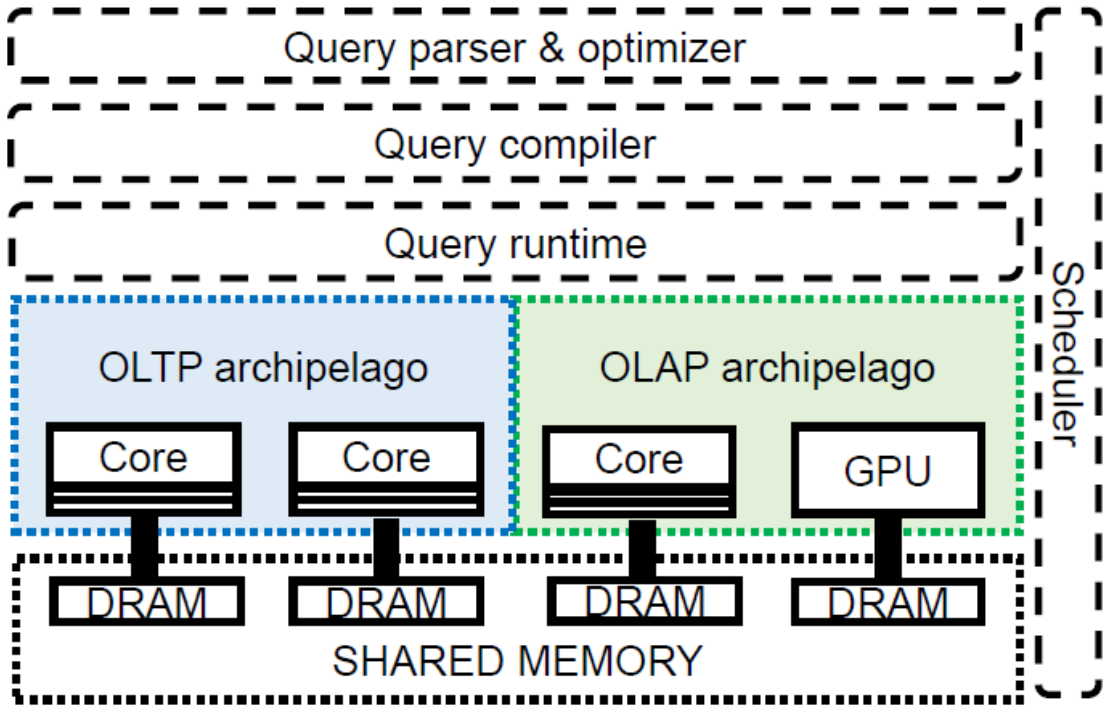


Figure 2: H^2TAP deployed over emerging server hardware.

HHTAP：

1. Heterogeneous HTAP

2. 非缓存一致的共享内存多核设计

3. GPGPU 通用图形处理器
   1. 可编程行和通用接口调用
   2. 支持CPU通用计算任务
   3. 并行处理、可编程的流水线

4. CPU调度TP操作；GPGPU调度AP操作

2.2.4 统一系统不同存储格式

实现方案：数据冷热分离、数据存储布局动态变更

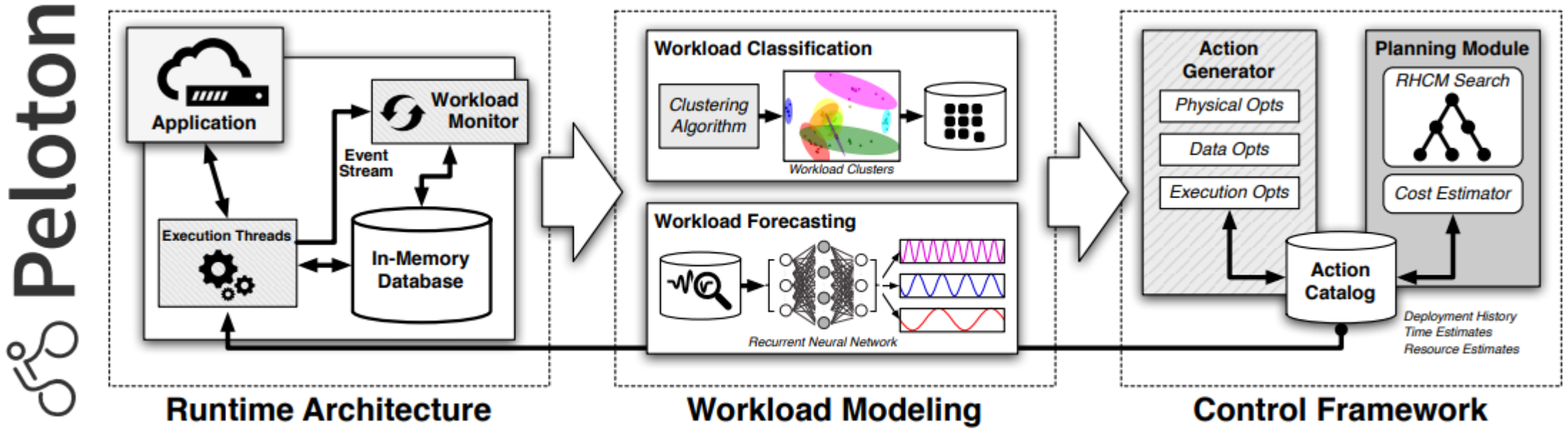
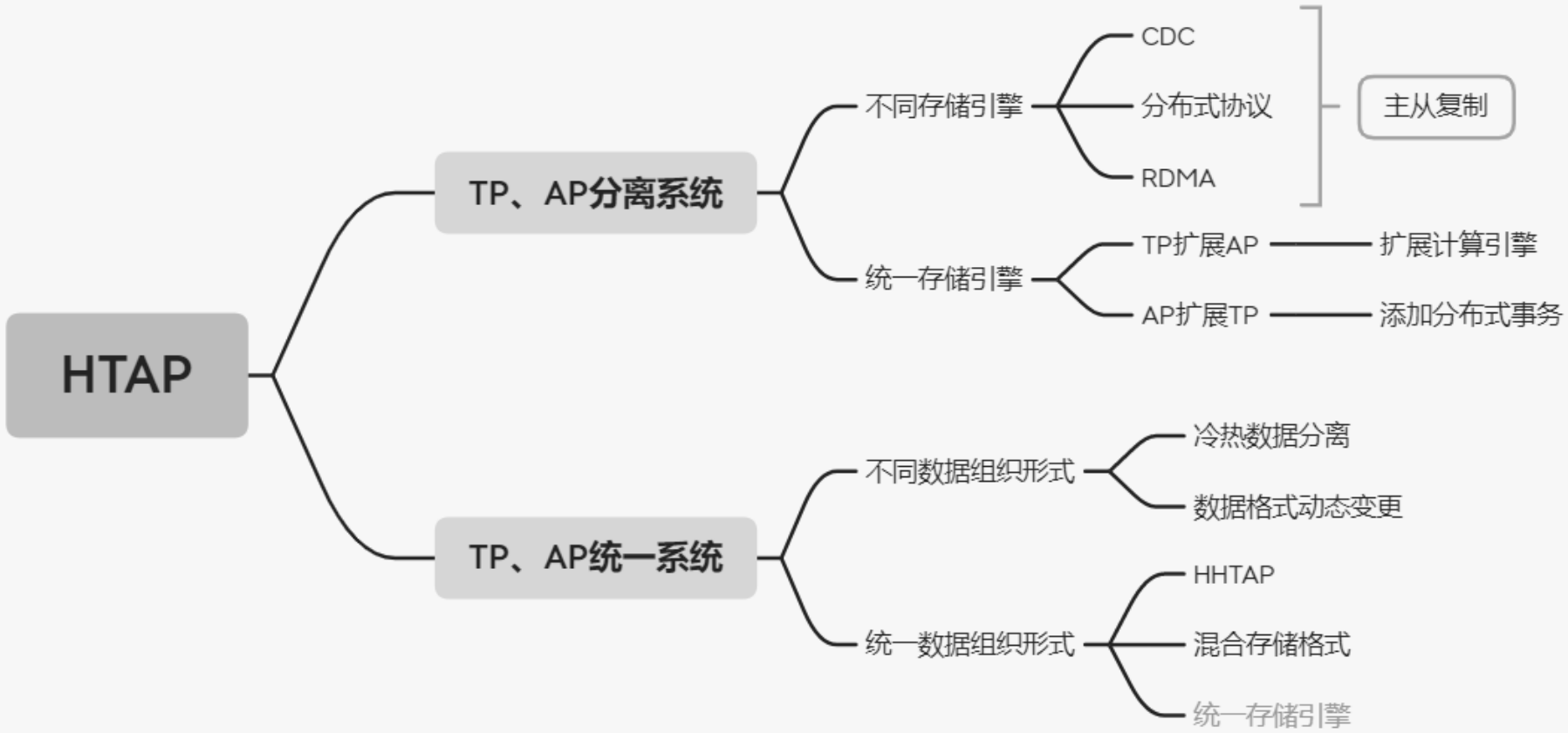


Figure 1: Peloton Self-Driving Architecture – An overview of the forecasting and runtime workflow of Peloton's self-driving components.

HTAP概述

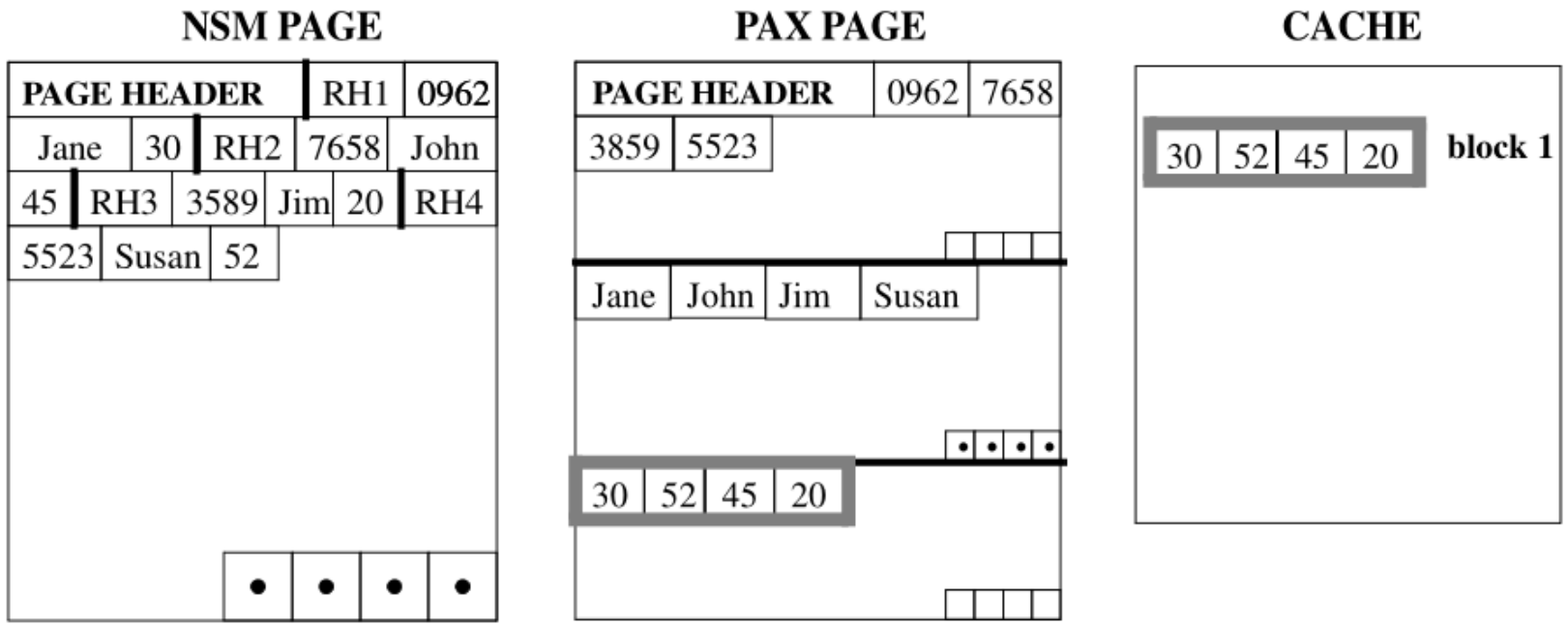
2.3 htap实现方案总结



03 ／ 混合存储格式-PAX

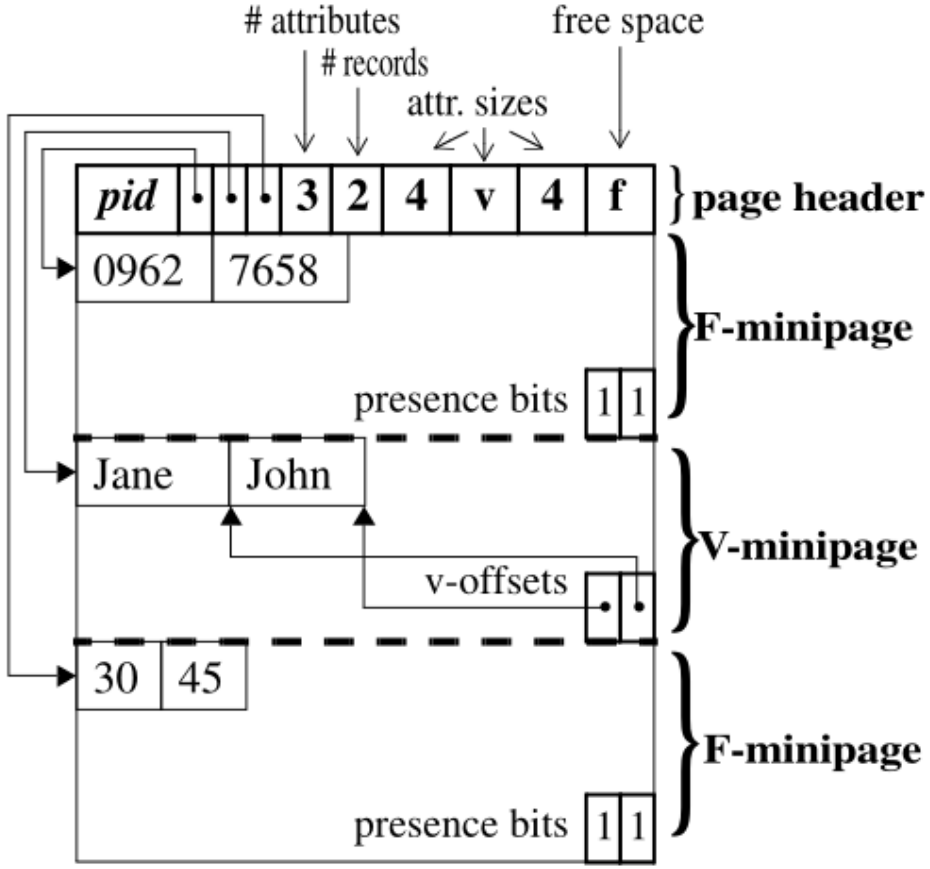
混合存储格式

- Partition Attributes Across
- 修复NSM的缓存行为，同时又不损害其相对于DSM的优势
- 该模型通过将每个页面中每个属性的所有值分组在一起来显著提高缓存性能。



混合存储格式

- N个属性有N个minipage，依次对应存储
- 每个页面的开头都有一个页头，它包含到每个迷你页面开头的偏移量
- 固定长度的属性值存储在F-minipages中
- 可变长度属性值存储在V-minipages中



03 混合存储格式

插入操作

- 当数据插入的时候，PAX会首先生成一个新的page，然后根据属性的大小分配好不同的mini page
- 这里需要注意变长属性，PAX会使用一些前几条数据来得到一个平均的属性大小。
- PAX会将这个元组里面的数据分别复制到不同的mini page上面。
  - 如果一个元组还能插入到这个page，但这个元组里面某一个某一个的属性不能插入到对应的mini page了，PAX 会重新调整不同mini page的边界。
  - 如果一个page已经满了，那么PAX就会重新分配一个page

03 混合存储格式

更新操作

- PAX会首先计算这个元组需要更新的属性在不同mini page里面的偏移量
- 对于可变属性来说，如果更新的数据大小超出了mini page可用空间，mini page就会尝试向周围的mini page借一点空间。
  - 如果邻居也没有额外的空间了，那么这个元组就会被移到新的page上面。

删除操作

- 当数据删除的时候，PAX会在page最开始会维护一个bitmap，用来标记删除的数据。
- 当删除标记越来越多的时候，就可能会影响性能，因为会导致mini page里面出现很多gap，并不能高效的利用cache。所以PAX会定期去对文件重新组织。

04 / 动态存储格式

04 动态存储格式

分布式存储与计算实验室

4.1 Hyper

1. Hyper是一款学术型的混合负载数据库
2. 主存数据库无需面向data page，更加灵活简单
3. Hyper支持行/列两种数据格式
4. 主存数据库问题：内存空间紧张
5. 核心思想：在不损耗TP性能下构建压缩列存格式

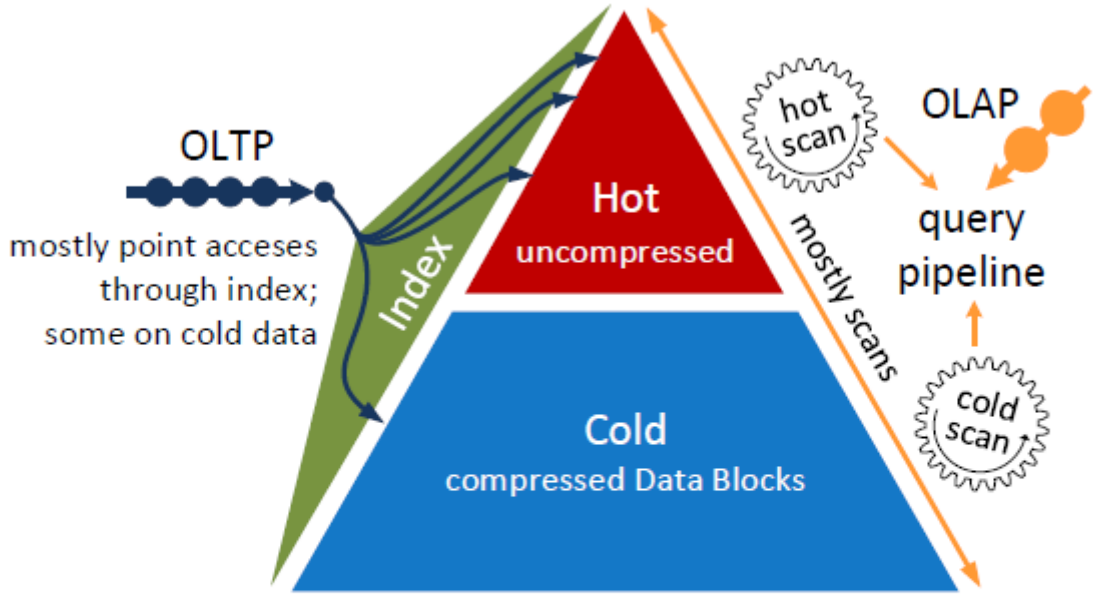


Figure 1: We propose the novel Data Block format that allows efficient scans and point accesses on compressed data and address the challenge of integrating multiple storage layout combinations in a compiling tuple-at-a-time query engine by using vectorization.

04 动态存储格式

4.1 Hyper

核心设计点：

1．将数据表划分为更小的单元chunk

2．以chunk为最小单位控制冷热数据转换

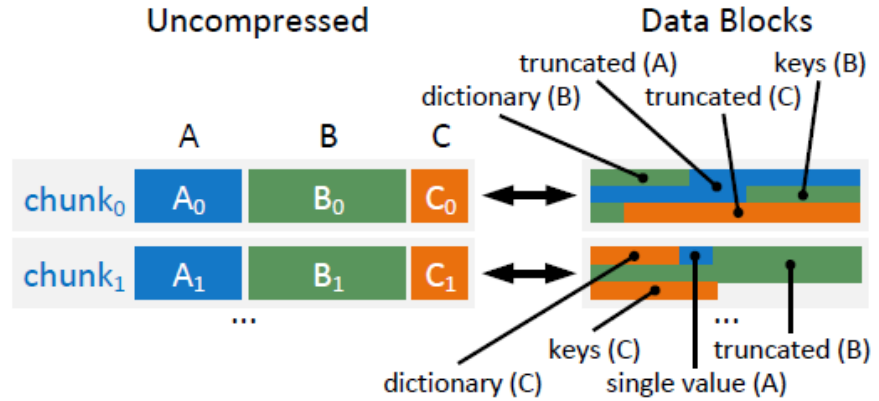
3．冷数据chunk进行压缩以及列式转换为Block

4．Block使用轻量级索引支持快速查找

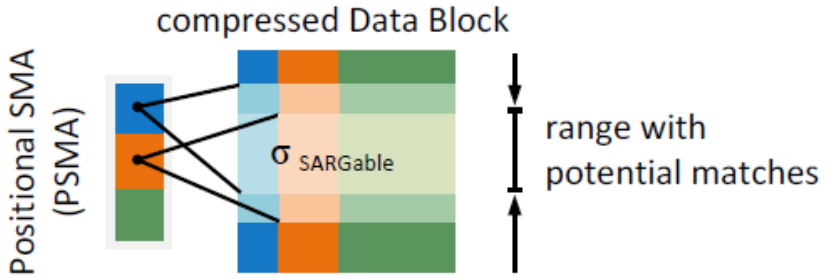
压缩设计：

1．轻量级压缩方案，仍支持字节寻址

2．每个属性单独选择最佳压缩方案



(a) Optimal compression techniques are chosen for each column in each block.



(b) Data Blocks store not only min and max values of attributes to skip blocks as a whole, but also light-weight indexes called Positional SMAs (PSMAs) that narrow the scan ranges on a per-attribute basis.

Figure 2: Overview of compressed Data Blocks

04 动态存储格式

4.1 Hyper

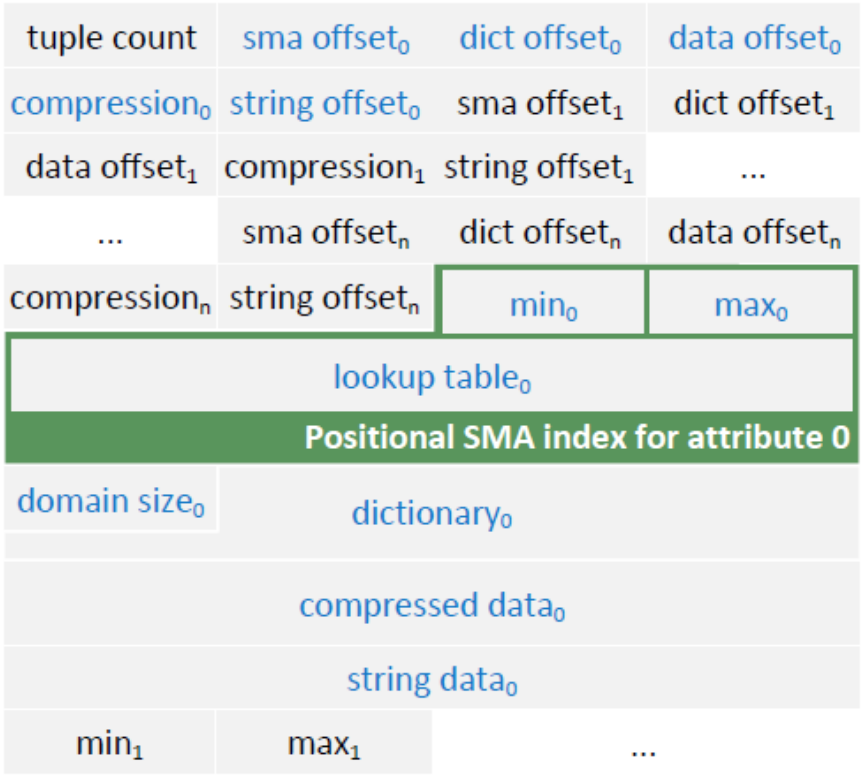


Figure 3: Layout of a Data Block for n attributes

Data Blocks核心设计：

1. 整体布局同PAX较为相似
    1. 第一个字段记录数据条目数
    2. 依次存储各属性基本元信息
    3. 指定各属性压缩方案

2. 每个Block大小根据记录数目确定（区别于常规的4KB限制）

3. Block块只读

04 动态存储格式

4.1 Hyper

SMA索引记录Block中最大和最小值

PSMA核心设计：

1．简单查询表，构建Block时创建此表

2．表项是对应查找值的范围区间

3．查找过程：
   1．求得查找值 V 与最小值的差值
   2．从差值中以第一个非零字节记为p,剩下字节值记为r
   3．对应值所在表位置即 i=p+r*256

4．多属性谓词判断可以根据交集缩小查找范围

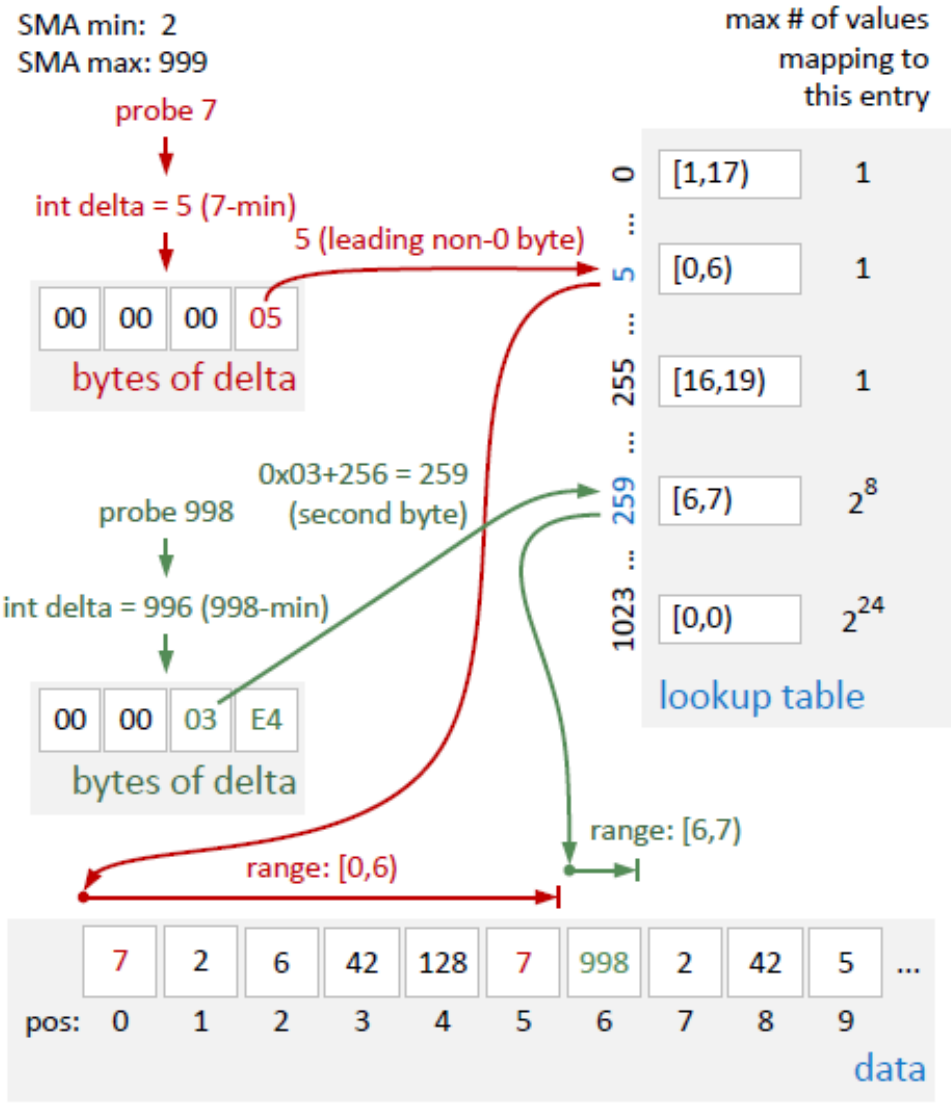


Figure 4: Examples of probing the lookup table of a Positional SMA index (PSMA)

04 动态存储格式

4.1 Hyper

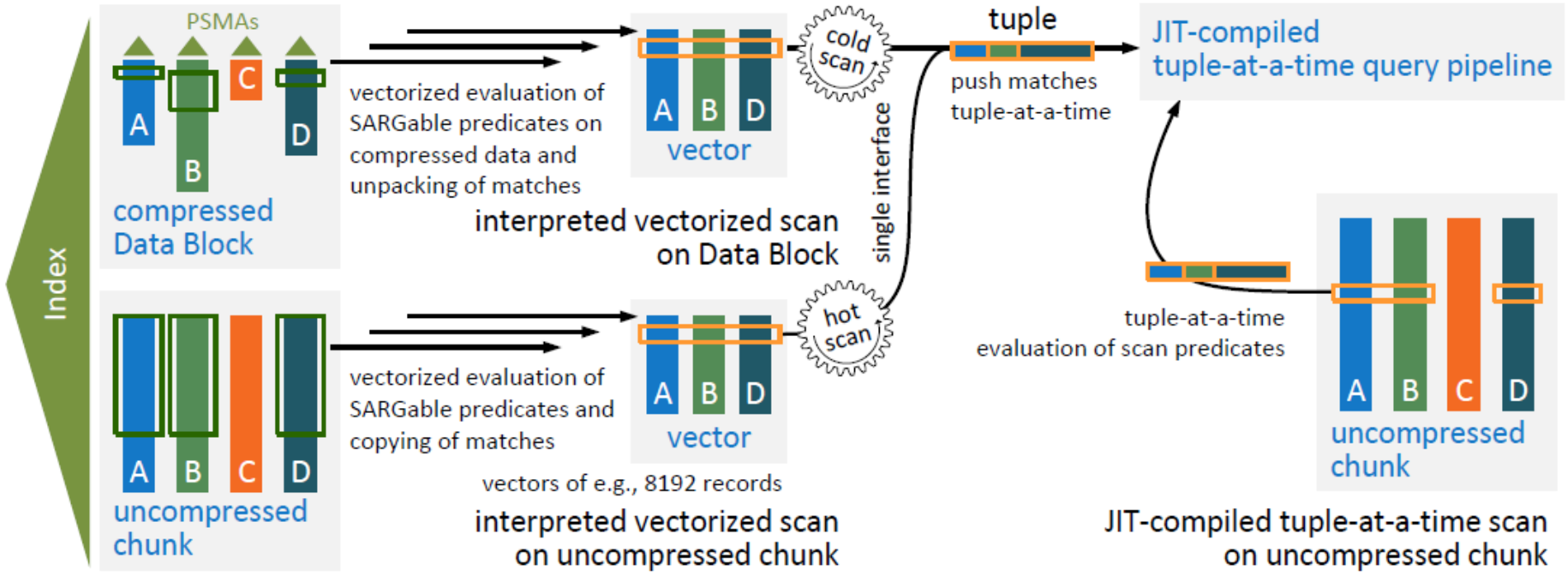
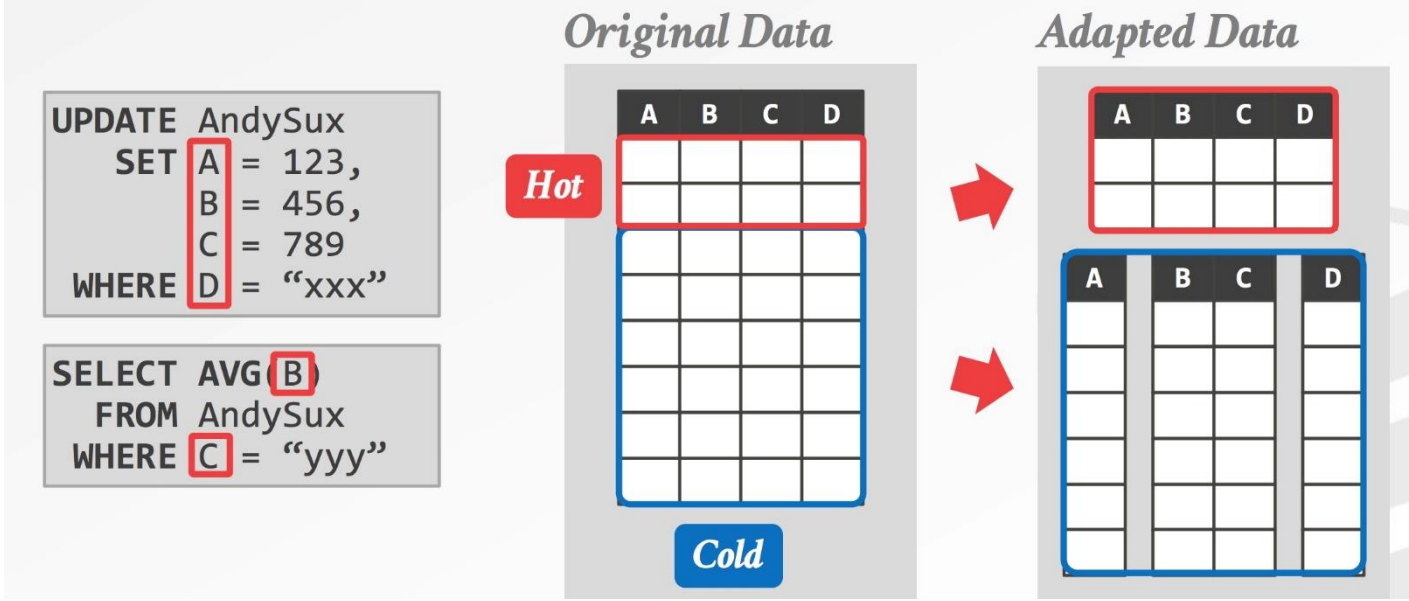


Figure 6: Integration of Data Blocks in our query engine: vectorized scans on Data Blocks and uncompressed chunks on the left share the same interface and evaluate SARGable predicates on vectors of records using SSE/AVX2 SIMD instructions (cf., Section 4.2). Matches are pushed to the query pipeline tuple at a time. The original JIT-compiled scan on the right evaluates predicates as part of the query pipeline.

04 动态存储格式

4.2 FSM

- 一个数据刚进入数据库的时候，它被称为热的，因为它有极高的可能性在近期内被修改。
- 而当这个数据的时期越来越大，它逐渐变冷，那么它被读取的次数将远大于修改的次数



04 动态存储格式

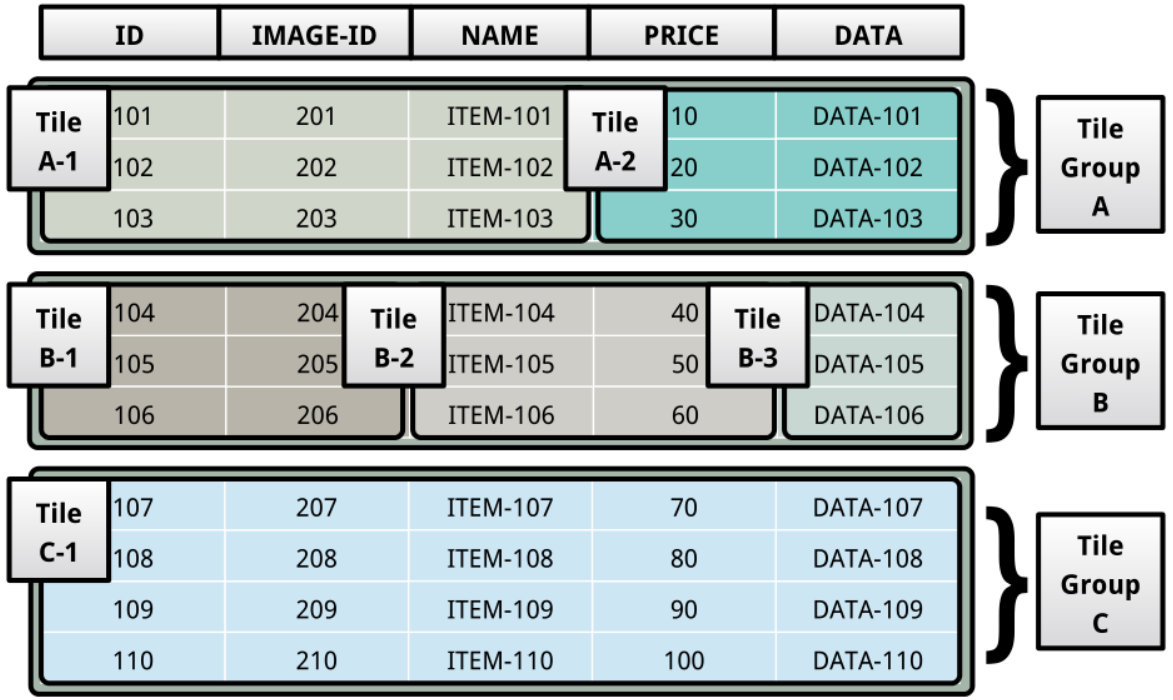
4.2 FSM

- Flexible Storage Model
- 支持同一数据库上的不同工作负载，支持多种混合存储布局
- 同一表中，热数据趋向于行存，冷数据趋向于列存
- 同一表中的不同数据段选择最优的布局，不断改进数据库的物理存储布局
- 新的在线重组技术
  - 无需手动调优，并且开销较小
- 忽略数据的存储布局
  - 使用抽象的逻辑块隐藏物理块细节

04 动态存储格式

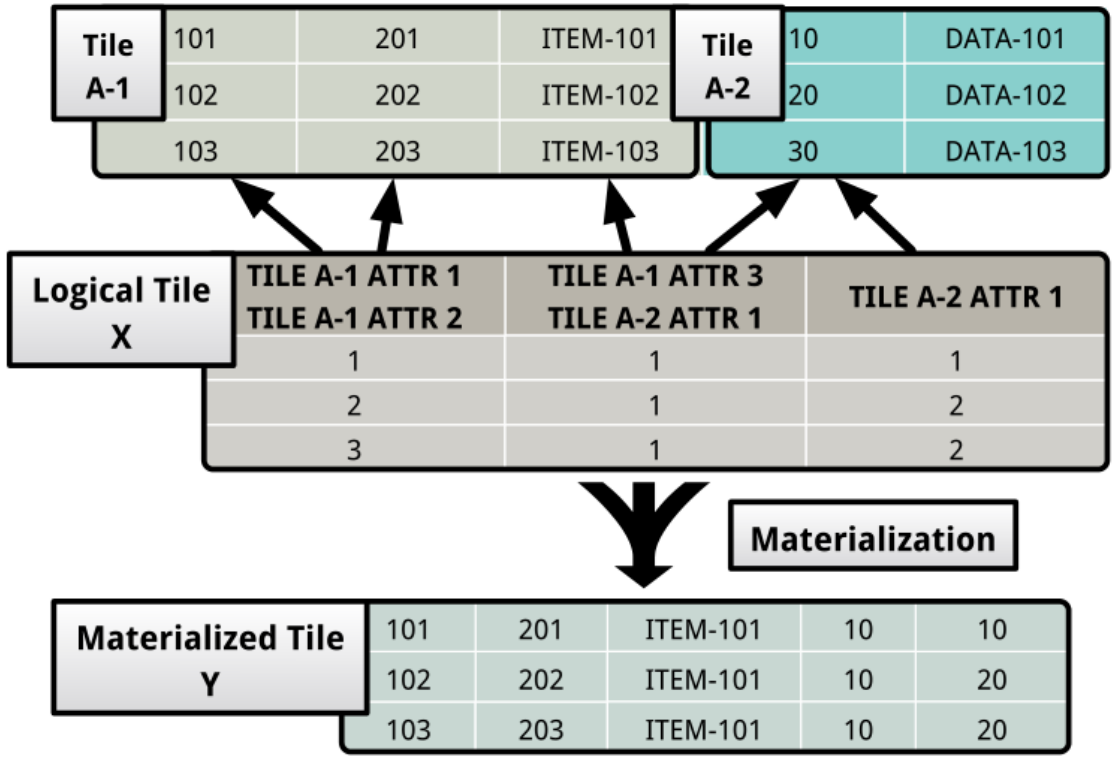
4.2 FSM

- 物理块：基于Tiles存储体系结构
- 元组可以随着时间的推移使用不同的布局存储在FSM数据库中
- 尽可能将经常一起访问的数据存到同一个tile中
- 在线重组步骤：
  - 跟踪查询负载
  - 周期性计算
  - 后台执行重组



04 动态存储格式

4.2 FSM



- 逻辑块简洁地表示分布在一个或多个表的物理块集合中的值
- 逻辑块的每一列都包含对应于底层物理块中的元组的偏移量列表。
- 优点：
  - Layout Transparency
  - Vectorized Processing
  - Flexible Materialization
  - Caching Behavior

04 动态存储格式

分布式存储与计算实验室

4.3 Real-Time LSM-Tree for HTAP

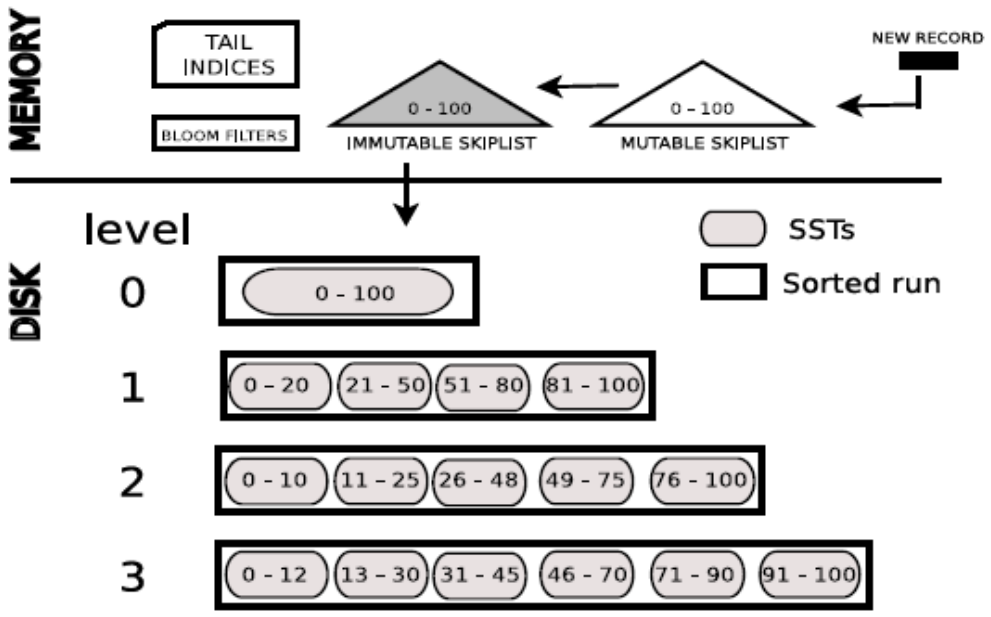


Figure 1: LSM-Tree with leveling merge strategy

LSM-Tree 特性:

1. 直接写入memtable提高写吞吐量
2. 分层存储，大小递增
3. 定期compaction，提高查询效率
4. 新数据在上层，旧数据在下层

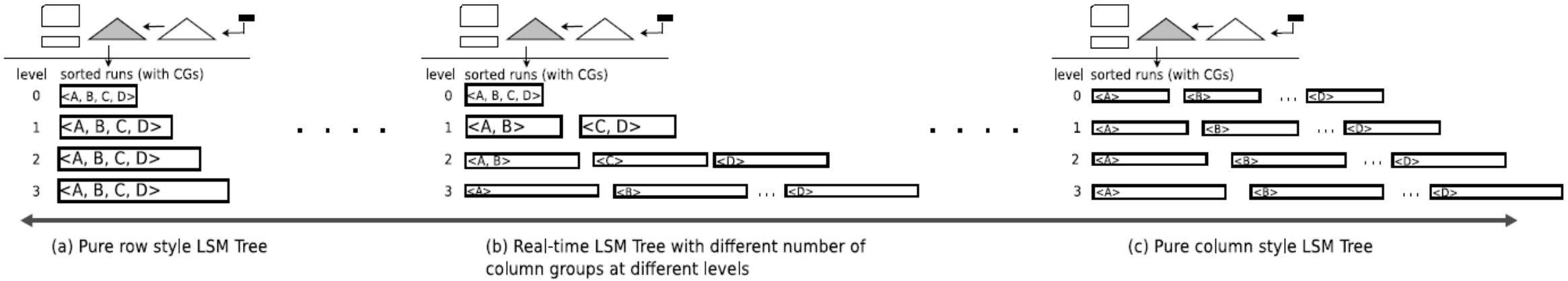
Real-Time LSM-Tree 特性:

1. 数据生命周期感知的数据布局
2. 针对HTAP将数据由行存逐渐转换为列存

04 动态存储格式

分布式存储与计算实验室

4.3 Real-Time LSM-Tree for HTAP



(a) Pure row style LSM Tree

(b) Real-time LSM Tree with different number of column groups at different levels

(c) Pure column style LSM Tree

列组CG:

1. 一个完整元组划分为不同的CG，将不同的列值放入CG

2. CG间列存，CG内行存

设计空间:

1. 每个level都会有特定的CG形式

2. Memtable与Level_0不变维持写吞吐量

3. 下层GG是上层CG的子集

04 动态存储格式

4.3 Real-Time LSM-Tree for HTAP

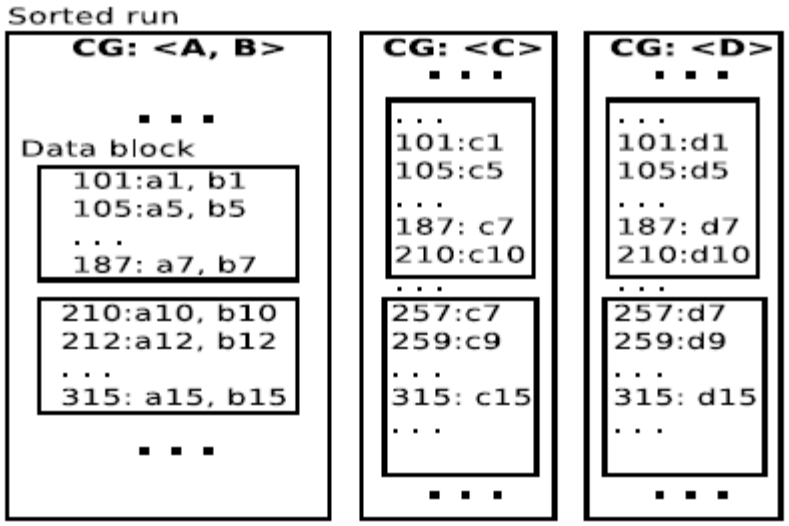


Figure 4: Simulated column-group representation

CG:

1. 同一level的CG可以视为一个sorted run
2. 每个CG都需要存储key才能定位
3. 每个CG内由多个sst组成，每个SST都有索引和布隆过滤器

投影Π：支持对元组特定列直接操作

1. read(key,Π)：读key的Π对应的列值
2. scan(key_l,Key_h, Π)：扫描范围Key在l和h之间的Π对应的列值
3. update(key,value_ Π})：更新key中指定的Π列的值

04 动态存储格式

分布式存储与计算实验室

4.3 Real-Time LSM-Tree for HTAP

LevelMergingIterators:

- 在范围查找和compaction时，从每个level获取和合并对应的元组，同时删除属性对应的旧版本值。

ColumnMergingIterators:

- 在相同level中将不同的CGs合并到一起，可能有列值为空的情况

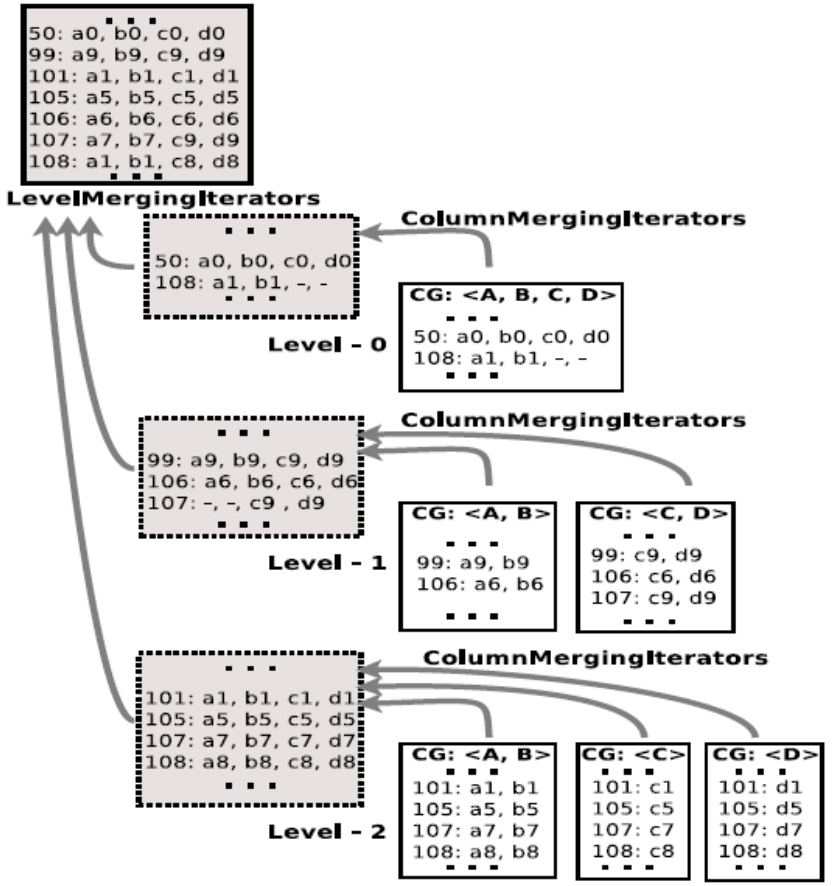


Figure 5: *ColumnMergingIterators* and *LevelMergingIterators*

04 动态存储格式

4.3 Real-Time LSM-Tree for HTAP

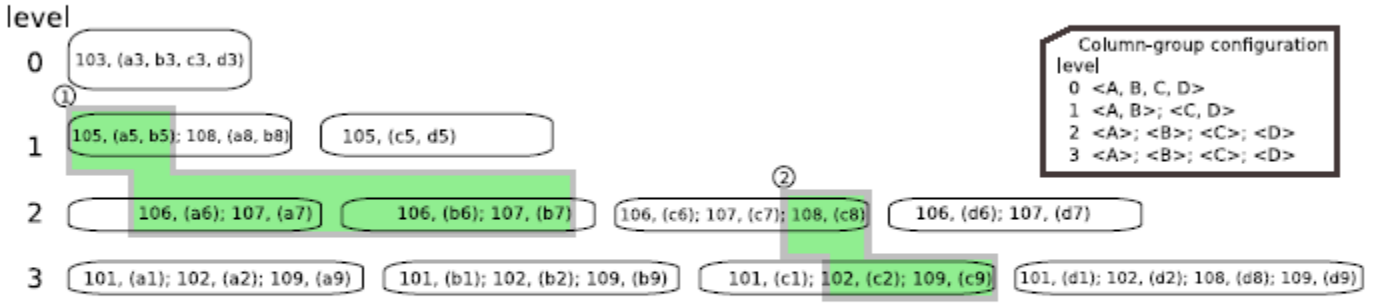


Figure 6: Sorted runs of a Real-Time LSM-Tree with two highlighted compaction jobs.

Compaction过程：

1. 检测溢出的Level
2. 从level中找出溢出的CG
3. 在下个level中寻找重叠CGs
4. 每个level开启ColumnMergingIterators
5. 开启LevelMergingIterators
6. 写入下一个level中

05 总结

05 总结

DSC 分布式存储与计算实验室

	思路	优点	缺点
PAX	1. 页面间行存 2. 页面内列存	1. 题上页面内列数据局部性 2. 设计简单高效 3. 提升CPU缓存命中率	1. 无法减少内存占用
Hyper Data Block	1. 冷热数据分离 2. 压缩减少内存 3. 轻量级索引	1. 减少内存空间占用 2. 列存压缩格式支持快速查找	1. 增加Block构建复杂度
FSM	1. 动态变更数据存储布局 2. 在线自动重组	1. 最大化提升CPU缓存命中率 2. 逻辑抽象使得上层业务无感知	1. 重组代价大 2. 动态布局算法模型开发难
Real-Time LSM	1. Compaction中变更存储布局 2. 数据生命周期自感知	1. 简单有效 2. 提高查询效率 3. 支持特定列的查找	1. 提升设计复杂度 2. 增加compaction任务压力

05 文献

DSC
分布式存储与计算实验室

[01] - Ailamaki, Anastassia, David J. DeWitt, and Mark D. Hill. "Data page layouts for relational databases on deep memory hierarchies." *The VLDB Journal* 11.3 (2002): 198-215.

[02] - Robert Binna, Eva Zangerle, Martin Pichl, Günther Specht, and Viktor Leis. 2018. HOT: A Height Optimized Trie Index for Main-Memory Database Systems. In Proceedings of the 2018 International Conference on Management of Data (SIGMOD '18). Association for Computing Machinery, New York, NY, USA, 521–534.

[03] - Lang, Harald, et al. "Data blocks: Hybrid OLTP and OLAP on compressed storage using both vectorization and compilation." *Proceedings of the 2016 International Conference on Management of Data*. 2016.

[04] - Saxena, Hemant, et al. "Real-Time LSM-Trees for HTAP Workloads." *arXiv preprint arXiv:2101.06801* (2021).