第12讲 大数据分析优化

1 查询性能的优化策略

2 平台功耗的优化策略

出租车**大数据**的**交互**式分析

时序分析

查看乘车数量如何随时间变化。一个特别有趣的分析是找出每个工作日每小时的乘车数量。

空间分析

查看旅途的空间分布模式。

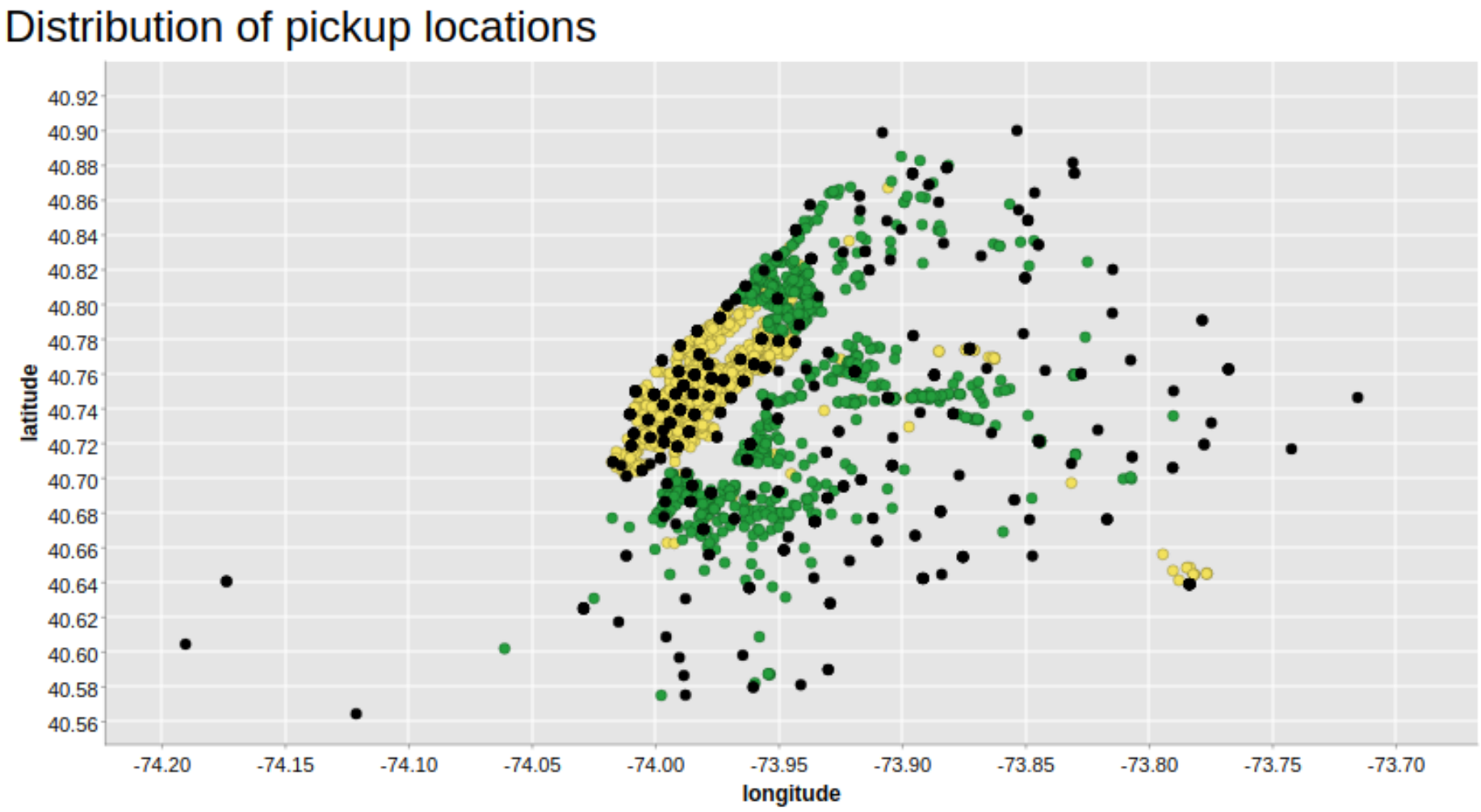
- 可以按旅途起点区域 ID 和下车地点区域 ID 将数据分组，计算每组的旅途数量，并可视化；
- 可以按下车地点区域 ID 对数据进行分组，并可视化；
- 可以探索出租车区域之间的交通情况；

数据集 – 纽约出租车行程

- https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page
- 属性：接送地点、时间、人数、距离、金额、小费等

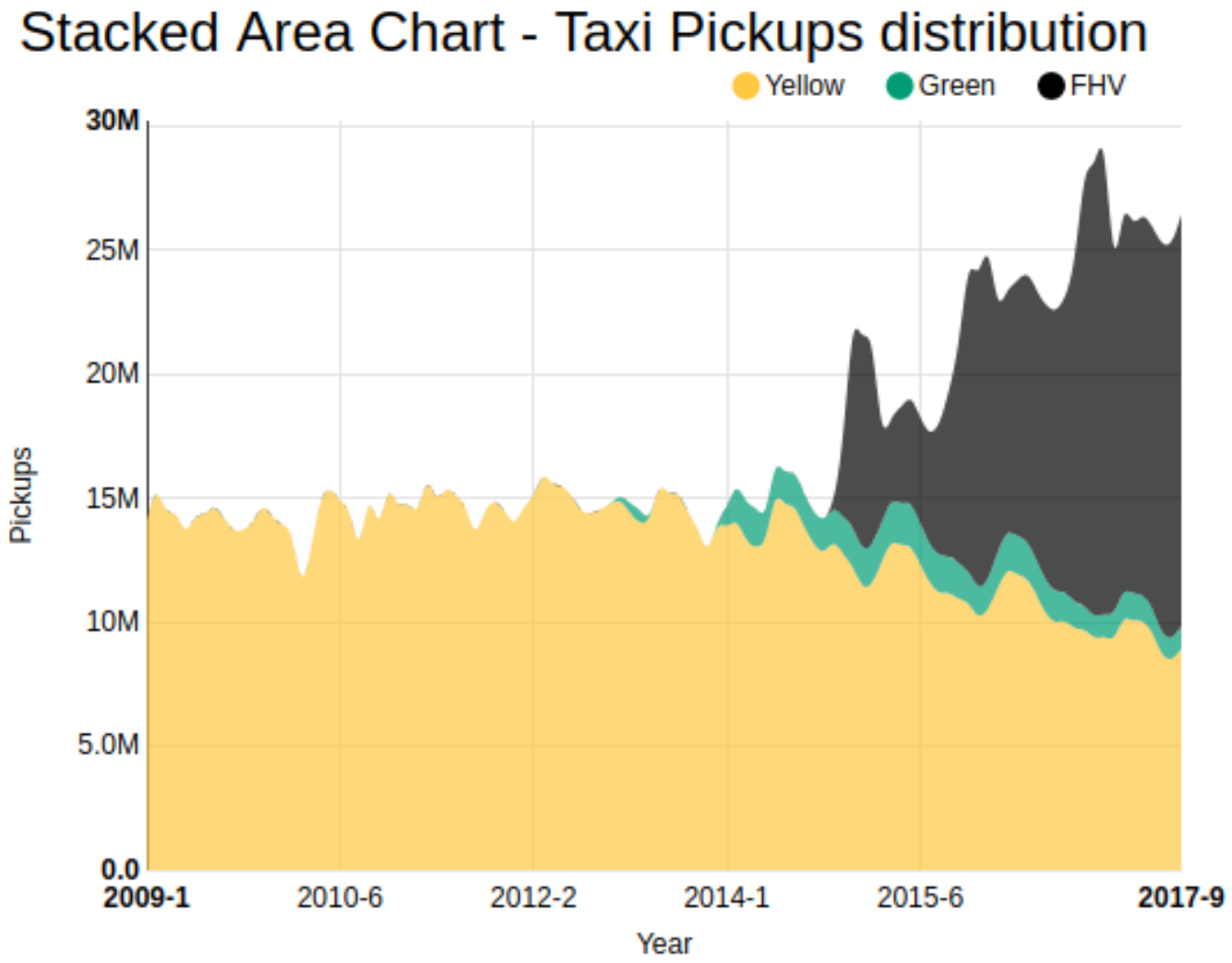
- 一个典型的交互式大数据分析过程：
  - ✓对整个数据集的概要分析
  - ✓聚焦到某个特定的年份
  - ✓查询特定类型出租车
  - ✓显示特定年份、特定行政区域、特定类型出租车的轨迹

出租车大数据的交互式分析：整体概要



the scatter plot map 显示的是采样数据（数据量太大）

出租车大数据的交互式分析：整体概要

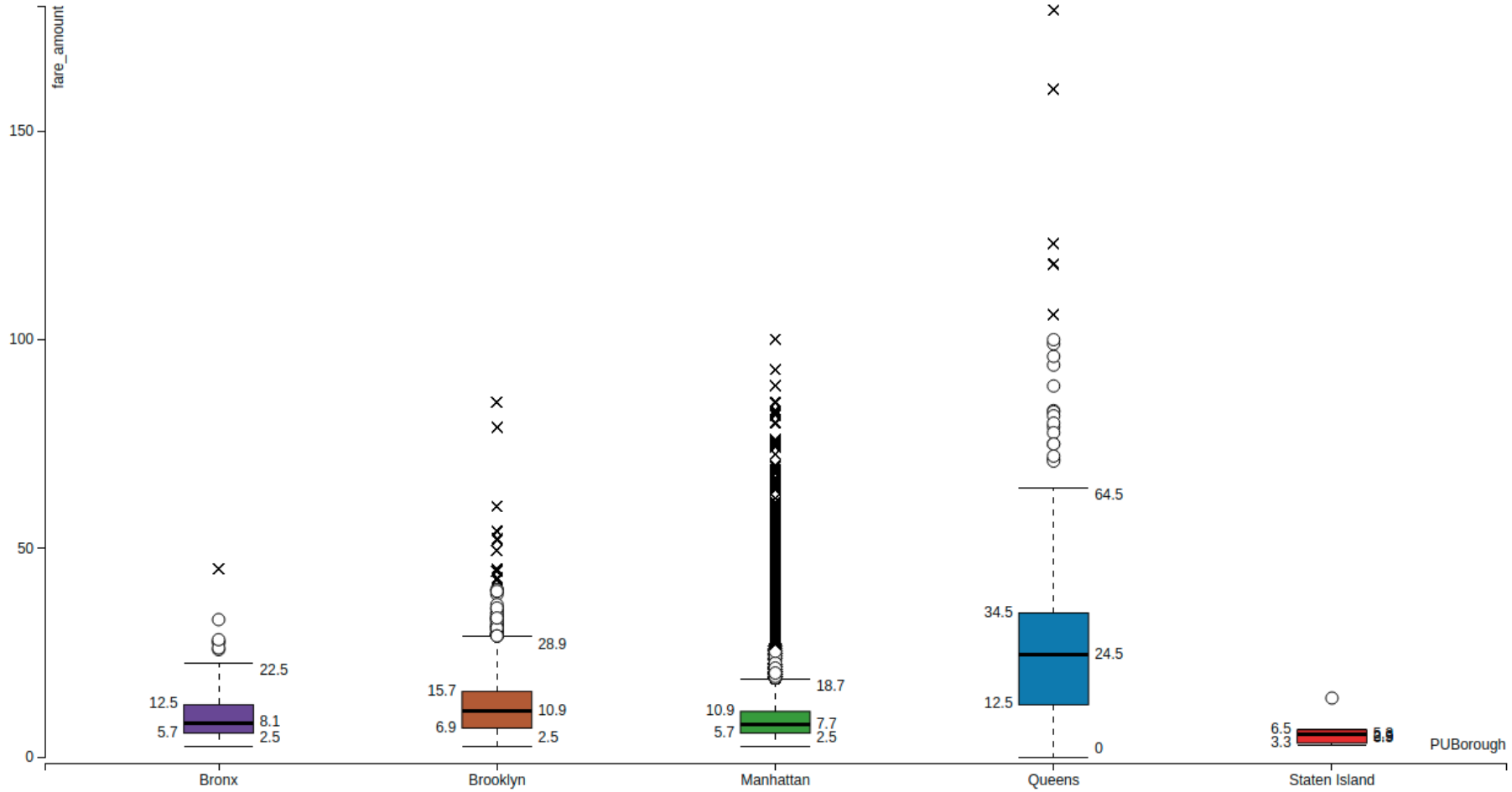


各类型出租车随年份变化的经营状况的改变（地位转变）

出租车大数据：yellow taxi每日经营情况分析

- 详细地展示了纽约市五个区黄色出租车的活动；
- 显然，即使允许黄色出租车在纽约市五个行政区的任何地方运行，曼哈顿地区（绿色）的搭乘数量也非常集中，这与其他行政区相比要多得多。

出租车大数据：各行政区车费的分布



- 曼哈顿（绿框）的票价中位数最低，同时其极端离群值也最多。较低的票价中位数可能意味着曼哈顿的大多数出行都是短途旅行，而离群值可能代表偶尔从曼哈顿到其他行政区甚至其他城市的较长旅行。
- 史丹顿岛（红色框）的票价大多是统一的，也许大多数的出租车旅行都在区内

出租车大数据：实时状态

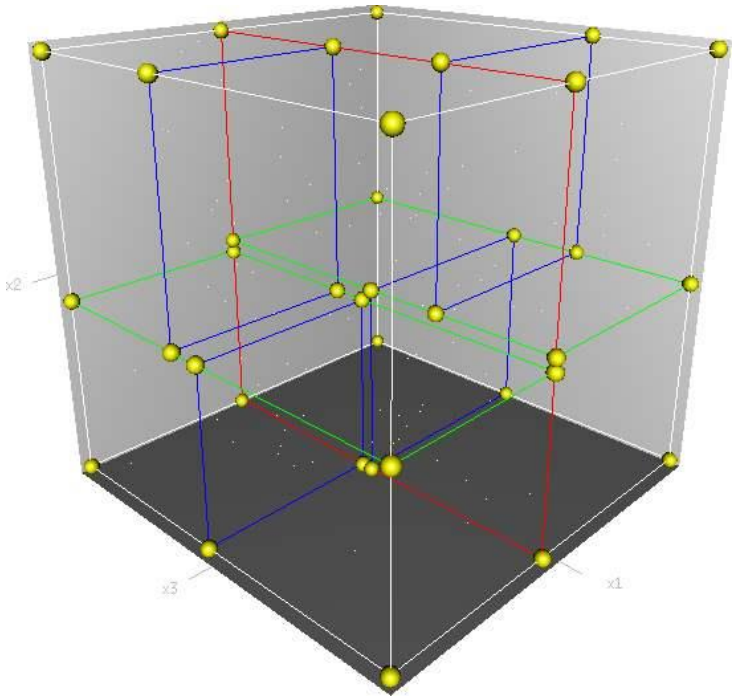

Hour: 0:00
Ridgefield Park
Palisades Park
Fort Lee
Ridgefield
Edgewater
Cliffside Park
Fairview
Guttenberg
Union City
Weehawken
Hoboken

交互式大数据分析

- 用户实时地与大数据进行交流
  - ✓每一个查询的处理时间必须是用户可忍受的
  - ✓通过扫描所有数据得到结果的方式无法胜任
- 如何加速大数据分析?
  - ✓减少数据访问：避免访问与分析查询不相关的数据
  - ✓减少计算：使用更小的数据作为输入
  - ✓减少数据传输：减少不同计算节点之间大规模数据传输

目录

MRA树: 空间聚集索引
（Multi-Resolution Tree）



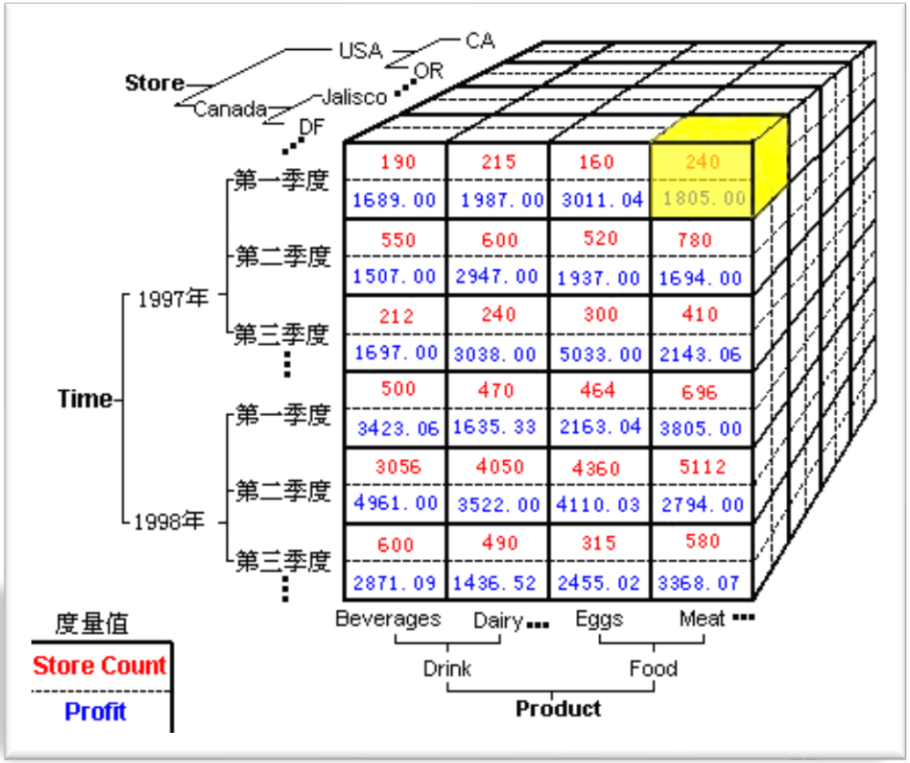
相关概念：多维数据分析

- 多维数据：

tpep_pickup_datetime	tpep_dropoff_datetime	trip_distance	PULocationID	DOLocationID	payment_type	fare_amount	tip_amount	total_amount
2019/1/1 0:46	2019/1/1 0:53	1.5	151	239	1	7	1.65	9.95
2019/1/1 0:59	2019/1/1 1:18	2.6	239	246	1	14	1	16.3
2018/12/21 13:48	2018/12/21 13:52	0	236	236	1	4.5	0	5.8
2018/11/28 15:52	2018/11/28 15:55	0	193	193	2	3.5	0	7.55
2018/11/28 15:56	2018/11/28 15:58	0	193	193	2	52	0	55.55
2018/11/28 16:25	2018/11/28 16:28	0	193	193	2	3.5	0	13.31
2018/11/28 16:29	2018/11/28 16:33	0	193	193	2	52	0	55.55

- OLAP中的CUBE模型：

数据立方体(超立方)



相关概念:聚集函数(Aggregate function)

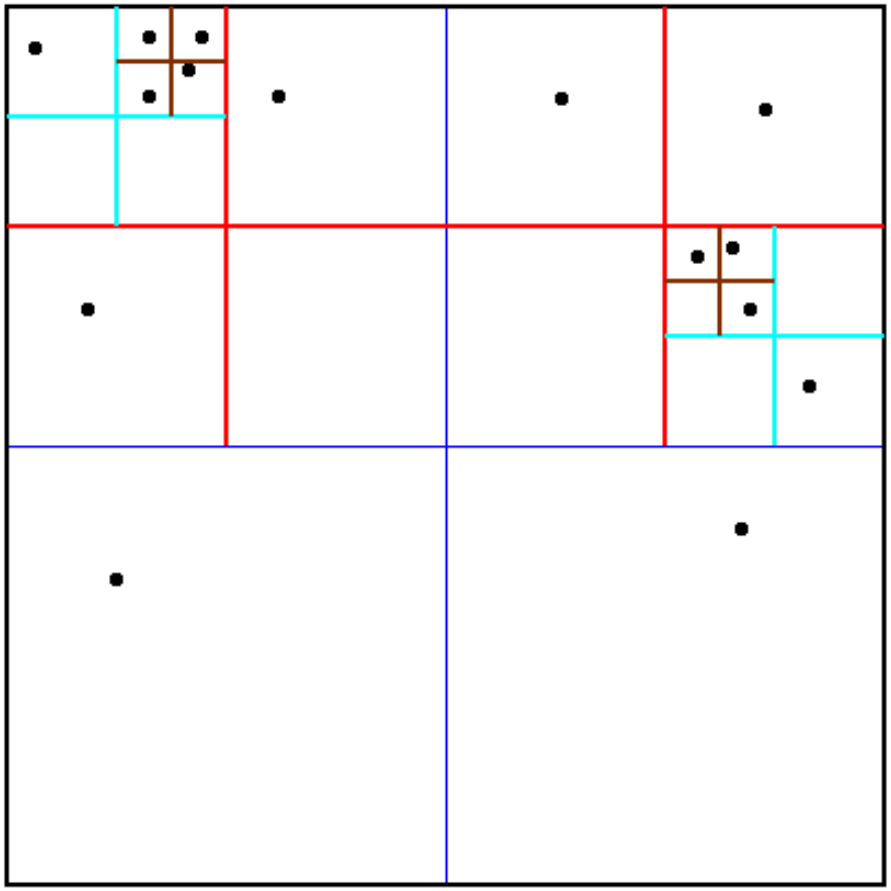
- 用来统计每个分组的信息
  - ✓Min、Max、Count、Sum、Avg…
- 聚集查询(Aggregate Queries)的用途:
  - ✓什么时间是打车的高峰时间?
  - ✓各个区域每天什么时间最拥堵?
  - ✓出租车司机如何能接到“大单”?
  - ✓什么时候“拼车”的人多?
  - ✓不同时间，什么地方上车的人多?人们都去哪了?
- 聚集查询通常带有条件(where谓词):
  - ✓聚集查询转换为多维空间上一个“子空间”的查询

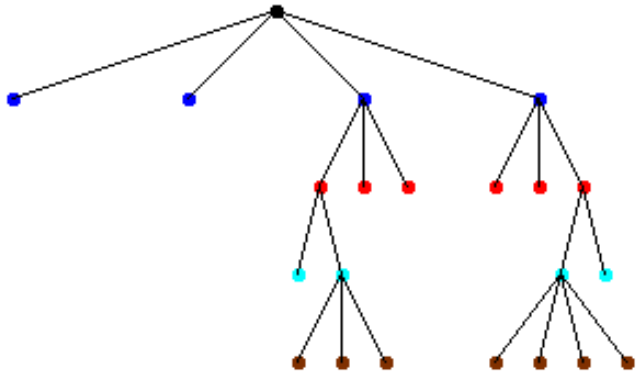
相关概念: 空间索引

- 空间索引:
  - ✓索引是为了提高数据集的检索效率
  - ✓高维数据空间上的聚集查询，首先要找到符合条件的子空间的数据，还要在上面执行聚集函数，空间索引更是效率的关键
  - ✓空间索引的作用是通过它的筛选，把大量与特定空间操作无关的空间对象排除
- 有很多空间索引:
  - ✓MRA建立在四叉树基础上（比较基础的一种）
  - ✓四叉树索引，对数据空间进行网格划分，递归进行四分来构建四叉树，直到自行设定的终止条件

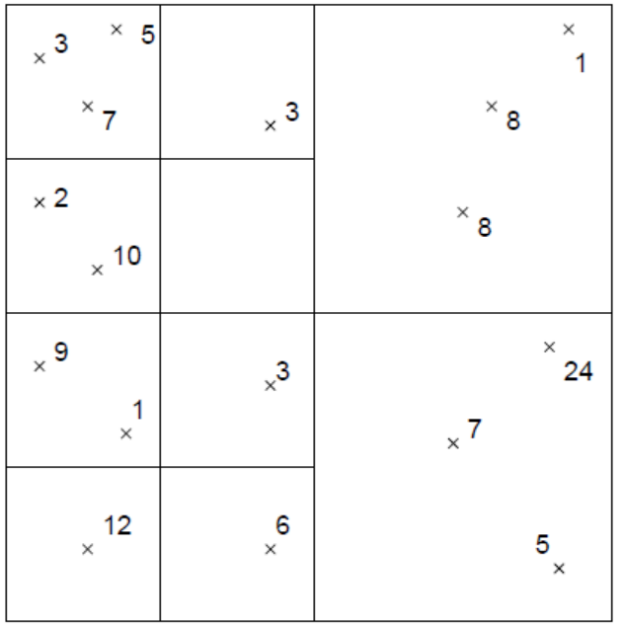
相关概念: 空间索引

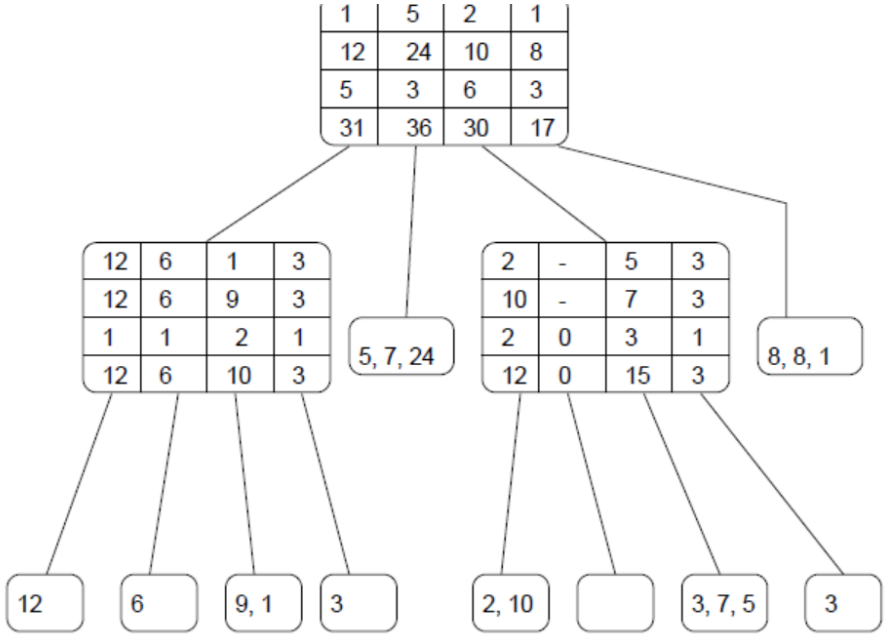
将所有黑点插入四叉树中构建四叉树





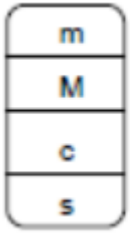
MRA树: MRA-四叉树





x, y, z, ...

Leaf Node containing data points with values x, y, z, ...



Child Record:
m: min c: count
M: max s: sum

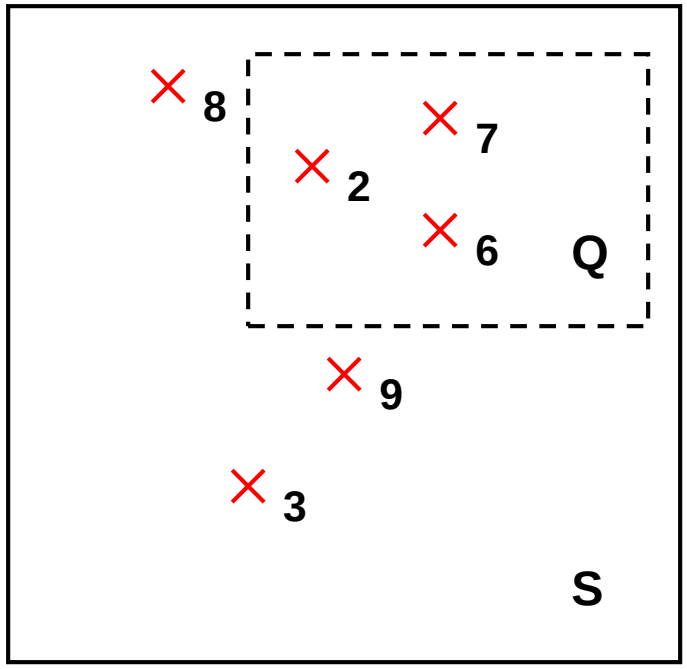
如何利用MRA完成“Progressive Approximate Aggregate Queries”？

问题定义

- 数据：以点(point)形式存在：<*loc*, *values*>
  - ✓$loc \in R_{space} \subseteq \mathcal{R}^d, values \in D^v$
- Query region：$R^Q \subseteq R_{space}$
- 聚集函数采用：COUNT, SUM, MIN, MAX, AVG
- 关键词：Progressive，Approximate Aggregate Queries

简单的例子

- 空间聚集分析
  - 计算给定空间范围内对象的统计值



$\mathbf{min}^{Q} = 2$

$\mathbf{max}^{Q} = 7$

$\mathbf{count}^{Q} = 3$

$\mathbf{sum}^{Q} = 2+7+6 = 15$

$\mathbf{avg}^{Q} = 15/3 = 5$

Progressive Approximate Aggregate Queries

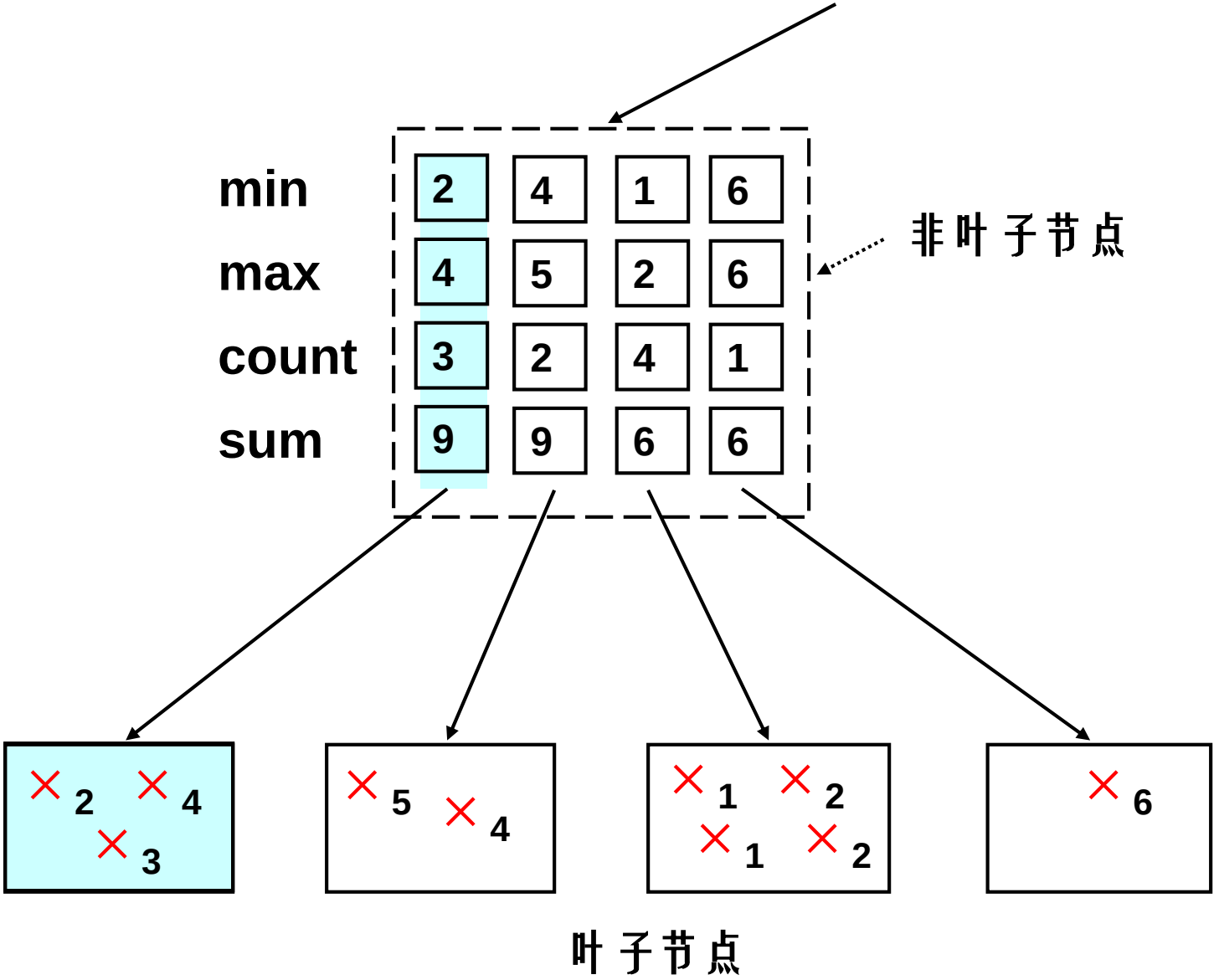
- 空间聚集分析
  - 精确计算
    - 扫描数据集合中所有对象并计算给定空间范围内的聚集值
    - 从空间索引中获取给定空间范围内的对象，并计算其聚集值

线性扫描和索引扫描可能引起巨大的查询代价

  - 近似计算
    - 牺牲聚集值一定的精度，换取计算时间的节省
    - 近似结果带有概率保证或者误差范围

- 如何利用MRA索引，逐步得到不同精度的近似值，直至精确结果?

聚集索引：MRA树节点结构

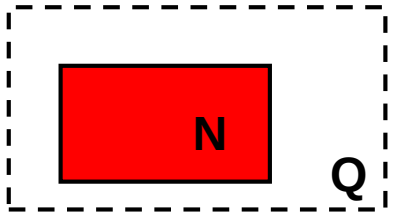


➢ MRA树聚集分析处理

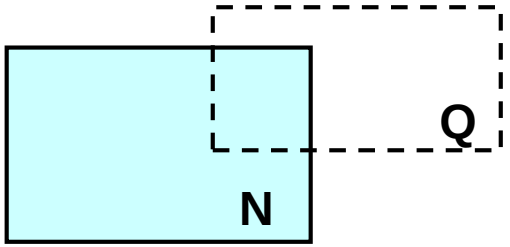
–自顶向下访问两类索引节点

- 完全包含于查询$\mathcal{N}^c$：$N \in \mathcal{N}^c \Rightarrow R^N \cap R^Q = R^N$
- 与查询部分相交$\mathcal{N}^p$：$N \in \mathcal{N}^p \Rightarrow R^N \cap R^Q \neq R^N \wedge R^N \cap R^Q \neq \Phi$

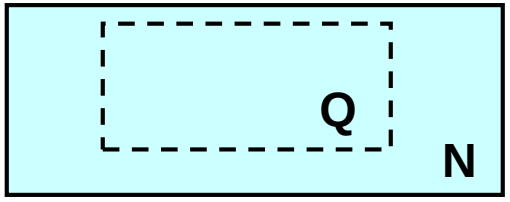
contained



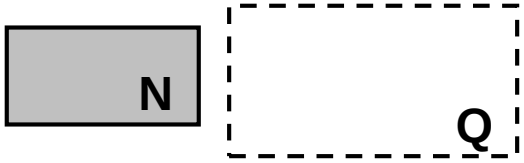
partially overlapping



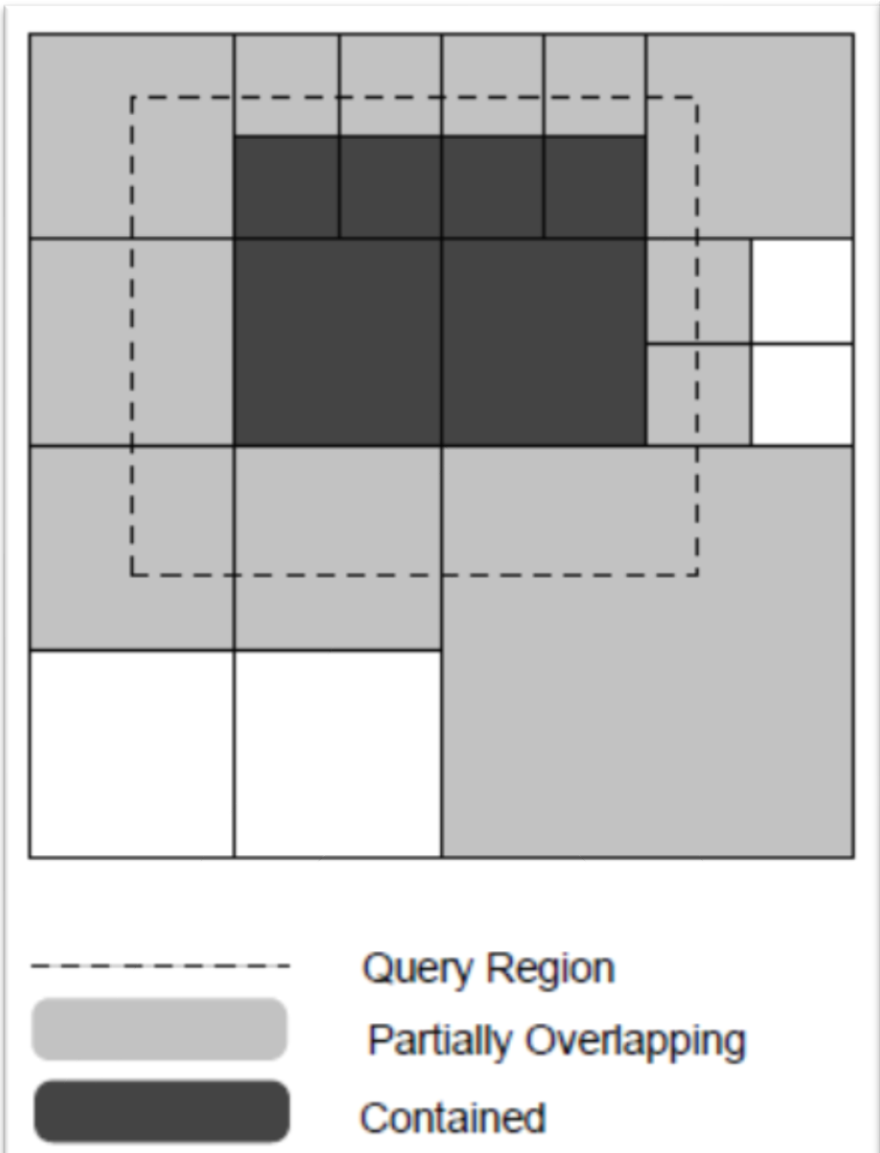
encloses

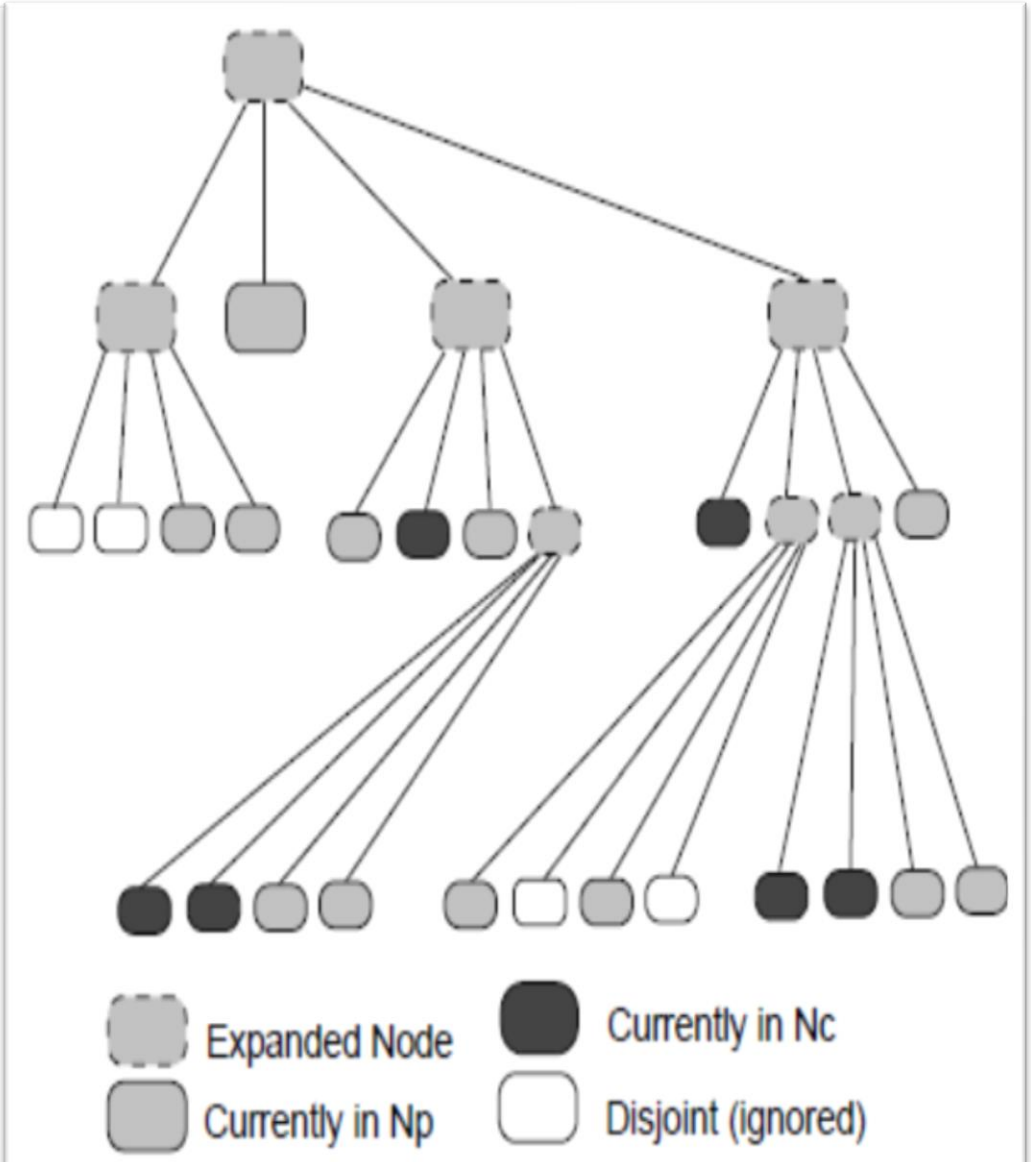


disjoint

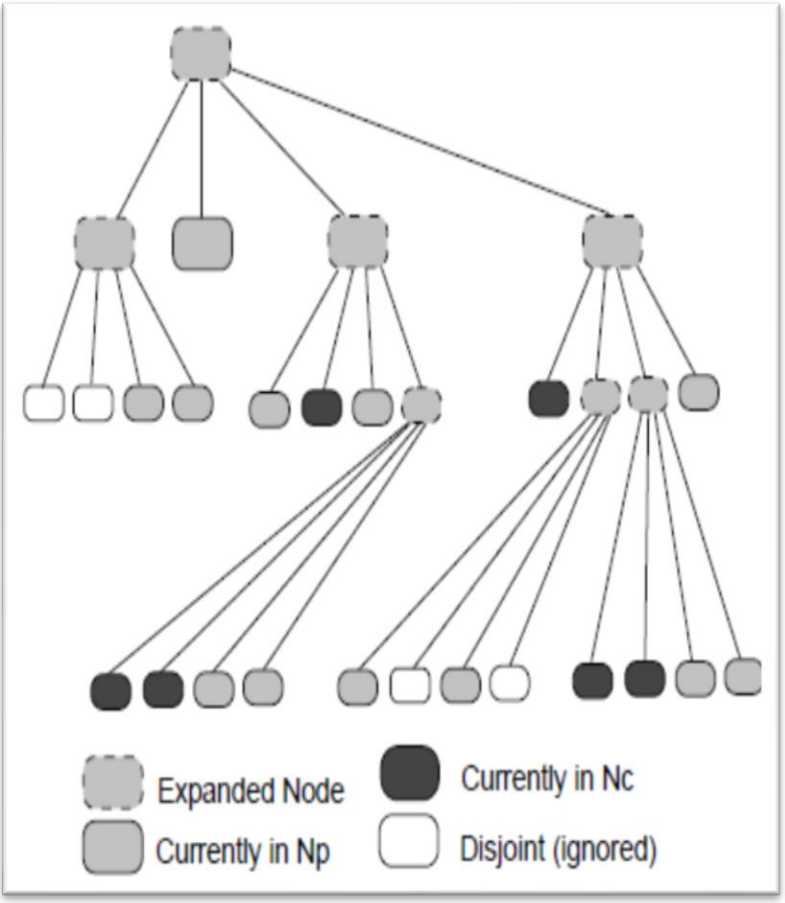


Progressive Aggregate Query Algorithm





Progressive Aggregate Query Algorithm



初始化：

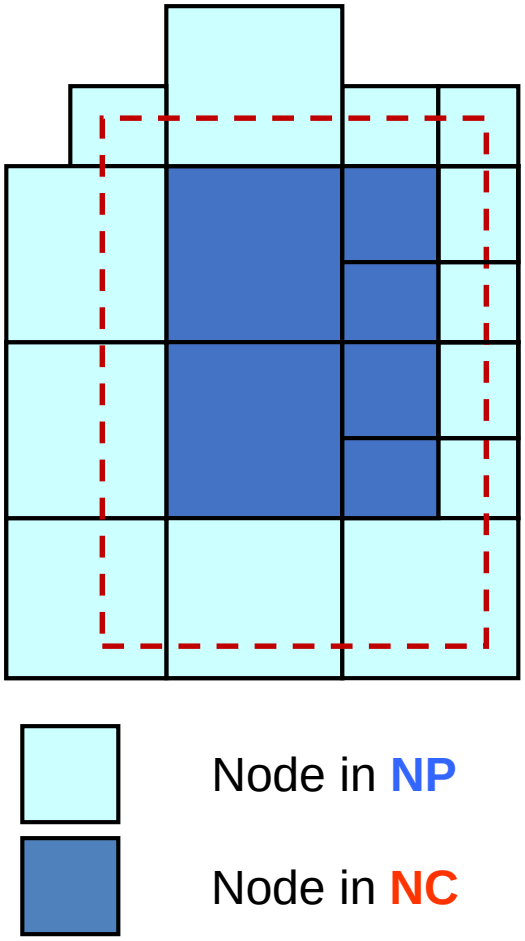
$R^Q := \mathrm{queryR};$
$\mathrm{agg} := \mathrm{aggType};$
$\mathbf{if}\ (R^Q \cap R^{root} = R^{root})$
$\quad \mathcal{N}^c := \{\mathrm{root}\};$
$\mathbf{else}$
$\quad \mathcal{N}^p := \{\mathrm{root}\};$

Progressive Aggregate Query Algorithm

迭代部分

```
if (N^p = ∅) {
   estimate ans from N^c;
   low := high := ans;
}
else {
   remove node N from N^p;
   forall children N_i : i = 1, ..., n_N of node N {
      if (R^Q ∩ R^N = ∅)
         ignore;
      else if (R^Q ∩ R^N = R^N)
         insert N_i into N^c;
      else
         if N_i is useful
            insert N_i into N^p;
   }
   estimate ans from N^p, N^c;
   calculate low, high from N^p, N^c;
}
```

遍历顺序?



- 为以下聚集值实例化算法{MIN,MAX,COUNT,SUM,AVG}:
  - 误差界.
  - 置信区间 $I=[L, H] : L \leq agg^{Q} \leq H$
  - 索引遍历策略.
    - 优先访问NP中能尽量缩小置信区间的节点
  - 估计值.
    - 提供一个目标聚集值的估计数值

聚集索引

➢ MRA树聚集分析处理Min(Max)

置信区间

$\mathbf{min^{NC}} = \mathbf{min}\{4, 5\} = 4$

$\mathbf{min^{NP}} = \mathbf{min}\{3, 9\} = 3$

$\mathbf{L} = \mathbf{min}\{\mathbf{min^{NC}}, \mathbf{min^{NP}}\} = 3$

$\mathbf{H} = \mathbf{min^{NC}} = 4$

因此，$\mathbf{I} = [3, 4]$

估计值

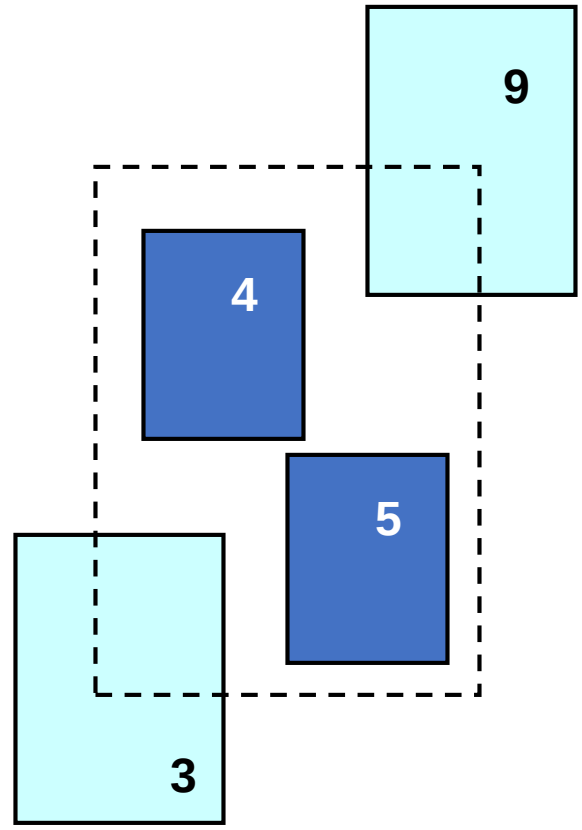
Lower bound:

$\mathbf{E(min^{Q})} = \mathbf{L} = 3$

遍历策略

选择 $\mathbf{N} \in \mathbf{NP}$:

$\mathbf{min^{N}} = \mathbf{min^{NP}}$



➢ 遍历策略(Min)

✓选择节点 $N_b^{Min} \in \mathcal{N}^p : min^{N_b^{Min},X} < min^{c,X} \wedge min^{N_b^{Min},X} \leq min^{N,X}$

聚集索引

➢ MRA树聚集分析处理COUNT(SUM)

置信区间

count^{NC} = 9+6 = 15

count^{NP} = 8+10 = 18

L = **count^{NC}** = 15

H = **count^{NC}** + **count^{NP}** = 33

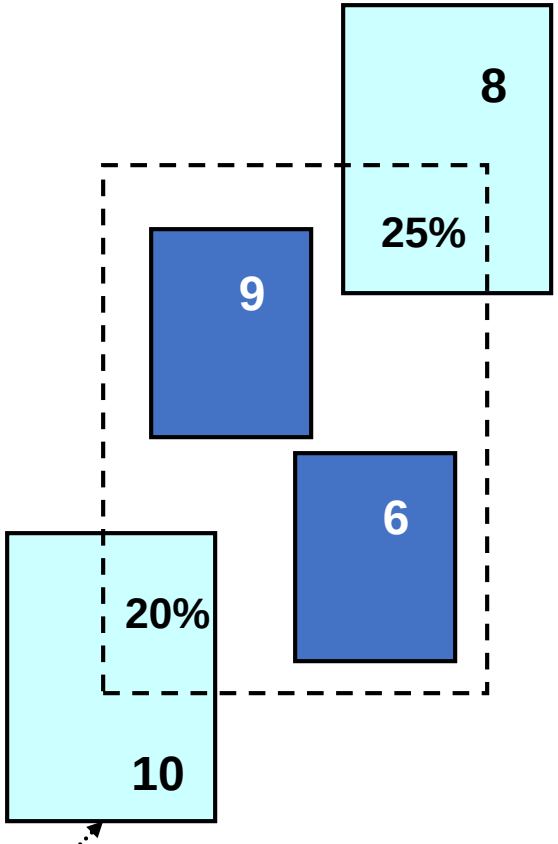
因此，**I** = [15, 33]

估计值

E(count^{Q}) = **L** + 0.25×8 + 0.2×10 = 19

遍历策略

优先选择 **N** ∈ **NP**: **count^{N}**≥**count^{M}**, ∀ **M** ∈**NP**



MRA树聚集分析：AVG

- 有了对Sum和Count的估计，Avg的估计比较容易：
  - ✓ $\widehat{Avg} = \frac{\widehat{Sum}}{\widehat{Count}}$
- 置信区间和节点遍历策略稍微复杂些
- 先思考两个小问题（后面会用到）

① 一个MRA节点N(带有统计信息：Min^N, Max^N, $Count^N$, Sum^N)中，最多可以有多少个Max^N?

② 集合UpperBoundSet含有n_1个值，平均值为avg。如果value＞avg，那么加入n_2个value到UpperBoundSet后，新的平均值avg_2与avg的关系？

聚集索引

➢ MRA树聚集分析处理AVG

置信区间

当前avg^{NC} = 55/10 = 5.5

最大可能值: (55+2×10) / (10+2) = 6.25

最小可能值: (55+3×5) / (10+3) = 5.38

因此, **I** = [**5.38**, **6.25**]

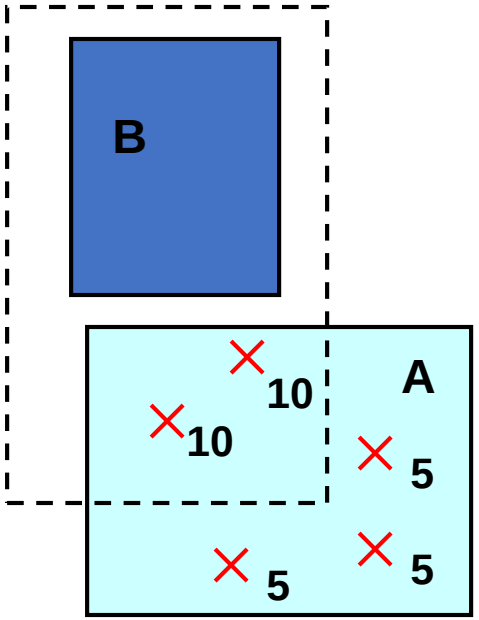
估计值(比较简单)

E(avg^{Q}) = E(sum^{Q})/ E(count^{Q})

遍历策略

– 最大的 **count^{N}或者**

– 最大的 (**max^{N}-avg^{NC}**), (**avg^{NC}-min^{N}**)



最多有多少个max值?

	min	max	count	sum
A	5	10	5	35
B	–	–	10	55

A中一种值的分布{**5**, **5**, **5**, **10**, **10**}

A中有2个10，比当前均值大，加入两个10!

聚集索引

- MRA树聚集分析处理AVG

置信区间

当前avg^{NC} = 40/10 = 4

最大可能值:

(40+2×10) / (10+2) = 60 / 12 = 5

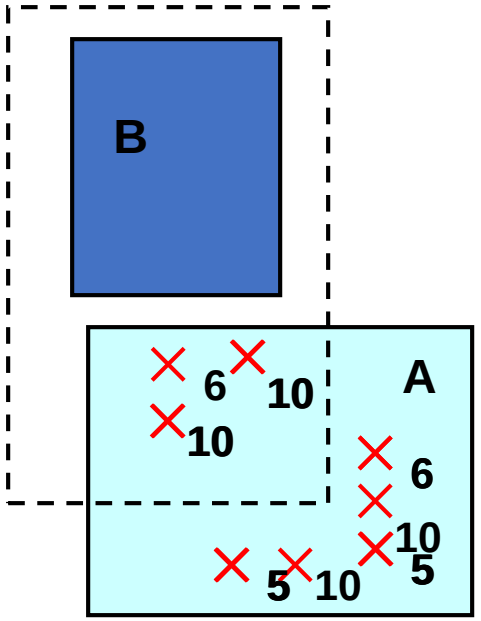
(60+6) / (12+1) = 66/13 = 5.07

最小可能值:

4

因此, **I** = [**4**, **5.07**]

A中还有1个6,
比当前均值大,
加入6



	min	max	count	sum
A	5	10	5	36
B	–	–	10	40

A中一种值的分布{**5**, **5**, **6**, **10**, **10**}

聚集索引

- MRA树聚集分析处理AVG

置信区间

当前avg^{NC} = 45/3 = 15

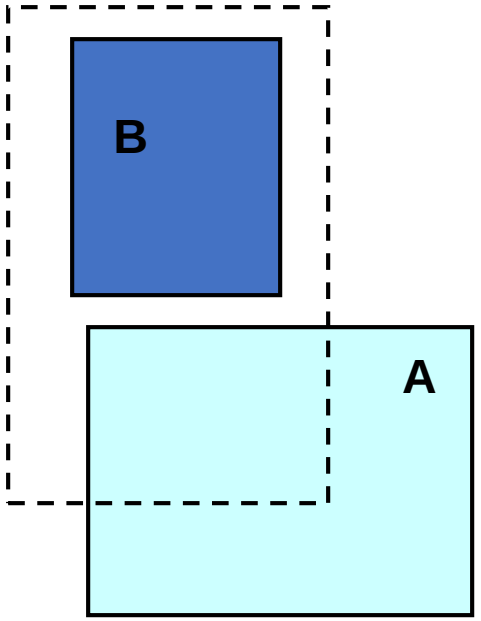
最大可能值:

15

最小可能值:

(45 + 15) / 6 = 10

因此, **I** = [**10**, **15**]



	min	max	count	sum
A	5	10	5	35
B	–	–	3	45

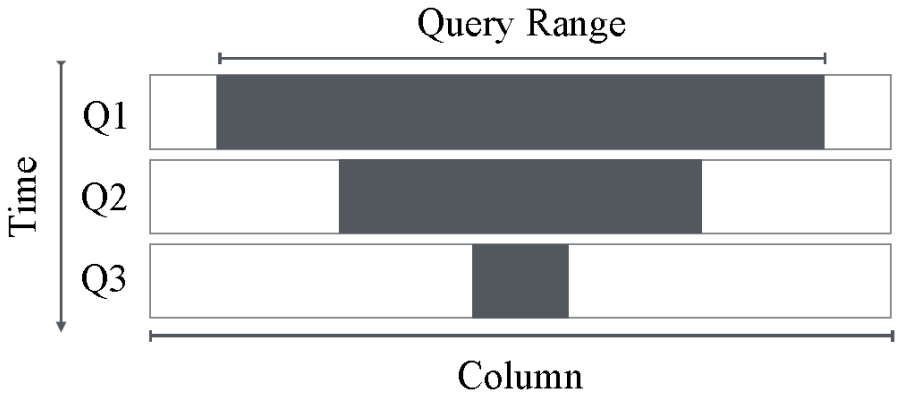
A中一种值的分布{**5**, **5**, **5**, **10**, **10**}

目录

数据分析过程中的一个现象

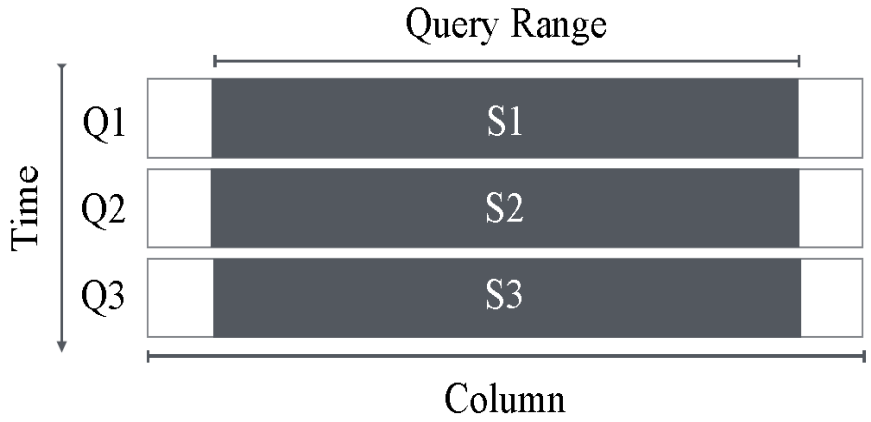
- 探索性、交互性分析中存在大量的重复部分 (Repetitive Calculation of Statistics)
  - ✓在数据的相同部分上连续计算不同的统计信息
  - ✓在与先前访问范围重叠的数据上计算相同的统计信息
- 在对数据的探索性分析过程中，找到最终模式之前，往往会执行很多次上述的重复查询

数据分析中重复计算的种类

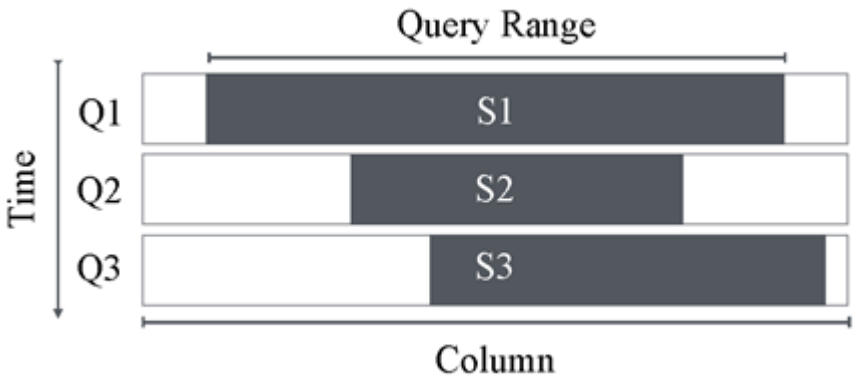


子范围

查询范围部分重叠



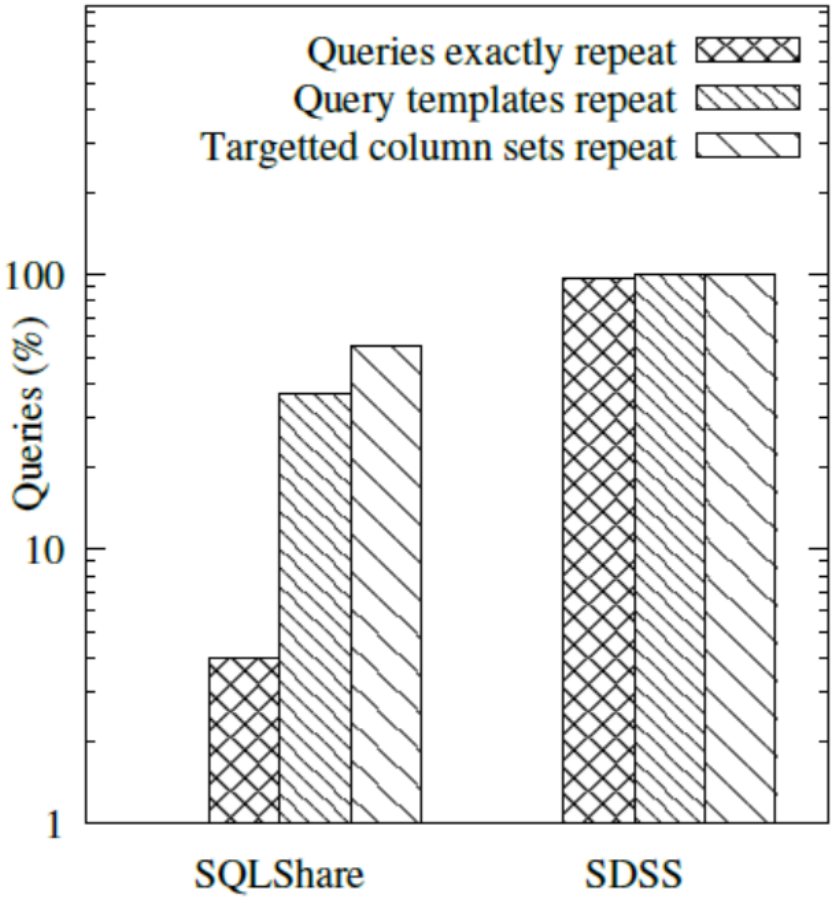
相同范围上不同统计值



混合型

不同工作负载中查询的重复特性分析

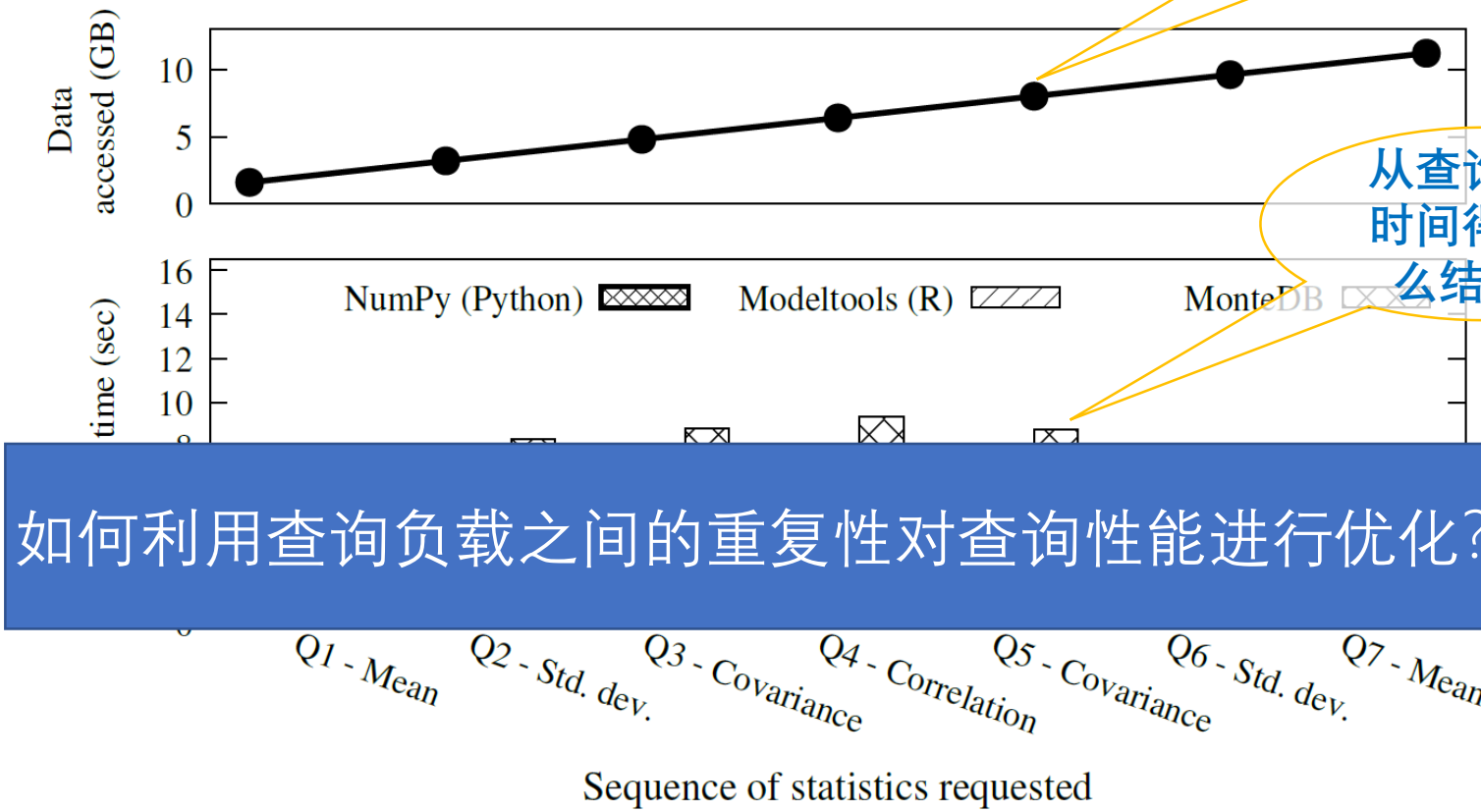
- 通过实验分析查询中的重复特性
- 公开的负载集合SDSS、SQLShare
- 包括手写与自动生成的查询：
  - SDSS中97%查询至少重复一次
  - SQLShare中55%的查询所涉及的属性集合有交集



常用的数据分析工具是否利用了查询之间重复特性呢？

现有数据管理系统的处理方式分析

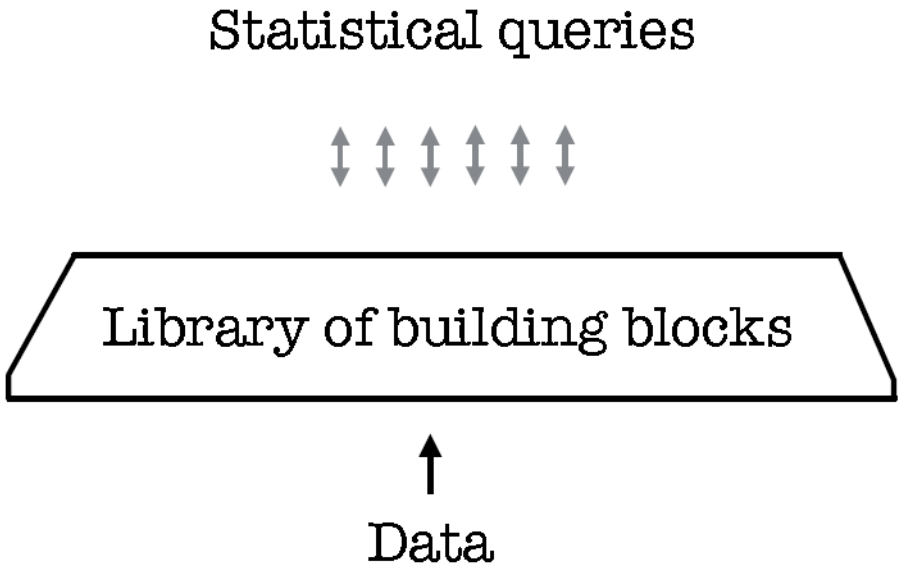
- 实验设置：
  - 工具: NumPy、Modeltools、MonteDB
  - 1亿行double数据
  - 连续提交7个查询、1~3与5~7完全重复



如何利用查询负载之间的重复性对查询性能进行优化?

利用查询的重复性进行优化

- 将工作负载分解为**基本计算单元**
- **缓存**并管理基本计算单元
- 用基本**计算**单元生成未来查询的（部分）结果



Data Canopy: Accelerating Exploratory Statistical Analysis, SIGMOD 2017

Data Canopy使用示例

- 数据集：National Centers for Environmental Information (NCEI)
  - 每小时收集一次温度数据
  - $t_0, t_1, \ldots, t_n$
- 查询内容：给定范围内温度数据（序列）的Mean、Variance、Standard Deviation等统计量
  - Q1: 计算每一天的平均温度
  - Q2: 计算每一周的平均温度
  - Q3: 计算每两周内温度的方差

Data Canopy 示例

Q1: 计算每一天的平均温度

Q2: 计算每一周的平均温度

Q3: 计算每两周内温度的方差

$$V_x = \frac{1}{N}\sum_{i=1}^{N}(xi - \bar{x})^2 = \left(\frac{1}{N}\sum_{i=1}^{N} x_i^2\right) - \left(\frac{1}{N}\sum_{i=1}^{N} x_i\right)^2$$

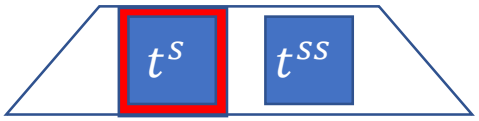
$$t_k^s = \{\sum_{i=0}^{11} t_{i+12k} \,|k \in \{0,1,\dots,728\}\} \quad t_k^{ss} = \{\sum_{i=0}^{11} t_{i+12k}^2 \,|k \in \{0,1,\dots,728\}\}$$

计算、分解、存储 → 重用

Q1访问原始数据
并存储聚集值

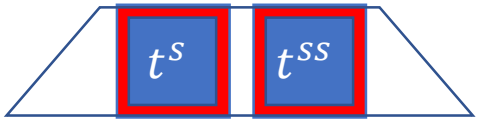
数据

Q2重用公共范围



数据

Q3重用公共
聚集值



数据

$$r_1 = \{\frac{1}{24}(t_k^s + t_{k+1}^s) | k \in \{0,2,\dots,728\}\}$$

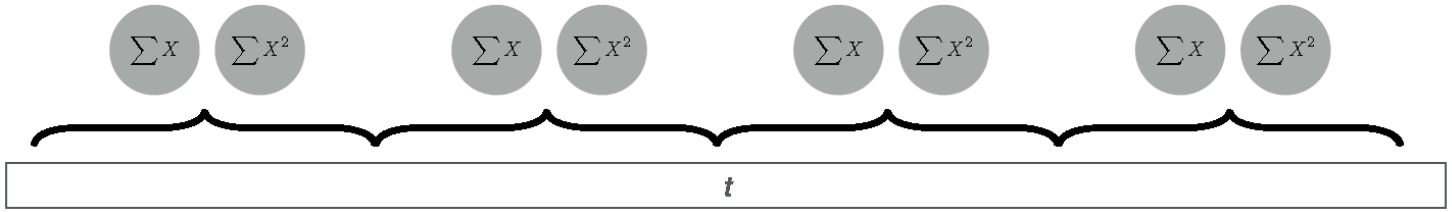
$$r_2 = \{\frac{1}{168}\sum_{i=0}^{13} t_{i+14k}^s \,|k \in \{0,1,\dots,51\}\}$$

$$r_3 = \{\frac{1}{336}\sum_{i=0}^{27} t_{i+28k}^{ss} - \left(\frac{1}{336}\sum_{i=0}^{27} t_{i+28k}^{s}\right)^2 \,|k \in \{0,1,\dots,25\}\}$$

- Data Canopy使用示例

Q: Monthly Variance

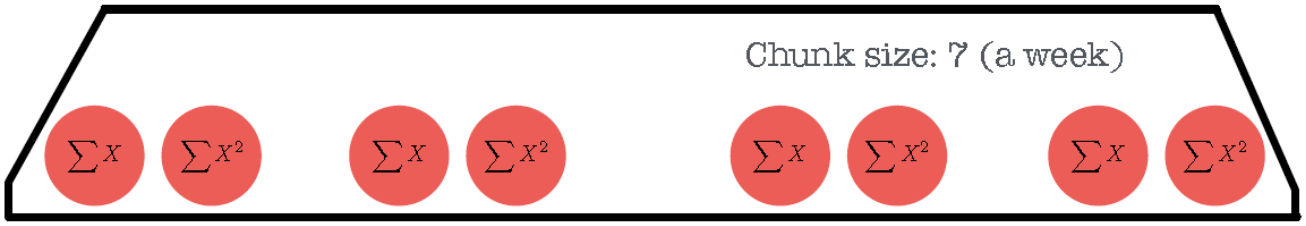
$$\text{Variance} = \left(\frac{1}{n}\sum X^2\right) - \left(\frac{1}{n}\sum X\right)^2 \longrightarrow \sum X \quad \sum X^2$$



Data Canopy

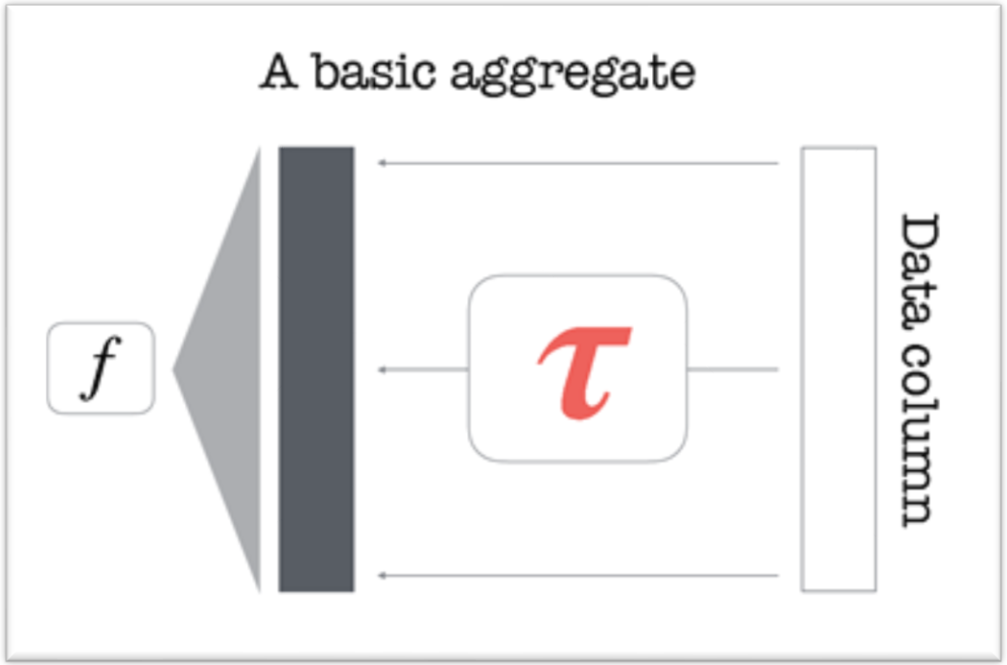
- Data Canopy使用示例

Q: Monthly Variance $\left(\frac{1}{n}\sum X^2\right) - \left(\frac{1}{n}\sum X\right)^2$



Data Canopy：基本聚集(Basic Aggregates)

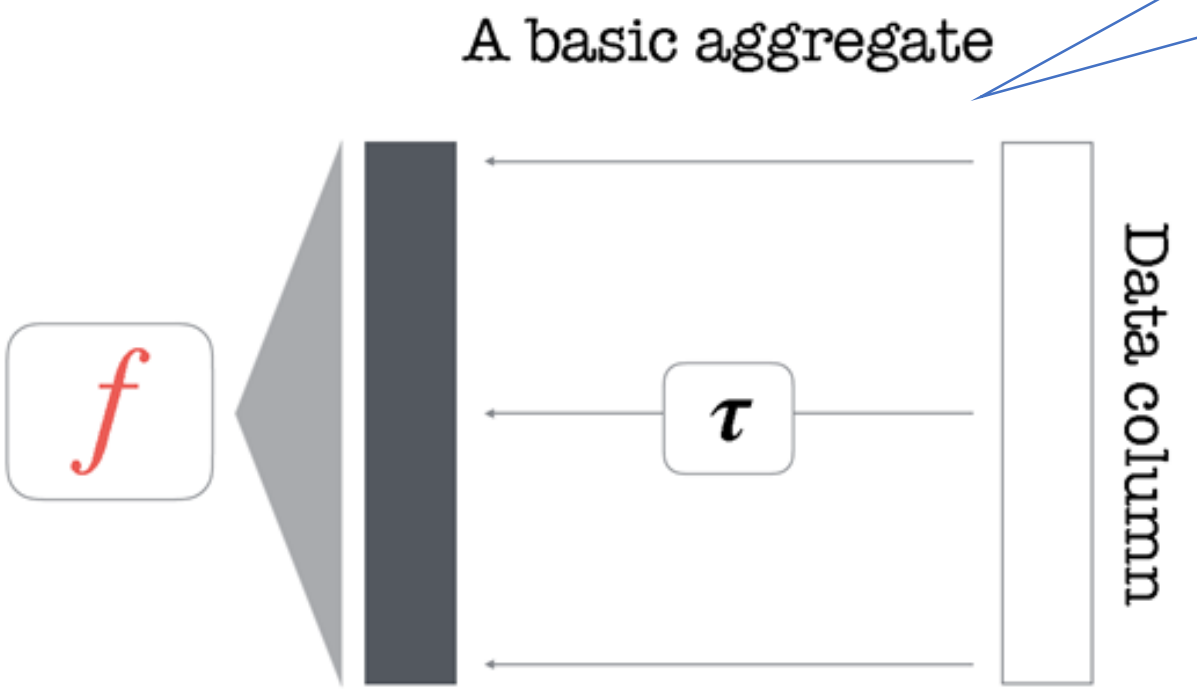
- 将统计度量分解为基本原语，称之为Basic Aggregates
- 将一定**范围**内数据的基本聚集定义为一个值：
  - ✓ 首先对该范围内的每个数据项执行转换τ
  - ✓ 然后使用聚合函数f组合结果而获得
- 例子中t^{SS}就是一个基本聚集：$f(\{\tau(x_i)\}) = \sum_i x_i^2$
  - ✓ $\tau(x_i)=x_i^2$, f是sum函数



Data Canopy：基本聚集(Basic Aggregates)

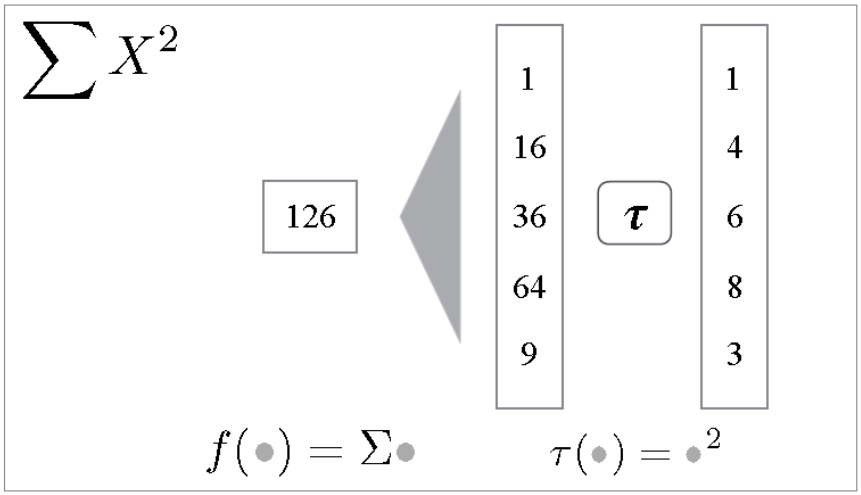
- 转换τ可以是对一个数据项的任意操作
- 聚合函数 f 必须满足结合律和交换律

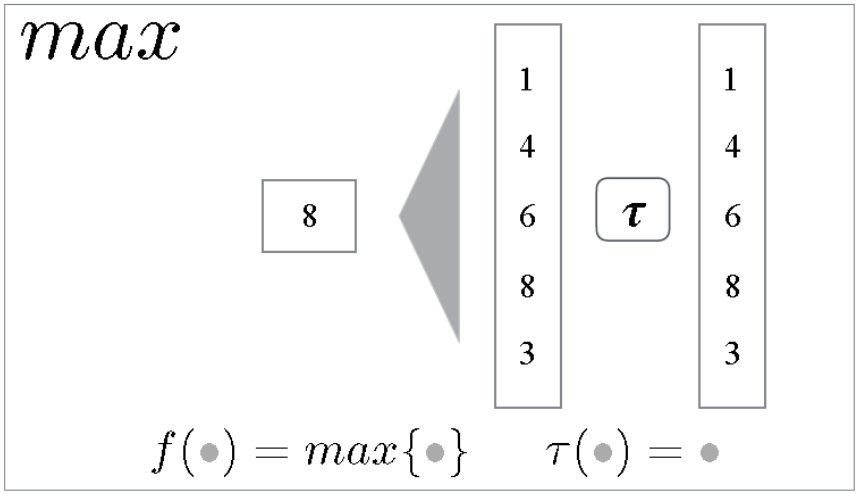
哪些常见的聚集函数满足？哪些不满足？



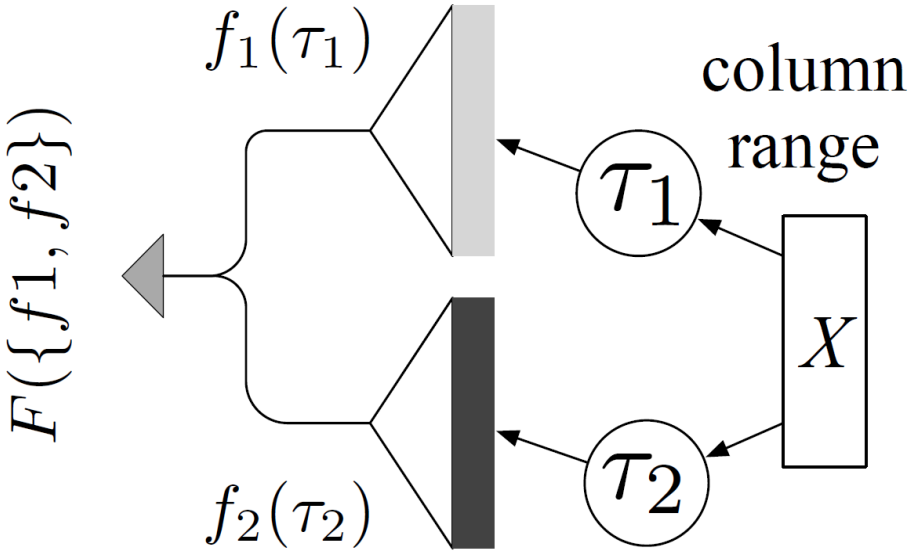
$$f(X) = f(\{f(X_1), f(X_2) \ldots f(X_n)\})$$

- 基本聚集值(Basic Aggregates)示例





- 将统计量(Statistics)分解为基本聚集值(Basic Aggregates)
  - 统计量S定义为基本聚集值的函数F
  - $S(X) = F(\{f(\tau\{x_i\})\})$

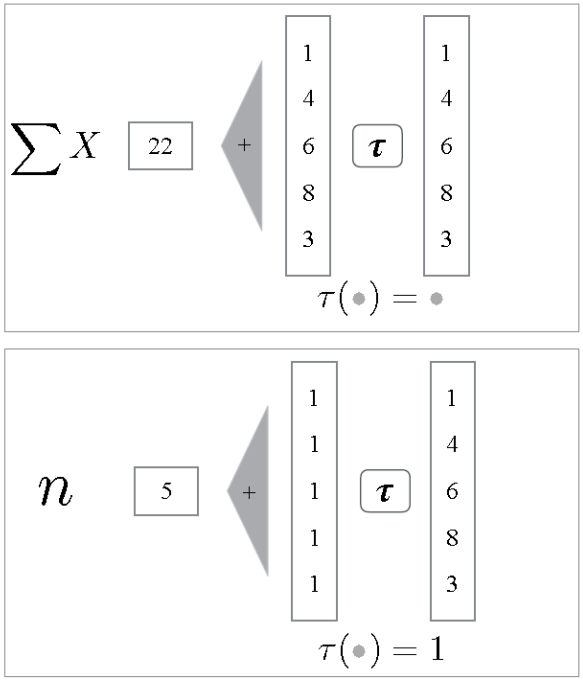


- 将统计量(Statistics)分解为基本聚集值(Basic Aggregates)
  - 示例: 计算算术平均值

$$\frac{1}{n}\sum X$$

Arithmetic Mean

4.4

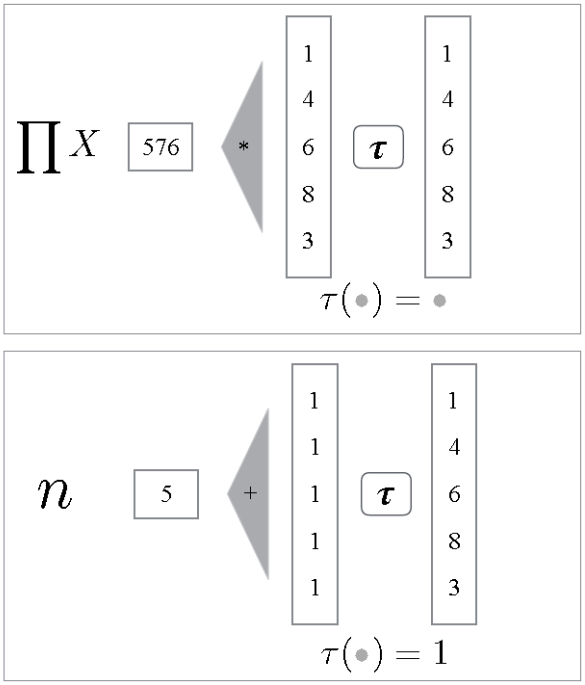


- 将统计量(Statistics)分解为基本聚集值(Basic Aggregates)
  - 示例: 计算几何平均值

$(\prod X)^{\frac{1}{n}}$

Geometric Mean

3.6

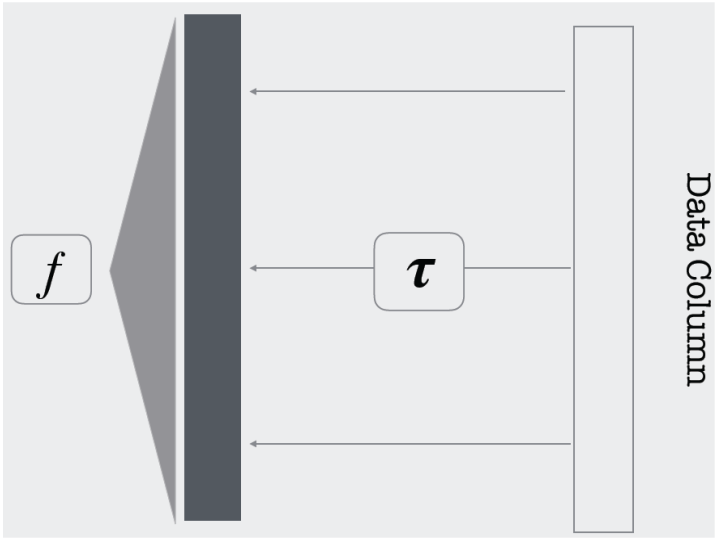


Data Canopy使用的基本聚集值

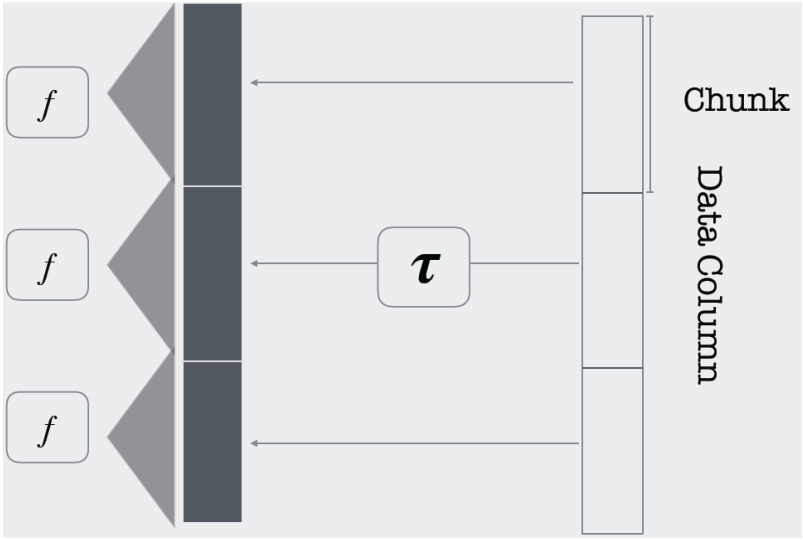
Statistics		Basic Aggregates				
Type	***Formula***	$\sum x$	$\sum x^2$	$\sum xy$	$\sum y^2$	$\sum y$
Mean (avg)	$\frac{\sum x_i}{n}$	■				
Root Mean Square (rms)	$\sqrt{\frac{1}{n} \cdot \sum x^2}$		■			
Variance (var)	$\frac{\sum x_i^2 - n \cdot \text{avg}(x)^2}{n}$	■	■			
Standard Deviation (std)	$\sqrt{\frac{\sum x_i^2 - n \cdot \text{avg}(x)^2}{n}}$	■	■			
Sample Kurtosis (kur)	$\frac{1}{n}\sum\left(\frac{x_i - \text{avg}(x)}{\text{std}(x)}\right)^4 - 3$	■	■			
Sample Covariance (cov)	$\frac{\sum x_i \cdot y_i}{n} - \frac{\sum x_i \cdot \sum y_i}{n^2}$	■		■		■
Simple Linear Regression (slr)	$\frac{\text{cov}(x,y)}{\text{var}(x)}, \text{avg}(x), \text{avg}(y)$	■	■	■		■
Sample Correlation (corr)	$\frac{n \cdot \sum x_i \cdot y_i - \sum x_i \cdot \sum y_i}{\sqrt{n \cdot \sum x_i^2 - (\sum x_i)^2}\sqrt{n \cdot \sum y_i^2 - (\sum y_i)^2}}$	■	■	■	■	■

Data Canopy synthesizes statistics from a library of basic aggregates

- 计算和维护基本聚集值的单位: 块 (Chunk)
  - Data Canopy使用的基本聚集值

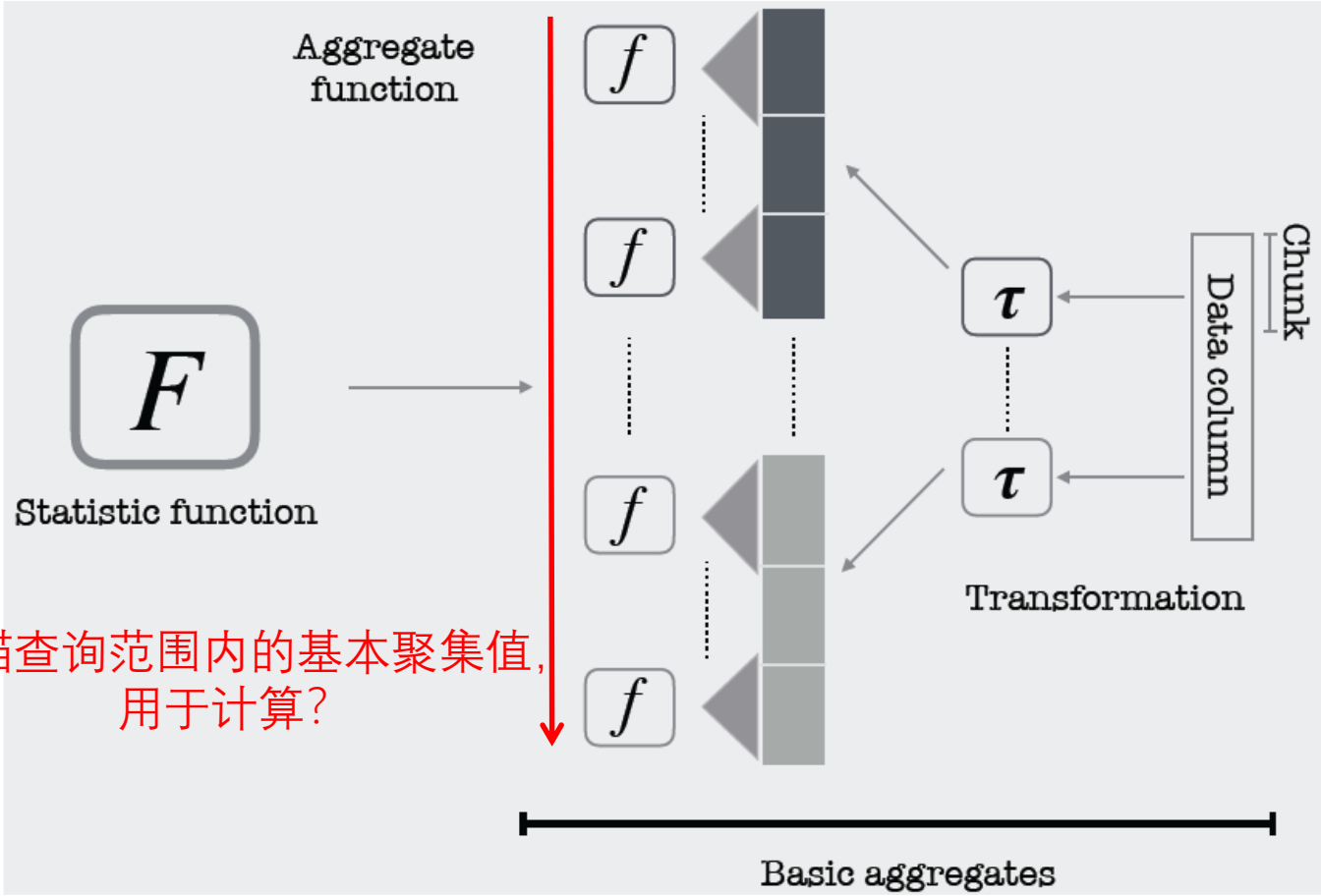


整个列上的基本聚集值重用性低



以块为单位计算基本聚集值: 提高重用性

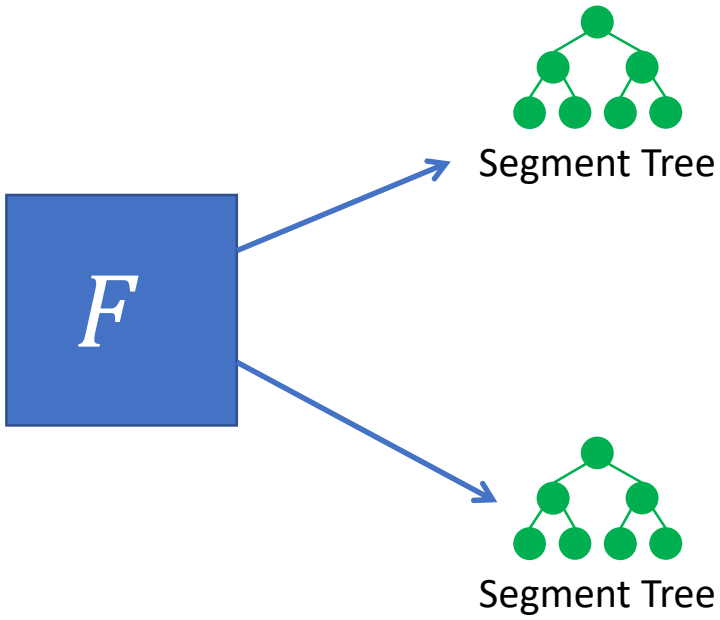
- 计算和维护基本聚集值的单位: 块 (Chunk)
  - 使用分块后的基本聚集值

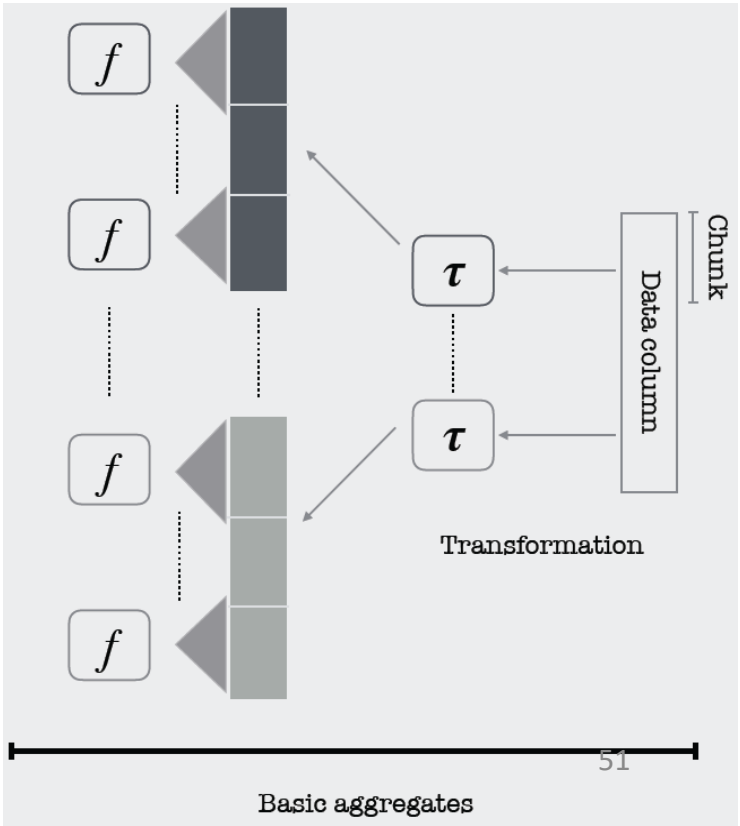


扫描查询范围内的基本聚集值，用于计算？

Data Canopy

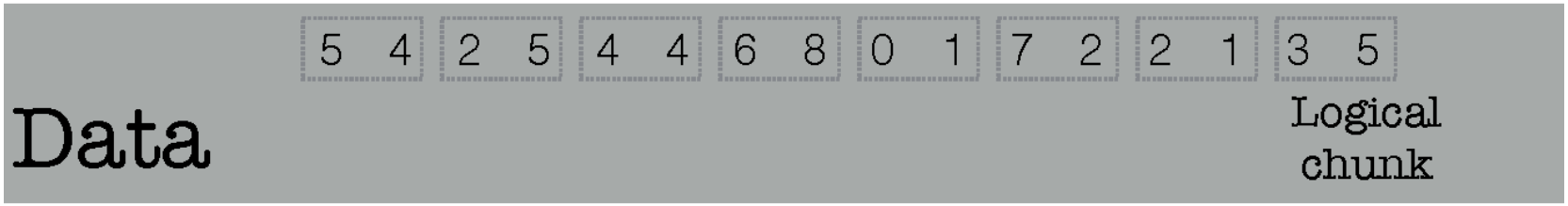
- Data Canopy的核心数据结构: Segment Tree集合
  - 为每个列的每个基本聚集值建立一个Segment Tree (ST)
  - 在Data Canopy的目录中建立哈希表
    - 存储指向各个Segment Tree的指针





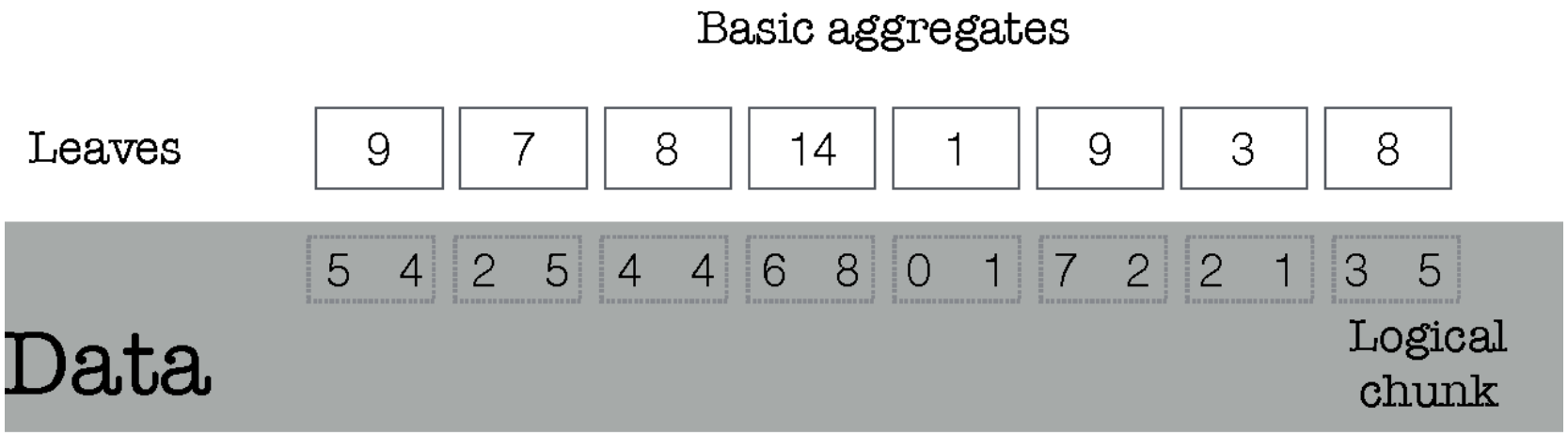
Data Canopy

- Data Canopy中的Segment Tree
  - 16个数据
  - 每个块容纳2个数据



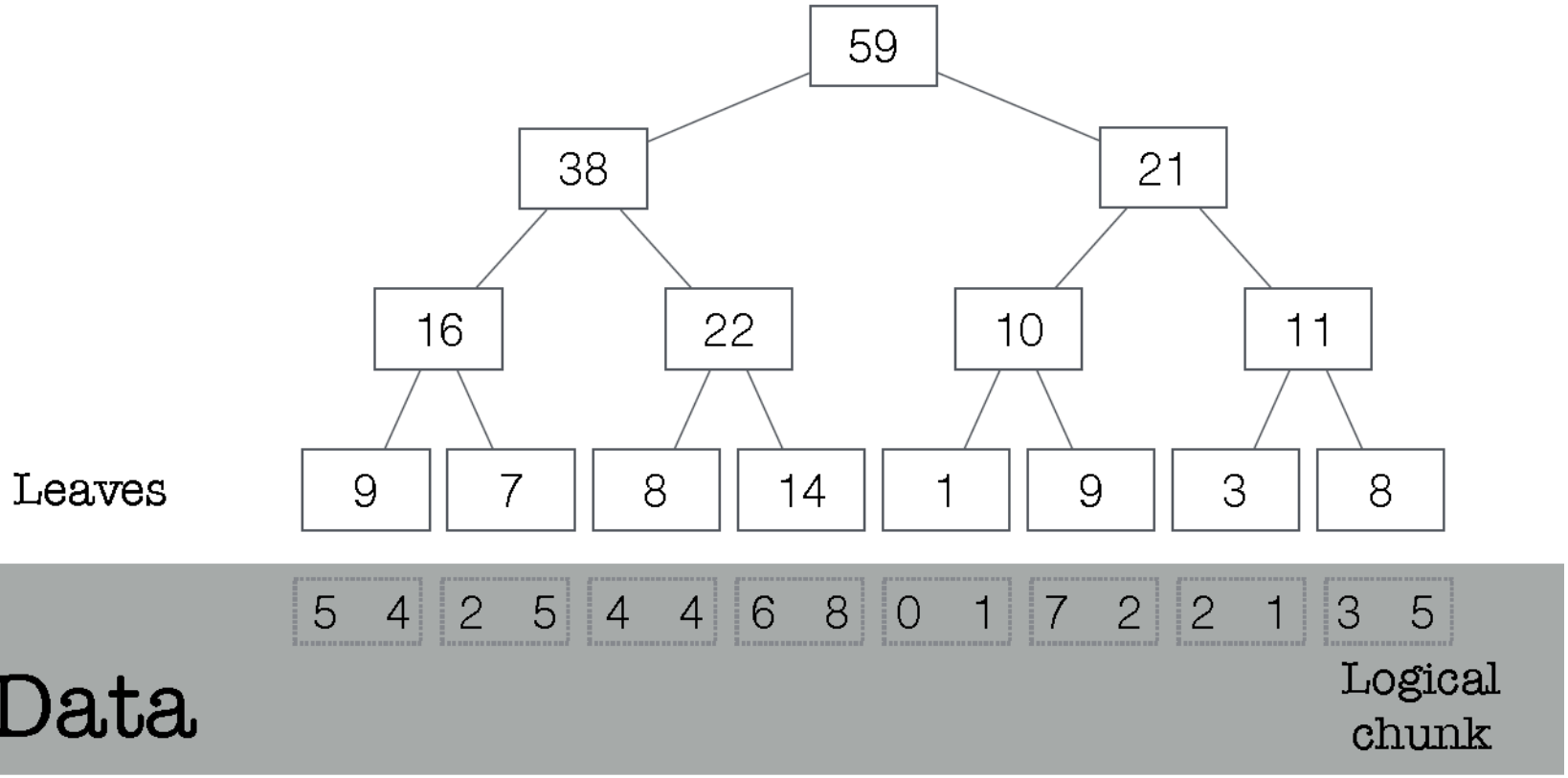
Data Canopy

- Data Canopy中的Segment Tree
  - 为每个块构建一个Segment Tree的叶子节点
  - 叶子节点中存储对应块的基本聚集值

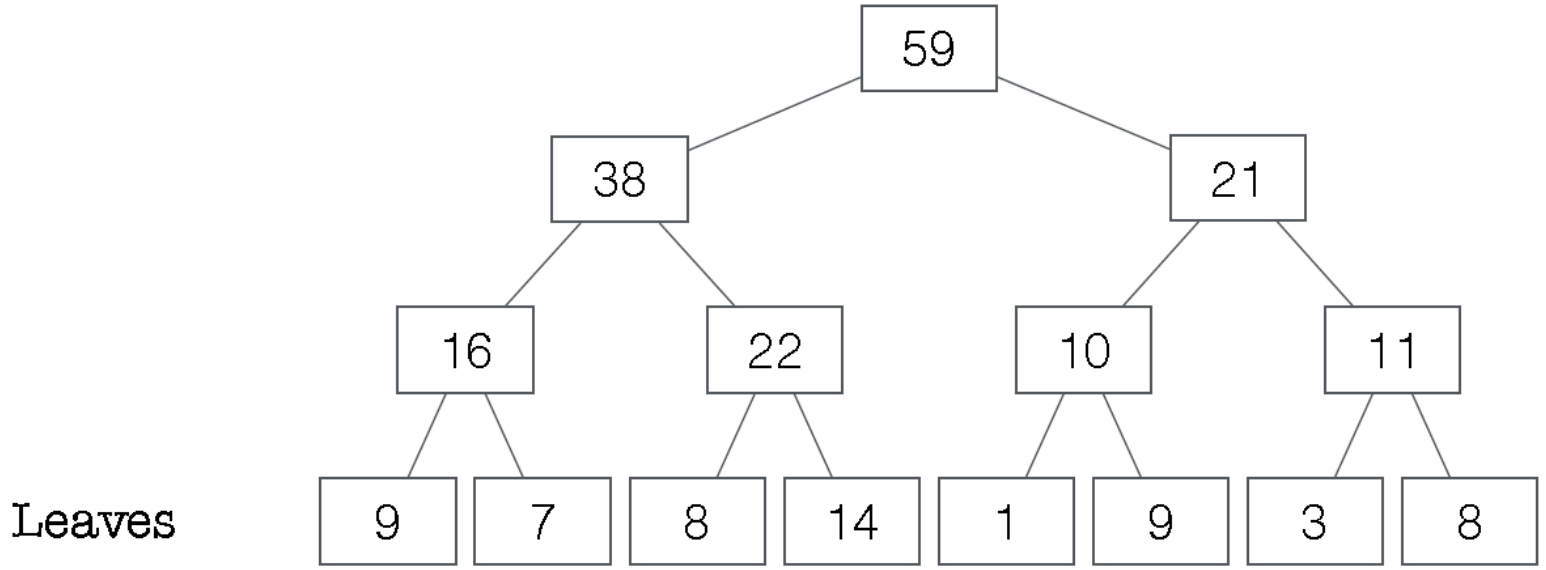


Data Canopy

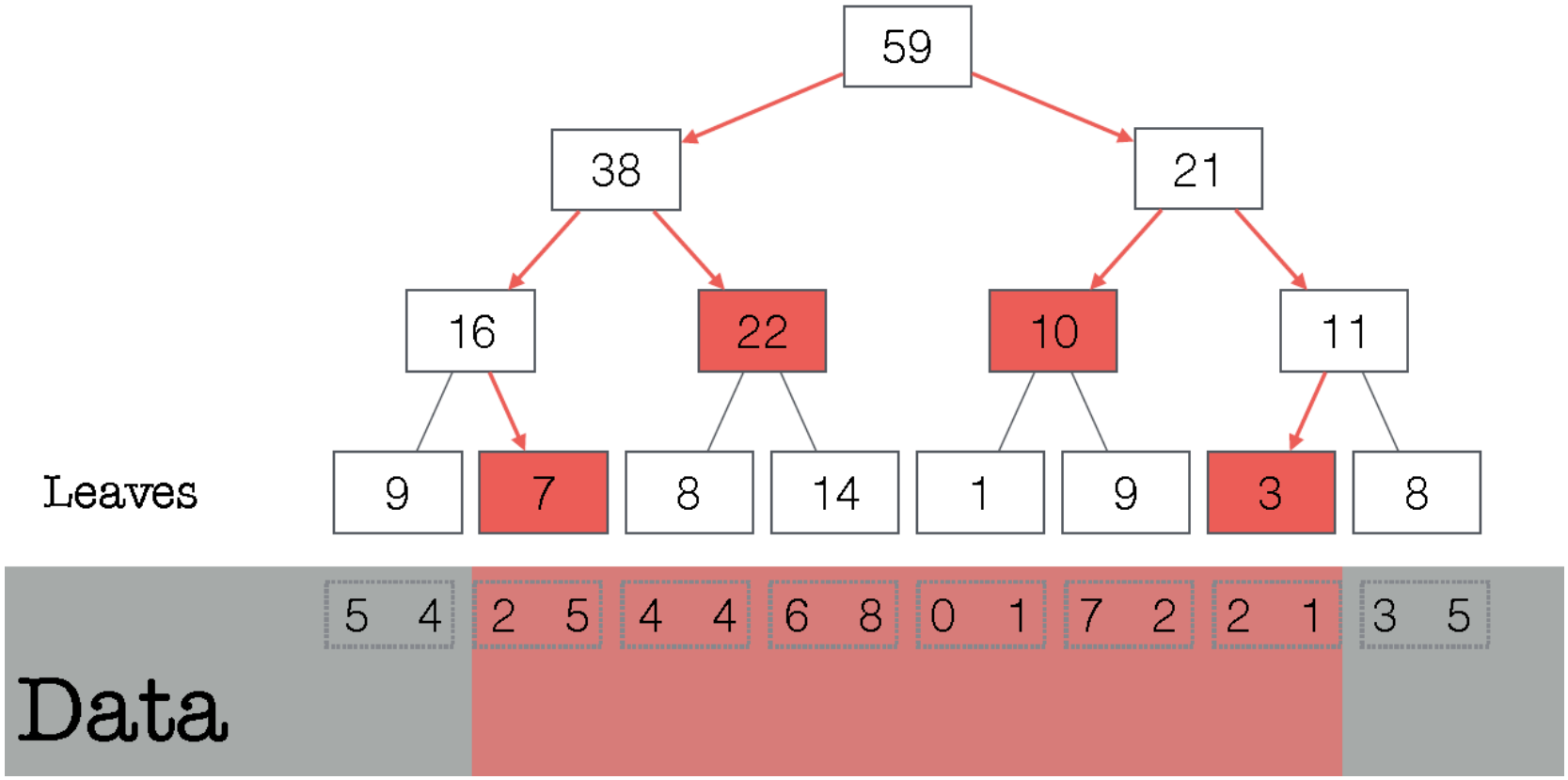
- Data Canopy中的Segment Tree
  - 从叶子节点向上构建一颗Segment Tree



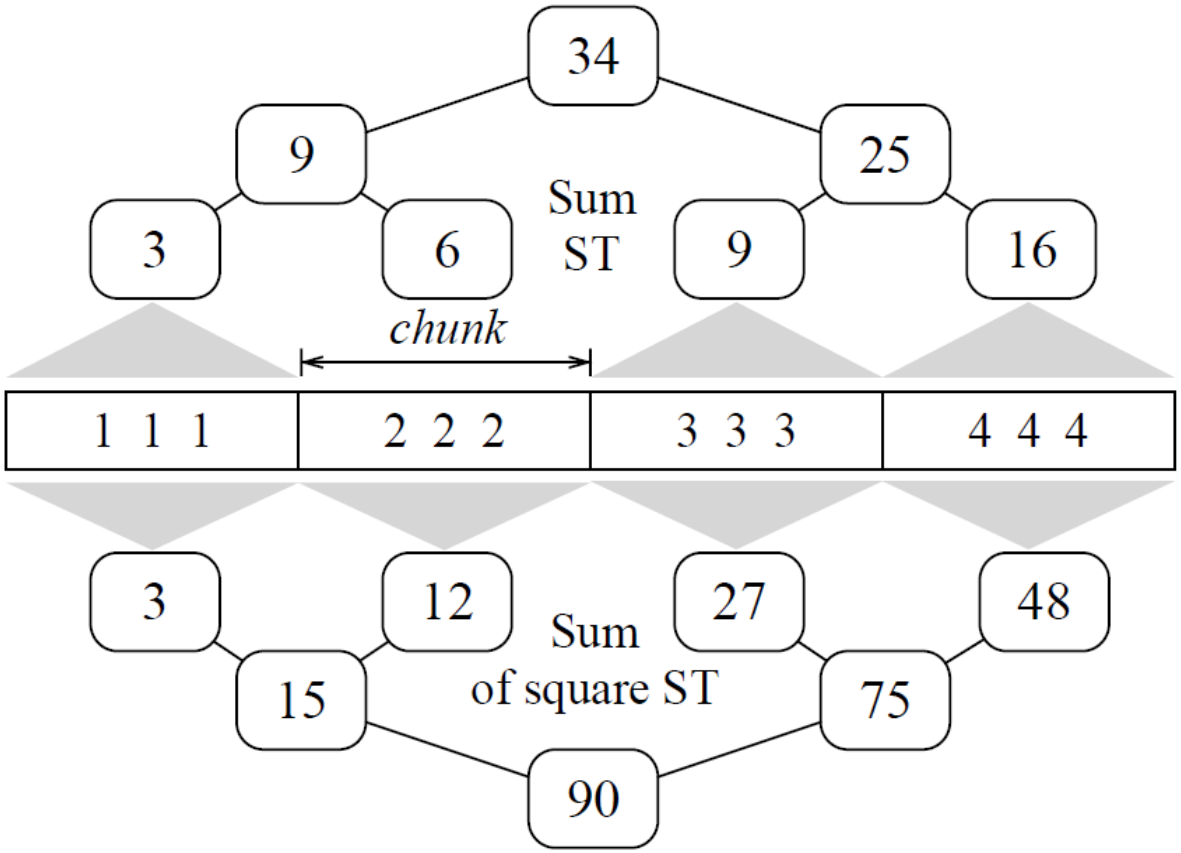
- Data Canopy中的Segment Tree
  - 查询范围从第3个数据到第14个数据



- Data Canopy中的Segment Tree
  - 查询范围从第3个数据到第14个数据



- Data Canopy中的Segment Tree示例
  - 两棵Segment Tree、块容纳3个数据



Data Canopy

- Data Canopy的核心数据结构: Segment Tree集合
  - 为每个列的每个基本聚集值建立一个Segment Tree (ST)
  - 在Data Canopy的目录中建立哈希表
    - 存储指向各个Segment Tree的指针
- 不同的Segment Tree构建方式
  - 为整个Data Canopy建立一个Segment Tree
  - 为每个列建立一个Segment Tree
  - 为每个基本聚集值建立一个Segment Tree
  - 为每个列的每个基本聚集值建立一个Segment Tree

内存开销、查询、更新代价有何不同?

Data Canopy

- 不同的Segment Tree构建方式
  - 内存开销: ST节点数量
  - 查询/更新代价: ST节点访问数量

Segment Tree选项	内存用量	查询/更新代价
ST per Data Canopy	$2 \cdot b \cdot c \cdot h - 1$	$O(\log b \cdot c \cdot h)$
ST per column	$2 \cdot b \cdot c \cdot h - c$	$O(\log b \cdot h)$
ST per statistic	$2 \cdot b \cdot c \cdot h - b$	$O(\log c \cdot h)$
ST per column per statistic	$2 \cdot b \cdot c \cdot h - b \cdot c$	$O(\log h)$

符号	内容
c	列的数量
b	基本统计量的数量
r	行的数量
h	桶的数量
s	桶的大小(bytes)
v_d	记录的大小(bytes)
v_{st}	ST节点的大小(bytes)
#	缓存行大小

Data Canopy为每个列的每个基本聚集值建立一个Segment Tree

Data Canopy

- 为每个列的每个基本聚集值建立一个Segment Tree
  - 更好的灵活性
    - 不必实现开辟大量内存空间
    - 针对新的列、基本聚集值灵活地添加Segment Tree
  - 更好的并行性
    - 建立、使用Segment Tree得到最大的并行性
    - 多线程
    - 多计算节点

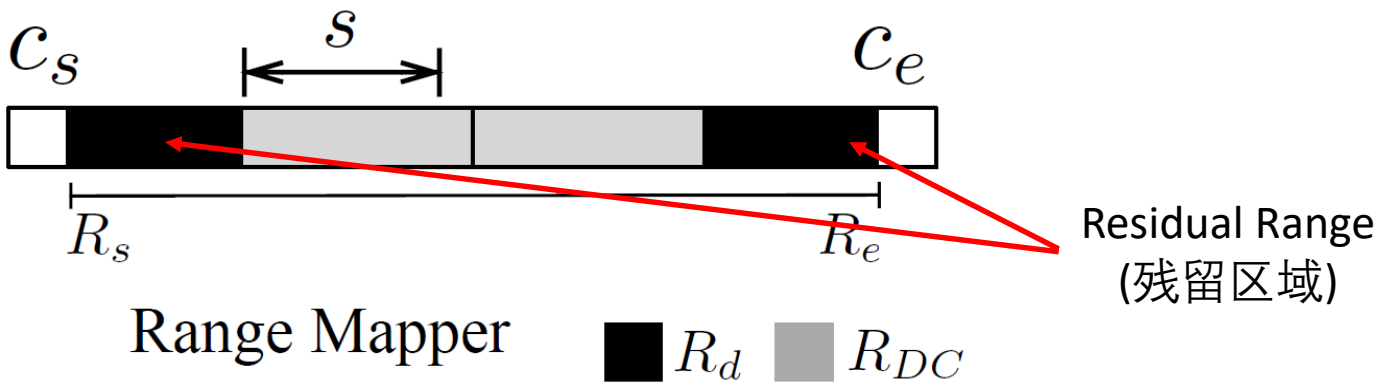
- Segment Tree的建立模式
  - Offline (离线建立模式)
    - 在查询到来前建立所有Segment Tree
    - 覆盖所有行、列、基本聚集值
  - Online (在线建立模式)
    - 在查询进行时建立或补全Segment Tree
    - 缓存查询使用的基本聚集值，并加入Segment Tree中
  - Speculative (投机建立模式)
    - 访问任何部分数据时，建立该部分数据上所有一元的基本聚集值

系统空闲时采用offline模式，繁忙时转为online模式

- Data Canopy查询处理
  - 查询定义
    - $Q = \{\{C\}, [R_s, R_e), S\}$

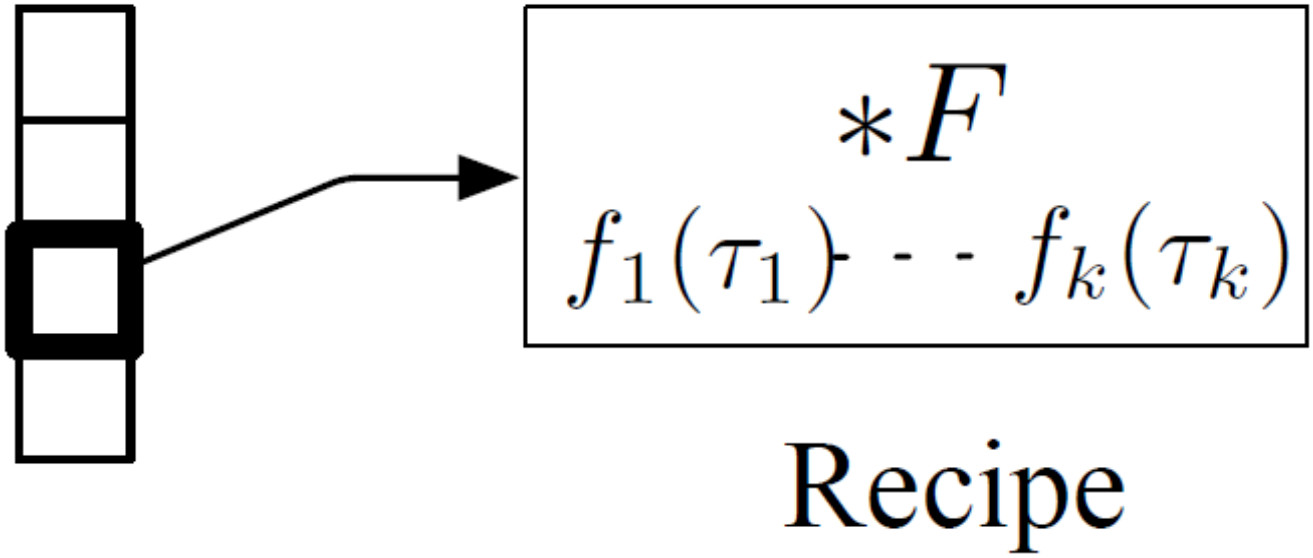
C：列的集合
$[R_s, R_e)$：查询数据范围
S：待计算的统计量

  - 第一步: 将查询范围映射到块范围
    - 根据块的容量、查询范围$[R_s, R_e)$计算块范围$[c_s, c_e]$



    - R_{DC}: 由包含于查询范围的块组成，计算使用块的基本聚集值
    - R_d: 查询范围两端与块部分相交的区域，计算需要扫描原始数据

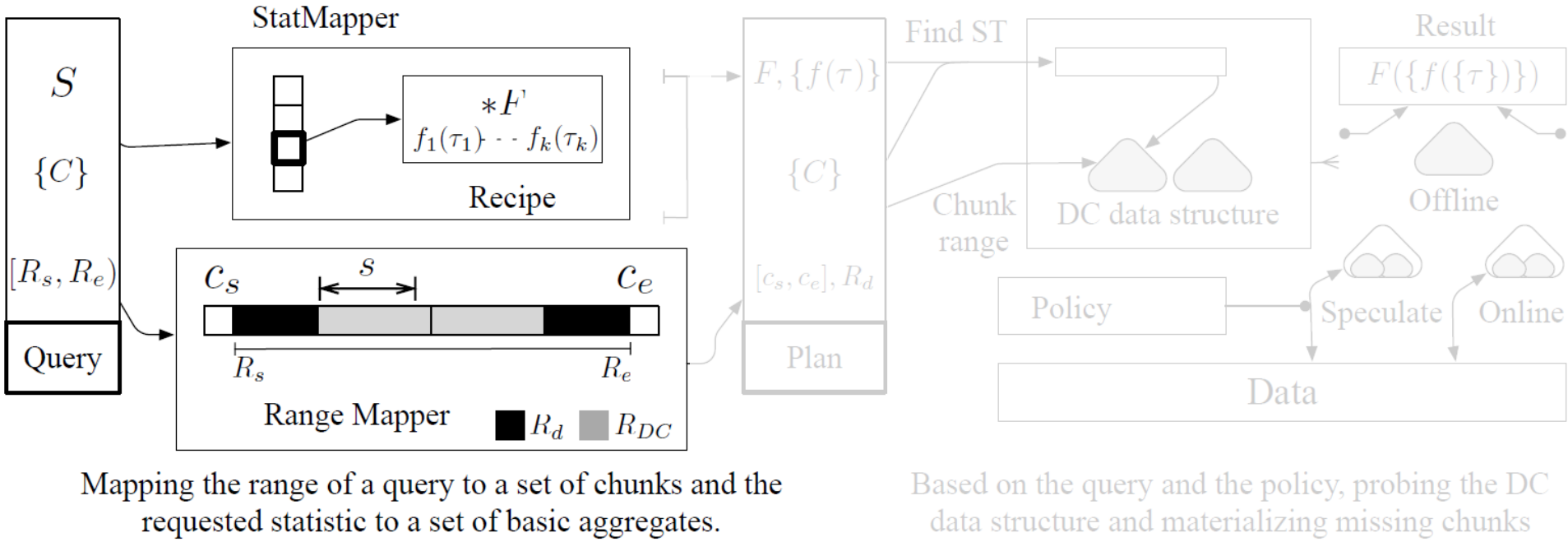
- Data Canopy查询处理
  - 第二步: 将统计量映射到基本聚集值
    - 维护一个哈希表，键为统计量的ID，值为一个recipe (食谱)
    - 统计量S对应的食谱中存储使用基本聚集值合成S的方法



- Data Canopy查询处理
  - 第三步: 执行查询计划
    - Offline模式:
      - 访问R_d中的数据、访问Segment Tree
      - 合成基本聚集值、合成统计量
    - Online模式:
      - 访问R_d中的数据
      - 如果所需基本聚集值不存在，访问原始数据并建立/更新Segment Tree
      - 合成基本聚集值、合成统计量

Data Canopy

- Data Canopy查询处理
  - 整体执行流程

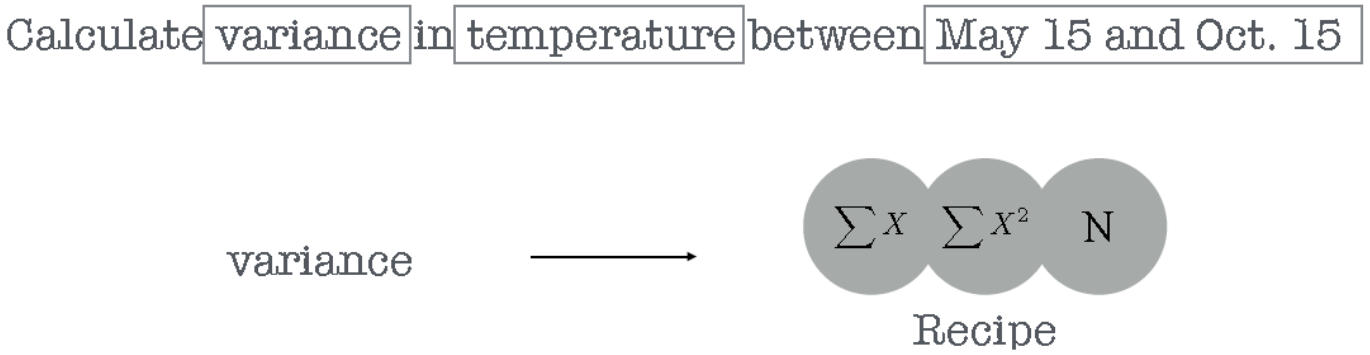


Mapping the range of a query to a set of chunks and the requested statistic to a set of basic aggregates.

Based on the query and the policy, probing the DC data structure and materializing missing chunks

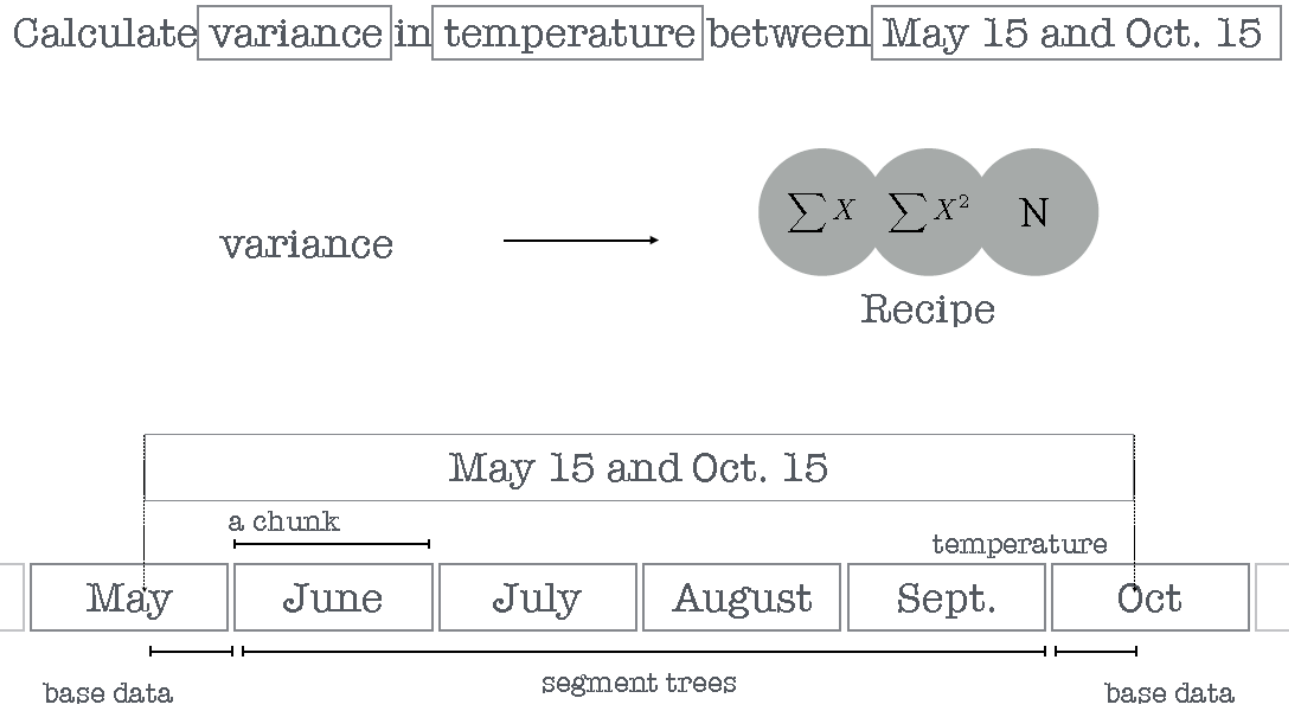
- Data Canopy查询处理
  - 示例

- Data Canopy查询处理
  - 示例



Data Canopy

- Data Canopy查询处理
  - 示例



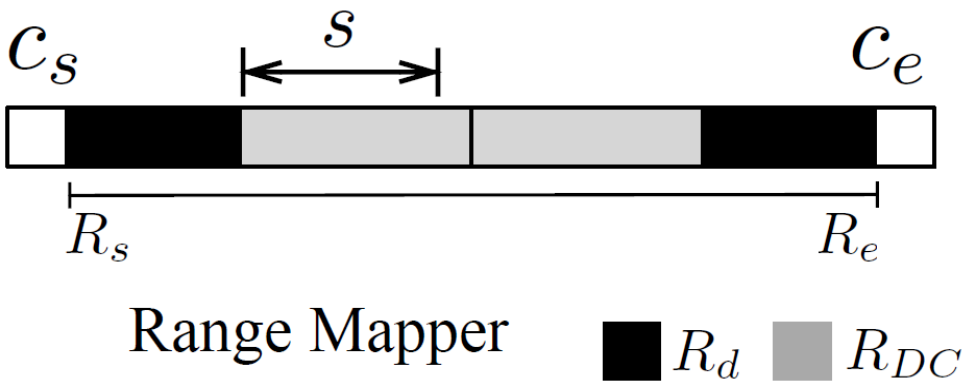
- Data Canopy查询代价分析
  - 建立内存中的代价模型
  - 用数据访问量衡量(缓存行读取数量)
  - 假设一个统计量S的计算涉及k个列，共需b个基本聚集值
    - 访问b个Segment Tree
    - 访问$2k$个residual range
  - 查询代价$C_{syn} = C_{st} + C_r$
    - C_{st}: 访问b个Segment Tree的代价
    - C_r: 扫描$2k$个residual range的代价

Variance: b = ? k = ?
Correlation: b=? k=?

符号	内容
c	列的数量
b	基本统计量的数量
r	行的数量
h	桶的数量
s	桶的大小(bytes)
v_d	记录的大小(bytes)
v_{st}	ST节点的大小(bytes)
#	缓存行大小

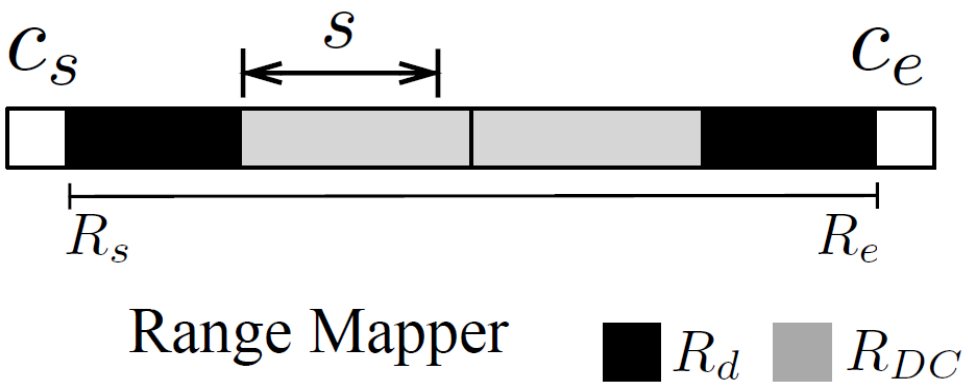
- Data Canopy查询代价分析
  - 用数据访问量衡量(缓存行读取数量)
  - 在Segment Tree中获取R_{DC}的基本聚集值的代价C_{st}
    - 读取缓存行的数量等于访问ST节点的数量
    - 该部分代价$C_{st} = 2 \cdot b \cdot \log_2(\frac{r \cdot v_d}{s})$

每棵ST的叶节点数是多少?



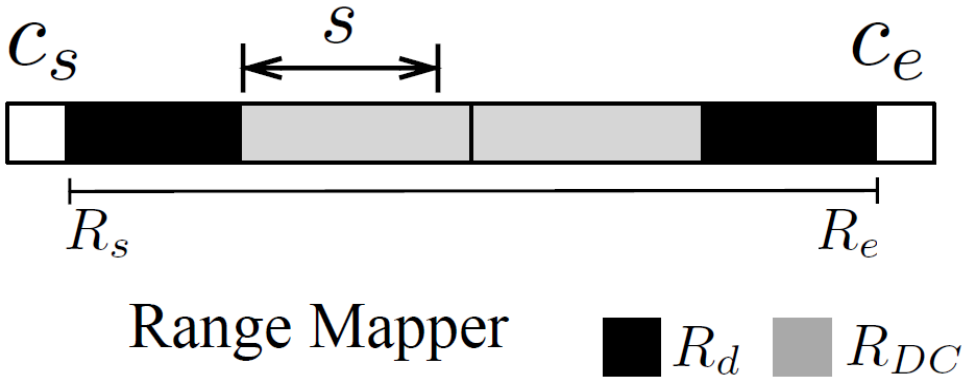
符号	内容
c	列的数量
b	基本统计量的数量
r	行的数量
h	桶的数量
s	桶的大小(bytes)
v_d	记录的大小(bytes)
v_{st}	ST节点的大小(bytes)
#	缓存行大小

- Data Canopy查询代价分析
  - 用数据访问量衡量(缓存行读取数量)
  - 计算扫描k个列上的residual range的代价C_r
    - 最多扫描$2k$个块
    - 每个块的扫描代价为$\frac{s}{\#}$
    - 故而$C_r = \frac{2 \cdot k \cdot s}{\#}$



符号	内容
c	列的数量
b	基本统计量的数量
r	行的数量
h	桶的数量
s	桶的大小(bytes)
v_d	记录的大小(bytes)
v_{st}	ST节点的大小(bytes)
#	缓存行大小

- Data Canopy查询代价分析
  - 用数据访问量衡量(缓存行读取数量)
  - $C_{syn} = C_{st} + C_r$
  - $C_{st} = 2 \cdot b \cdot \log_2\left(\frac{r \cdot v_d}{s}\right)$
  - $C_r = \frac{2 \cdot k \cdot s}{\#}$
  - $C_{syn} = 2 \cdot b \cdot \log_2\left(\frac{r \cdot v_d}{s}\right) + \frac{2 \cdot k \cdot s}{\#}$



符号	内容
c	列的数量
b	基本统计量的数量
r	行的数量
h	桶的数量
s	桶的大小(bytes)
v_d	记录的大小(bytes)
v_{st}	ST节点的大小(bytes)
#	缓存行大小

- Data Canopy查询代价分析
  - 用基本统计量合成永远比扫描快吗？
  - 扫描查询范围R的代价为

$$C_{scan} = \frac{R \cdot v_d}{\#}$$

$$\frac{R_b \cdot v_d}{\#} = 2 \cdot b \cdot \log_2\left(\frac{r \cdot v_d}{s}\right) + \frac{2 \cdot k \cdot s}{\#}$$

  - 扫描与合成代价相等的临界值R_b

$$R_b = \frac{2 \cdot k \cdot s}{v_d} + \frac{2}{v_d} \cdot \# \cdot b \cdot \log_2\left(\frac{r \cdot v_d}{s}\right)$$

  - $R > R_b$时，使用合成的计算方式
  - $R \leq R_b$时，使用扫描的计算方式

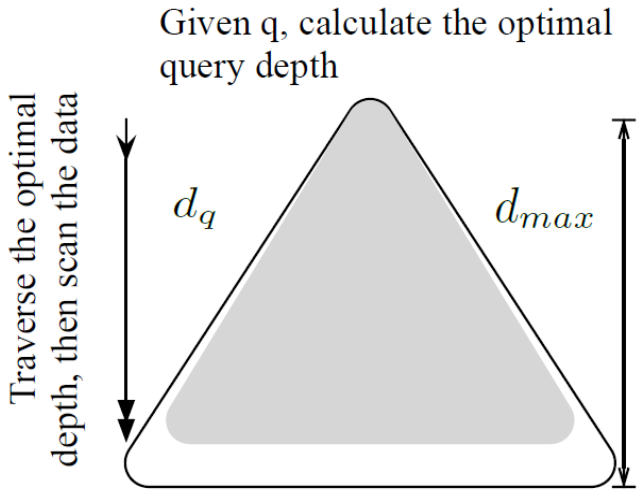
符号	内容
c	列的数量
b	基本统计量的数量
r	行的数量
h	桶的数量
s	桶的大小(bytes)
v_d	记录的大小(bytes)
v_{st}	ST节点的大小(bytes)
#	缓存行大小

- Data Canopy中块的容量对查询性能的影响
  - $C_{syn} = 2 \cdot b \cdot \log_2\left(\frac{r \cdot v_d}{s}\right) + \frac{2 \cdot k \cdot s}{\#}$
  - 与数据相关的参数: r 、 v_d
  - 与查询相关的参数: b、k
  - 与硬件相关的参数: $\#$
  - 与Data Canopy中Segment Tree相关的参数: s（块的大小）
    - 给定r 、 v_d、 b、 k、 $\#$，求s使得C_{syn}最小
    - 解得$s_0 = \frac{b \cdot \#}{k \cdot \ln 2}$

对于不同的查询，最优的块容量是不相同的

符号	内容
c	列的数量
b	基本统计量的数量
r	行的数量
h	桶的数量
s	桶的大小(bytes)
v_d	记录的大小(bytes)
v_{st}	ST节点的大小(bytes)
$\#$	缓存行大小

- 对于不同的查询，最优的块容量是不相同的
  - 在每个列的每个基本聚集值上为不同查询建立不同的ST？
    - 内存开销巨大，不可行！
  - 建立一个块容量最小的Segment Tree
    - $\frac{b}{k} \geq 1$，令$b = k = 1$，ST高度为d_{max}
    - 对于查询q，计算其最优块容量$s_q = \frac{b_q \cdot \#}{k_q \cdot \ln 2}$
    - 计算q在ST中的最优探索深度$d_q = \log_2(\frac{r \cdot v_d}{s_q})$
    - 扫描至d_q后，得到子树中块容量之和接近q的最优块容量



- Data Canopy处理内存不足的方法
  - 第一阶段:
    - Segment Tree向上收缩，直至块大小超过磁盘块，进入第二阶段；
  - 第二阶段:
    - 替换数据页面到磁盘，使用LRU机制
    - 查询过程中外存访问次数有界: $2k$次外存I/O
  - 第三阶段:
    - 内存中没有任何数据页面
    - 替换单位是整棵的Segment Tree
    - Serialize()、Deserialize()