项目介绍

PostgreSQL 是一款功能强大的开源对象-关系型数据库管理系统（ORDBMS），以严格遵循 SQL 标准、支持完整 ACID 事务、多版本并发控制（MVCC）以及高度的可扩展性（如支持自定义函数、数据类型和扩展如 PostGIS）著称。其市场地位稳固，是**全球第二大开源数据库、2025年StackOverflow调研中最流行、最受喜爱和需求量最高的数据库，开发者使用率超55%**，领先优势明显

pgvector是postgresql向量数据库实现的**事实标准**

我们正为pgvector添加**首个向量查询的gpu加速支持**

ivfflat算法

1. 聚类：对data向量做k-means聚类，每个聚类建立倒排表
2. 粗筛：计算query向量与每个聚类中心的聚类，排序得到n_probe个最近的的候选聚类
3. 精筛：计算query向量与n_probes个候选聚类中每个向量的距离，排序得到k近邻

距离可以是**欧氏距离**或**余弦距离**

适合场景：所有索引能够存在内存中

Ivfpq乘积量化

分治：把大向量切成小向量再做K-means

如何复原：把小向量的聚类中心拼起来

实现：

1. 将向量沿着维度划分为多个子空间
2. 每个子空间做Kmeans（一般是$\sqrt{n}$）并保存聚类中心
   查询的时候从聚类里面找

变体：

1. OPQ：先旋转矩阵（向量组）再做PQ
2. stack PQ：对第一次PQ的误差再做K-means

Ivfjl

J-L引理：为了保存N个向量，可以把高维向量映射为低维向量，各个向量之间的距离高概率分布

映射矩阵可以用：

1. 正态分布随机生成
2. 可以用1，-1随机矩阵、甚至是稀疏矩阵

向量传输空间变少、距离计算更简单

K-means 聚类算法

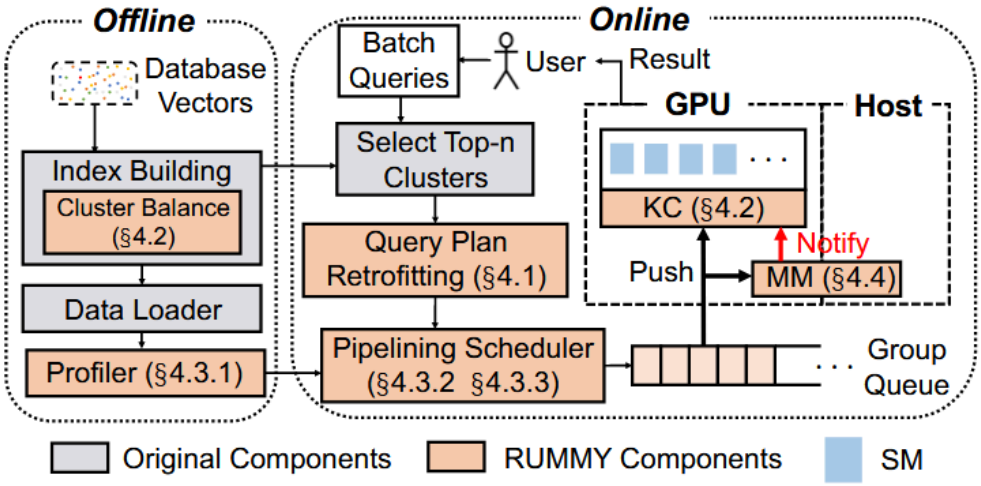
1. 初始化n个聚类中心
2. 加入新向量，看他们离哪个聚类中心最近，就属于哪个聚类。更新聚类中
3. 迭代直至收敛或满足精度要求

Rummy

开源：https://github.com/pkusys/Rummy.

Overview

数据库 [N,D] N是十亿级别



三个优化点

1. 不同查询之间造成数据加载冗余
2. 如果一个聚类只使用一个SM，那么GPU使用会不充分
3. 优化查询顺序使得传输时间和计算时间尽可能重合

符号表

Symbol	Description
C_i	The i_{th} cluster
Q_i	The i_{th} query vector
G_i	The i_{th} group
$\{C_1, C_2 \ldots\}$	A set of original clusters
$[C_1, C_2 \ldots]$	The execution order of original clusters
$\{B_1, B_2 \ldots\}$	A set of balanced clusters
$[B_1, B_2 \ldots]$	The execution order of balanced clusters
ρ	The fixed size of balanced clusters (the number of vectors)
$T(G)$	The transmission (time) of group G
$E(G)$	The computation (time) of group G
$Q_i \rightarrow B_j$	The computation (thread block) of Q_i on B_j

Table 1: Key notations in the design.

优化点1：query重排序

对query的改造为两个部分1.**intra-batch** 2. **inner-batch**
区别在于infra-batch不改变查询顺序，inner-batch改变
传输的cluster数量一定不低于涉及的cluster数量
最高目标是每个涉及的cluster只传输一次

优化点2：Cluster-based Retrofitting

dynamic kernel padding with cluster balancing
straggler problem：某些线程块执行时间过长，导致拖慢整体运行效率

Cluster Balancing（集群平衡）
解决时间利用率

1. 集群拆分：
   • 原始集群 C_i 被拆分为多个大小均衡的子集群 {B_{i_1}, B_{i_2}, ...}，每个子集群大小固定为 ρ。
2. 离线处理：
   • 在主机内存（Host Memory）中完成均衡化，运行时仅将均衡后的集群传输到GPU，减少在线开销。

Dynamic Kernel Padding动态内核填充）
解决空间利用率

目标：通过运行时动态填充线程块，实现100% SM利用率和GPU占用率（Occupancy）。

技术实现

1. 内核填充原理：
   • 将线程块 Q_i \to B_j 动态拆分为更小的块（如拆分为8份，填充24个块，图6(b)）。
   • 数据指针操作：仅调整指针（如 P_j 和 $P_j + \rho \times d/2$），运行时开销极低。？
2. 动态决策：
   • Kernel Controller 根据SM数量和查询计划动态决定拆分数量，无需离线干预。

优化点3：batch重排序

传输和计算的公式和图例

$$\Upsilon(O[1:i]) = max(\Upsilon(O[1:i-1]) + E(B_{o_i}), \sum_{k=1}^{i} T(B_{o_k}) + E(B_{o_i})). \quad (1)$$

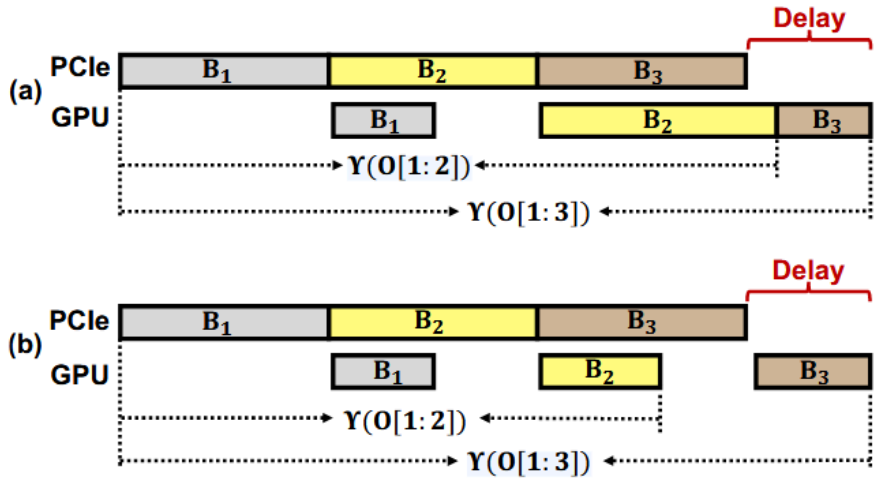


Figure 7: Two cases of the recurrence formula in formulation.

两个改善思路：

1. 不需要传输的cluster前移（为什么会有不需要传输的查询）
2. 计算量很大的cluster前移
   图例：

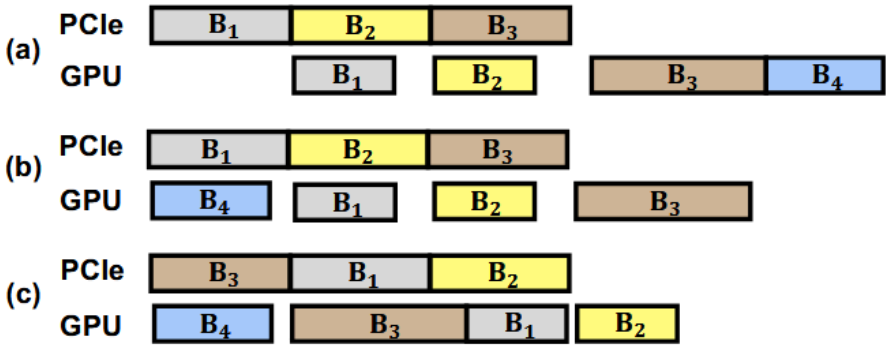


Figure 8: Examples for the reordering algorithm.

最优贪心算法（最优性被数学证明）

Algorithm 1 Optimal greedy reordering algorithm

```
function FINDOPTORDER({B_1, ..., B_m})
    opt_order ← ∅
    for i = 1 → m do
        if T[B_i] == 0 then
            B_i.priority ← +∞
        else
            B_i.priority ← E[B_i]
        opt_order.append(B_i)
    Sort opt_order in descending order based on B_i.priority
    return opt_order
```

寻找最优查询分组：DP+启发式剪枝
剪枝：计算一个节点的子树的下界，如果下界超过了最优时间，就剪枝

GPU页面置换策略

GPU内存管理
rummy将GPU内存视为连续的地址空间

1. 以页为粒度分配：页是分配内存的最小单位，允许传输任务的集群存储在不连续的空间，使得任何空闲页都能被分配，减小了外部碎片
2. 页大小优化：cluster balancing固定了cluster大小，将页大小设为集群尺寸，确保了每页被完全应用，彻底消除了内部碎片

GPU页面置换

1. intra-batch：固定当前批次、未来需使用、已驻留CPU的集群
2. inter-batch：记录每个集群的引用计数，置换未被固定且计数最小的集群

页锁定内存Pinned Memory
通过NVIDIA GPU的cudaMallocHost分配固定内存存储cluster，GPU可直接访问该内存区域。

Scaling GPU-Accelerated Databases beyond GPU Memory Size

传统数据库的GPU优化
observation:

(Section 2): 1) in modern column-store database engines, scan operators on the CPU, despite processing highly compressed data, can achieve throughput close to main memory bandwidth, e.g., 80GB/s on an A100 GPU VM, which far exceeds the PCIe bandwidth; 2) in contrast, join operators on the CPU typically run significantly slower than the PCIe speed, making it beneficial to offload their execution to the GPU, despite the cost of data movement; 3) scans serve as data-reduction operators, often filtering out a large fraction of rows, thereby reducing data processed by downstream operators.

1. 做在CPU上提前filter掉较多的数据，减少数据传输量
2. CPU用来做读写密集的scan操作
3. GPU用来做计算密集的JOIN操作

SPANN: Highly-efficient Billion-scale Approximate Nearest Neighbor Search

处于聚类边缘的向量可以处于多个聚类中