A strategy to improve cache performance in vector query

Cache-aware design in Milvus

董若扬

2024-10-11

ADSLAB, USTC

Outline

Introduction

Faiss Solution

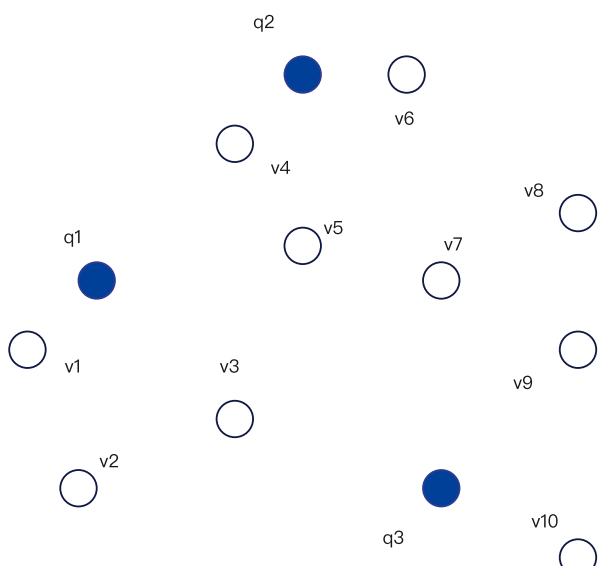
Milvus Solution

Problem

给定一个包含 m 个查询 $\{q_1, q_2, ..., q_m\}$ 的集合和一个包含 n 个数据向量 $\{v_1, v_2, ..., v_n\}$ 的集合，如何快速为每个查询 q_i 找到其最相似的前 k 个向量？

最相似的向量

可以通过欧几里得距离或者余弦角来量化相似程度。

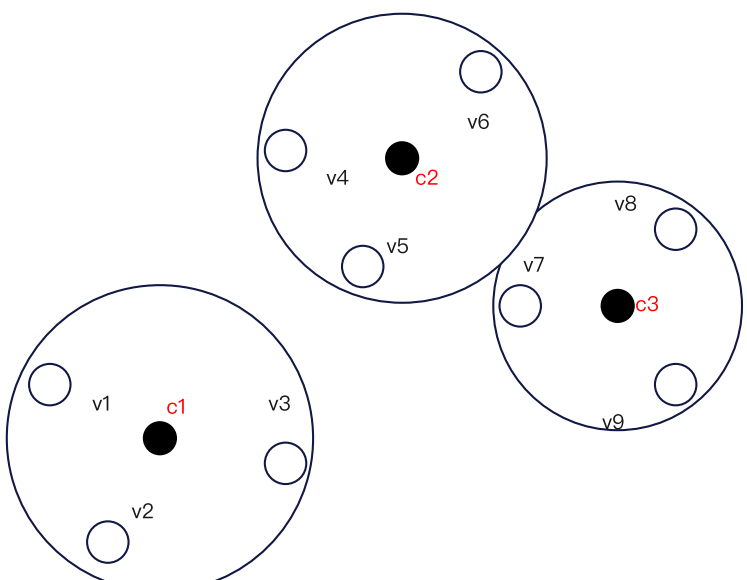


Before Search

在开始搜索之前，还需要对数据库中的向量创建索引以加快搜索过程。创建索引的方法有基于量化的方法和基于图的方法。相较于基于图的索引，在取得较好结果的情况下，基于量化的索引占用更少的内存，并且运行速度更快。

索引创建过程

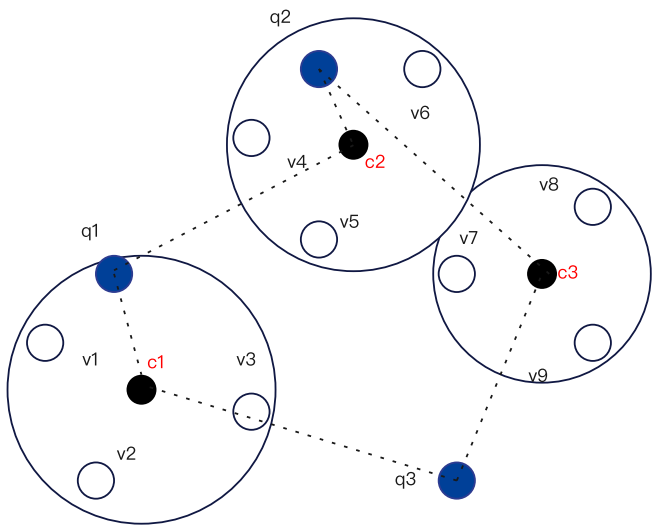
使用 K-means 聚类算法构建码本（其中每个码字是一个质心），应用量化器将向量映射到码字上，码字是距离向量最近的质心。



Filter

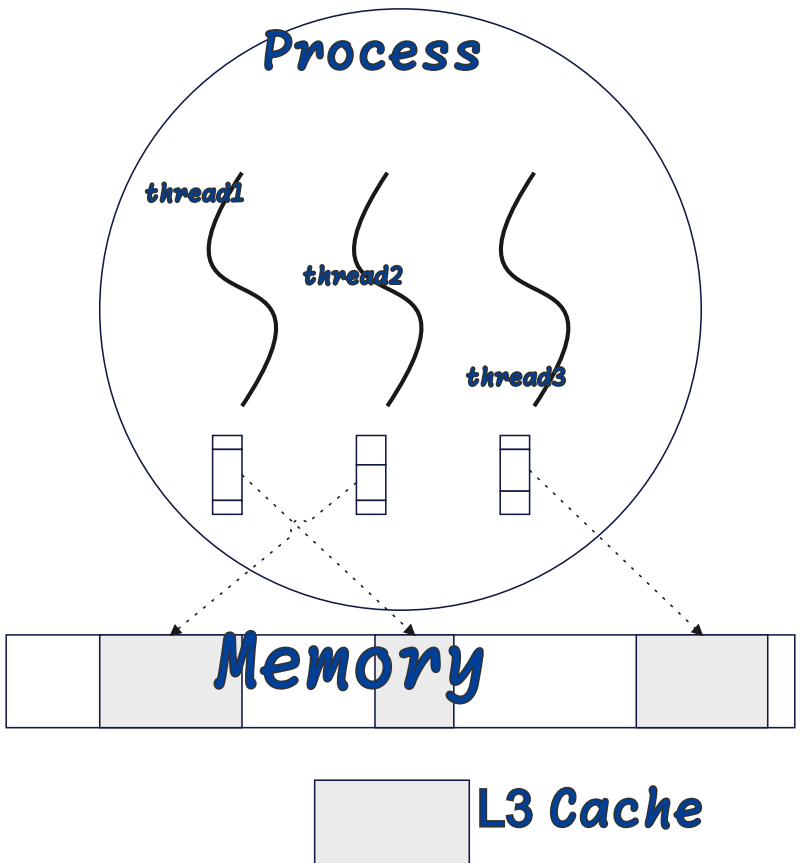
根据查询 q 和每个桶质心 c 之间的距离，找到最近的 n_{probe} 个桶。例如，$n_{\text{probe}} = 2$，则每个查询会在所有聚类中找到质心离自己最近的两个。

参数 n_{probe} 控制准确性和性能之间的权衡。较大的 n_{probe} 值可以产生更高的准确性，但性能可能会下降。



Thread Model

查询进程 P 中有多个线程 T_i 并行执行，每个线程 T_i 都有自己的堆栈，这些堆栈中的内容存放在内存中，并受操作系统的调度放进 Cache 里。



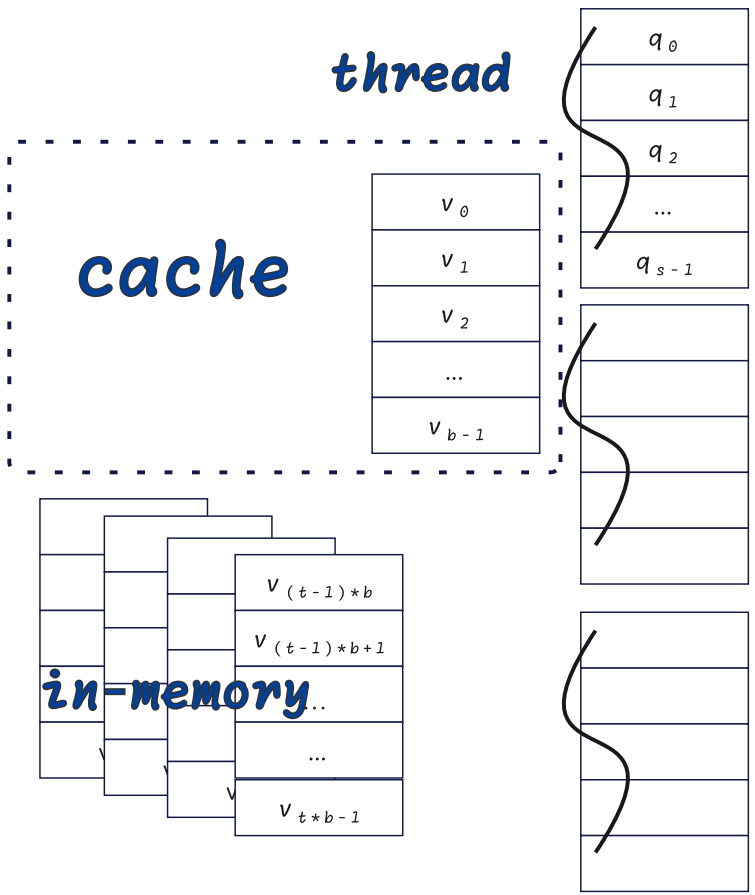
Outline

Solution

Faiss 使用 OpenMP 多线程并行处理查询。

Faiss 将查询向量 q_i 放进线程中，将待比较的数据向量 v_i 放进 cache

当 q_i 完成当前 cache 数据查询后，切换下一页数据向量 v 进入 cache



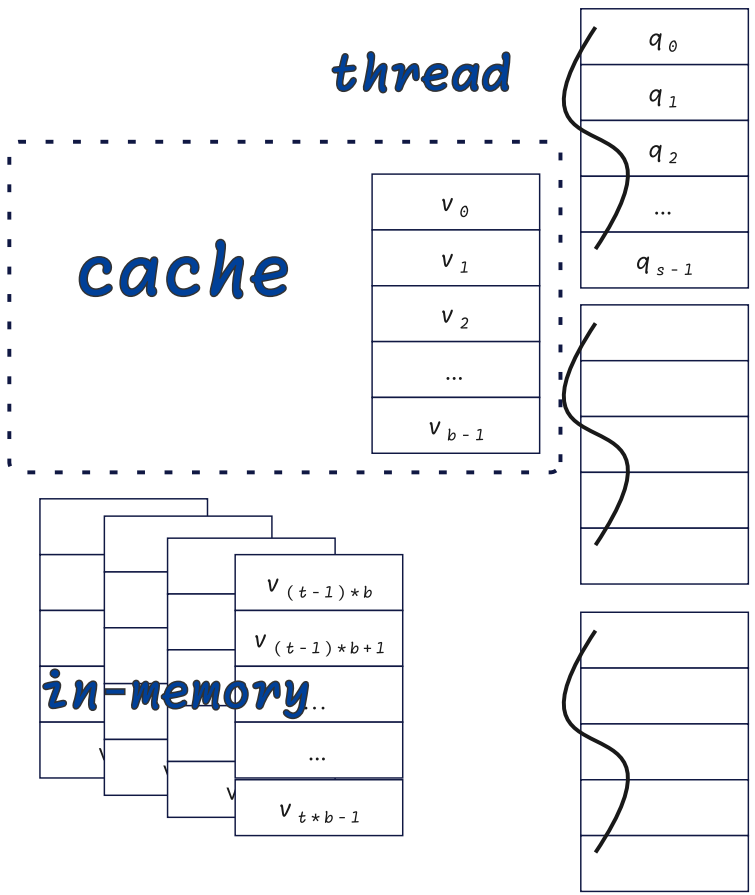
Cost

m 个查询分布在 t 个线程中，每个线程分配 $\frac{m}{t}$ 个查询

每个线程每次只能进行一个查询，所有线程可以并行查询。

对于线程的每一次查询，需要访问一遍完整的数据向量。

所以每个线程都要访问 $\frac{m}{t}$ 倍的完整数据集。



Limitation

Incurs many cache misses

对于每个查询，整个数据集需要流式地放入 cache 中，无法用于下一个查询。

Cannot fully leverage multi-core parallelism

查询向量 m 批次较小，分配的线程数量有限。

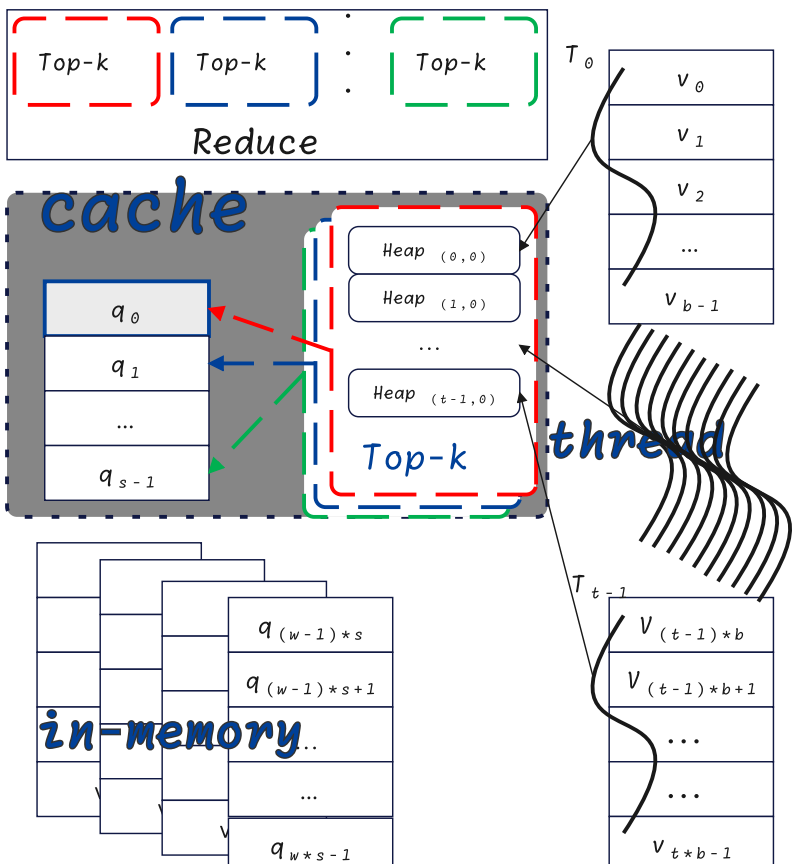
Outline

Solution

Milvus 将数据向量 v_i 放入线程中，将查询向量 q_i 放进 cache

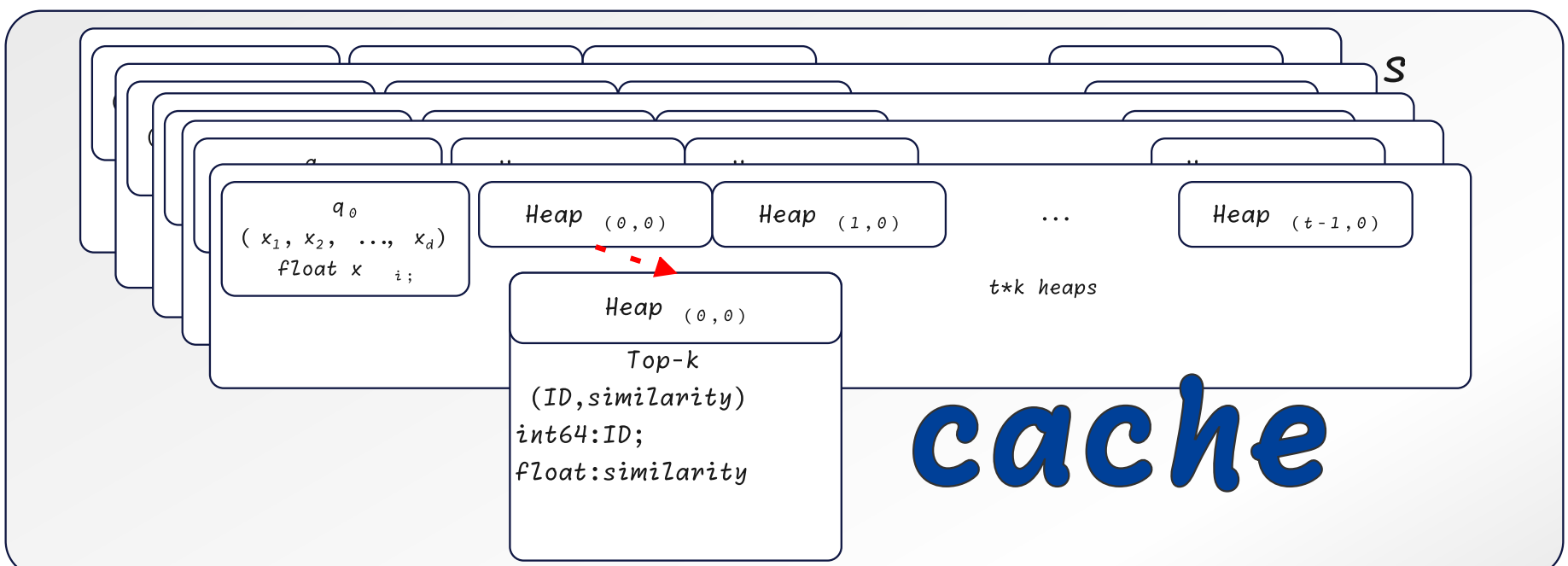
Milvus 使用多线程同时计算每个查询块的 top-k 结果。每个线程将其分配的数据向量与缓存中的整个查询块（含有 s 个查询）进行比较。

为了最小化同步开销，Milvus 为每个线程分配一个查询堆。查询 q_i 的结果分布在 t 个线程的堆中。最后，需要合并各线程的堆来得到最终的 top-k 结果。



Calculat s

$$s = \frac{\text{L3's cache size}}{d \times \text{sizeof(float)} + t \times k \times (\text{sizeof(int64)} + \text{sizeof(float)})}$$



Cost

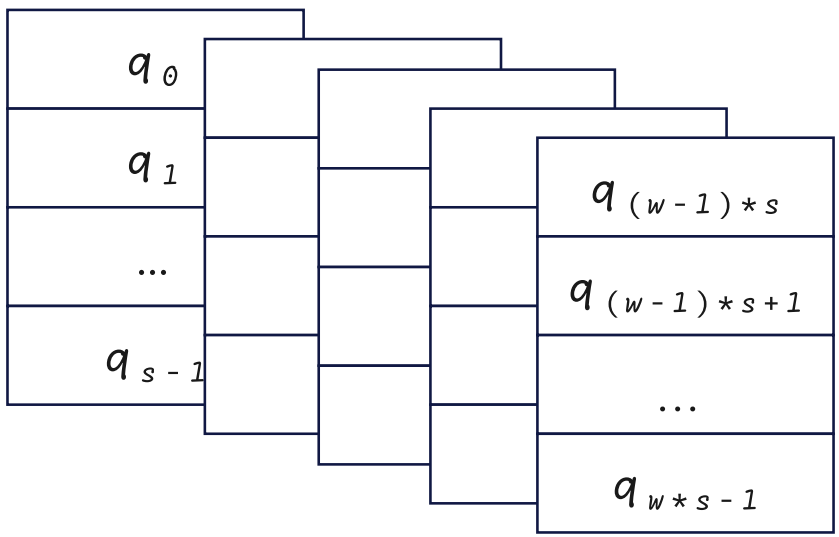


Figure 9: m 个查询，每 s 个查询划分为一个查询块，共 $\frac{m}{s}$ 个查询块。

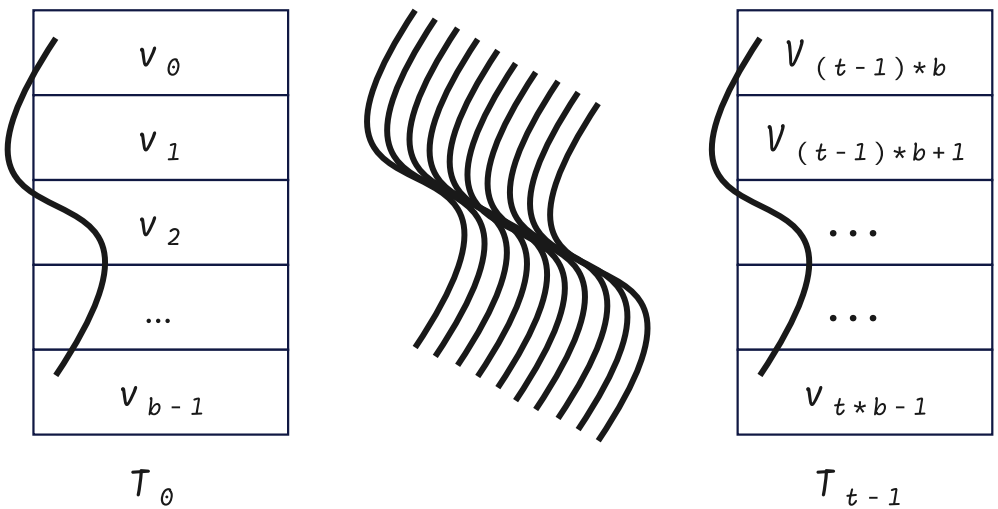


Figure 10: 每个线程分到完整数据集的 $\frac{1}{t}$

从每个查询块的视角看，t 个线程是并行访问查询块的，所以每个线程访问 $\frac{m}{s} \times \frac{1}{t} = \frac{m}{s \times t}$ 的完整数据集。

Comparison

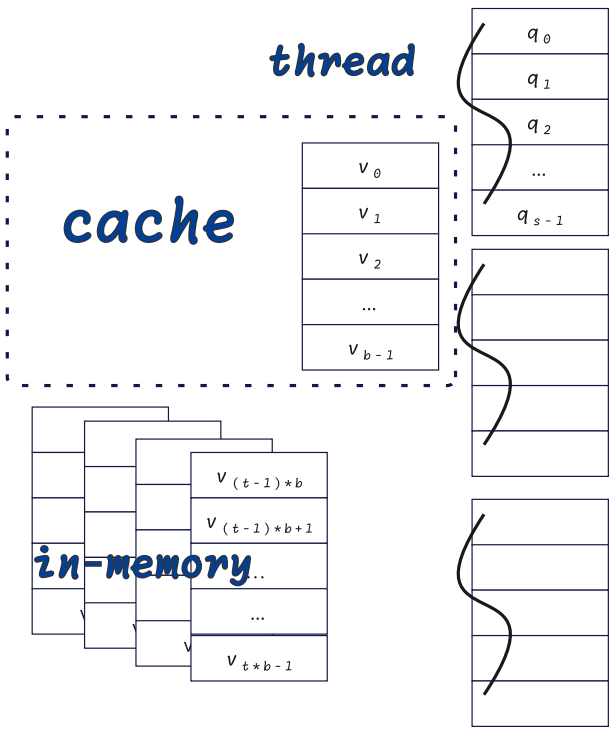


Figure 11: each thread accesses $\frac{m}{t}$ times of the entire data

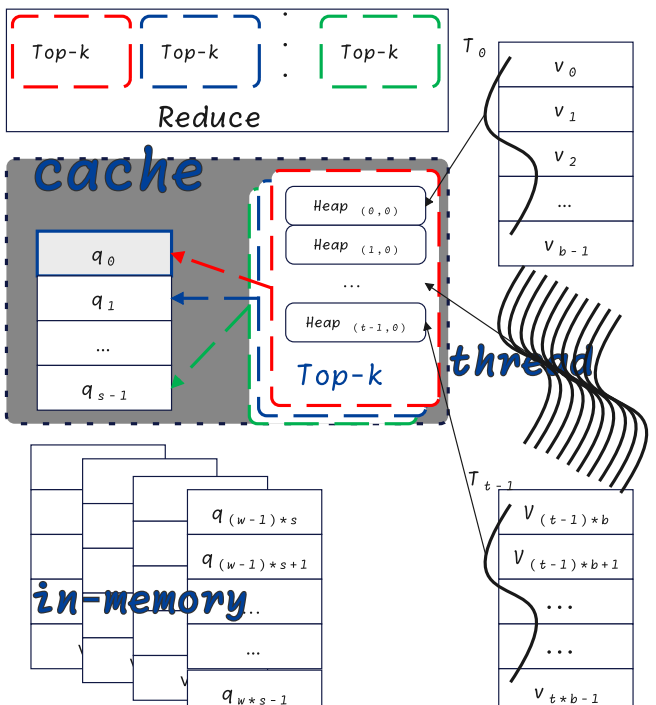


Figure 12: each thread accesses $\frac{m}{s \times t}$ times of the entire data