AntKV：蚂蚁实时计算 KV 分离5x性能提升实践

蚂蚁集团 高级开发工程师 刘达

关于我

刘达

- 毕业于华中科技大学
- IBM CSL Storage，SVC/Storwize
- 蚂蚁计算智能部，目前主要负责 AntKV 相关研发

大纲

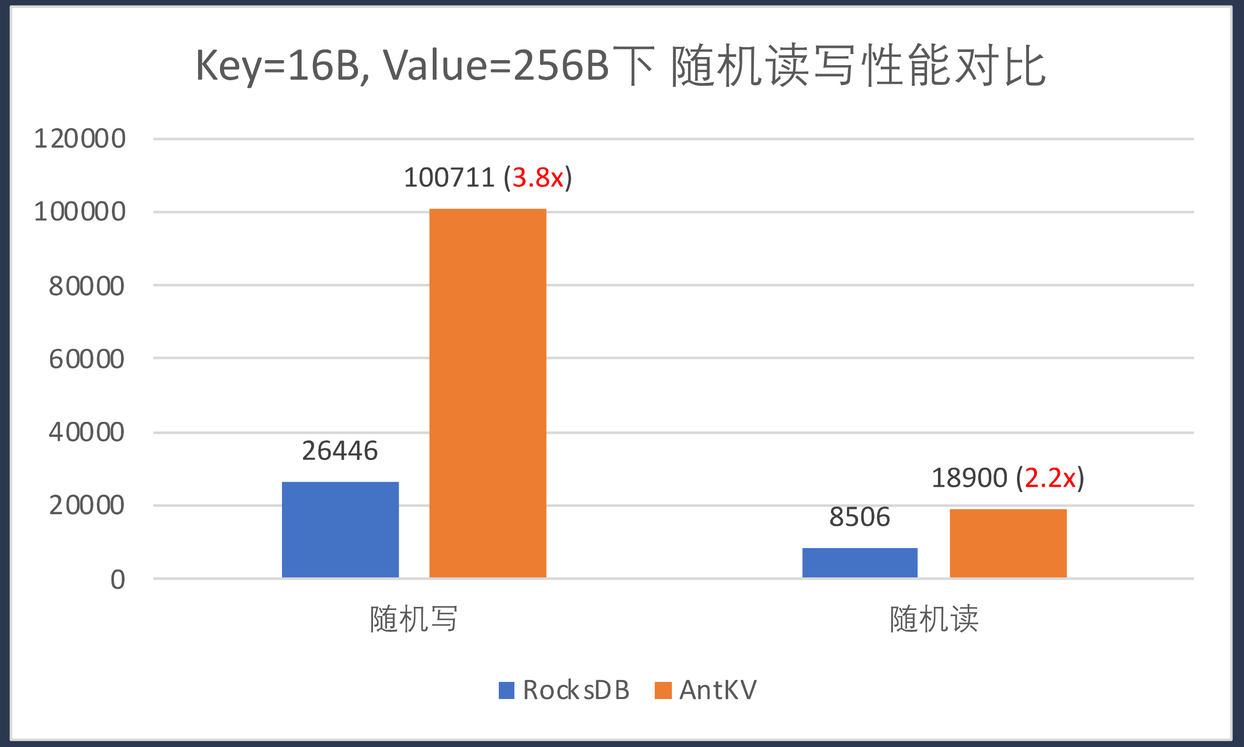
- AntKV 概述
- 针对 KV 分离的 Scan 优化探索
- 借助 Learned Index 优化查询
- 总结

AntKV 概述

什么是 AntKV?

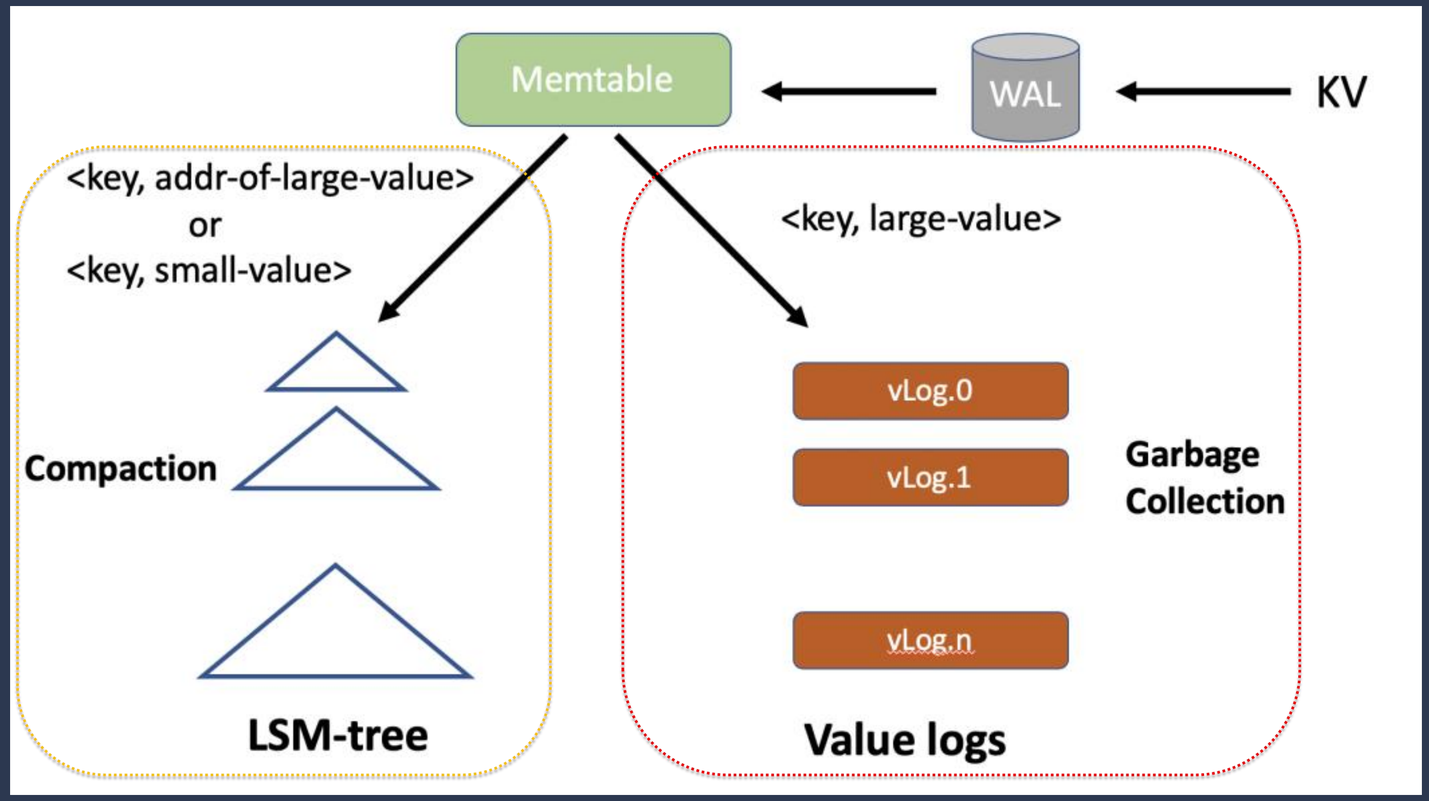
一款基于 RocksDB 的高性能 KV 存储引擎，实现了 WiscKey 提出的 KV 分离设计

- 解决了大 Value 场景下 Compaction 带来的毛刺问题
- 具有优秀的点读和写入性能



目前 AntKV 在蚂蚁支撑了流计算/在线学习平台等业务

AntKV 整体架构


Memtable
WAL
KV
<key, addr-of-large-value>
or
<key, small-value>
Compaction
LSM-tree
<key, large-value>
vLog.0
vLog.1
vLog.n
Garbage
Collection
Value logs

AntKV 核心功能

KV分离
- 元数据管理

空间回收
- GC
- TTL

数据版本
- Checkpoint
- Ingest Value Log Files

特性支持
- 异步恢复Checkpoint
- Table API

性能优化
- Scan 优化
- 流控优化
- Learned Index

AntKV 最佳实践

业务	痛点	接入后性能	最终结果
AntFlink	大状态双流 Join 作业，无法实时处理	提高2–5倍	解决了大状态双流 Join 不可用的问题
Ray	大 Job 写入性能不满足，后台 Compaction 占用过高	提高2–4倍	改善了写入瓶颈，支撑了无法支撑的业务
SofaMQ	小型化站点形态，RocksDB 性能不满足	满足了性能要求	小型化输出目标实现，节约50%+成本

针对 KV 分离的 Scan 优化探索

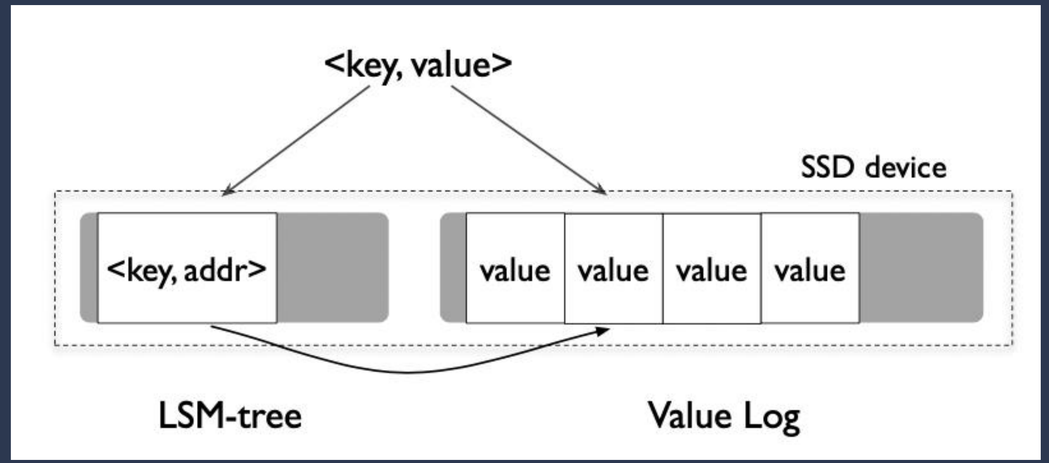
Scan 优化：背景分析

KV 分离的现状

- Value 物理上不连续
- 访问 Value 多一跳

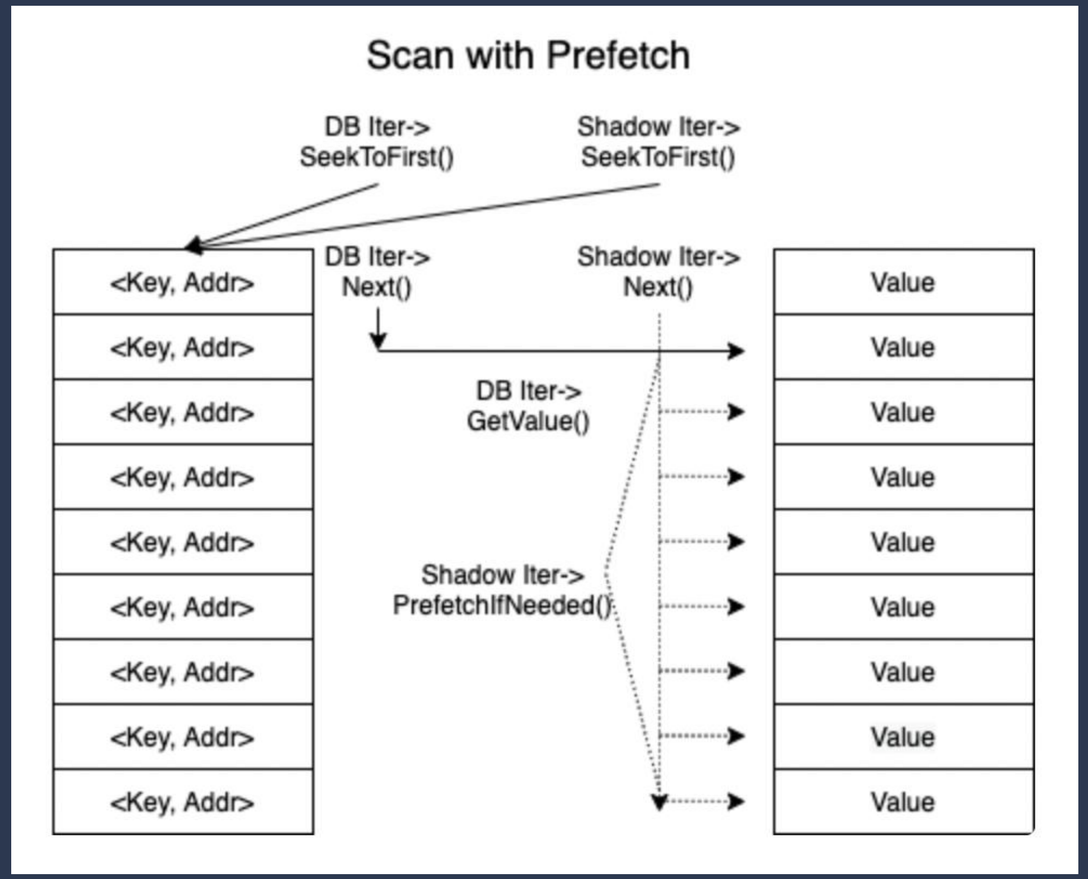
带来的问题

- Scan 性能下降



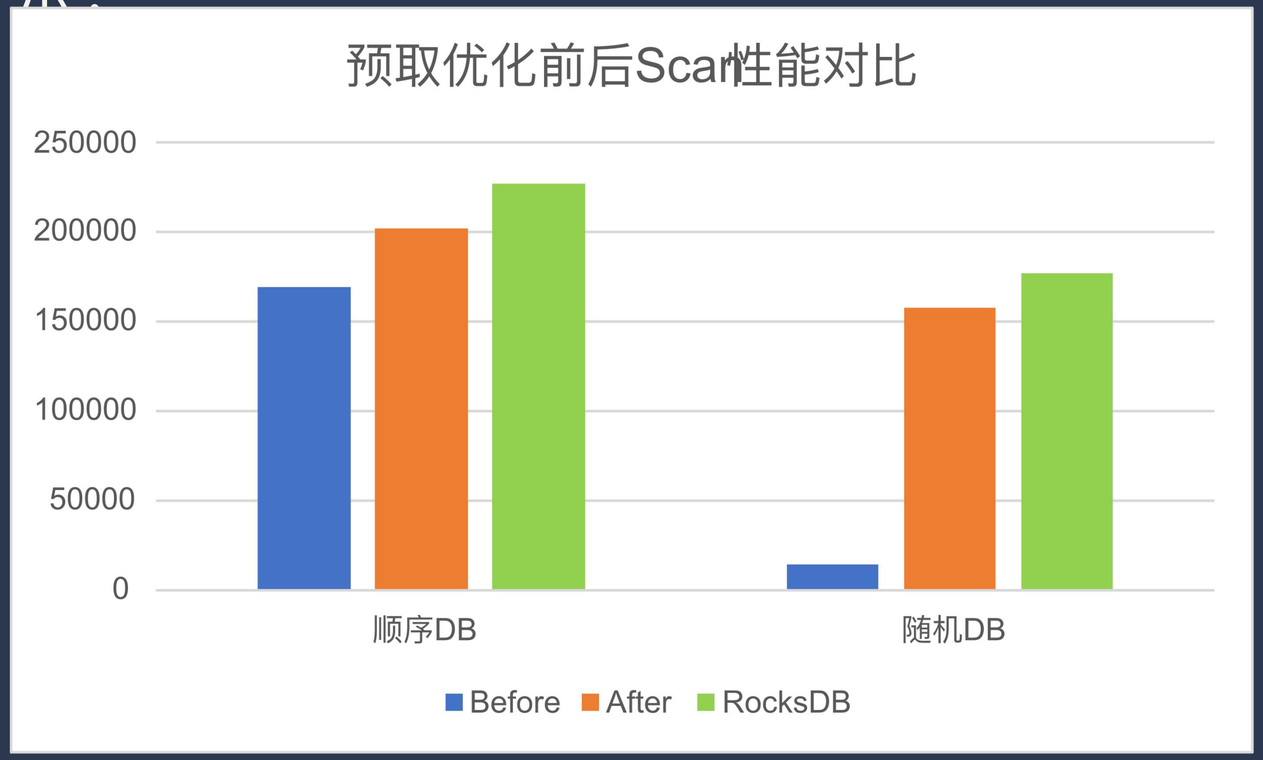
Scan 优化：多线程预取加速

- 根据用户访问 Pattern 或者 Range Hint 发起异步预取
- 发挥 NVMe SSD 能力并行预取
- 利用 Block Cache 实现数据同步



Scan 优化：预取加速效果

在 Key Size=16B, Value Size=1KB 的条件下，Scan 性能 (IOPS) 如图所示：



在两种数据分布下，Scan 性能分别有20%/10倍左右的提升

Scan 优化：分离数据的连续性问题

新的问题

中等大小（如256B）Value 情况下，Scan 仍然比 RocksDB 差很多

原因

Block 中数据不连续，磁盘带宽即便打满，大多内容都是无效数据

8 Scan 优化：基于DiffKV的数据分布优化

ATC 21: Differentiated Key-Value Storage Management for Balanced I/O Performance

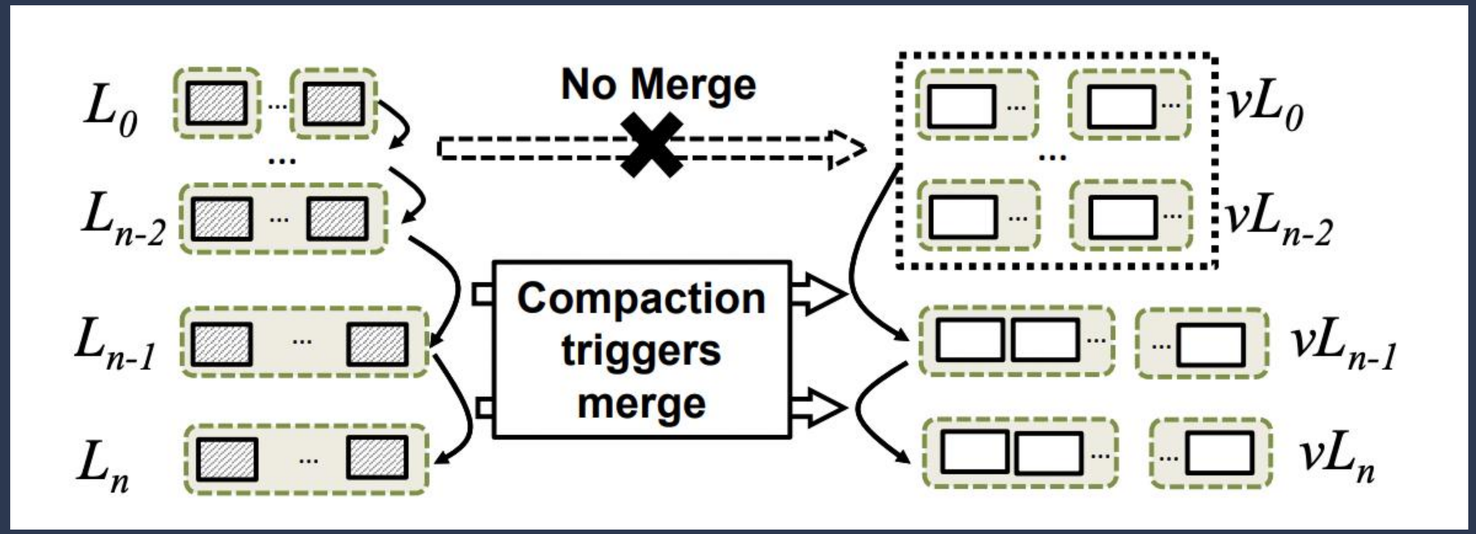
核心思路

对于中等大小的 KV pairs，对 Value Log Files也进行分层处理，增强局部连续性

Scan 优化：基于DiffKV的数据分布优化

基于 Compaction 的重写：

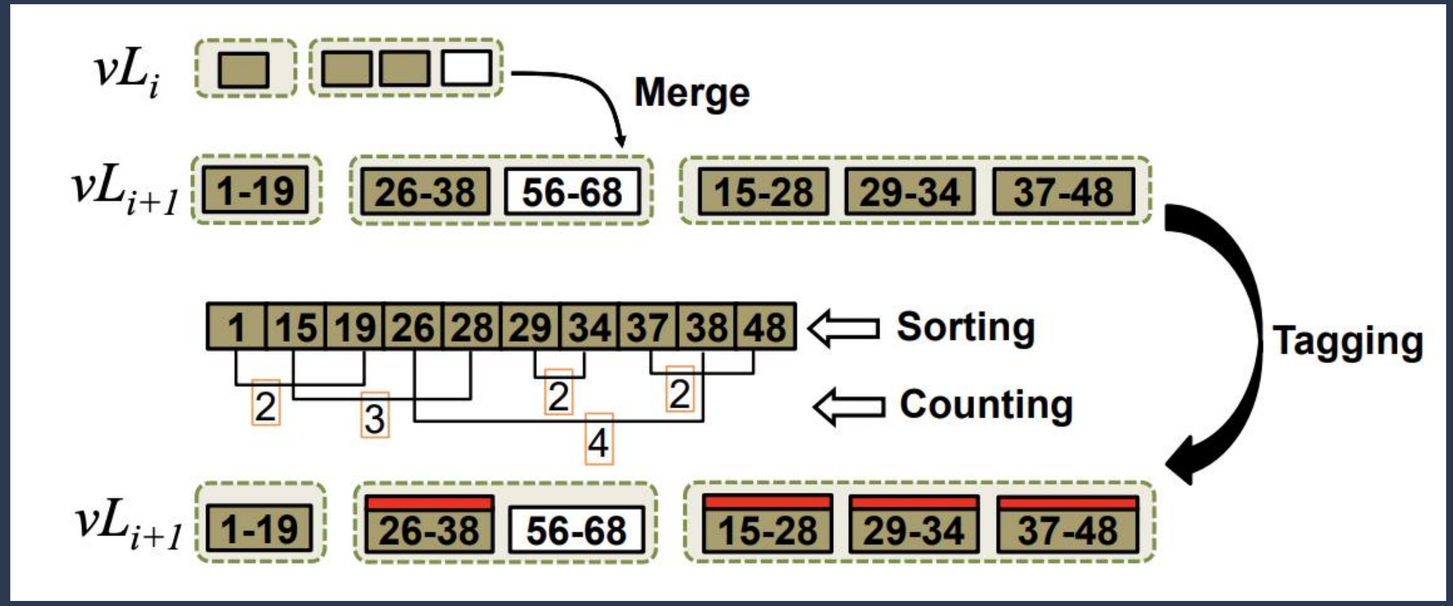
- Level N–2 及以下的层级 不做重写
- Level N–1 及以上的层级 在 Compaction 时重写 Value Log Files



Scan 优化：基于DiffKV的数据分布优化

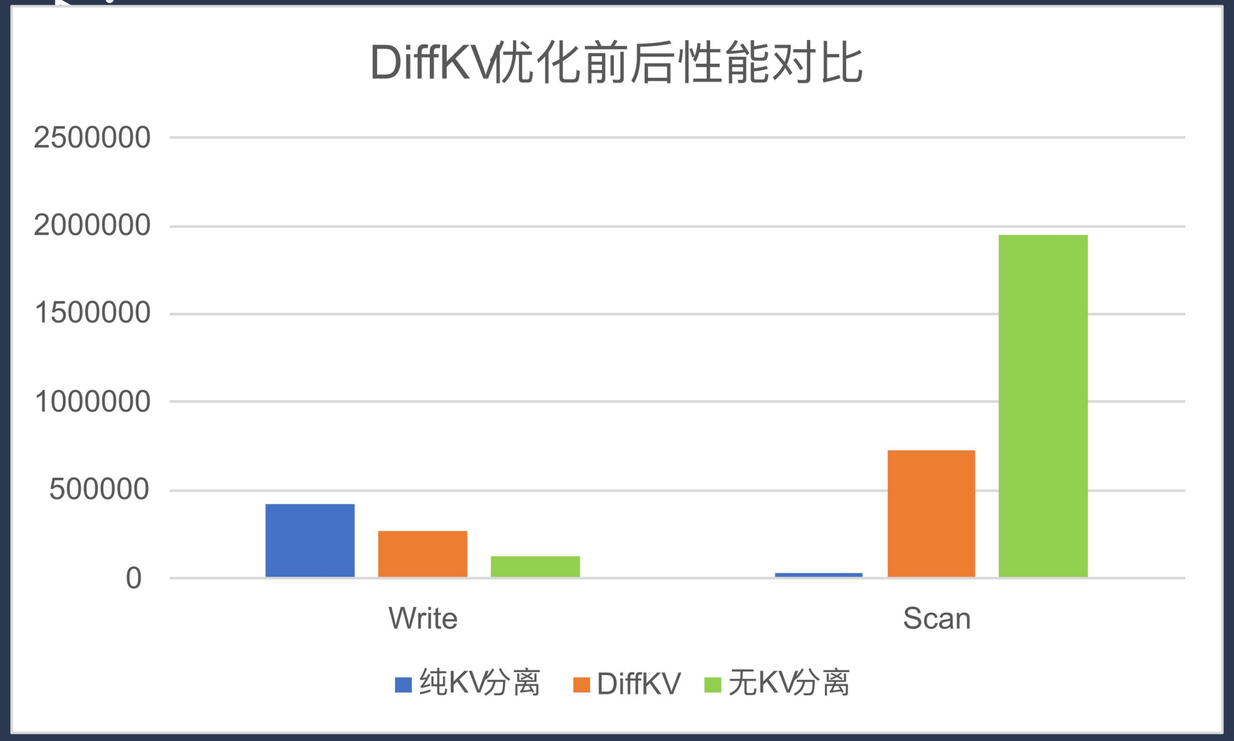
针对 Scan 优化的重写：

- Compaction 过程中，对本轮参与的 Value Log Files 进行重叠记数
- 当发现某文件重叠记数超过阈值，则标记相关文件后续进行重写



Scan 优化：DiffKV 效果

在 Key Size=16B, Value Size=256B 的条件下，写入和 Scan 性能 (IOPS) 如下：



相较于纯KV分离的版本，DiffKV 以35%左右写性能的代价，大幅度提高了 Scan 的性能。

为用户提供了一种可以配置的性能模型，在写入和 Scan 性能之间进行 Tradeoff。

借助 Learned Index 优化查询

Learned Index: 背景

传统的索引结构

未利用数据分布的特点

Learned Index

通过学习数据分布，训练基于 key 的 index 模型

- O(logn)–>O(1)
- 索引大小减少

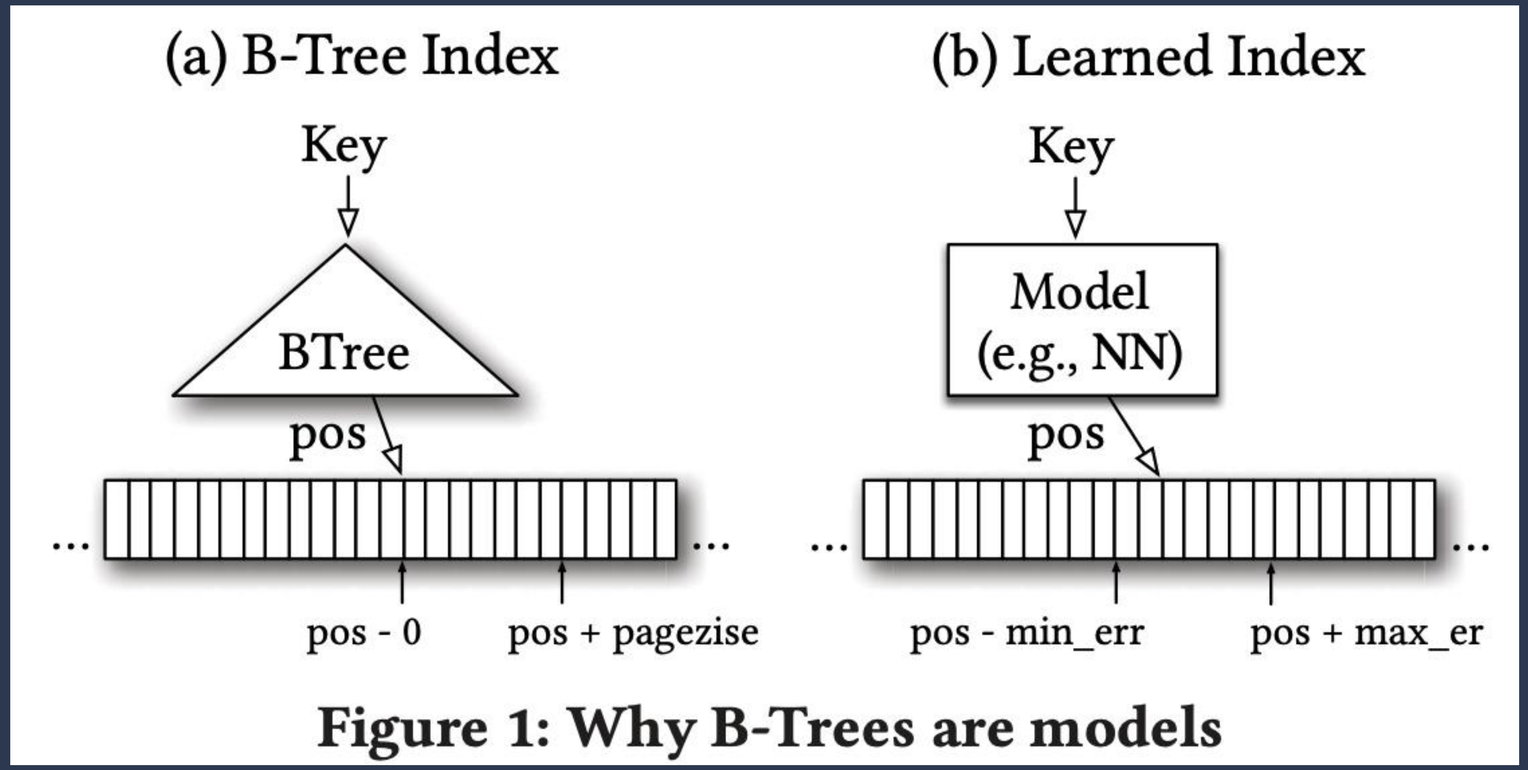
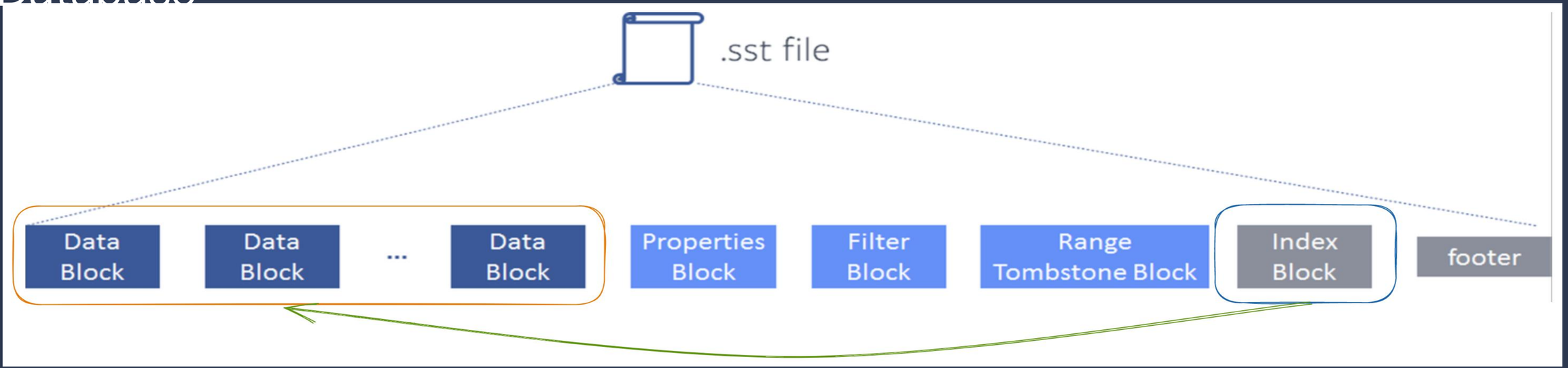


Figure 1: Why B-Trees are models

Learned Index: 背景

NeurIPS 2020: Learned Indexes for a Google-scale Disk-based Database

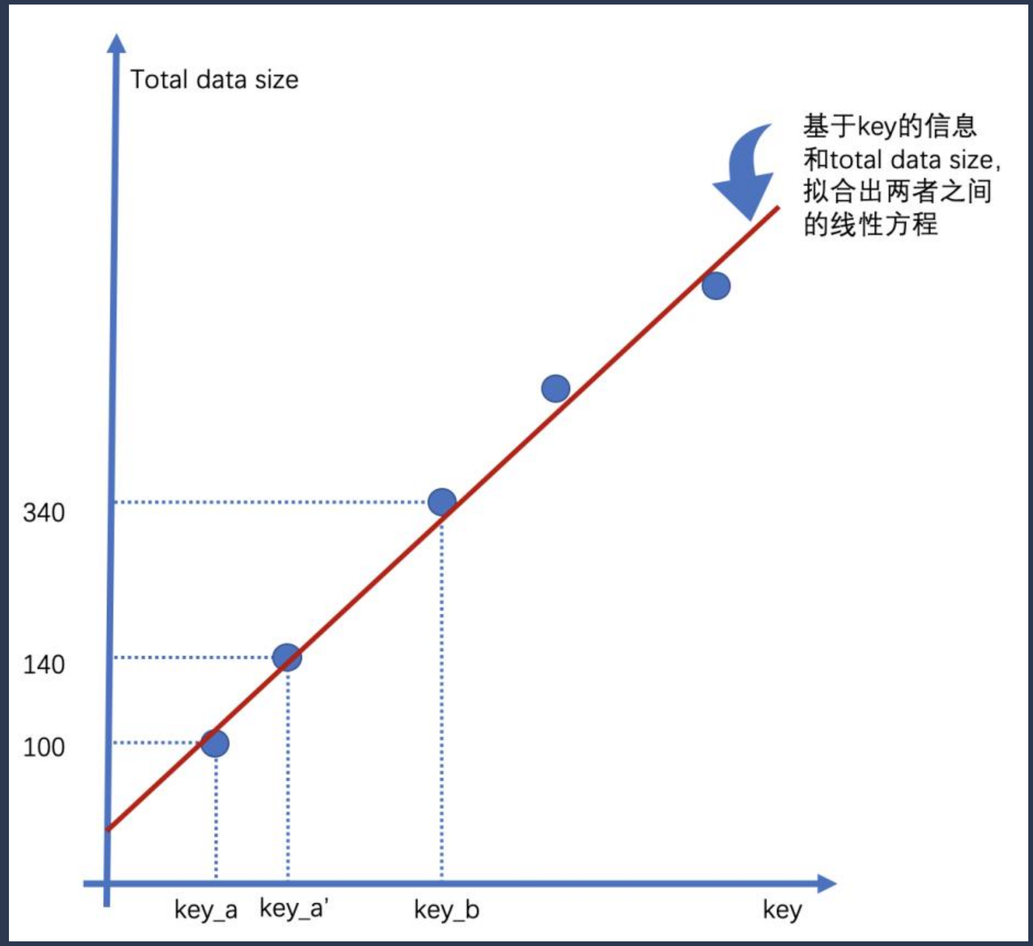


F(key) -> ~~entry_pos~~ block_pos

Learned Index: 模型

$$total\_data\_size = F(key) = a * key + b$$

注意：这个线性模型求出的是近似的current data size

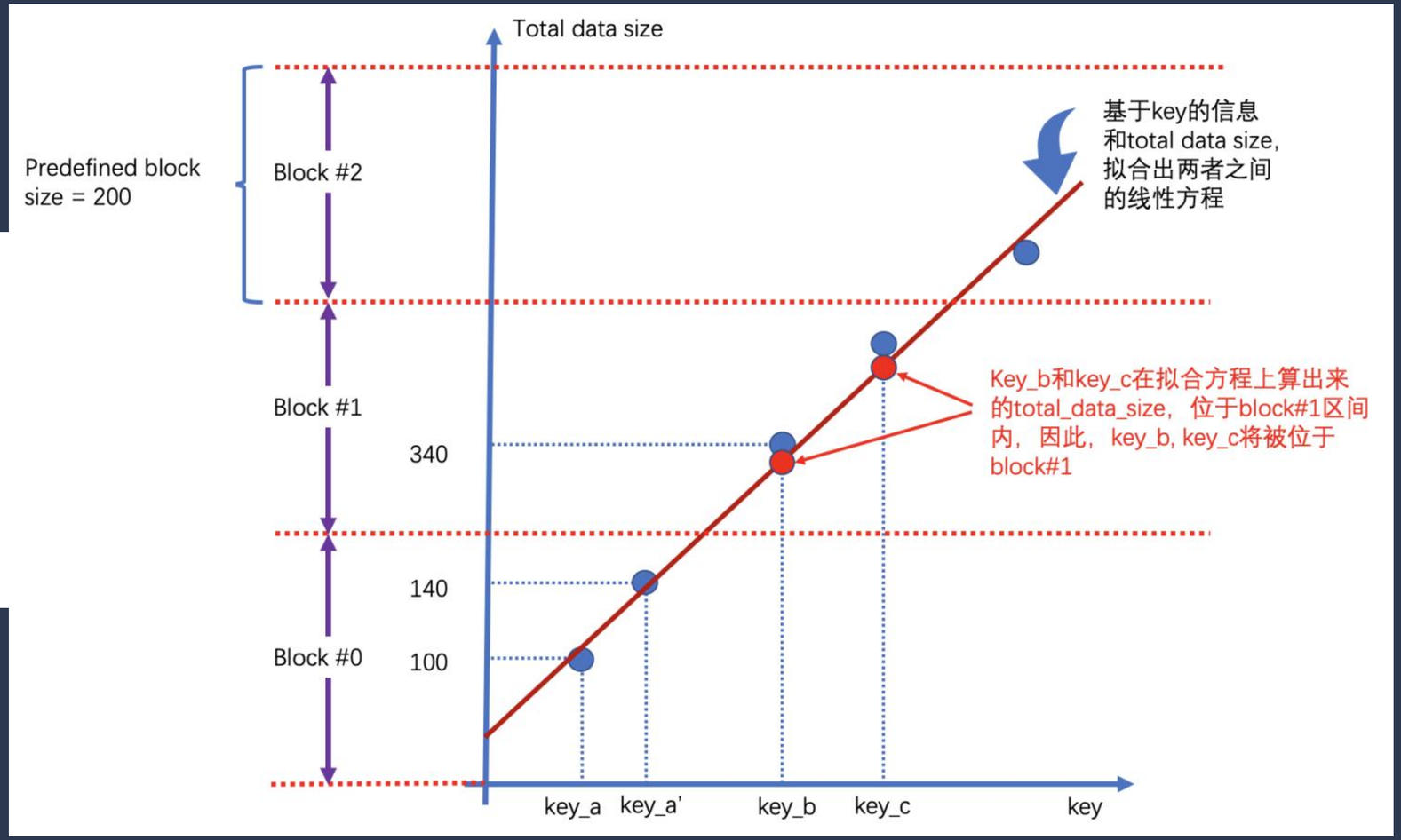


Learned Index: 模型

$$total\_data\_size = a * key + b$$

当我们通过拟合方程算出key对应的total_data_size时，基于下面的公式，就能计算出对应的该key位于哪个block中

$$block\_no(key) = \frac{total\_data\_size(key)}{predefined\_block\_size}$$



Learned Index: 字符串转换

因为实际 SST 保存的 key 为 string 类型，非 integer，因此需要进行转换

要求

唯一性：不同的 key，转换出来的 key_digest 不能相同
保序性：如果 key1 < key2，那么转换后的 key_digest_1 < key_digest_2

问题

字符串长度是随机的，并且可能很长

Learned Index: 字符串转换

思路

按位计算进制，减少转换后的数值大小。举例：

"ABX", "BCZ", "BDY"

第0位: 'A'~'B', 5种编码
第1位: 'B'~'D', 6种编码
第2位: 'X'~'Z', 6种编码

编码 / 值 / 位数	0	1	2	3	4	5
d[0]	空值	小于min	'A'	'B'	大于max	N/A
d[1]	空值	小于min	'B'	'C'	'D'	大于max
d[2]	空值	小于min	'X'	'Y'	'Z'	大于max

"BCZ" –> d[0]['B'] * 6 * 6 + d[1]['C'] * 6 + d[2]['Z'] = 130
"XBA" –> d[0][大于max] * 6 * 6 + d[1]['B'] * 6 + d[2][小于min] = 157

Learned Index: 生成过程

在构建新的SST过程中，会缓存待写入的所有KV数据，在Finish时进行建模并持久化相关参数。

- 不会在L0构建Learned Index
- 不会对大小在阈值以下的SST进行构建
- 当不满足构建条件时，退化为默认的Binary Index

Learned Index: 效果对比

假设一个64MB SST，按压缩前128MB计算，Key=100B, Value=100B,
按照4K Block Size计算:

Binary Index Block： 3.7MB
Learned Index Block: 512KB ↓86%

	P50 latency(us)	P95 latency(us)	Index Block Hit ratio
Binary Index	130	10230	85%
Learned Index	109	230	99%

总结

总结

AntKV 作为一款基于 KV 分离的本地存储引擎，在性能方面：

- 解决了大 Value 场景下的随机读写性能问题
- 一定程度弥补了分离引入的 Scan 性能下降问题
- 通过机器学习提高了SST索引的访存效率

欢迎交流 :)

THANKS

软件正在重新定义世界

Software Is Redefining The World