

DTCC

2017第八届中国数据库技术大会

DATABASE TECHNOLOGY CONFERENCE CHINA 2017

使用 TerarkDB 提升 MyRocks 的性能

——随机读场景下的 MySQL 性能优化

雷鹏 peng@terark.com

CTO <http://terark.com>

大纲

- MyRocks 简介
- 为什么选择 MyRocks
- 传统数据库的块压缩
- 硬件的发展趋势
- TerarkDB 和 MyRocks 的关系
- TerarkDB 的原理、优势、劣势
- 新的问题，以及优化建议

MyRocks 简介

- MyRocks = MySQL + RocksDB(存储引擎)
- 基于 MySQL-5.6 : [代码\(MySQL on TerarkDB\)](#)
 - 增加了一些 MySQL-5.7 的 Feature
 - 增加了一些针对 RocksDB 特有的语法和功能
 - 有部分功能不支持, 详见 : [MyRocks-limitations](#)
- 深入了解 MyRocks : [MyRocks Deep Dive](#)

MyRocks 主要功能

- 类似 InnoDB 的功能
- 事务
 - 基于 MVCC
 - Read Committed, Repeatable Read (PostgreSQL-style)
- 在线备份
 - 逻辑备份 (使用 mysqldump)
 - 二进制备份 (使用 myrocks_hotbackup)

为什么选择 MyRocks

- 成熟稳定，满足绝大部分功能需求
- 使用单一 DB，适合云 DB 多租户场景
- 基于 RocksDB，直接兼容 TerarkDB
- 基于 LSM，对 SSD 友好

传统数据库都使用块压缩——

- InnoDB
- LevelDB/RocksDB...
- TukoDB
- WiredTiger
- Cassandra/ScyllaDB
- CouchDB
- HBase
-

块压缩有着难以克服的，固有的缺陷，我们看看它的原理，分析一下——

块压缩的原理

- 多条数据打包成一个块，索引定位到块
 - 增大块：索引更小，单个块中的数据条数更多，无效解压更多
 - 减小块：索引更大，压缩算法无法有效工作，压缩率更低

- 块缓存

- 每次访问都解压一个块，**太慢了**
- 解压出来的块放入缓存

问题 1

Direct IO 导致缓存中只有解压后的数据，一定程度上减小了内存利用率

问题 2

Direct IO 需要对齐访问，最坏情况下需要跨边界多读两个块

使用 Direct IO

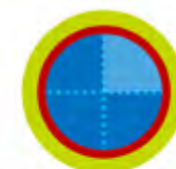
硬件的发展趋势

- 机械硬盘的 容量/价格 仍有巨大优势
 - 更耐写，更安全；氦气硬盘，HAMR 技术.....
- SSD 越来越便宜，NVMe/PCIe 已广泛应用
 - 不耐写，写坏了数据就永久丢失
 - NVMe/PCIe 接口的 SSD 更快，延迟更低
- 3D XPoint，比 SSD 更快，更耐写
 - 第一代产品 Optane(闪腾) 已经上市！
 - 使用 DIMM 接口，可当做内存用.....

Bridging the Memory-Storage Gap

Intel® Optane™ Technology

Performance & Lower costs: Integrated solutions

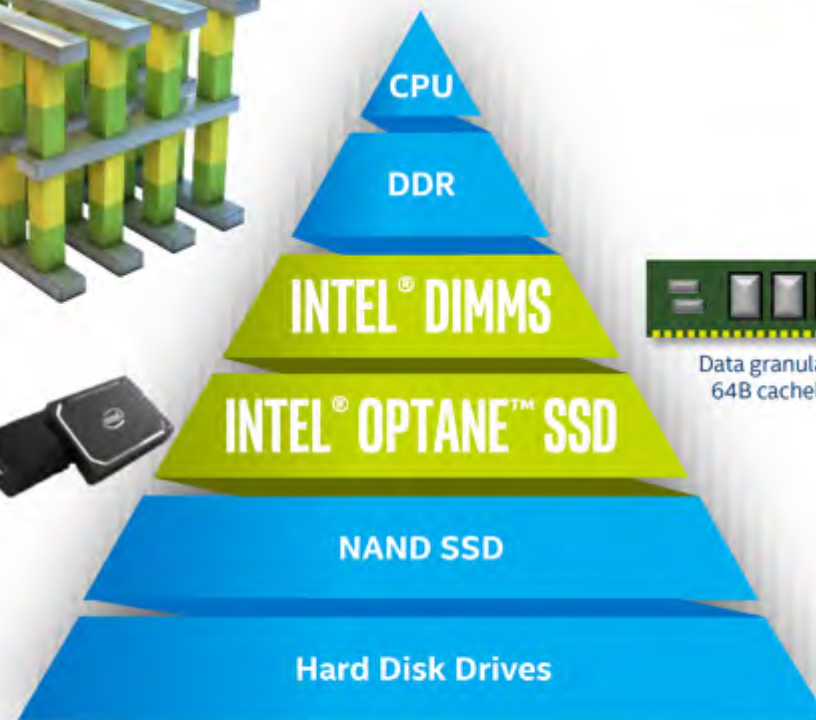
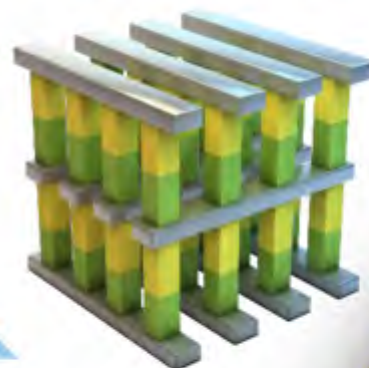


Intel® Scalable System Framework

1000X
FASTER
Than NAND¹

1000X
ENDURANCE
Of NAND²

10X
DENSER
Than DRAM³



WSCAD 2016

DTCC

2017年第八届中国数据库技术大会
DATABASE TECHNOLOGY CONFERENCE CHINA 2017

SequeMedia

KT100

mpub

ChinaUnix

硬件的发展趋势

| | 带宽(MB/s) | IOPS | 延迟(us) | 访问粒度 |
|-----------|----------|-----------|--------|------------|
| HDD | 200 | 200 | 10,000 | 4K/512B |
| SSD | 1,000 | 50,000 | 200 | 4K |
| NVMe | 5,000 | 500,000 | 20 | 4K |
| 3D XPoint | 10,000 | 1,000,000 | 2 | 64B |

更适合 Terark PA-Zip

| Storage Latencies | ns: Nano seconds | us: Micro seconds | If L1 Access is 1 second |
|---------------------------------|-------------------------|--------------------------|--|
| L1 Cache Reference | 0.5 | | 1 secs |
| L2 Cache Reference | 7 | | 14 secs |
| DRAM Access | 200 | | 6 mins, 40 secs |
| Intel Optane 3D Xpoint | 7,000 | 7 | 3 hours, 53 mins, 20 secs |
| Micron 9100 NVMe PCIe SSD Write | 30,000 | 30 | 16 hours, 40 mins, 00 secs |
| Mangstor NX NVMeF Array Write | 30,000 | 30 | 16 hours, 40 mins, 00 secs |
| DSSD D5 NVMeF Array | 100,000 | 100 | 2 days, 07 hours, 33 mins, 20 secs |
| Mangstor NX NVMeF Array Read | 110,000 | 110 | 2 days, 13 hours, 33 mins, 20 secs |
| NVMe PCIe SSD Read | 110,000 | 110 | 2 days, 13 hours, 33 mins, 20 secs |
| Micron 9100 NVMe PCIe SSD Read | 120,000 | 120 | 2 days, 18 hours, 40 mins, 00 secs |
| Disk Seek | 10,000,000 | 10,000 | 7 months, 10 days, 11 hours, 33 mins, 20 secs |
| DAS Disk Access | 100,000,000 | 100,000 | 6 years, 4 months, 00 days, 19 hours, 33 mins, 20 secs |
| SAN Array Access | 200,000,000 | 200,000 | 9 years, 6 months, 02 days, 17 hours, 20 mins, 00 secs |

块压缩的缺点，我想静静.....

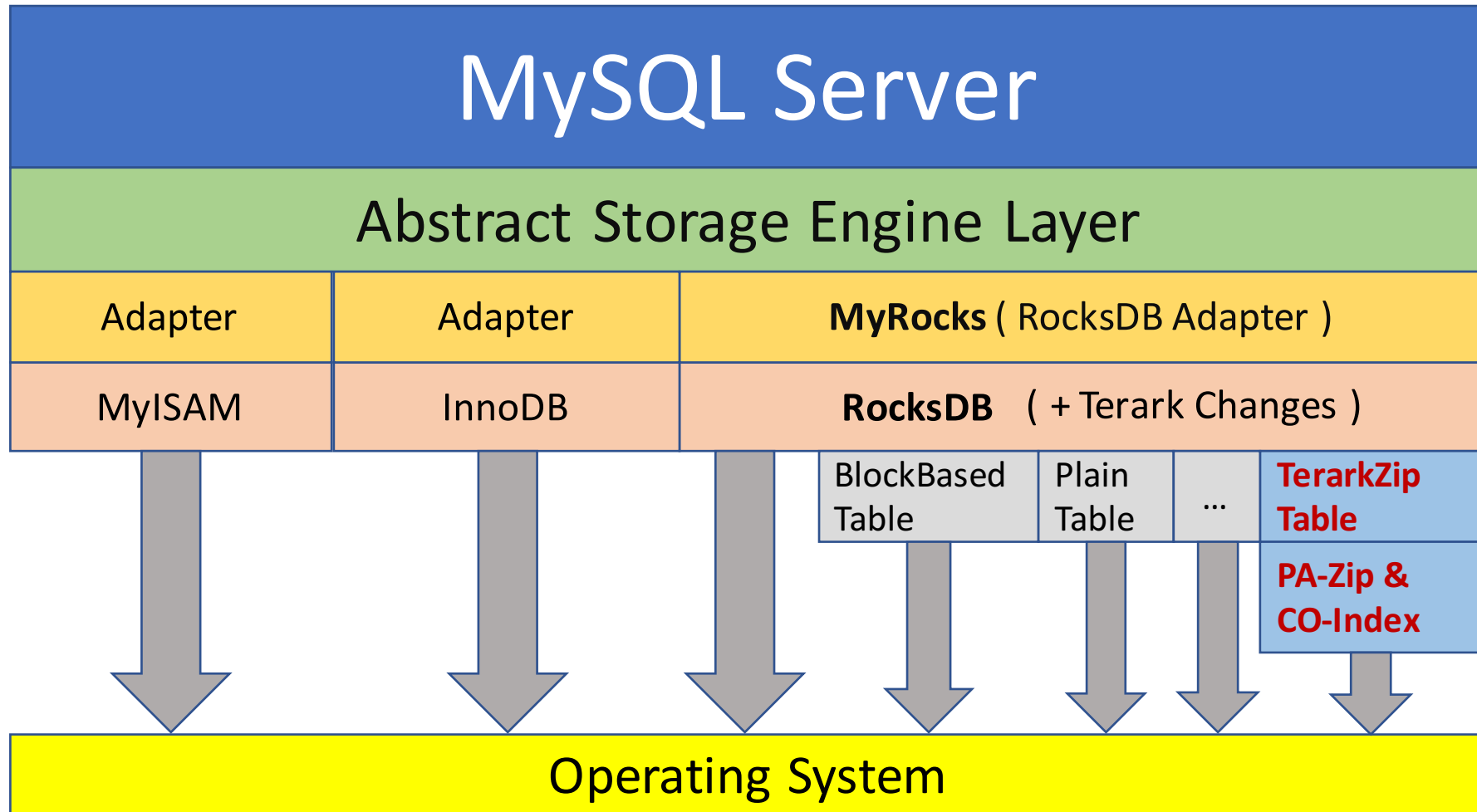
- 压缩率**太低**
- 随机读**太慢**
- 内存用量**太大**
- 与硬件发展趋势**失配**

怎么改进



且听分解.....

TerarkDB 和 MyRocks 的关系



TerarkDB 和 RocksDB 的关系

- TerarkZipTable
 - 使用 Terark 可检索压缩技术 的 SSTable
- Terark Modified RocksDB
 - SSTable 创建过程中支持两遍扫描, 以及其它小修改
 - 二进制兼容原版 RocksDB
- TerarkDB 可使用环境变量进行参数配置

确认 **TerarkZipTable** 是否已启用

```
select * from  
information_schema.ROCKSDB_CF_OPTIONS  
where OPTION_TYPE='TABLE_FACTORY_NAME';
```

| CF_NAME | OPTION_TYPE | VALUE |
|------------|--------------------|-----------------------|
| __system__ | TABLE_FACTORY_NAME | TerarkZipTable |
| default | TABLE_FACTORY_NAME | TerarkZipTable |

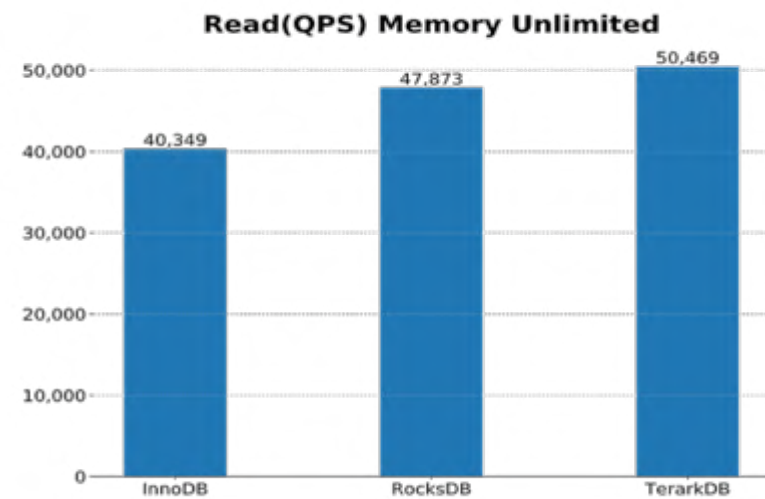
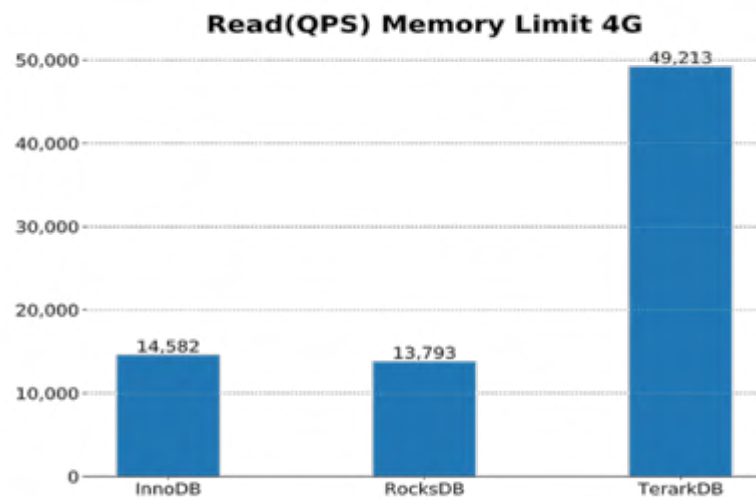
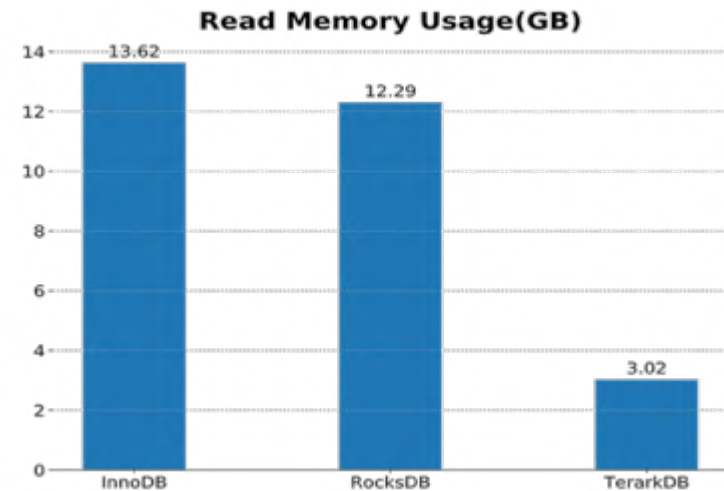
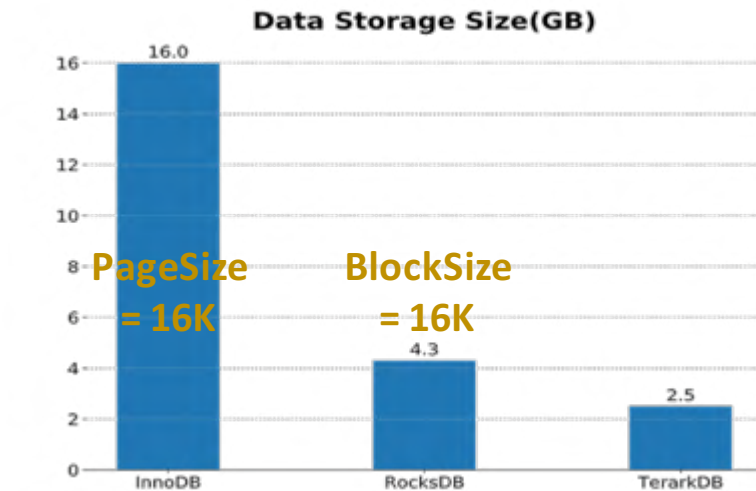
使用 optimize table 执行 compact

optimize table TableName;

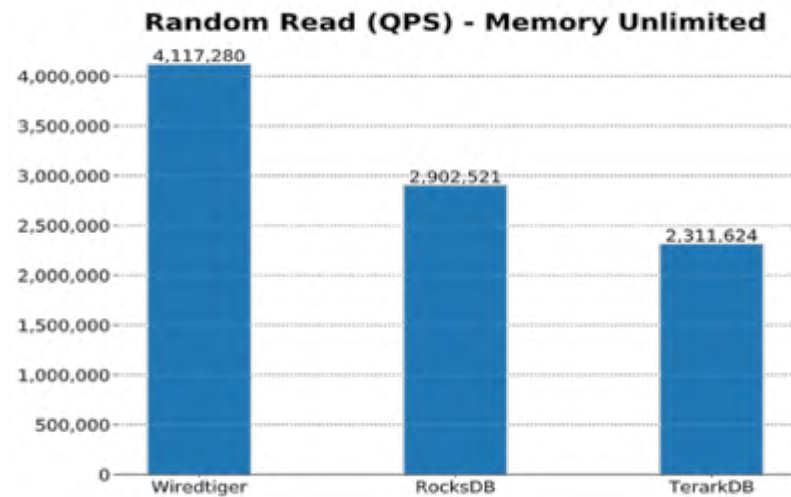
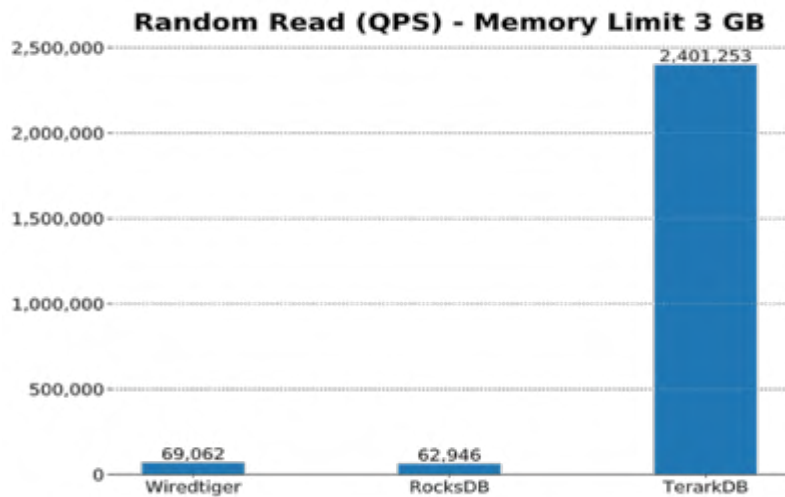
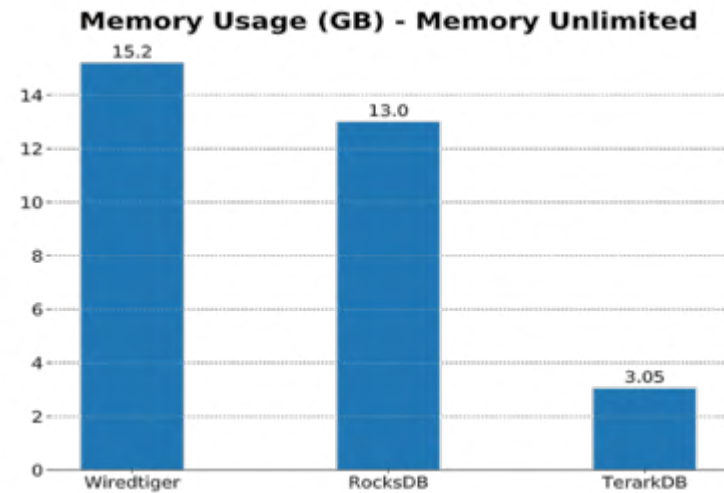
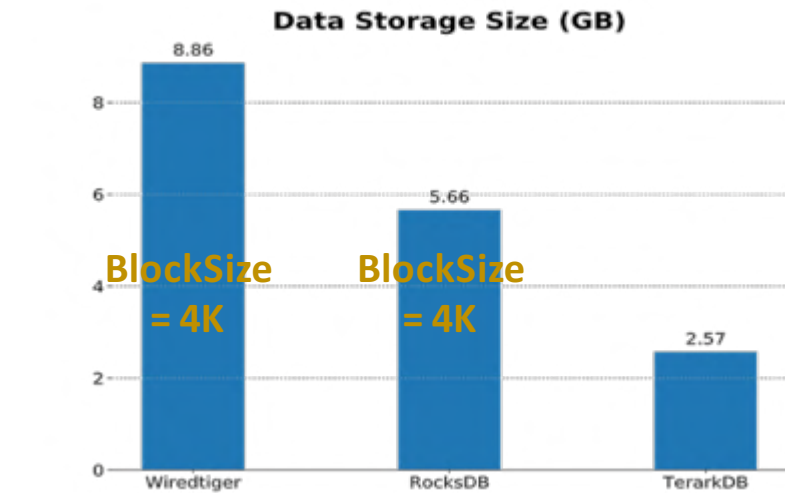
- optimize table 会触发 rocksdb 的 compact
- compact 会减小磁盘和内存占用，并提高性能
- compact 会花一些时间，并占用一些 CPU 和内存
- compact 会随着数据的写入自动运行
 - 一般不需要手工 optimize table 去触发 compact
 - 手工 optimize table 最好在系统负载较低时进行

YCSB on MySQL,

Amazon movies data



Bare Metal on Engine, Amazon movies data



Terark 可检索压缩技术：全局压缩

- 压缩（**写**）的单位是**整个数据集**
 - 压缩过程中 CPU 和内存消耗都比较大
- 访问（**读**）的单位是**单条数据**
 - 直接读压缩的数据
 - 只读需要的数据，没有无效读取和解压
- Index(**Key**) 和 Data(**Value**) 使用不同的算法
 - **CO-Index** (**C**ompressed **O**rdered **I**ndex), 压缩的、有序的索引
 - **PA-Zip** (**P**oint **A**ccessible **Z**ip), 可定点访问的压缩

CO-Index (Compressed Ordered Index)

- CO-Index 是个逻辑概念，有多种不同的物理实现
- Terark 应用最广的实现是 Nested Succinct Trie
 - 多个嵌套起来的 Trie 树，使用 Succinct 方式表达
 - 随机定点查询 (Point Search) 很快
 - 随机范围查询 (Range Search) 较慢
 - 顺序遍历较慢 (比定点查询快，但比传统索引的顺序遍历慢很多)
 - 查询的结果是一个内部 ID

PA-Zip (Point Accessible Zip)

- **PA-Zip** 是个逻辑概念，有多种不同的物理实现
- Terark 应用最广的实现是 **Global Dict Zip**
 - 顾名思义：全局字典压缩，套用分类标准，属于 **lz77** 系列
 - 支持按 ID 定点提取单条数据
 - ID 即数据的序号（可以看做是一个数组下标）
 - 按 ID 随机读取，大于 300MB/s，最高到 3GB/s
 - 按 ID 顺序读取，大于 500MB/s，最高到 7GB/s

PA-Zip (Point Accessible Zip)

- PA-Zip 是个逻辑概念，有多种不同的物理实现
- Terark 应用最广的实现是 **Global Dict Zip**

- 主要使用**全局字典**压缩
- 配合使用**局部字典**压缩
- 压缩过程需要**两遍**扫描

全局字典最大需要 **12G** 内存

默认配置下，压缩 67G 数据时，需要 **12G**
数据更多时，也不会超过 **12G**

- ◆ 第一遍生成全局字典，第二遍执行压缩

- Pipeline 多线程压缩：

使用 TerarkDB 的 MyRocks

- 用 TerarkDB 替换 MyRocks 中的 RocksDB
 - 替换 librocksdb.so (必须动态连接, 不可静态连接)
 - 将 Terark lib 目录加入 LD_LIBRARY_PATH
- 设置环境变量
 - LD_PRELOAD=libterark-zip-rocksdb-r.so
 - TerarkZipTable_localTempDir=/some/dir

TerarkDB 引擎的优势

- 压缩率更高
 - 磁盘文件更小，SSD 寿命更长
 - 数据规模越大，压缩率越高
- 内存用量更少
 - 只需 mmap 压缩的文件，无双缓存问题
- 随机访问更快
 - 直接访问压缩的数据
 - 只访问需要的数据，没有无效解压
 - 顺应硬件发展趋势：随机少量读，高 IOPS

TerarkDB 在 MyRocks 中的特殊优势

- MyRocks 使用单一 DB，适合云 DB 多租户场景
 - 多租户会导致非常多的 Table & Index
 - 使用 Key 前缀区分不同的 Table & Index
 - ◆ 大大减小碎片问题：碎片文件，碎片内存.....
- 单一 DB 更适合 TerarkDB 的场景
 - 数据规模越大，压缩率越高，内存利用率也越高
 - 云厂商可能会为租户提供一系列 Table & Index 模板
 - ◆ 每个租户的数据可能都很少
 - ◆ 但数据的模式（表结构）相似，更有利于 TerarkDB 的全局压缩

TerarkDB 引擎的劣势：压缩过程

- 速度慢

- 算法复杂，计算量大
- 算法的内存访问局部性较差，CPU Cache miss 较高

- 内存用量大

- 全局压缩，需要更多工作内存

- 解决：限制内存 & 优化调度

- 限制压缩算法的并发线程数、内存上限.....

TerarkDB 引擎的劣势：顺序读

- 顺序读包括**全表扫描**和**区间扫描**
- 顺序读慢在 **CO-Index**, **PA-Zip** 的顺序读是很快的
- **CO-Index** 顺序读的性能比起**随机读**, 只高一点点
 - 严格讲是 Nested Succinct Trie, 它的数据高度压缩, 结构复杂
 - 即使在顺序读的情况下, 内存访问的局部性也很一般
 - ◆ CPU Cache & TLB miss 居高不下, 使用 HugePage 会有所改善 (TLB Miss 减小)
- 传统存储引擎的块压缩, 非常适合顺序读
 - 顺序读比随机读快几百倍, 甚至几千倍, 也快于 TerarkDB 的顺序读
 - 只需要一个工作块 (Working Block), 相当于块缓存中只需要一个块
 - 不需要访问索引, 工作块中已解压的数据也都会被用到

新的问题， 以及优化建议

- 引擎层太快， 执行层 相对性能损失 超过 10 倍
- 发动机马力增大时
 - 需要增大 变速箱、传动系统、轮胎... 的承载能力
- 使用更强的 CPU
 - 传统引擎瓶颈在 IO， 增加 CPU 无效
 - TerarkDB 的瓶颈在 CPU 和内存带宽
- 使用更快的网络（万兆网）
 - 引擎层太快， 千兆网跟不上

传统存储引擎在
内存足够的情况下，
瓶颈才会转移到
CPU 和 网络



谢谢！

THANKS

SequeMedia
盛拓传媒

IT168.com

ITPUB

ChinaUnix