DBAplus

云数据库架构设计与实践沙龙

# 随机读场景下的 MongoDB 性能优化

雷 鹏

# 大纲

- MongoRocks 简介
- 传统数据库的块压缩
- 硬件的发展趋势
- TerarkDB 和 MongoRocks 的关系
- TerarkDB 的原理、优势、劣势
- 对 MongoRocks 的 oplog 做特殊处理
- 新的问题，以及优化建议
- TerarkDB 与 WiredTiger

# MongoRocks 简介

- MongoRocks = MongoDB + RocksDB(存储引擎)
- 完整支持 MongoDB 的上层功能
- 成熟稳定，经历实战考验(facebook parse)
- 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——

- WiredTiger
- LevelDB/RocksDB...
- TukoDB
- 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
  - 块大了：索引更小，单个块中的数据条数更多，**无效解压**更多
  - 块小了：索引更大，压缩
- 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

使用 Direct IO

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**

# 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

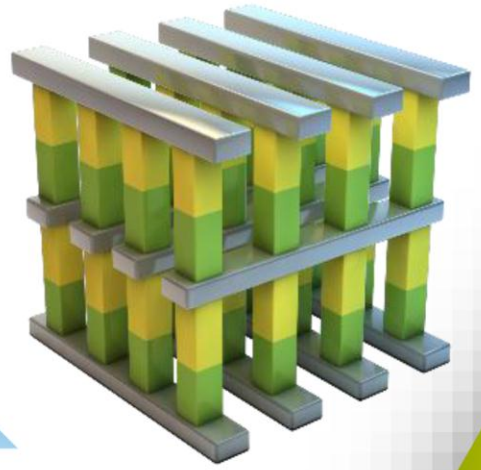
*Performance & Lower costs: Integrated solutions*

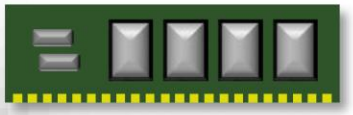
Intel® Scalable System Framework

**1000X**
**FASTER**
Than NAND[1]

**1000X**
**ENDURANCE**
Of NAND[2]

**10X**
**DENSER**
Than DRAM[3]





Data granularity:
64B cacheline

# 硬件的发展趋势

更大

| | 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 | |

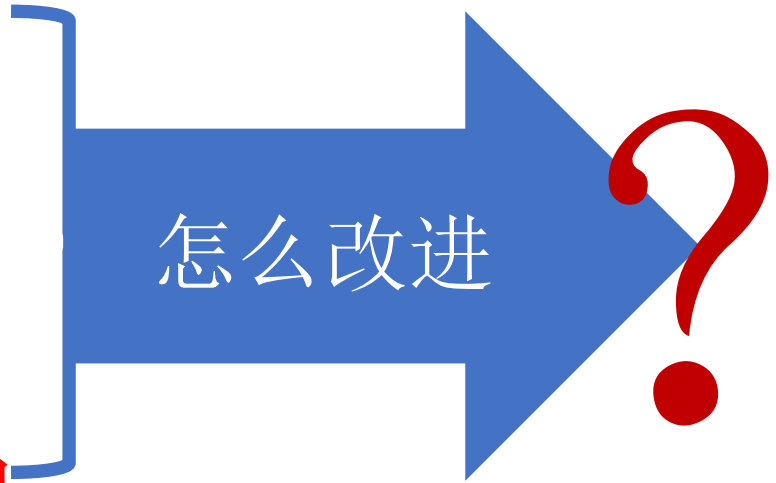
更快

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 |
| | | | 9 years, 6 months, 02 days, 17 hours, 20 mins, |

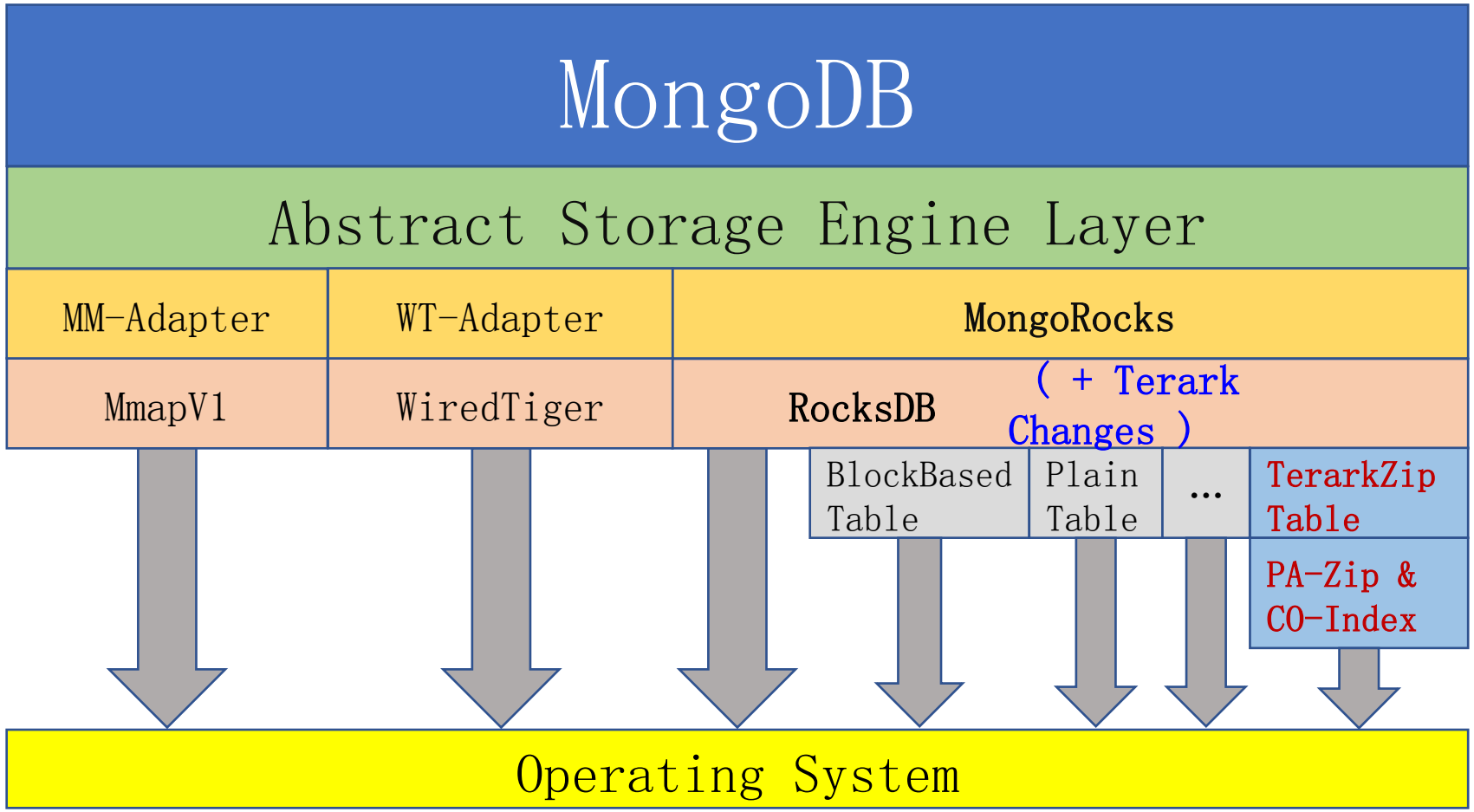
# 块压缩的缺点，我想静静……

- 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




且听分解……

# TerarkDB 和 MongoRocks 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

```
db.serverStatus().rocksdb["table-name"]
```

返回：

```
  TerarkZipTable
```

---

更多信息：

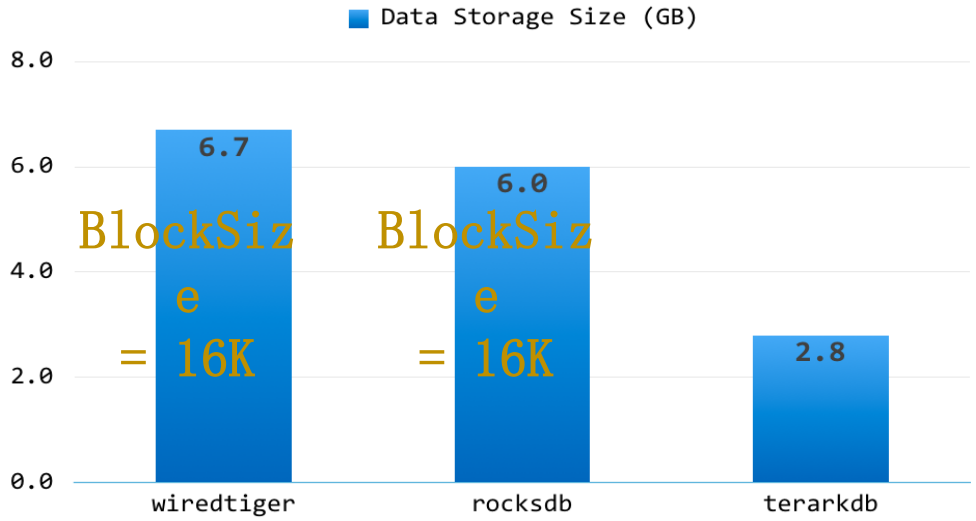
```
db.serverStatus().rocksdb["table-options"]
```

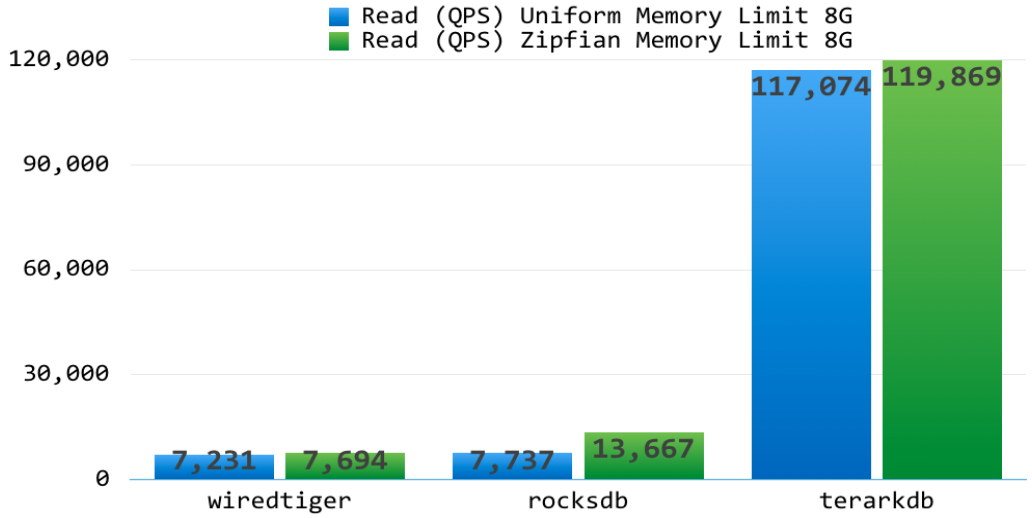
# 执行 compact

```
db.runCommand ( { compact: '<collection>'} )
```

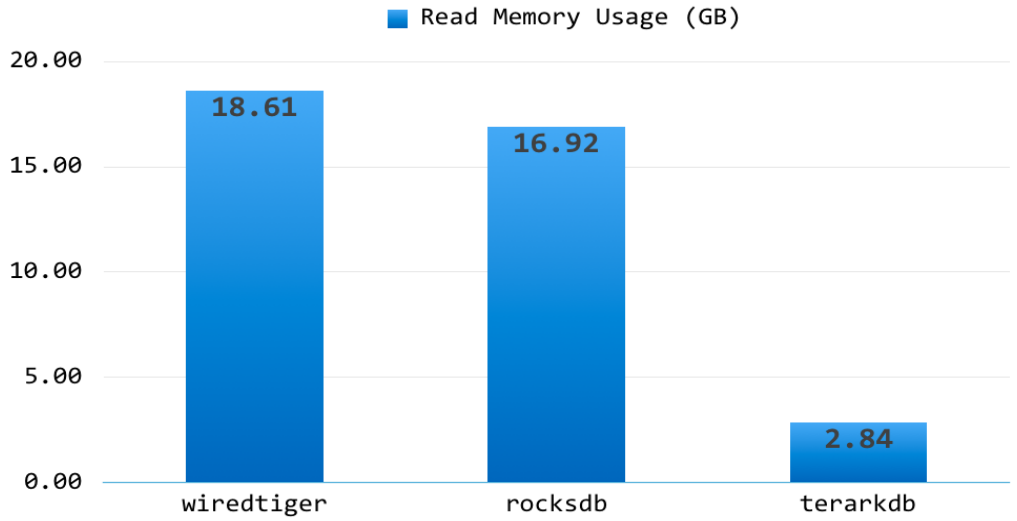
- compact 会减小磁盘和内存占用，并提高性能
- compact 会花一些时间，并占用一些 CPU 和内存
- compact 会随着数据的写入自动运行
  - 一般不需要手动 compact
  - 手动 compact 最好在系统负载较低时进行
- TerarkZipTable 默认使用 universal_compaction

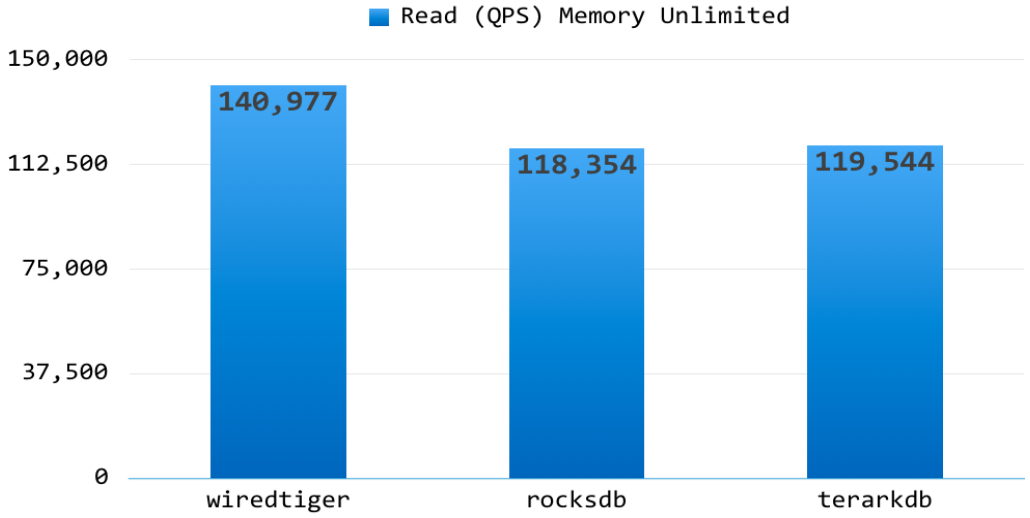
# YCSB on MongoDB,



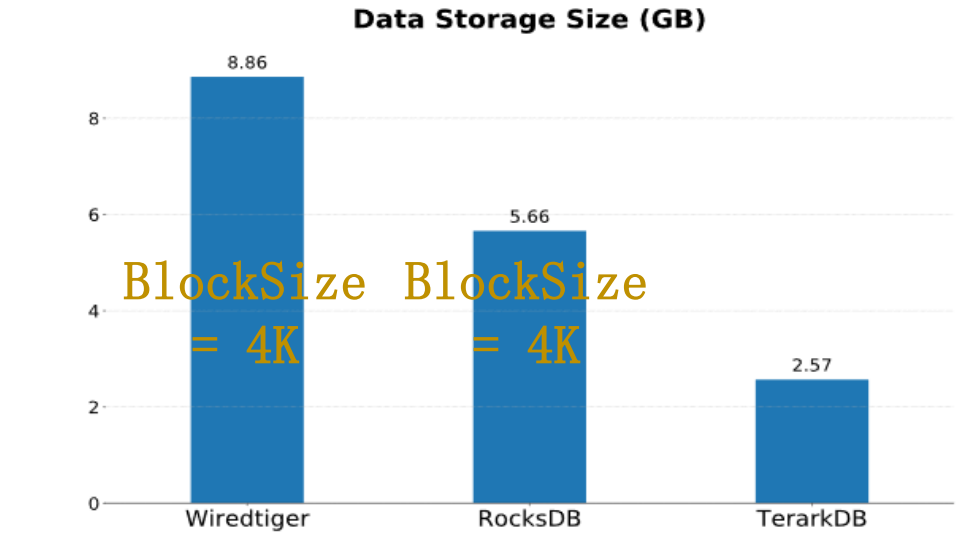




# Amazon movies data



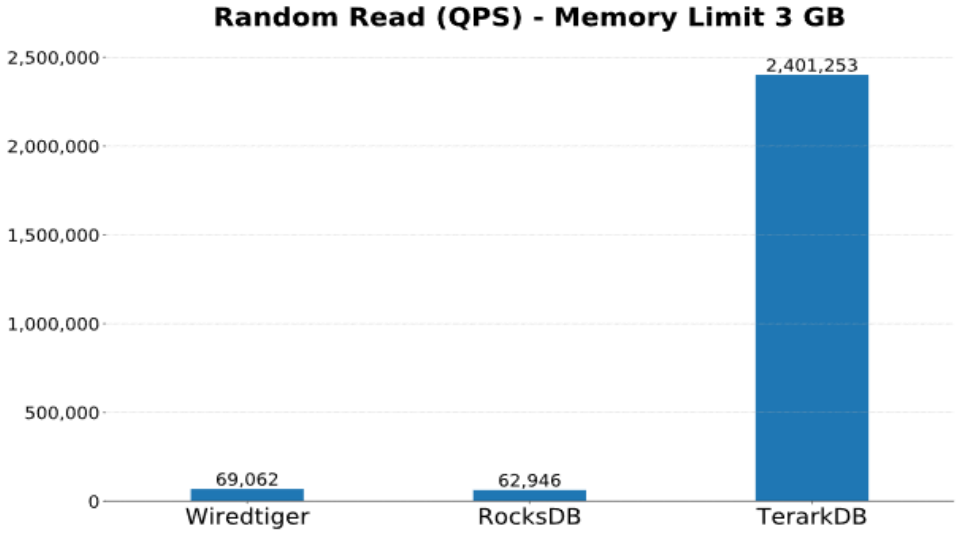



# Bare Metal on Engine.

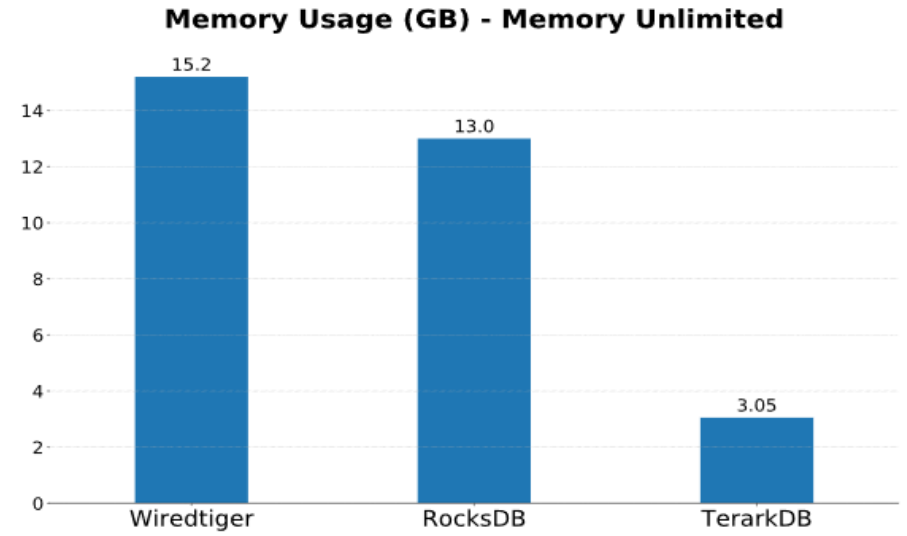


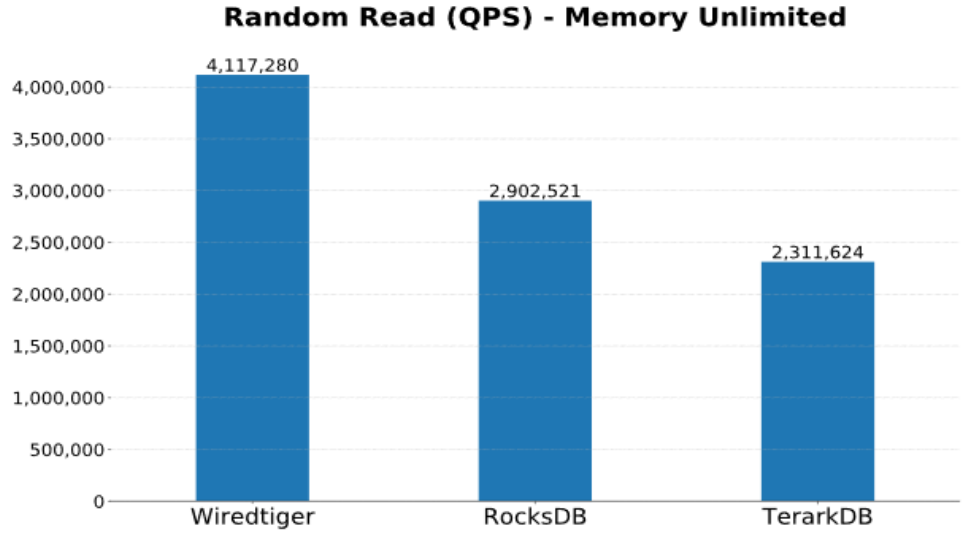

BlockSize = 4K

BlockSize = 4K




# 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 Index ），压缩的、有序的索引
  - **PA**-Zip（ **P**oint **A**ccessible Zip ），可定点访问的压缩

# CO-Index（Compressed Ordered Index）

- CO-Index 是个逻辑概念，可以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，可以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，可以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
    - 第一遍生成全局字典，第二遍执行压缩
  - Pipeline 多线程压缩

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

# 使用 TerarkDB 的 MongoRocks

1. 用 TerarkDB 替换 MongoRocks 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
2. 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 MongoRocks 中的特殊优势

- MongoRocks 使用单一 DB，适合云 DB 多租户场景
  - 多租户会导致非常多的 Collection & Index
  - 使用 Key 前缀区分不同的 Collection & 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 Collection & Index 模板
    - 每个租户的数据可能都很少
    - 但数据的模式（Schema）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
  - 限制压缩算法的线程数、内存上限，提高小作业的优先级
  - 使用 universal compaction，减小写放大，提高 compact 速度

稍慢于 gzip ，远快于 bzip2

# 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

# 对 MongoRocks 的 oplog 做特殊处理

- oplog 主要用来做数据同步，有容量限制（Capped）
- oplog 的生存时间一般很短，并且以 FIFO 的方式删除
  - 所以 oplog 不适合使用 TerarkDB
  - 把 oplog 放入一个专有的 ColumnFamily: oplog
  - 把 oplog ColumnFamily 放入 TerarkDB 的 BlackList
    - 通过环境变量配置 BlackList，从而使用默认的 BlockBasedTable

# 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
  - 传统引擎的瓶颈在 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**

# TerarkDB 与 WiredTiger

- WiredTiger 也有 LSM
- WiredTiger SSTable by CO-Index + PA-Zip
  - 将 CO-Index 和 PA-Zip 集成到 WiredTiger
- In-progress, POC 阶段……


DBAplus
www.dbaplus.cn
THANK YOU