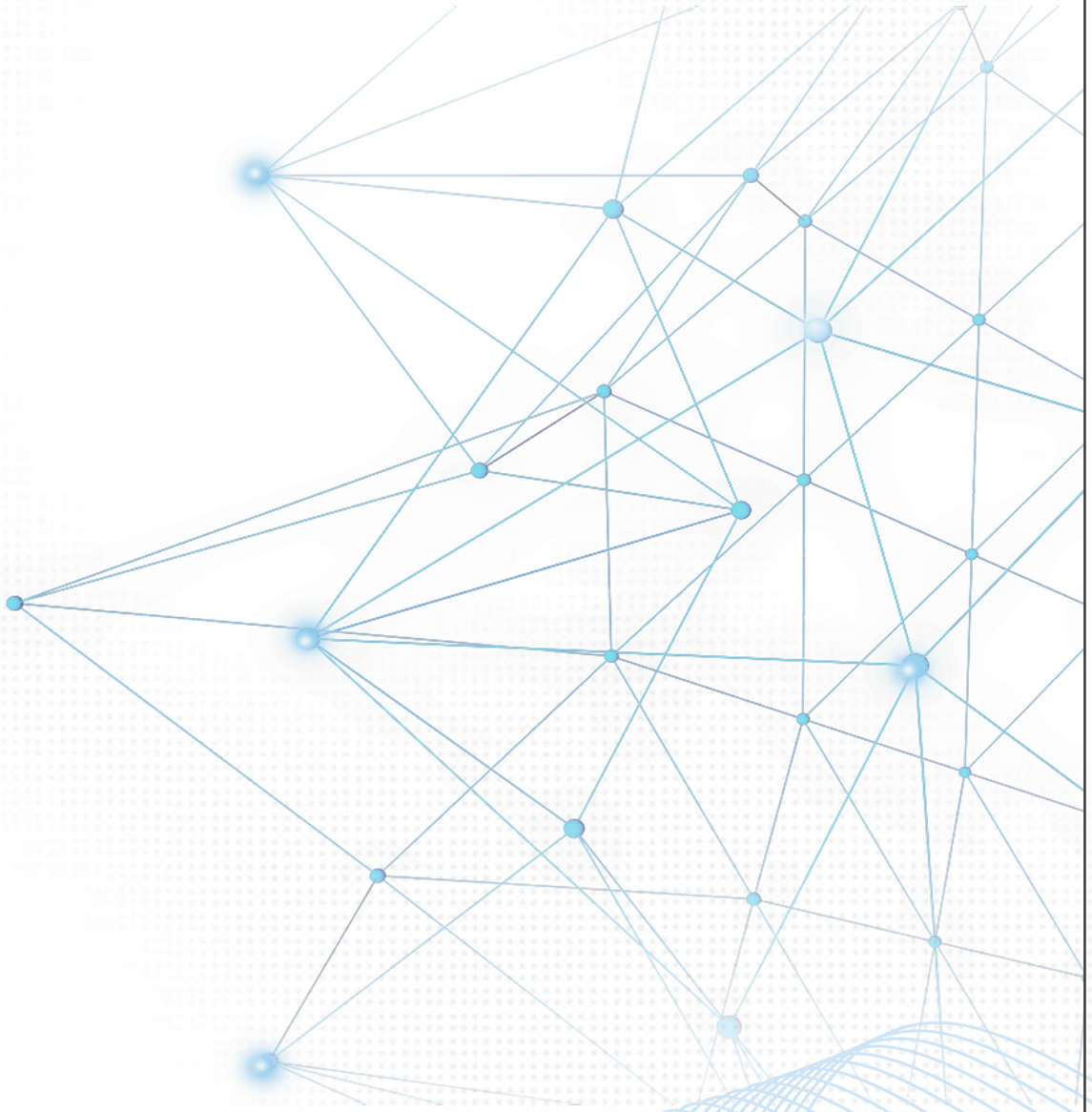
第二届华为云数据库挑战赛决赛答辩

汇报人：小组织　　时间：2020.9.11

Background& Limitations

01



1 Background

元数据管理器

管理日志的版本和存储信息

写入日志 WriteDeltaPacket

读取回放到指定版本数据 ReadDataByVersion

崩溃恢复

1 Background

正确性验证： 30线程并发写（崩溃恢复）

并发写： 30线程并发写（随机写）

并发读： 30线程并发读（随机读）

并发同时读写： 15线程并发写，15线程并发读

1 Limitations

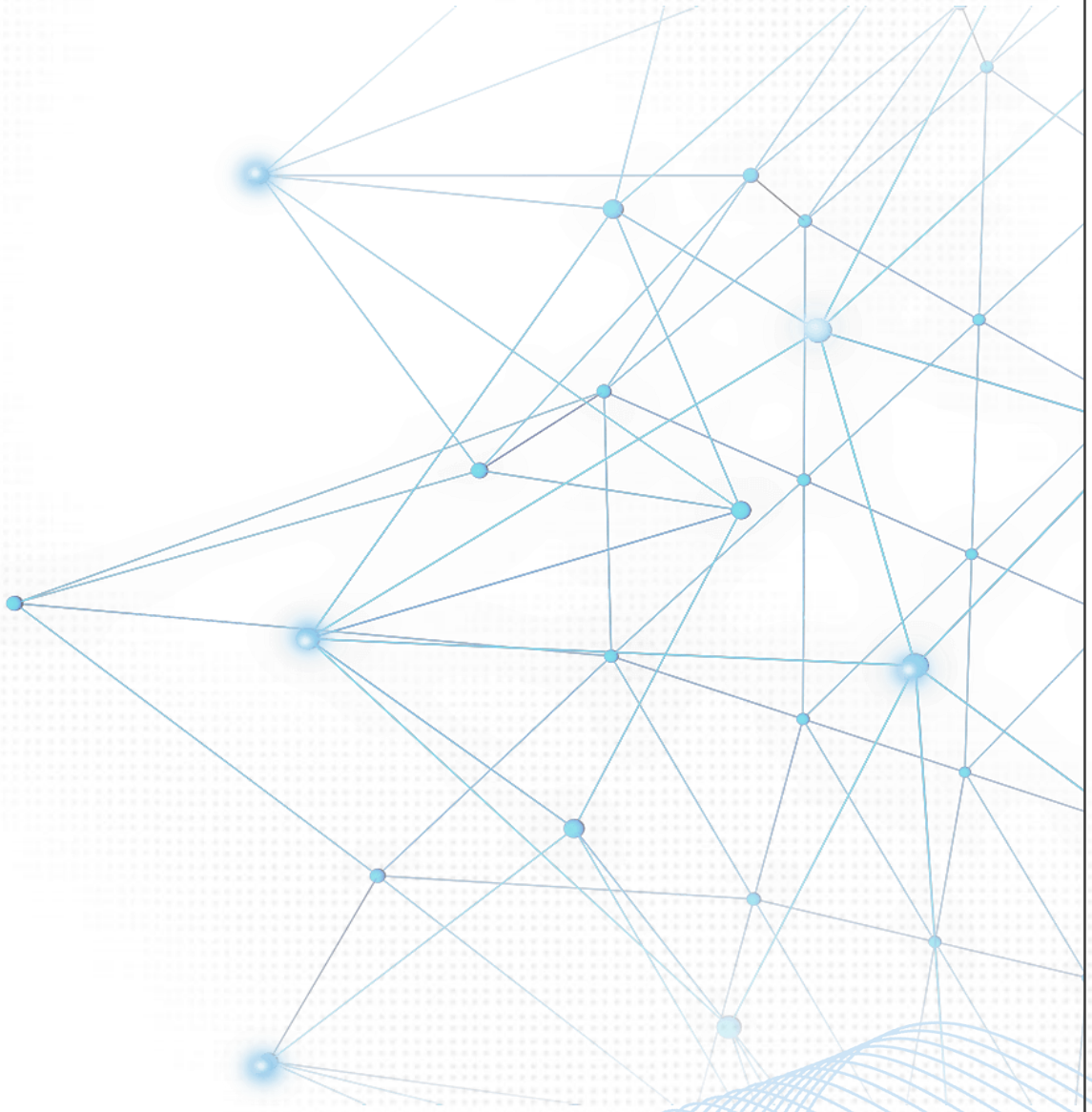
磁盘： SSD 100G

内存： 4G（评测程序1G）

性能瓶颈：ReadByVersion操作（读多个Version数据）

Idea

02

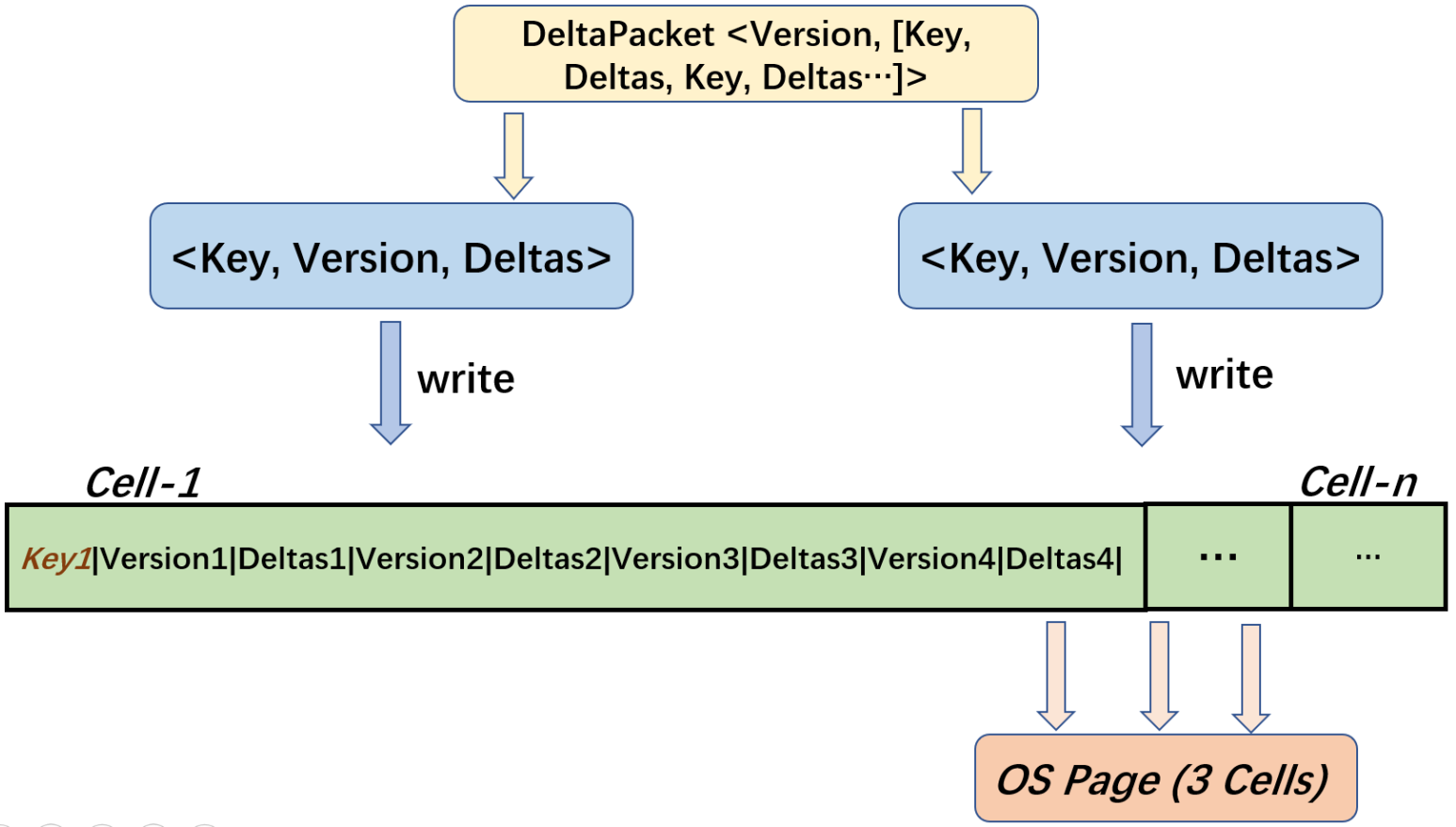


2 存储设计

前提：
每个Key仅含有4个不同Version的Deltas，大小在4K以内

结论：
围绕ReadByVersion设计。相同Key的所有Version数据连续存储（Cell，一个OS Page可以存储3个Cell），读取任意Version时只需要加载操作系统的一个页

2 存储设计



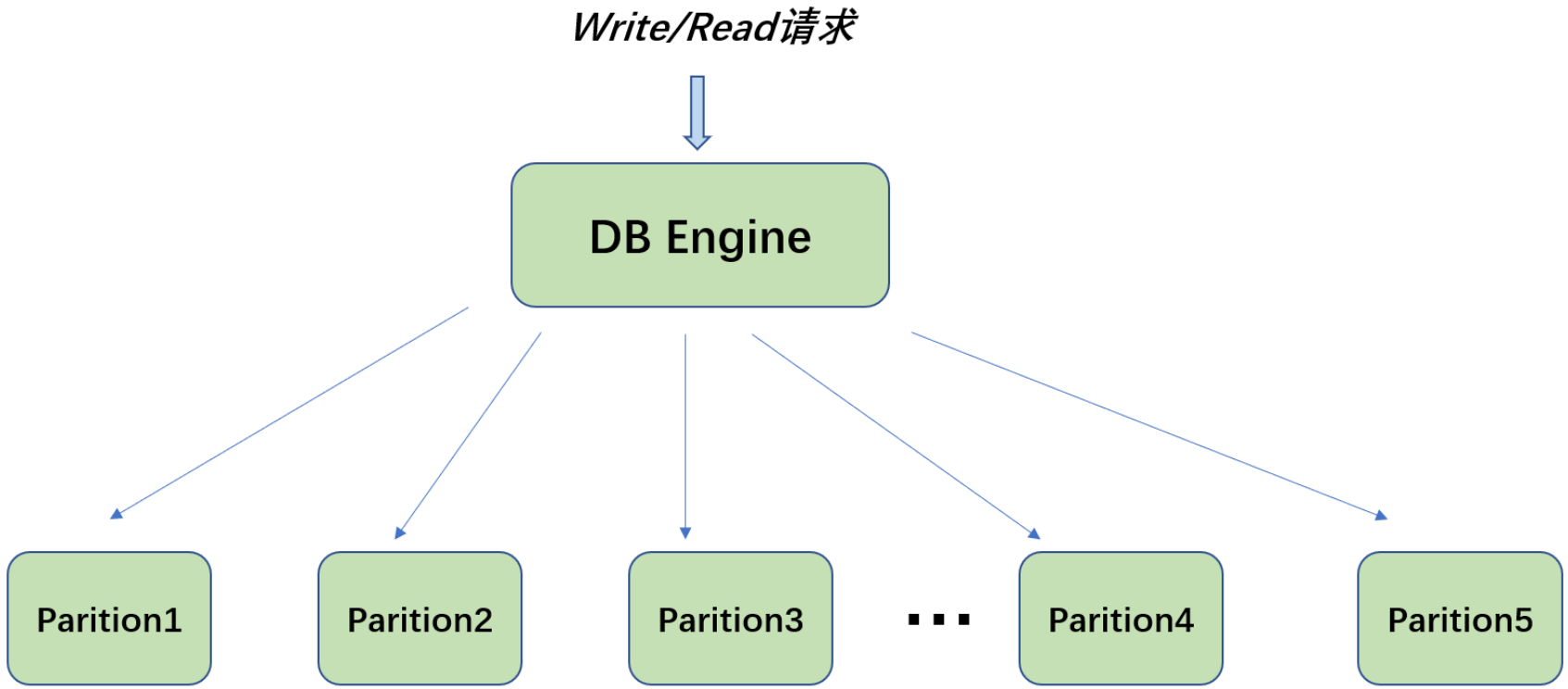
<Key1, Offset>
<Key2, Offset>
<Key3, Offset>
<…,…>
<Keyn, Offset>

Disk Organization

In Memory Index Organization

2 存储设计

Partiton：减小并发冲突， 按key的hash分区，实测512个分区效果最好



2 写入流程

1）每次WriteDeltaPacket调用分解为多个<Key,Version,Deltas>的写入

2）根据Key的hash选择对应partition写入

3）采用mmap方式，直接写入mmap,每次写入新的Key 时占用磁盘上一个Cell以内的大小，写入已经存储过的Key时直接在对应Cell位置插入

4）写入时维护内存索引，记录每个Key在磁盘的Cell位置和每个Cell下次写入的位置

5）每个Parition使用一把锁，防止并发冲突

2 读取流程

1）根据Key的hash，选择对应partition读取

2）读内存索引，计算对应Key所在Cell的磁盘位置

3）读磁盘同样采用mmap的方式，因为每个Cell大小小于一页，所以最多只需要一次磁盘IO就可以得到所有版本数据

4）根据函数调用Version参数所需的对应版本，过滤掉大于该Version的数据再累加

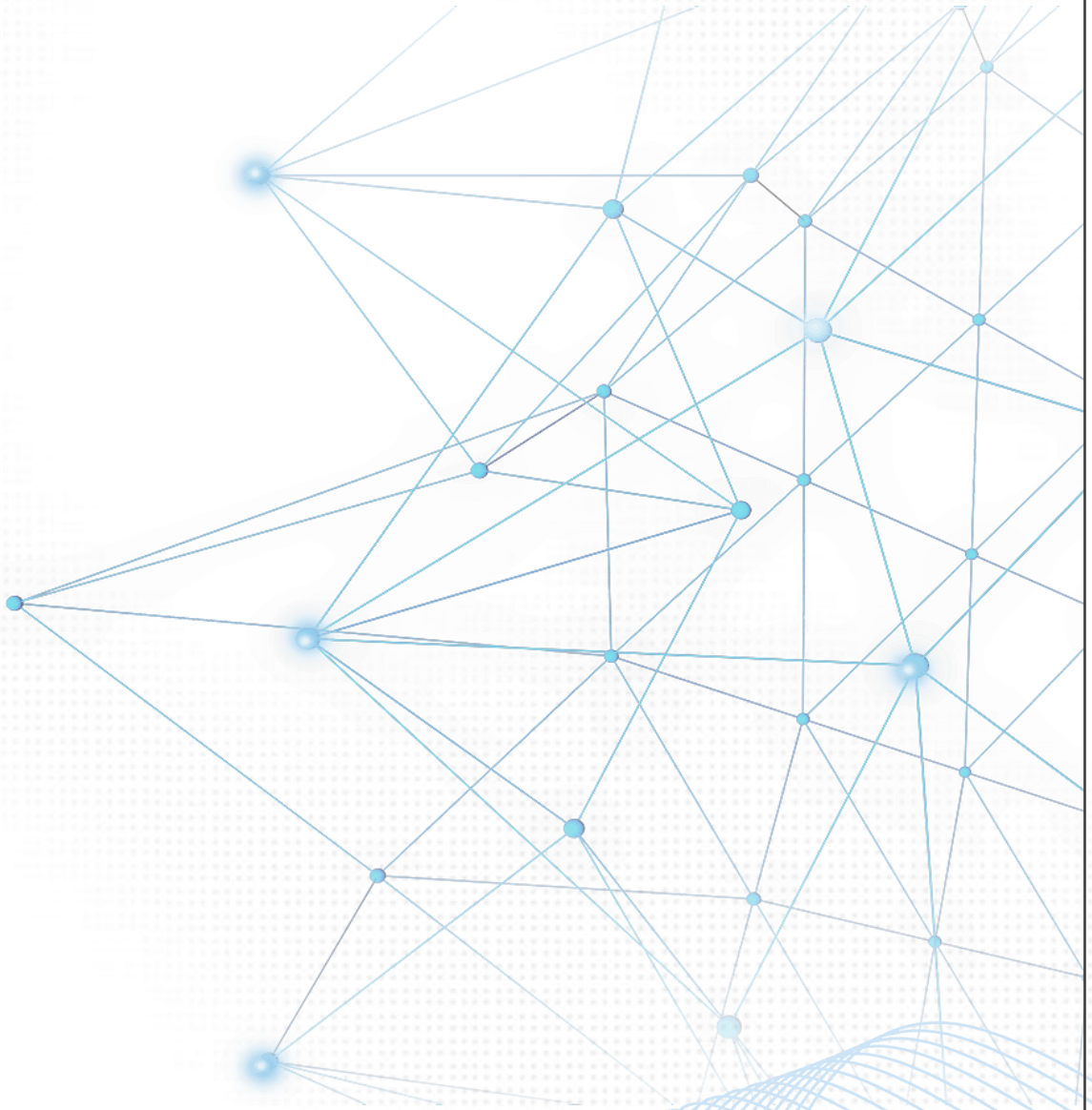
2 崩溃恢复

1）每次都直接写入mmap，保证了重启后数据没有丢失

2）写入时不需要维护磁盘索引，重启时顺序扫描磁盘上的数据重建内存索引

Optimization

03



3 Optimization

1）调用WiteDeltaPacket时先合并相同Key的数据，减少磁盘写入大小

2）数据压缩：Deltas使用40位即可表示，设计了U40数据类型，占用磁盘空间更小

```
struct u40_4{
  uint8_t no1;
  uint8_t no2;
  uint8_t no3;
  uint8_t no4;
  uint32_t no1_;
  uint32_t no2_;
  uint32_t no3_;
  uint32_t no4_;
};
```

3 Optimization

```
void Utils::memcpyToU40(u40_4* dst,  const uint64_t* src){
  for(int i = 0; i < DATA_FIELD_NUM; i += 4){
    int ii = i >> 2;
    dst[ii].no1_ = src[i];
    dst[ii].no1 = src[i] >> 32;
    dst[ii].no2_ = src[i+1];
    dst[ii].no2 = src[i+1] >> 32;
    dst[ii].no3_ = src[i+2];
    dst[ii].no3 = src[i+2] >> 32;
    dst[ii].no4_ = src[i+3];
    dst[ii].no4 = src[i+3] >> 32;
  }
}
```

```
void Utils::memcptToU64(uint64_t* dst,  u40_4* src){
  for(int i = 0; i < DATA_FIELD_NUM; i += 4){
    int ii = i >> 2;
    dst[i] = ((uint64_t)(src[ii].no1) << 32) + src[ii].no1_;
    dst[i+1] = ((uint64_t)(src[ii].no2) << 32) + src[ii].no2_;
    dst[i+2] = ((uint64_t)(src[ii].no3) << 32) + src[ii].no3_;
    dst[i+3] = ((uint64_t)(src[ii].no4) << 32) + src[ii].no4_;
  }
}
```

3）内存索引不直接存储offset，而是存储Key对应Cell的序号（递增），可以将Long由Int代替，索引使用内存减小一半

4）数据的partition对Key hash使用 murmur hash，可以让每个分片的数据分布大小尽量均匀

5）通过mmap读取一页时，预读下一页内容缓存到L3，增加局部性

4 感想

赛题很有挑战性

通过比赛学习了很多数据库和内核底层知识

认识了很多志同道合的极客们

感谢主办方提供了这样一个技术交流的平台

Thank You

汇报人：小组织 时间：2020.9.11