[Scaled KV 1](#_Toc13159773)

[1. layered index 1](#_Toc13159774)

[1.1 初始化 & B层 1](#_Toc13159775)

[1.2 A层： 2](#_Toc13159776)

[1.3 C层： 3](#_Toc13159777)

[2 Dynamic threads and concurrency 4](#_Toc13159778)

[3. Scale out 5](#_Toc13159779)

[4. KV ops and consistency 5](#_Toc13159780)

[5. 整个数据结构优缺点总结： 6](#_Toc13159781)

[6. 论文需要解答的问题： 6](#_Toc13159782)

## Scaled KV

Key的类型: key一定要等长，否则每个key node size不一致，通过A层算不出来。

## 1. layered index



### 1.1 初始化 & B层

Motivation：B+树，B-tree增长到一定容量后，Level增加，查找性能和写性能下降。数据量小于（N key）时采用原B+tree结构。数据直接插入B+树，形成一定规模后（N Key）将叶子节点读出全排序，产生有序数组B。B+ tree的数据量能够保证存入各Key range的数据量达到平衡。

优点：不需要log，数据量大，产生的数列可以使每个range的数据量相对平衡



B是(key, point)连续追加组成的有序数列。每个元素8byte + 8 byte = 16 byte。

Remain：**B**的key个数N（敏感性测试）。



### 1.2 A层：

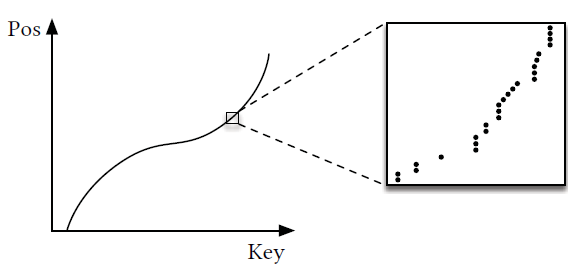
CDF统计B层数据分布情况，存储函数信息，生成CDF数组**A**，加速定位有序数列**B。**

1）CDF B全部KV项的分布情况，共N个KV项

2）设span=0.01，选择x= 0.01, 0.02, …, 1.00，100个坐标点，坐标点个数=1/span.

3）在数组CDF[i]中，

4）如果 , 说明i~i-1之间的key过多，B层查找开销大。记录i~i-1区间内设span=span/10，选择x=0.0i0, 0.0i1, … 0.0i9, 10个 (x,y) 坐标点。

5） 容量分析

数组中只存y，key在B中的相对位置。Y=float.

Span=0.01, 100 entry，A=1/span\* 4byte=0.4KB

Span=0.001, 1000 entry，A=4KB

……

**查找key:**

1. 如果,
   1. 在Position CDF[i-1]~Position CDF[i]间的B层做二分查找
2. 如果,
   1. 在Position CDFi[j-1]~Position CDFi[j]间的B层做二分查找

### 1.3 C层：



Put&Update

1）B层的每个key的指针首先指向自己的数据

2）当新的写请求到来，如ki对应key range为k1<ki<k2，为k2分配一个新的节点（256byte），指针指向新节点，将k2和ki写入新的节点。

a. 节点内部如下图所示，首先由标志位标记node，区别于KVitem. 其次记录prefix长度，bitmap标记有效数据，4个元素标记指向孩子节点的元素。

b. node 1:

node 2:

node 3:

node 4:

3）元素少于16，追加，更新bitmap

4）No.16插入，Put-off sorted操作，因为256byte的节点需要生成ECC，利用读出的元素对节点内元素排序，生成sorted arrary。

5）No.17插入，创建一个新的节点。

读和查找：根据sorted array 二分查找。



## 2 Dynamic threads and concurrency



线程池由M个locate线程和K个range服务线程组成，M+K=N

在任务队列中的个请求，取M个分配给M个线程，locate到有序数列B对应的指针。然后由serving线程，负责特定key range内的具体操作。 M个线程之间可以无条件并行，K线程之间，每个key range只允许一个线程访问。当Req\_key与当前服务的线程不在同一个key range则分配一个serving线程给该请求。

## 3. Scale out

Resize without data migration and foreground requests blocking

当树的深度达到引起性能下降的阈值，将full node 加入B，删除B中的无效数据

**Step 1：**50%的子树深度达到N；或者个别子树深度达到M（M>>N）；触发scale out操作。

**Step 2：**依据缩放规则，构造新数组B’。

·for depth n2 < tree < depth n3, 增加根节点元素到数组B。

·for depth n3 < B+ tree, 增加根节点及根节点的子节点元素到数组B。

（将所有full node 加入B，还是多少层的数据加入B）

**Step 3：**根据数组B’重新生成CDF A’

并行迁移数据到B’及服务前台请求。

·标记scale out的当前位置 key i，比较请求与key i的大小关系：

·Key i以前的数据通过二分查找B’服务读写请求

·Key i以后的数据通过原CDF，在B服务读写请求



## 4. KV ops and consistency

4.1 Delete

1）查找

2）删除B数列中指向的value，将B数列元素的8byte指针置空

3）删除B数列所指向Bucket/tree中的value，修改bitmap。

4.2 Get

1）见1.2 查找CDF。

2）在最小的key range内，通过指针查找C层

·如果是KV item，则只需比较一个key。

·如果是type I node， 遍历节点。

·如果是Type II node，binary search sorted array, 继续查找所在range的孩子节点。

4.3 Scan

1）（start\_key, end\_key）所有B层overlap的range并行读，merge

2）（start\_key, count）后k个range并行读，merge，cut4.2 一致性

4.4Update

1）B有序数列直接指向的KV items

2）B 指向的节点

Algorithm 2 UPDATE(op, key, value)

1: orig kv item = get original item(index entry);

2: new kv item = alloc space(sizeof(kv item));

3: set new kv item according to key and value;

4: persist(new kv item, sizeof(kv item));

5: AtomicWrite(pointer, new entry);

6: persist(pointer, sizeof(index entry));

7: free space(orig kv item);

3）B 指向的树

## 5. 整个数据结构优缺点总结：

1. 构造更快的索引，
2. 增加并行性，优化写；优化定位到查找的流程，优化写！
3. 优势：并行+层数减少+scan+约等于hash的查找性能
4. 天然前缀压缩
5. 缺点：key是定长整数，顺序写问题
6. 缺点：没有减少B/B+ tree的数据迁 （减少了）<https://www.youtube.com/channel/UCbowfWkY0jVt2eM_r9oOUVA>
7. **问题：**读size 64byte；写size 256byte；读写大小不对称，节点大小增加后怎么优化读性能？cacheline命中率，以及减少cacheline访问。缓存CDF

## 6. 论文需要解答的问题：

1. 为什么是NVM-optimized： 256byte：Put-off sorted
2. what’s your competitor? related work and your superiority
   1. HiKV