MV-Sketch实现

函数简介

1.构造函数

```
MVSketch::MVSketch(int depth, int width, int lgn)
```

depth：散列函数个数
width：每个散列函数对应的桶个数（宽度）
lgn：键值长度
其中使用AwareHash()生成用于进行散列的种子

```
unsigned long seed = AwareHash((unsigned char*)name, strlen(name), 13091204281, 228204732751, 6620830889);
for (int i = 0; i < depth; i++) {
    mv_.hardner[i] = GenHashSeed(seed++);
}
```

2.更新函数

```
void MVSketch::Update(unsigned char* key, val_tp val)
```

key：用于散列的键值（用于溯源的键值）
val：用于票选的值（重量）
对于每一个键值，需要根据所有散列函数的返回值将其插入桶中。若桶规模为depth*width，则对于一个键值，要根据depth个散列函数将其插入至depth个桶中，其中一个depth函数对应一行数量为width桶。对于一次循环，首先使用初始化MVSketch时生成的种子进行散列，根据返回值得到本次要更新的桶的下标。若桶为空，则标识该桶的键值更新为key并且该桶的重量自增val；若该桶的键值与key相同，则该桶的重量自增val；若该桶的键值与key不同，将该桶的重量减去val，此时若该桶的val小于等于0则将该桶的键值更新为key。如此重复depth次。

```
for (int i = 0; i < mv_.depth; i++) {
    bucket = MurmurHash64A(key, keylen, mv_.hardner[i]) % mv_.width;
    int index = i*mv_.width+bucket;
    MVSketch::SBucket *sbucket = mv_.counts[index];
    sbucket->sum += val;
    if (sbucket->key[0] == '\0') {
        memcpy(sbucket->key, key, keylen);
        sbucket->count = val;
    } else if(memcmp(key, sbucket->key, keylen) == 0) {
        sbucket->count += val;
    } else {
        sbucket->count -= val;
        if (mv_likely(sbucket->count < 0)) {
            memcpy(sbucket->key, key, keylen);
            sbucket->count = -sbucket->count;
        }
    }
}
```

3.查询函数

```
void MVSketch::Query(val_tp thresh, std::vector<std::pair<key_tp, val_tp> >&results)
```

thresh：用于确定重流的因子，当一个桶的重量大于thresh时它才有可能成为重流。
results：用于存放查询结果的一个vector，其中key_tp为键值、val_tp为重量。
对于每一个候选结果，查询它在depth行中的最小重量，若最小重量仍大于thresh，则将该候选插入结果。

```
for (auto it = res.begin(); it != res.end(); it++) {
    val_tp resval = 0;
    for (int j = 0; j < mv_.depth; j++) {
        unsigned long bucket = MurmurHash64A((*it).key, mv_.lgn/8, mv_.hardner[j]) % mv_.width;
        unsigned long index = j*mv_.width+bucket;
        val_tp tempval = 0;
        if (memcmp(mv_.counts[index]->key, (*it).key, mv_.lgn/8) == 0) {
            tempval = (mv_.counts[index]->sum - mv_.counts[index]->count)/2 + mv_.counts[index]->count;
        } else {
            tempval = (mv_.counts[index]->sum - mv_.counts[index]->count)/2;
```

```
        }
        if (j == 0) resval = tempval;
        else resval = std::min(tempval, resval);
    }
    if (resval > thresh ) {
        key_tp key;
        memcpy(key.key, (*it).key, mv_.lgn/8);
        std::pair<key_tp, val_tp> node;
        node.first = key;
        node.second = resval;
        results.push_back(node);
    }
}
```

问题讨论

1.估计流量是否准确？
由于散列函数的特殊性，两个大小不同的键值经过同一个哈希函数散列后得到的值可能相同，因此估计出的流量大小可能偏大。
2.如何提高准确率？
若散列函数的数量设置的足够多或者根据数据内容将宽度进行调整，可以减少散列得到的值重复的可能性，但是过多的散列函数将导致空间占用过大，宽度过长也将导致退化问题。