2021 级

《大数据存储系统与管理》课程

课 程 报 告

姓　名　刘兴元

学　号　U202115663

班　号　计算机本硕博 2101 班

日　期　2024.04.19

目　　录

一、选题背景

在我们工作中，如果遇到如网页 URL 去重、垃圾邮件识别、大集合中重复元素的判断一般想到的是将集合中所有元素保存起来，然后通过比较确定。如果通过性能最好的 Hash 表来进行判断，那么随着集合中元素的增加，我们需要的存储空间也会呈现线性增长，最终达到瓶颈。

所以很多时候会选择使用布隆过滤器来做这件事。布隆过滤器通过一个固定大小的二进制向量或者位图（bitmap），然后通过映射函数来将存储到 bitmap 中的键值进行映射大大减少了空间成本，布隆过滤器存储空间和插入/查询时间都是常数 O(K)。但是随着存入的元素数量增加，布隆过滤器误算率会随之增加，并且也不能删除元素。

为了解决布隆过滤器中不能删除，且存在误判的缺点，引入了一种新的哈希算法->Cuckoo Filter，它既可以确保该元素存在的必然性，又可以在不违背此前提下删除任意元素，仅仅比 bitmap 牺牲了微量空间效率。

所以在布隆过滤器的基础上，布谷鸟过滤器诞生了。

二、Cuckoo Filter 原理分析

布谷鸟过滤器（Cuckoo Filter）是一种用于快速查询的概率型数据结构，通常用于判断某个元素是否存在于一个集合中。它基于布谷鸟哈希（Cuckoo Hashing）算法，相比于传统的哈希表，在空间和查询时间上具有更好的性能。

我将简要介绍其基本思想：

1. 使用两个哈希函数 h1 (x) 、 h2 (x) 和两个哈希桶 T1、T2 。
2. 插入元素 x：
   - 如果 T1 [h1 (x)] 、T2 [h2 (x)] 有一个为空，则插入；两者都空，随便选一个插入。
   - 如果 T1 [h1 (x)] 、T2 [h2 (x)] 都满，则随便选择其中一个（设为 y ），将其踢出，插入 x。重复上述过程，插入元素 y。
   - 如果插入时，踢出次数过多，则说明哈希桶满了。则进行扩容、ReHash 后，再次插入。
3. 查询元素 x：读取 T1 [h1 (x)] 、T2 [h2 (x)] 和 x 比对即可。

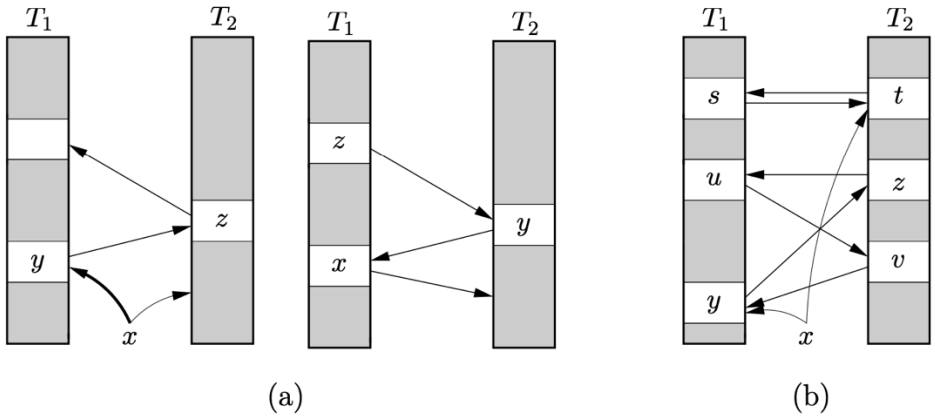

Fig. 1. Examples of CUCKOO HASHING insertion. Arrows show possibilities for moving keys. (a) Key x is successfully inserted by moving keys y and z from one table to the other. (b) Key x cannot be accommodated and a rehash is necessary.

布谷鸟（Cuckoo），即大杜鹃，喜欢在别的鸟窝里产蛋。布谷鸟幼鸟出生后，会将其他蛋踢出，借巢长大。布谷鸟哈希的关键设计正在于 “踢出”（kicks out）这个动作。

可以证明，布谷鸟哈希插入操作的均摊时间复杂度是 O(1)。相较于其他哈希方法，布谷鸟哈希具有空间利用率高、查找速度快等优势。

三、算法设计与实现

我根据布谷鸟过滤器的基本原理：布谷鸟过滤器的核心思想是利用两个哈希表和两个哈希函数来解决哈希冲突，当一个桶中的元素已满时，尝试将元素插入另一个桶中，或者随机选择一个桶进行替换。这样可以有效地减少哈希冲突，并提高过滤器的性能。

我使用 c++ ，使用自带的库哈希函数，编写了一个最基本简单的布谷鸟过滤器 demo：

1. 成员变量和构造函数：

CuckooFilter 类包含了几个成员变量，其中包括两个哈希表 T1 和 T2，以及两个 c++自带的哈希函数 hashFunction1 和 hashFunction2。构造函数 CuckooFilter() 初始化了这些成员变量，包括使用 random_device 初始化了随机数引擎，并且定义了一个均匀分布 distribution。

```
// 计算数据应该插入的两个桶的索引
 void getIndices(const std::string& item, int& index1, int& index2)
{
    index1 = hashFunction1(item) % NUM_BUCKETS;
    index2 = (hashFunction2(item) % (NUM_BUCKETS - 1)) + 1;
}
```

2. 插入数据：

insert 函数用于向布谷鸟过滤器中插入数据。首先，我根据哈希函数计算

出元素应该插入的两个桶的索引。然后，我尝试将元素插入第一个哈希表的对应桶中。如果该桶已满，则尝试将元素插入第二个哈希表的对应桶中。如果两个桶都已满，就选择一个哈希表，随机选择一个桶，将该桶中的元素替换为待插入的元素，并尝试将被替换出的元素重新插入另一个哈希表。

```
1.  bool insert(const std::string& item) {
2.          int index1, index2;
3.          getIndices(item, index1, index2);
4.
5.          if (insertToTable(item, &T1[index1]))
6.              return true;
7.          if (insertToTable(item, &T2[index2]))
8.              return true;
9.
10.         // 随机选择一个哈希表进行替换
11.         int tableIndex = chooseTable();
12.         std::vector<std::string>* table;
13.         if (tableIndex == 0)
14.             table = &T1[index1];
15.         else
16.             table = &T2[index2];
17.
18.         // 随机选择一个桶进行替换
19.         int bucketIndex = distribution(rng) % BUCKET_SIZE;
20.         std::string temp = table->at(bucketIndex);
21.         table->at(bucketIndex) = item;
22.
23.         // 尝试将替换出的元素插入另一个哈希表
24.         if (tableIndex == 0)
25.             return insertToTable(temp, &T2[index2]);
26.         else
27.             return insertToTable(temp, &T1[index1]);
28.     }
```

3. 检查数据是否存在：

Contains 函数用于检查指定的数据是否存在于布谷鸟过滤器中。通过哈希函数计算出数据应该存储的两个桶的索引，然后在这两个桶中查找数据。如果数据在任何一个桶中被找到，则返回 true，表示数据存在；否则返回 false，表示数据不存在。

四、测试与分析

编写一个简单的程序进行测验：

```cpp
int main() {
    CuckooFilter filter;

    filter.insert("apple");
    filter.insert("banana");
    filter.insert("orange");

    std::cout << std::boolalpha;
    std::cout << "Contains apple? " << filter.contains("apple") << std::endl;
    std::cout << "Contains grape? " << filter.contains("grape") << std::endl;

    return 0;
}
```

运行程序，发现可以正确实现 filter 的基本的插入查询的功能。

```
[Running] cd "/Users/liuxingyuan/csLearning/课程报告/" && g++ test.cpp -o test && "/Users/liuxingyuan/csLearning/
Contains apple? true
Contains grape? false

[Done] exited with code=0 in 1.327 seconds
```

本次实验中,我所写的布谷鸟过滤器实际上非常之基础，而在实际上的布谷鸟过滤器要涉及到复杂的多的哈希算法，好的哈希函数应该能够将数据均匀地分布到哈希表中，以减少冲突并提高过滤器的性能。且实际布谷鸟过滤器可能需要支持动态调整大小，以适应数据量的变化。这可能涉及到动态调整哈希表的大小、重新哈希已有的元素等操作。

五、总结

在本次实验中，我学习了布谷鸟哈希算法，并尝试实现了一个相对简化的版本。在确定选题后，我积极地在网上查阅了相关资料，并深入阅读了关于布谷鸟哈希的相关博客和论文。这个过程锻炼了我自学和查找资料的能力，并让我意识到这些技能在未来的学习和研究生阶段中的重要性。

通过实践，我深入了解了布谷鸟哈希算法的原理和实现细节，学会了如何设计和编写一个简单的布谷鸟过滤器。虽然我实现的版本相对简化，但这个过程为我打下了坚实的代码基础，为将来更深入地研究做好准备。

在这个过程中，我也体会到了自学的乐趣和挑战。通过不断查阅资料、阅读文献和思考问题，我逐渐提高了解决问题的能力，并且对布谷鸟哈希算法有了更深入的理解。这将对我未来的学习和研究产生积极的影响，使我能够更加自信和独立地探索新的知识领域。

参考文献

[1] R. Pagh and F. Rodler, “Cuckoo hashing,” Proc. ESA, pp. 121-133, 2001.
[2] Yu Hua, Hong Jiang, Dan Feng, "FAST: Near Real-time Searchable Data Analytics for the Cloud", Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC), November 2014, Pages: 754-765.
[3] Yu Hua, Bin Xiao, Xue Liu, "NEST: Locality-aware Approximate Query Service for Cloud Computing", Proceedings of the 32nd IEEE International Conference on Computer Communications (INFOCOM), April 2013, pages: 1327-1335.
[4] Qiuyu Li, Yu Hua, Wenbo He, Dan Feng, Zhenhua Nie, Yuanyuan Sun, "Necklace: An Efficient Cuckoo Hashing Scheme for Cloud Storage Services", Proceedings of IEEE/ACM International Symposium on Quality of Service (IWQoS), 2014.
[5] B. Fan, D. G. Andersen, and M. Kaminsky, “MemC3: Compact and concurrent memcache with dumber caching and smarter hashing,” Proc. USENIX NSDI, 2013.
[6] B. Debnath, S. Sengupta, and J. Li, “ChunkStash: speeding up inline storage deduplication using flash memory,” Proc. USENIX ATC, 2010
[7] Go 语言实现布谷鸟过滤器, https://www.luozhiyun.com/archives/453
[8] 布谷鸟哈希和布谷鸟过滤器, https://www.qtmuniao.com/2021/12/07/cuckoo-hash-and-cuckoo-filter/