2021 级

《大数据存储与管理》课程

实 验 报 告

姓　名　骆传威

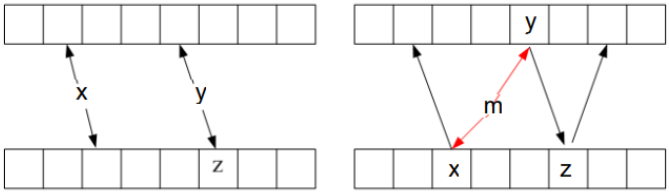
学　号　U202115451

班　号　计算机 2104 班

日　期　2024.04.18

一、选题

选题 3： Cuckoo-driven Way
如何确定循环，减少 cuckoo 操作中的无限循环的概率和有效存储。



Insert item x and y　　Insert item m

二、基本介绍

Cuckoo-driven Way 是一种基于 Cuckoo Hashing 算法的散列表技术。Cuckoo Hashing 算法是一种高效的散列表算法，它通过两个散列表(或者称为桶)来减 少散列冲突。

Cuckoo Hashing 算法的基本思想是:使用两个散列表，每个元素在其中一个散列表中占据一个桶，如果在另一个散列表中发现该桶已经被占用，则将该元素插入到该桶所对应的位置，同时将原来占据该位置的元素插入到另一个散列表中。这样，即使散列函数存在冲突，也可以保证元素可以被正确地插入散列表中。

传统的哈希冲突解决在最坏情况下查找时间为 O(n),而 Cuckoo Hashing 的最坏情况下查找时间为Ω(1)。Cuckoo Hashing 使用多个哈希函数和多个哈希表。对于插入操.作，如果在某个哈希表中的对应位置为空，则直接插入，如果所有哈希表对应位置都不为空，则选取一个哈希表中对应位置的元素进行驱逐，对被驱逐的元素重新进行哈希操作，就像杜鹃鸟占据别的鸟巢并把鸟巢中原有的蛋踢掉的行为。对于查找和删除操作，Cuckoo Hashing 的时间复杂度永远为Ω(1)，因为对每个哈希表只会检查其中的一个位置

Cuckoo Hashing 插入元素的操作与查询操作相比需要更大的时间开销，是改进和优化的关键。在插入踢除的过程中，如果一个元素在某个位置被踢除，但是在后续的操作中又重新占据该位置，那么就会产生踢除的无限循环，在实现中需要选择合理的提出策略并及时检测是否发生无限循环的可能。

Cuckoo-driven Way 则是在 Cuckoo Hashing 算法的基础上进行了改进,双哈希函数，同时使用两个哈希表存储关键字，以减少哈希冲突的概率。在此基础上，Cuckoo-driven Way 还提供了一种循环移位的策略，以减少 Cuckoo 操作中的无限循环的概率和有效存储。踢除策略并及时检测是否有发生无限循环的可能。

总体来说，Cuckoo-driven Way 是一种高效的散列表技术，可以在大规模数据处理中发挥重要作用，尤其适用于需要高速查询的应用场景。

三、数据结构设计与实现

分析可知，Cuckoo-driven Way 核心操作是插入和查找。

在参考了网上的 libcuckoo 库后，数据结构定义与初始化代码如下

```
struct CUCKOO_TABLE_NAME {
  tbl_t t;
  CUCKOO_TABLE_NAME(size_t n) : t(n) {}
};
typedef struct CUCKOO_TABLE_NAME CUCKOO_TABLE_NAME;

#define CUCKOO_KEY_ALIAS CUCKOO(_key_type)
typedef CUCKOO_KEY_TYPE CUCKOO_KEY_ALIAS;
#define CUCKOO_MAPPED_ALIAS CUCKOO(_mapped_type)
typedef CUCKOO_MAPPED_TYPE CUCKOO_MAPPED_ALIAS;

CUCKOO_TABLE_NAME *CUCKOO(_init)(size_t n) {
  CUCKOO_TABLE_NAME *tbl;
  try {
    tbl = new CUCKOO_TABLE_NAME(n);
  } catch (std::bad_alloc &) {
    errno = ENOMEM;
    return NULL;
  }
  tbl->t.minimum_load_factor(0);
  tbl->t.maximum_hashpower(libcuckoo::NO_MAXIMUM_HASHPOWER);
  return tbl;
}
```

查找与插入功能函数声明如下：

```
// find
bool CUCKOO(_find)(const CUCKOO_TABLE_NAME *tbl, const CUCKOO_KEY_ALIAS *key,
                   CUCKOO_MAPPED_ALIAS *val);
// insert
bool CUCKOO(_insert)(CUCKOO_TABLE_NAME *tbl, const CUCKOO_KEY_ALIAS *key,
                     const CUCKOO_MAPPED_ALIAS *val);
```

插入操作的基本思路是:首先将关键字插入到第一个哈希表中，如果发生了哈希冲突，就将冲突的元素移动到第二个哈希表中。如果在第二个哈希表中也发生了哈希冲突，就将冲突的元素移动回第一个哈希表中，以此类推，直到插入成功或者达到最大迭代次数。

具体来说，插入操作的流程如下:

1)使用两个哈希函数将关键字分别映射到第一个和第二个哈希表中。

2)如果第一个哈希表中的对应位置为空，就将关键字插入到该位置中，插入完成。

3)如果第一个哈希表中的对应位置已经被占用了，就将该元素移动到第二个哈希表中，并将第一个哈希表中的对应位置设置为空。

4)如果第二个哈希表中的对应位置为空，就将关键字插入到该位置中，插入完成。

5)如果第二个哈希表中的对应位置已经被占用了，就将该元素移动回第一个哈希表中，并将第二个哈希表中的对应位置设置为空。

6)重复上述步骤，直到插入成功或者达到最大迭代次数。

查找操作的基本思路是:首先使用两个哈希函数将关键字分别映射到第一个和第二个哈希表中，然后在两个哈希表中查找关键字。如果关键字在其中一个哈希表中存在，就返回 true;否则返回 false。

具体来说，查找操作的流程如下:

1)使用两个哈希函数将关键字分别映射到第一个和第二个哈希表中。

2)在第一个哈希表中查找关键字，如果存在，就返回 true。

3)在第二个哈希表中查找关键字，如果存在，就返回 true。

4)如果在两个哈希表中都没有找到关键字，就返回 false。

四 理论分析

为了减少 Cuckoo-driven Way 中无限循环的概率，可以采取以下措施:
1.设定最大重试次数:在进行 Cuckoo 操作时，可以设置一个最大重试次数,
如果超过该次数则认为当前操作失败。这可以防止出现死循环的情况。但是会导致一部分极端情况的正常操作被判定为死循环操作，无法从根源上解决死循环问题，并且要根据处理的数据量大小每次都要改动成一个合适的值。

2.增加哈希表容量:增加哈希表的容量可以减少哈希冲突的概率，从而降低 Cuckoo 操作失败的概率。这种方法通过利用硬件资源存储优势来缓解软件程序设计的弊端，在硬件资源差的时候无法使用。

3.使用更多的哈希函数:使用更多的哈希函数可以增加元素与哈希表项之间的映射关系，从而减少哈希冲突的概率。这种方法通过增加程序的设计复杂性来实现，过度使用反而会导致程序执行整体时间无法接受。

4.使用随机化:在 Cuckoo-driven Way 中，可以使用随机化来选择哈希函数，从而降低哈希冲突的概率，减少 Cuckoo 操作失败的概率。由于引入了随机化，所以优化的程度性能也具有随机性，条件允许可多次测是分析提升性能。

为了有效存储，在 Cuckoo-driven Way 中，可以采取以下措施:
1.压缩哈希表:可以使用一些压缩技术来减少哈希表的存储空间，如哈希表压缩。

2.合并哈希表项:可以将一些相邻的哈希表项合并成一个大的哈希表项，从而减少哈希表的存储空间。

3.使用紧凑的哈希表:可以使用紧凑的哈希表来减少哈希表的存储空间，如线性探测哈希表。
总之，Cuckoo-driven Way 可以通过调整各种参数来优化其性能，包括减少无限循环的概率和有效存储。

五 实验心得

在本课程中深入学习了大数据存储有关的知识，老师详细的解释了为什么需要大数据存储以及如何发展大数据存储技术，讲解了为什么电脑一掉电便会导致数据丢失和如何解决数据丢失问题。

在本实验中选择 `Cuckoo-driven Way` 作为选题，分析了如何减少死循环的方法，并在容器满后重新哈希，在实验过程中理解到了大数据存储中的关键问题，即在数据容量增大的同时如何保证原有的程序依旧能正常运行。这对编程思路有了极大的启发与开拓，今后也会加深这方面的思考学习。

参考文献

- R. Pagh and F. Rodler, “Cuckoo hashing,” Proc. ESA, pp. 121–133, 2001.
- 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.
- 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.
- 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.
- B. Fan, D. G. Andersen, and M. Kaminsky, “MemC3: Compact and concurrent memcache with dumber caching and smarter hashing,” Proc. USENIX NSDI, 2013.
- B. Debnath, S. Sengupta, and J. Li, “ChunkStash: speeding up inline storage deduplication using flash memory,” Proc. USENIX ATC, 2010
- Libcuckoo library. https://github.com/efficient/libcuckoo.