2021 级

《大数据存储系统与管理》课程

课 程 报 告

姓　名 汪鑫

学　号 U202115339

班　号 CS2101

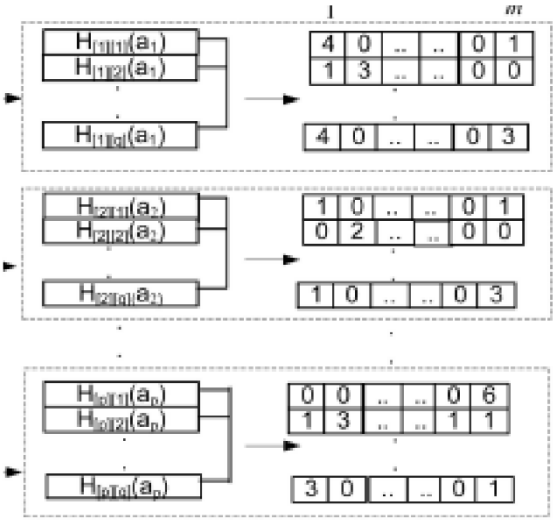
日　期 2024.04.22

目　　录

一、选题

选题 1：基于 Bloom Filter 的设计。

内容：基于 Bloom Filter 的多维数据属性表示和索引。



Bloom Filter 是一种用于快速检查一个元素是否属于一个集合的概率型数据结构。它具有高效的查询速度和占用较少内存的特点，因此在大数据集的处理中得到了广泛应用。本实验旨在探讨基于 Bloom Filter 的多维数据属性表示和索引，即将多维数据映射到 Bloom Filter 中，并利用其进行快速的检索。

二、实验理论

首先，选择一个包含多维属性的数据集作为实验对象。假设有一个包含 N 条记录的数据集，每条记录包含 M 个属性。为了表示这些多维属性，将采用将每个属性映射到 Bloom Filter 中的方法。则需要 M 个 Bloom Filter。

针对每个属性，创建一个独立的 Bloom Filter。在构建 Bloom Filter 时，需要确定合适的哈希函数数量和位数组大小，以保证较低的误判率和较小的内存占用。

对于给定的查询，将查询条件中的多维属性映射到各自的 Bloom Filter 中进行检查。如果所有属性的 Bloom Filter 均返回 True，则认为该记录可能存在于查询结果中。

除此之外，还应该认识到多维数据具有整体性，上述的方法只是分散的检测了每一个属性而并没有考虑到整体性，可能会导致错误。例如在二维数据中有 A{1,3}, B{2,4},此时我们查询 C{1,2},若没考虑到整体性，则会认为 C 存在，产生了误判，因此需要在上述的多维 Bloom Filter 的前提下，在使用一个 Bloom Filter 来存储所有属性的整体哈希值。如此可以减少误判。

三、算法设计与实现

使用 C++来实现具体的代码。

首先定义 Bloom Filter 的类。存储位数组使用 C++标准库中的 bitset，计算哈希值使用 C++标准库中提供的 std::hash。如下图 3.1 所示。

```cpp
#include<bitset>
#include<stdlib.h>
class Bloom_Filter
{
private:
    std::bitset<1024>bits;
    std::hash<std::string>hash_f1;
    std::hash<std::string>hash_f2;
    /* data */
public:
    void add(sonst std::string& item)
    {
        auto hash1=hash_f1(item)%bits.size();
        auto hash2=hash_f2(item)%bits.size();
        bits.set(hash1);
        bits.set(hash2);
    }

    bool seek(const std::string& item) const
    {
        auto hash1=hash_f1(item)%bits.size();
        auto hash2=hash_f2(item)%bits.size();
        return bits.test(hash1)&&bits.test(hash2);
    }
};
```

图 3.1

为了处理多维数据，需要创建一个包含多个 Bloom Filter 的结构，每个维度都有一个对应的 Bloom Filter。最后再使用一个 Bloom Filter 用于存储属性的联合值。如下图 3.2 所示。

```cpp
class MulDimenBloomFilter
{
private:
    std::array<Bloom_Filter,3>filters;
    Bloom_Filter unionFilter;//总检查
    /* data */
public:
    void add(const std::array<std::string,3>& items)
    {
        std::string combined;
        for(size_t i=0;i<items.size();i++)
        {
            filters[i].add(items[i]);
            combined+=items[i];
        }
        unionFilter.add(combined);
    }

    bool seek(const std::arry<std::string,3>& items)
    {
        std::string combined;
        for(size_t i=0;i<items.size();i++)
        {
            if(!filters[i].seek(items[i]))
                return false;
            combined=combined+items[i];
        }
        return unionFilter.seek(combined);
    }

};
```

图 3.2

简单定义一个主函数来测试上述功能，如下图 3.3 所示。

```
int main()
{
    MulDimenBloomFilter mdbf;
    mdbf.add({"1","3","5"});
    mdbf.add({"2","4","6"});
    std::cout <<"TEST '1','2','3':"
        <<(mdbf.seek({"1","2","3"}))?"Found":"Not Found"<<std::endl;
    std::cout <<"TEST '1','2','4':"
        <<(mdbf.seek({"1","2","4"}))?"Found":"Not Found"<<std::endl;
    std::cout <<"TEST '1','3','5':"
        <<(mdbf.seek({"1","3","5"}))?"Found":"Not Found"<<std::endl;

}
```

图 3.3

四、测试与分析

测试集所得到的结果如下图 4.1 所示。

```
TEST '1','2','3': Not Found
TEST '1','2','4': Not Found
TEST '1','3','5': Found
```

图 4.1

正确的得到了相应的结果，由于增加了一个额外的的 Bloom Filter，空间开销也会相应的变大。设位数组的长度为 L，则所花费的空间开销为(M+1)*L。另外经查阅资料，所花费的时间与所选取的哈希函数的选择、哈希函数的个数、错误率的要求等也有密不可分的关系。

五、总结

基于 Bloom Filter 的多维数据属性表示和索引是一种高效的数据结构设计，旨在解决大规模多维数据集的快速检索和查询问题。通过将数据对象的多维属性映射到 Bloom Filter 中，并利用其快速的成员检测特性，我们可以实现快速的数据对象定位和查询操作。

参考文献

• F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese, “Beyond Bloom Filters: From Approximate Membership Checks to Approximate State Machines,” Proc. ACM SIGCOMM, 2006.

• Y. Zhu and H. Jiang, “False Rate Analysis of Bloom Filter Replicas in Distributed Systems,” Proc. Int’l Conf. Parallel Processing (ICPP ’06), pp. 255-262, 2006.

• S. Dharmapurikar, P. Krishnamurthy, and D.E. Taylor, “Longest Prefix Matching Using Bloom Filters,” Proc. ACM SIGCOMM, pp. 201-212, 2003.

• L. Fan, P. Cao, J. Almeida, and A. Broder, “Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol,” IEEE/ACM Trans. Networking, vol. 8, no. 3, pp. 281-293, June 2000.

• B. Xiao and Y. Hua, “Using Parallel Bloom Filters for Multi- Attribute Representation on Network Services,” IEEE Trans. Parallel and Distributed Systems, vol. 21, no. 1, pp. 20-32, Jan. 2010.

• Y. Hua, Y. Zhu, H. Jiang, D. Feng, and L. Tian, “Scalable and Adaptive Metadata Management in Ultra Large-scale File Systems,” Proc. 28th Int’l Conf. Distributed Computing Systems (ICDCS ’08), pp. 403-410, 2008.

• D. Guo, J. Wu, H. Chen, and X. Luo, “Theory and Network Application of Dynamic Bloom Filters,” Proc. IEEE INFOCOM, 2006.

•【程序员都必须会的技术，面试必备【布隆过滤器详解】，Redis 缓存穿透解决方案】 https://www.bilibili.com/video/BV1zK4y1h7pA/?share_source=copy_web&vd_source=c962480b9f97695a0c0bed7f17c46218