

**2019** 级

《大数据存储系统与管理》课程

**基于Bloom Filter的设计**

**姓 名 钟逸**

**学 号 U201914978**

**班 号 计算机1903班**

**日 期 2023.05.28**

**目 录**

[一、 Bloom Filter 1](#_Toc22298)

[1.1 Bloom Filter 1](#_Toc9693)

[1.2 Multi-dimension Bloom Filter 1](#_Toc4030)

[二、数据结构设计 1](#_Toc27278)

[三、操作流程分析 2](#_Toc32758)

[四、理论分析 2](#_Toc16544)

[五、 实验性能测试 4](#_Toc25771)

[六、课程总结 5](#_Toc19553)

[参考文献 5](#_Toc18413)

# Bloom Filter

## 1.1 Bloom Filter

Bloom Filter是一种空间效率高、误判率低的数据结构，常用于判断一个元素是否属于一个集合。特别是在大规模数据处理中，Bloom Filter可以有效地判断一个元素是否存在于一个集合中，因为它只需要使用很少的空间来存储判断信息。

它的主要思想是使用多个哈希函数对元素进行哈希操作，并将哈希结果映射到一个位向量中。当元素需要加入集合时，分别对元素进行哈希操作并将对应的位向量置为1；当判断一个元素是否在集合中时，同样对元素进行哈希操作，并判断对应位向量的值是否为1，若所有位向量都为1，则认为该元素存在于集合中。

Bloom Filter的优点是空间占用小，查询速度快，在大规模数据集上应用广泛。但是由于哈希函数的限制，Bloom Filter可能会出现误判情况，即认为一个元素存在于集合中，但实际上并不存在。因此在对查询结果的可靠性要求比较高的情况下，Bloom Filter可能无法进行应用。

## 1.2 Multi-dimension Bloom Filter

多维Bloom Filter是在原有的Bloom Filter基础上，将元素的多个属性（维度）同时考虑，建立多个位向量来快速定位元素是否在集合中。

在传统Bloom Filter中，哈希函数的输出只有一个，而多维Bloom Filter中，每个哈希函数的输出都能够表示元素在不同属性上的取值情况。

比如，在一个三维Bloom Filter中，每个元素有三个属性，分别为A、B、C。可以使用三个不同的哈希函数，将元素的三个属性分别映射到三个位向量中，每个位向量只负责一个属性的信息。在加入元素时，对每个属性分别做哈希操作，并将对应的位向量上的位设置为1。在查询元素是否存在时，同样对每个属性做哈希操作，并判断对应的位向量上的位是否都为1。

多维Bloom Filter的优点是可以使得在数据中含有多个特征属性的情况下，查询元素是否存在，其相比与单一Bloom Filter更加快速和精确。但是其缺点是需要占用更多的空间，因为每个属性需要保留一个位向量。

# 二、数据结构设计

本实验中，设计了一个多维Bloom Filter，对有三个属性的数据集进行存储然后判断。多维Bloom Filter中有三个Bloom Filter，分别对数据的三个属性进行处理。

每个Bloom Filter有三个属性值，分别为filter\_size、seed\_list和bits。

filter\_size用于传入这个Bloom Filter的位向量的长度。seed\_list是用于设定hash函数的随机数list。bits是一个由filter\_size个0组成的表，用于储存位向量。

每个Bloom Filter中都有两个函数，分别为storage和contain。

storage用于将数据通过hash转换为位，存入bits中。

1. def storage(self, data):
2. """
3. 将data通过hash存入bits
4. :param data: 传入的属性值
5. :return: 无
6. """
7. for i in range(len(self.seed\_list)):
8. hash\_i = hashlib.sha256(self.seed\_list[i].encode('utf8'))
9. hash\_i.update(data.encode('utf8'))
10. idx = int(hash\_i.hexdigest(), 16)
11. idx = idx % self.filter\_size
12. self.bits[idx] = 1

contain在所有数据存完之后，对数据是否存在进行判断。

1. def contain(self, data):
2. """
3. 判断data是否在bits中
4. :param data: 传入的单属性值
5. :return: 是，则true
6. """
7. for i in range(len(self.seed\_list)):
8. hash\_j = hashlib.sha256(self.seed\_list[i].encode('utf8'))
9. hash\_j.update(data.encode('utf8'))
10. idx = int(hash\_j.hexdigest(), 16)
11. idx = idx % self.filter\_size
12. if self.bits[idx] == 0:
13. return False
14. return True

完成了每个Bloom Filter的设计，在使用时创建三个Bloom Filter对象，分别对数据的三个属性进行处理即可。

# 三、操作流程分析

本实验中，为了方便处理数据，设计了一个DataSetter类，用于生成拥有三个属性的数据集，并生成用于不同hash的seed\_list。

对于数据集，先随机地生成三个0~99999之间的小数，然后转为字符串，存入一个list。将这个list添加到data\_list之后，由此得到指定长度的三属性表。

对于seed\_list，生成指定个0~4096之间的随机数，存入seed\_list之中。

因此使用时，首先创建一个DataSetter实例，然后利用seed\_list初始化三个Bloom Filter对象。本实验中选择sha256作为hash函数，选择filter\_size为10000000，hash函数将有3个，数据集长度为1000000。

# 四、理论分析

Bloom Filter存在着False Positive误报的情况。

False Positive指的是当在Bloom Filter中检查一个元素时，Bloom Filter可能会错误返回这个元素在集合中存在的结果。这是因为Bloom Filter本身设计的过程中，存在多个不同元素会哈希到同一个位值上的可能性，即不同元素在哈希后的结果可能相同，造成误判。

而由于Bloom Filter并不删除已有元素，因此并不存在False Negative漏报的情况。

接下来结合实验设计，具体分析False Positive的情况。



图1 多维Bloom Filter

假设一个hash函数将元素映射到一个位相量各位上的概率是平均的，位向量有m个比特位，同时有k个哈希函数。

插入一个数据时，Bits A的特定位置没有被某个hash置1的概率是：

插入一个数据时，Bits A的特定位置没有被所有hash置1的概率是：

插入n个元素后，Bits A的特定位置仍为0的概率为：

所以Bits A中特定位置比特为1的概率是：

检测一个不在集合中的元素，经过哈希之后，Bits A中每个位置都有1的概率则是：

而对于三个属性，每个Bits中的每个位置都有1的概率1为：

带入m=10000000，k=3，n=1000000，则概率为，n个数据中，应有5.27个False Positive。

# 实验性能测试

在main函数中，首先设定参数，然后创建3个Bloom Filter，进行测试：

1. if \_\_name\_\_ == '\_\_main\_\_':
2. *# 本实验假设元素的属性值有3个*
3. filter\_size = 10000000
4. hash\_num = 3
5. data\_length = 3
6. data\_size = 1000000
7. ds = data.DataSetter(data\_length, data\_size, hash\_num)
8. bf1 = BloomFilter(filter\_size, ds.seed\_list)
9. bf2 = BloomFilter(filter\_size, ds.seed\_list)
10. bf3 = BloomFilter(filter\_size, ds.seed\_list)
11. cnt = 0
12. for data in ds.data\_list:   *# 三个属性值通过三个bf存储进不同bits*
13. if bf1.contain(data[0]) and bf2.contain(data[1]) and bf3.contain(data[2]):
14. cnt = cnt + 1
15. bf1.storage(data[0])
16. bf2.storage(data[1])
17. bf3.storage(data[2])
18. print(cnt)
19. print(data\_size)

在存入数据时，判断该数据是否已经有记录过，以此判断False Positive。用cnt记录False Positive数量，当数据的三个属性都已经有记录时，将cnt加1.

最后输出结果：

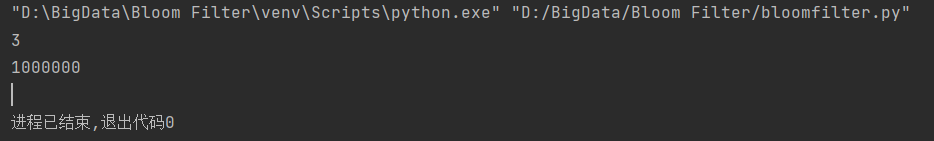


图2 实验输出结果

可见，与前文推导出的结果相近。

1. 查询延迟：Bloom Filter基于hash函数进行计算，所以查询的速度都是很快的，增加到k个hash函数也只是常量级的计算时间；
2. 空间开销：Bloom Filter的空间复杂由位向量大小和哈希函数个数决定。例如对于本实验中的三维Bloom Filter，需要三个长度为10000000的位相量，3\*3个hash对象；
3. 错误率：当位向量足够大、hash函数也适宜时，错误率是很低的。

# 六、课程总结

本次实验，我完成了一个较为简单的多维Bloom Filter的设计。而虽然较为简单，也花费了很多心思，并且通过设计，对Bloom Filter有了更深的理解。

由于是使用随机数生成数据，所以一开始由于设计的随机数过于简单（0~9的int型数值），导致False Positive率极高，想了很久才发现是这方面的问题。

总之，通过本次课程设计，我对大数据存储了解了更多，未来若有机会也可以更加深入地进行学习。

# 参考文献

1. Bonomi F, Mitzenmacher M, Panigrah R, et al. Beyond Bloom filters: From approximate membership checks to approximate state machines[J]. ACM SIGCOMM computer communication review, 2006, 36(4): 315-326.
2. Zhu Y, Jiang H. False rate analysis of Bloom filter replicas in distributed systems[C]//2006 International Conference on Parallel Processing (ICPP'06). IEEE, 2006: 255-262.
3. Dharmapurikar S, Krishnamurthy P, Taylor D E. Longest prefix matching using bloom filters[C]//Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications. 2003: 201-212.
4. Fan L, Cao P, Almeida J, et al. Summary cache: a scalable wide-area web cache sharing protocol[J]. IEEE/ACM transactions on networking, 2000, 8(3): 281-293.
5. Xiao B, Hua Y. Using parallel bloom filters for multiattribute representation on network services[J]. IEEE Transactions on parallel and distributed systems, 2009, 21(1): 20-32.
6. Hua Y, Zhu Y, Jiang H, et al. Scalable and adaptive metadata management in ultra large-scale file systems[C]//2008 The 28th International Conference on Distributed Computing Systems. IEEE, 2008: 403-410.
7. Guo D, Wu J, Chen H, et al. Theory and network applications of dynamic bloom filters[C]//Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications. IEEE, 2006: 1-12.