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

1.1 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。

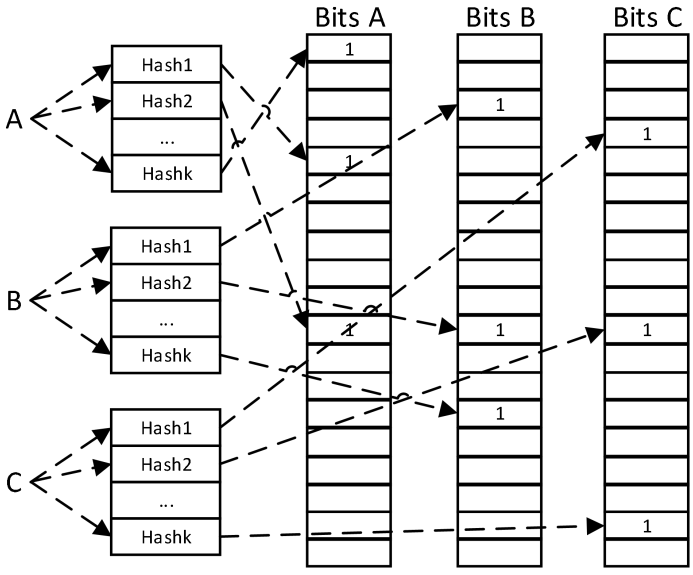

图 1 多维 Bloom Filter

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：

$$1-\frac{1}{m}$$

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：

$$(1-\frac{1}{m})^k$$

插入 n 个元素后，Bits A 的特定位置仍为 0 的概率为：

$$(1-\frac{1}{m})^{kn}$$

所以 Bits A 中特定位置比特为 1 的概率是：

$$1-(1-\frac{1}{m})^{kn}$$

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：

$$(1-(1-\frac{1}{m})^{kn})^k$$

而对于三个属性，每个 Bits 中的每个位置都有 1 的概率 1 为：

$$(1-(1-\frac{1}{m})^{kn})^{3k}$$

带入 m=10000000，k=3，n=1000000，则概率为 5.27×10^{-6}，n 个数据中，应有 5.27 个 False Positive。

五、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.
8.      ds = data.DataSetter(data_length, data_size, hash_num)
9.      bf1 = BloomFilter(filter_size, ds.seed_list)
10.     bf2 = BloomFilter(filter_size, ds.seed_list)
11.     bf3 = BloomFilter(filter_size, ds.seed_list)
```

```
12.
13.       cnt = 0
14.       for data in ds.data_list:   # 三个属性值通过三个bf 存储进不同bits
15.           if bf1.contain(data[0]) and bf2.contain(data[1]) and bf3.co
    ntain(data[2]):
16.               cnt = cnt + 1
17.           bf1.storage(data[0])
18.           bf2.storage(data[1])
19.           bf3.storage(data[2])
20.
21.       print(cnt)
22.       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：

```
"D:\BigData\Bloom Filter\venv\Scripts\python.exe" "D:/BigData/Bloom Filter/bloomfilter.py"
3
1000000

进程已结束,退出代码0
```

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.