分片计数布隆过滤器及其在 Hbase 二级索引的应用①

黄 璨, 方旭昇, 张朝泉

(南京航空航天大学 经济与管理学院, 南京 211106)

摘 要: 针对 Hadoop Database(Hbase)仅支持主索引结构, 即通过主键和主键的 range 来检索数据的问题, 提出利用 Counting Bloom Filter 的新变体建立二级索引来支持非主键数据的检索. 分析了已有的 Counting Bloom Filter(CBF)技术, 针对 CBF 溢出概率高的问题, 提出一种新的 Split Counting Bloom Filter(SCBF)技术, SCBF 将标准 CBF 分成多个相互独立的区域, 由这多个区域共同存储元素的 fingerprint. 实验结果表明, 与标准 CBF 相比, SCBF 降低了溢出概率, 充分提高了过滤器的性能, 可以很好地用来建立 Hbase 二级索引.

关键词: Hbase; 二级索引; 非主键数据; 计数布隆过滤器; 分片计数布隆过滤器

Split Counting Bloom Filter and its Application in Hbase Secondary Index

HUANG Can, FANG Xu-Sheng, ZHANG Chao-Quan

(School of Economics and Management, Nanjing University of Aeronautics & Astronautics, Nanjing 211106, China)

Abstract: A new variant of Counting Bloom Filter was set up to build Hbase secondary index to support the retrieval of non-primary key data, which solved the problem that Hbase only supported the main index structure and retrieve data through the primary key and the primary key range. The new variant, Split Counting Bloom Filter (SCBF) , was proposed according to the high overflow probability problem of Counting Bloom Filter (CBF) after analyzing existing CBF technology. SCBF divided standard CBF into multiple independent regions, which stored elements' fingerprint by all these areas. Comparing SCBF with CBF, the experimental result shows that, SCBF contributes to much lower overflow probability, which improves the performance of filter, and can be used to build the Hbase secondary index.

Key words: Hbase; secondary index; non-primary key; data counting bloom filter; split counting bloom filter

HBase-Google 的 BigTable 模型的开源实现, 是一个高可靠性、高性能、面向列、可伸缩的分布式存储系统, 利用 HBase 技术可在廉价 PC Server 上搭建起大规模结构化存储集群, 适合于各种非结构化和半结构化的松散数据的存储和管理, 目前已经被很多大型企业用于处理海量数据. 然而, 它本身还存在着一些问题, 如仅支持主索引结构, 不支持跨行事务等, 这些问题都严重制约着 Hbase 的进一步发展.

在电商网站中, 商品的数量可以达到千万、甚至亿的级别, 并且每件商品都有十几个甚至上百个属性, 如尺寸、功能、名称、类目、价格等. 用户往往会从多个不同的角度对商品进行查询从而最终确定要购买的商品. 而上文中提到 Hbase 仅支持主索引结构, 导致用户只能从单一角度了解商品, 所以本文准备在 Hbase 中建立二级索引来解决此问题, 使得在实际应用中, 支持对非主键的查询.

很多学者开展了 Hbase 中索引技术的研究工作, 取得了很大的进展, 但主要存在三方面的问题. 第一方面是在分布式集群中很多未存储目标记录的节点也触发了检索过程; 第二方面是通过索引用主键去定位主表数据的时候存在大量的随机读; 还有一方面是整个过程中 Remote Procedure Call Protocol(RPC)操作太多, 这些都会造成大量的不必要的计算资源的浪费. 为了解决上述三方面的问题, 本文准备采用层次结构来组织索引. 首先, 在主节点上, 本文准备用提出的分片计数布隆过滤器(Split Counting Bloom Filter)做全

① 收稿时间:2015-06-24;收到修改稿时间:2015-09-06

局索引, 针对每一个 region server 建立一个可扩展分片计数布隆过滤器(SSCBF). 其次, 针对每一个 region 利用 Hbase 协处理器(Coprocessor)技术建立倒排索引做局部索引. 当检索任务到达, 先通过全局索引定位到存储目标数据的 region server, 然后利用局部索引在存储目标数据的 region server 上定位目标数据. 针对局部索引问题, hindex 已经提出了很好的解决方案, 在此不再累赘, 本文主要就全局索引做详细说明, 介绍分片计数布隆过滤器(SCBF)并与标准计数布隆过滤器做比较.

本文其余部分的结构如下. 第一部分讨论相关的工作. 第二部分介绍布隆过滤器和计数布隆过滤器. 第三部分, 介绍分片计数布隆过滤器(SCBF). 第四部分, 本文将 SCBF 与 CBF 进行比较. 本文在第五部分总结全文.

1　相关工作

Burton Howard Bloom 于 1970 年提出标准 Bloom Filter 用以解决某些元素是否为集合中元素的判断问题, 它突破了传统哈希函数的映射和存储元素的方式, 通过一定的错误率换取了空间的节省和查询的高效[1]. Li Fan 等提出 Counting Bloom Filter(计数布隆过滤器-CBF), 通过将标准 Bloom Filter 位数组的每一位扩展为一个计数器来增加元素的删除操作[2]. d-Left Counting Bloom Filter 利用 d-left hashing 的方法存储 fingerprint, 从一个 hash value 可以同时用作地址和 fingerprint 出发, 将 hash value 的这两种用途相结合来存储信息, 并利用 d-lefting hashing 解决负载均衡问题[3]. 几何布鲁姆过滤器(Geometric Bloom Filter, GBF)通过引入哈希指纹、布鲁姆过滤器两次分割、基于桶负载存放的方法,实现了集合元素的简洁存储、快速查询[4]. K 分组合型 Bloom Filter 由两个基本的 Bloom Filter 构成,每个 Bloom Filter 都作为 K 分组合型的一部分参加运算[5]. 二路平衡动态布隆过滤器按向量组的方式扩充存储空间,新元素的插入是在向量组中查找插入位置, 使得组向量中新置为 1 的位置增加最少[6]. 同源组合布鲁姆过滤器包含流抽样(sample)和分组计数(packet)2 个计数器向量组合, 2 个计数器向量宽度不同, 以相同的散列源函数计算散列位置[7]. Dynamic Counter filter(DCF)使用两个长度相同的数组来存储所有的 Counter, 在基本 CBF 的基础上增加了另外一个用来处理 Counter 溢出的数组[8]. 在 Partial Bloom Filter 中, 位数组被等分成 k(k 为哈希函数的个数)个区域, 每个哈希函数只映射到其中一个区域, 映射范围都是 {0, … , m/k - 1}, 相互并不重叠, 所以 k 个哈希函数可以并行访问位数组[9].

2　Bloom Filter

2.1 标准 Bloom Filter

Bloom Filter 是一种精简的数据结构, 用一个长度为 m 的位数组表示集合中的元素, 每个位的初始值为 0. 为了表达 S={x1, x2,…,xn}这样一个 n 个元素的集合, Bloom Filter 使用 k 个相互独立的哈希函数(Hash Function)分别将集合中的每个元素映射到位数组的 k 个位中, 其中每个元素的映射范围为{0,???,m-1}. 对任意一个元素 x, 第 i 个哈希函数映射的位置 hi(x)就会被置为 1(1≤i≤k). 当查询元素是否属于 S 时, 用 Bloom Filter 的 k 个哈希函数将元素映射到位数组中, 如果哈希函数映射到的位都为 1, 则认为元素以一定的误判率属于 S, 否则元素不属于 S. Bloom Filter 的优点是它的插入和查询时间都是常数, 另外它查询元素却不保存元素本身, 具有良好的安全性.

2.2 标准 Counting Bloom Filter

虽然标准 Bloom Filter 存在上述的优点, 但是它不支持元素的删除操作, 而标准计数布隆过滤器(Counting Bloom Filter - CBF)通过将标准 Bloom Filter 位数组的每一位扩展为一个计数器(Counter)增加了删除操作. 在插入元素时将对应的 k(k 为哈希函数个数)个 Counter 的值分别加 1, 删除元素时将对应的 k 个 Counter 的值分别减 1.

如图 1 所示: x_1、x_2 为 S 中的元素, Hash 函数为 3 个, x_1、x_2 利用 Hash 函数生成相应的 Hash 值, 将二进制存储空间对应的位置进行设置. 如图 2 所示: 删除 x2 时, 将其对应的 3 个 Counter 的值分别减 1.

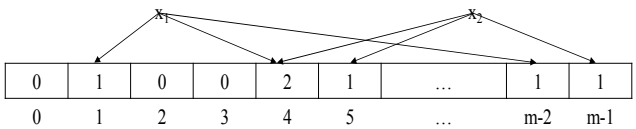

图 1　插入 x_1, x_2 的标准 Counting Bloom Filter

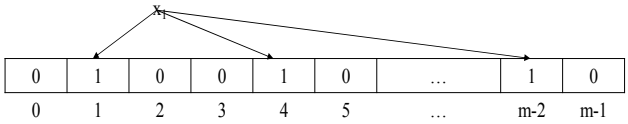

图 2 删除 x_2 的标准 Counting Bloom Filter

3 Split Counting Bloom Filter

3.1 Split Counting Bloom Filter 定义

虽然 CBF 增加了元素的删除操作, 但是它在溢出概率方面还存在着问题, 所以本文在 CBF 的基础上提出分片计数布隆过滤器(SCBF)来解决问题.

SCBF 用一个长度为 M 的位数组表示集合中的元素, 位数组被等分成连续的 k 片, 其中 k 为哈希函数个数, 每片的位数为 . 在 SCBF 中, 每个哈希函数 hi(), 其中 1≤i≤k, 只映射到其中的一片, 映射范围都是 {0, … , M/k - 1}, 所以每个元素由相互独立的 k 位共同索引. 同 CBF 一样, 插入元素时将对应的 k 个 Counter 的值分别加 1, 删除元素时将对应的 k 个 Counter 的值分别减 1.

如图 3 所示, 位数组的长度 M=16, 哈希函数个数 k=4, 初始状态时位数组被等分成连续的 4 片, 每一片的位数为 4 位. 如图 4 所示, 向 SCBF 中插入元素 x1, x2, 利用 4 个相互独立的 hash 函数分别对应每一片生成相应的 hash 值, 将二进制存储空间设置为相应的值. 如图 5 所示, 删除 x_2 时, 将其对应的 4 片上的 4 个 Counter 的值分别减 1.

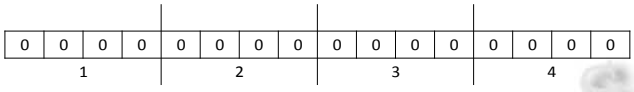

图 3 初始状态的 Split Counting Bloom Filter

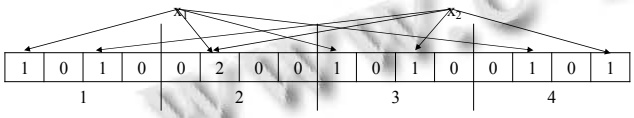

图 4 插入 x_1,x_2 的 Split Counting Bloom Filter

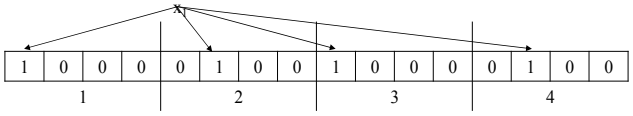

图 5 删除 x2 后的 Split Counting Bloom Filter

3.2 错误率分析

CBF 在判断一个元素是否属于它表示的集合时会有一定的错误率(false positive rate), SCBF 也存在这种情况. 假设每一片中一个确定位被设置为 1 的概率为 f, 当 m 足够大时, 可以认为所有片中一个确定位被设置为 1 的概率相同, 所以错误率为 $P=f^k$. 假设元素 hash 后的值服从均匀分布, 则在每一片中一个确定位被设置为 1 的概率为 $1/m$, 相反的它被设置为 0 的概率是 $1-1/m$, 当把集合 S 中的 n 个元素全部插入 SCBF 中后, 每一片中一个确定位被设置 0 的概率是 $(1-1/m)^n$, 所以它被设置为 1 的概率是 $f=1-(1-1/m)^n$. 在给定 M 和 P 的条件下, 本文下面来确定最优的哈希函 数个数 k 使得 SCBF 存储尽可能多的元素.

当 $|1/m|$ 很小时, $1-1/m\approx e^{-1/m}$, 所以 $f\approx 1-e^{-n/m}$. 由此可得 $n\approx -m\ln(1-f)$, 又因为 $M=km$, $P=f^k$, 所以 $n\approx M(\ln f\ln(1-f)/\ln(1/P))$. 当 P 与 M 一定时, $f=1/2$ 时 n 最大. 即当每一片有一半是满的时候, SCBF 存储了最多的元素. 此时 $n\approx M((\ln 2)^2/|\ln P|)$. 又因为 $P=f^k$, 所以得到最优的哈希函数个数为 $k=\log_2(1/P)$. 同时当 $f=1/2$ 时, 由 $f\approx 1-e^{-n/m}$ 可得 n 的另一个表达式 $n=m\ln 2$.

由上面的分析可以知道当 SCBF 存储了最多的元素时, 哈希函数个数 k 的最优值为 $k=\log_2(1/P)$. 由此可以得出最优情况下 SCBF 的错误率 P 与哈希函数个数 k 的表达式为 $P=2^{-k}$. 根据指数函数的定义, 可以得出 P 随着 k 的增大而减小.

假设 SCBF 的位数组长度 M=45KB, 则最优情况下 SCBF 的各个参数如表 1 所示.

表 1 M=45KB 时最优情况下 SCBF 的各个参数

P(%)	k	m	n
0.1	10	36864	25639
0.01	14	26331	19229
0.001	17	21684	15383
0.0001	20	18432	12819

从表 1 中我们可以得出, 当 SCBF 的位数组长度一定时, 错误率随着哈希函数个数的增大而减少, 同时当错误率减小的时候, SCBF 所存储的最大元素个数也在减少. 由以上结果我们便可以确定 SCBF 的最优参数. 下面来探讨分片计数布隆过滤器的计数位数应该怎样确定.

3.3 溢出概率分析

本文先计算第 i 个 Counter 被增加 j 次的概率, 得到 $P(c(i)=j)=C_n^j(1/m)^j(1-1/m)^{n-j}$, 其中 n 为集合元

素个数, m 为每一片的位数且 $m = M / k$, k 为哈希函数个数. 通过进一步计算, 第 i 个 Counter 的值大于等于 j 的概率可以限定为 $P(c(i) \ge j) \le C_n^j (1/m)^j$. 根据斯特林公式 $n! \approx \sqrt{2\pi n}(n/e)^n$, 可以得到 $P(c(i) \ge j) \le C_n^j (1/m)^j \le (en/(jm))^j (1/\sqrt{2\pi j})$.

上文中得出, 当 SCBF 达到最大利用率时, 存储的元素个数 n 可以表示为 $n \approx m \ln 2$, 所以得到最终结果 $P(\max(c(i)) \ge j) \le (e \ln 2/j)^j (1/\sqrt{2\pi j})$. 可以看到, 经过这样处理后, 溢出概率不再和 n、m 直接相关. 所以本文得出如下结论, 对应于 SCBF 最优的情况, 溢出概率不再和 SCBF 表的分片长度以及要存储的元素个数直接相关, 而只是和计数位数相关. 计算最优情况下, 即当 $f = 1/2$ 时, 分片计数布隆过滤器的溢出概率, 得到表 2 的结果.

表 2　最优情况下分片计数布隆过滤器的溢出概率

计数位数	j	$P(\max(c(i)) \ge j)$
1	2	0.03388
2	4	1.80E-4
3	8	4.48E-10
4	16	1.54E-23
5	32	3.89E-55

从表 2 中可以看出, 采用 4 位计数已经可以满足绝大部分的应用要求. 可是由于不可能总是预先确定要存储的元素个数, 如果一开始就建立一个可以存储大量元素的 SCBF 将会造成不必要的浪费. 解决这个问题的思路是先建立一个存储少量元素的 SCBF, 然后根据实际的情况, 后续动态添加更多的 SCBF.

3.4 可扩展的分片计数布隆过滤器

解决上述问题的方法是建立可扩展分片计数布隆过滤器. 具体方案是针对每一个 region server 建立可扩展分片计数布隆过滤器, 该可扩展分片计数布隆过滤器包含一个或多个完全相同的分片计数布隆过滤器, 首先建立一个小尺寸的 SCBF, 随着 SCBF 中插入元素的增加, 当 $f = 1/2$ 即填充率达到一半的时候增加一个新的 SCBF. 以此类推, 当可扩展分片计数布隆过滤器中所有的 SCBF 的填充率都达到一半的时候继续增加新的 SCBF. 下面具体探讨应该增加多少个 SCBF 使得分片计数布隆过滤器达到最优.

假设可扩展分片计数布隆过滤器中包含 d 个 SCBF, 则错误率为 $P_d = 1 - (1 - P_{SCBF})^d$, 其中, P_{SCBF} 为单个 SCBF 的错误率; d 为 SCBF 的数量. 由上文的分析可以知道, 在 M 和 P_{SCBF} 给定的情况下, 可以设置单个 SCBF 的其他参数使它达到最优. 由错误率表达式可以得出, 当 P_{SCBF} 一定时, 错误率 P_d 随着 d 的增大而增大. 所以当给定 M、P_{SCBF} 和 P_d 时, 可以确定最优情况下可扩展分片计数布隆过滤器所包含的 SCBF 的数量.

当查询元素是否存储在某个 region sever 中, 查询该 region server 所对应的可扩展分片计数布隆过滤器中所有的 SCBF, 如果该元素不属于其中任何一个 SCBF, 则可以确定该元素没有存储在该 region server 中, 但是如果该元素属于其中的一个或多个 SCBF, 则就认为该元素存储在该 region server 中. 删除元素时先删除该元素, 然后再删除其对应的索引. 插入元素时, 找到一个填充率还没有达到一半的 SCBF 进行插入操作.

针对 Hbase 仅支持主索引结构的不足, 提出利用 Counting Bloom Filter 的新变体建立二级索引来支持非主键数据的检索. 针对 Counting Bloom Filter 技术溢出概率高的问题, 本文提出一种 SCBF 技术, 详细讨论了 SCBF 的原理和实现细节, 通过与 CBF 的比较, 证明 SCBF 显著降低了溢出概率, 充分提高了过滤器性能. 此外, 针对 Hbase 的非主键数据的检索问题, 提出 SCBF + hindex 结构的系统实现方案, 通过层次索引来快速定位要检索的数据.以后的研究工作还可以对 SCBF 做进一步的改进, 使其在错误率和负载均衡上有大的突破.

4　SCBF与CBF的溢出概率比较

假设要表示的集合 S 有 n 个元素, SCBF 的位数组长度为 M, 错误率为 P, 哈希函数的个数 k 使用最优值 $k = \log_2(1/P)$, 每一片的位数为 $m = M/k$. 由上文可得 SCBF 的溢出概率为

$$P(\max(c(i)) \ge j) \le (e \ln 2/j)^j (1/\sqrt{2\pi j}),$$

其中 i 表示某个 Counter, $\max(c(i))$ 表示所有 Counter 的最大值, j 表示 Counter 增加的次数. 计算最优情况下, 即当 $f = 1/2$ 时, SCBF 的溢出概率, 得到表 3 的结果.

表 3　最优情况下分片计数布隆过滤器的溢出概率

计数位数	$P(\max(c(i)) \ge j)$
1	0.03388
2	1.80E-4
3	4.48E-10
4	1.54E-23

假设对于同样 n 个元素的集合 S, CBF 的位数组长度为 m, 哈希函数的个数 k 使用最优值 $k=\ln 2(m/n)$, 错误率为 $P=1/2^k$. 文献[10]中得出 CBF 的溢出概率为

$$P(\max(i)\succ c)\approx 1-e^{-\lambda}\sum_{j=0}^{c}(\lambda^j/j!),\lambda=nk/m,$$

其中 i 为实际计数值, c 为最大不溢出计数值, j 表示 Counter 增加的次数. 最优情况下, 即 $m=nk/\ln 2$, $\lambda=\ln 2$ 时, 计算标准 CBF 的溢出概率得到表 4 所示的结果[10].

表 4 最优情况下标准计数布隆过滤器的溢出概率

计数位数	$P(\max(c(i))\geq j)$
1	0.1534
2	5.56E-3
3	7.15E-7
4	2.22E-16

比较表 3 和表 4 的结果可以发现, 当 SCBF 和 CBF 分别达到其最优情况时, SCBF 的溢出概率低于 CBF, 而且随着计数位数的增加, 两者的溢出概率差距越来越大. 对于计数 Bloom Filter 而言, 计数位数决定了计数 Bloom Filter 的性能, 所以本文完全可以说 SCBF 的性能优于 CBF.

5 结语

针对 Hbase 仅支持主索引结构的不足, 提出利用 Counting Bloom Filter 的新变体建立二级索引来支持非主键数据的检索. 针对 Counting Bloom Filter 技术溢出概率高的问题, 本文提出一种 SCBF 技术, 详细讨论了 SCBF 的原理和实现细节, 通过与 CBF 的比较, 证明 SCBF 显著降低了溢出概率, 充分提高了过滤器性能. 此外, 针对 Hbase 的非主键数据的检索问题, 提出 SCBF + hindex 结构的系统实现方案, 通过层次索引来快速定位要检索的数据.以后的研究工作还可以对 SCBF 做进一步的改进, 使其在错误率和负载均衡上有大的突破.

参考文献

1 Burton HB. Space/time trade-offs in hash coding with allowable errors. Communications of the ACM, 1970, 13(7): 422–426.

2 Fan L, Cao P, Almeida J, et al. Summary cache: A scalable wide-area web cache sharing protocol. IEEE/ACM Trans. on Networking (TON), 2000, 8(3): 281–293.

3 Bonomi F, Mitzenmacher M, Panigrahy R, et al. An improved construction for counting bloom filters. Algorithms-ESA 2006 Lecture Notes in Computer Science, 2006. Zurich: Springer Berlin Heidelberg. 2006. 684–695.

4 张震,汪斌强,陈庶樵,等.几何布鲁姆过滤器的设计与分析.电子学报,2012,40(9):1852–1857.

5 李珺,刘晓光,王刚,等.K 分组合型 Bloom Filter 方法的设计.计算机研究与发展,2008,45:48–52.

6 孙智超,徐蕾.二路平衡动态布隆过滤器.数学的实践与认识,2014,44(5):199–205.

7 侯颖,郭云飞,黄海,等.基于同源组合布鲁姆过滤器的早期流量抽样算法.通信学报,2014,35(10):117–126.

8 Aguilar-Saborit J, Trancoso P, Muntes-Ulero V. Dynamic count filters. New York. ACM, 2006: 26–32.

9 Meng J. Partial Bloom Filter. http://blog.csdn.net/jiaomeng/article/details/1502910. [2015-04-13].

10 魏静波,蒋平,朱劲.计数型 Bloom Filter 及其在机器人导航中的应用.微计算机信息,2008,24(35):241–243.

11 王键.d-Left CBF 技术在 P2P 中的研究.计算机工程与设计,2008,29(7):1711–1713.

12 严华云,关佶红.Bloom Filter 研究进展.电信科学,2010,26(2):31–35.

13 Paulo SA, Carlos B, Nuno P, et al. Scalable bloom filters. Information Processing Letters, 2007, 101(6): 255–261.

14 Cheng X, Li HY, Wang Y, et al. BF-matrix: A secondary index for the cloud storage. In: Li FF, Li GL, eds. Web-Age Information Management: Lecture Notes in Computer Science. Macau: Springer International Publishing, 2014: 384–396.

15 严华云,关佶红.基于 Bloom 滤波器的对等网多关键字检索.计算机应用,2010,30(9):2335–2443.

16 张华,郑世珏,童德茂.Bloom Filter 技术及应用.阜阳师范学院学报(自然科学版),2014,31(3):62–66.