**海量数据排重服务**

**mc\_bloom\_filter项目设计文档**

作者：何跃 (heyue1@staff.sina.com.cn) (分机5898 2375) 186 1081 2028 时间：2012年09月10日

本文档修订历史：

|  |  |  |
| --- | --- | --- |
| **版本号** | **修订人** | **修订说明** |
| 1.0 2012-09-10 | 何跃 | 创建mc bloom filter 过滤器的第一个版本 |
| 1.1 2012-09-21 | 何跃 | 更新mc bloom filter的详细设计 |
| 1.2 2012-10-15 | 汤晓刚 | 根据实际开发情况补充文档； |
| 1.3 2012-10-15 | 胡鸿 | 更新 Hash算法选择 |
| 1.4 2012-10-19 | 何跃 | 更新详细的文档 |
| 1.5 2012-11-08 | 何跃 | 多线程版本发布，添加多线程部分 |

目录

[1 bloom filter的使用场景和原理 2](#_Toc338667965)

[1.1 Bloom Filter原理及使用场景 2](#_Toc338667966)

[1.2 Bloom filter相比mc、redis等做排重的优势和缺点 2](#_Toc338667967)

[1.3 在我们产品线上可能用到的地方 2](#_Toc338667968)

[1.3.1 排重 2](#_Toc338667969)

[1.3.2 我们的产品中，用来判断用户的首次操作 3](#_Toc338667970)

[1.3.3 其他的场景，请搜索“bloom filter 使用场景” 3](#_Toc338667971)

[2 我们的Mc bloom filter 的设计和详细实现 3](#_Toc338667972)

[2.1 设计思想 3](#_Toc338667973)

[2.2 定义下文的几个术语 3](#_Toc338667974)

[2.3 网络协议、内存设计、命令设计 3](#_Toc338667975)

[2.4 Hash算法选择 5](#_Toc338667976)

[2.5 存储、日志设计 5](#_Toc338667977)

[2.6 运维设计（同memcached） 5](#_Toc338667978)

[2.7 使用详细说明和运维（内存预测、评估等） 5](#_Toc338667979)

[2.7.1 Bloom filter的安装 5](#_Toc338667980)

[2.7.2 Bloom filter的安装 6](#_Toc338667981)

[2.7.3 内存评估 7](#_Toc338667982)

[3 Mc bloom filter 的开发计划 7](#_Toc338667983)

# bloom filter的使用场景和原理

## Bloom Filter原理及使用场景

**Bloom filter 的简介：**

|  |  |
| --- | --- |
| 百度百科： | <http://baike.baidu.com/view/1912944.htm> |
| **Google黑板报:** | <http://www.google.com.hk/ggblog/googlechinablog/2007/07/bloom-filter_7469.html> |
| 算法详细介绍：  | <http://blog.csdn.net/v_july_v/article/details/6685894> |

## Bloom filter相比mc、redis等做排重的优势和缺点

**优点：**

1. bloom filter不存储字符串本身，需要的内存极少，不因为字符串变长，而需要大量的内存。
2. bloom filter速度很快，时间复杂度x倍的O(1)，x是hash函数查找的次数，x一般是1~20以内。
3. bloom filter大小一旦确定，内存就不会随业务的增长而增加。

做10亿URL的排重，每个URL长度平均50字节

|  |  |  |
| --- | --- | --- |
| Mc | 不计算其他数据结构，只计算这个字符串 | 把url做md5再排重 |
|  | **50\*10000000000 = 50G** | **32\*1000000000 = 32G** |
|  |  |  |
| Bloom filter（开辟10亿空间） | 误判率0.00001 | 误判率0.0001 |
|  | **2.7G** | **2.3G** |

**缺点：**

1. 不能删除项
2. 有误判率，可以判断一个元素肯定不在这个容器中，但是没法判断一个原始一定在这个容器中（也就是说，在判断某个元素一定在容器中的时候，有误判）。
3. 一旦创建一个bloomfilter之后，扩容就只能新建一个更大的bloom filter，无法扩容。

## 在我们产品线上可能用到的地方

### 排重

1. url排重（比如爬虫爬URL的时候）
2. 昵称不能重复的排重（比如微博昵称，某个昵称是否已经被用了，哪个昵称还能用）
3. 垃圾邮件地址的排重（把所有的垃圾邮箱全放bloom filter中，就可以实现单个邮箱反垃圾）
4. 拼写检查（把常用的词语组合放入bloom filter，如果来了一个组合不对，就可能是拼写错误）
5. Google的BigTable也使用了Bloom Filter，以减少不存在的行或列在磁盘上的查询，大大提高了数据库的查询操作的性能。

### 我们的产品中，用来判断用户的首次操作

1. 比如用户是否访问过某个TAG或者用户是否访问过某页面。
2. 在产品新功能发布的时候，会希望用户在第一次使用这个功能的时候显示引导，显示一次后再次访问则不再提示；

我们则可以使用bloomfilter建立访问过该新功能的用户过滤器，每当用户访问新功能的时候先判断用户uid是否在这个过滤器中，如果在，则表明用户之前访问过，如果不在，则说明用户是第一次访问，并将这个uid加入到bloom过滤器中；

### 其他的场景，请搜索“bloom filter 使用场景”

# 我们的Mc bloom filter 的设计和详细实现

## 设计思想

* 只要内存够，可以申请几十G的内存来做bloom filter（没有上百G的机器，本人只测试过20多G的，理想容量100亿）
* 允许一个实例下面存在多个bloom filter
* 可以随意的增、删、查一个bloom filter，每创建一个bloom filter，申请一块内存；每删除一个bloom filter，释放这块内存。
* 采用memcache的网络层，用mc的get、set、add、delete stats等命令
* 可以随时查看bloom filter的运行状态

## 定义下文的几个术语

|  |  |
| --- | --- |
| **key** | **是指bloom filter的名字，32字节内，不能有空格和|** |
| **subkey** | **是指bloom filter需要排重的内容，比如需要排重的url、邮箱、昵称等，不能有空格** |
| **|** | **竖线是一个申请的设计，我们用它来做我们的分割符号，下面有它的详细说明** |

## 网络协议、内存设计、命令设计

采用mc协议，可以允许一个实例中有多个bloom filter。命令使用add、set、get、status命令。

**为啥采用mc协议：**

 从我们的需求上看，mc协议比较简单，我们需要的命令刚好和mc的原有的命令相同，而且对现有应用比较友好，目前所有平台都能找到对应的mc客户端程序，所以可以通过mc协议无缝扩展到bloomfilter的支持。

|  |  |
| --- | --- |
| add 命令 | 创建一个bloom filter |
| set 命令 | 向一个bloom filter中添加某个值 |
| get 命令 | 判断这个值是否在某个bloom filter中 |
| delete 命令 | 删除一个bloom filter |
| stats | Bloom filter的状态 |
| stats blooms | 所有bloom 的状态 |
| stats bloom key | 查看某个bloom filter的状况 |

**采用mc协议的风险和我们不得不做的妥协：**

 Mc的key只能是250字节，这个从原理上限制了这个长度的问题，这个得研究mc的客户端是否限制了key的长度。

|  |  |
| --- | --- |
| **key** | **32字节，不能有“|”和“ ”空格** |
| **subkey** | **217字节，不能有“ ”空格** |

 为啥会有这么多限制？首先是mc 的get命令，限制了长度是250，在php、c等客户端的实现上，也控制了mc的key的长度。

 在我们改写的mc的命令中，采用**“get key|subkey”**的方式，这两个字符串长度加起来不能超过250，而bloom filter的key最长不能超过32个字节，所以subkey的长度也不能超过217字节。

**内存设计：**

 采用单实例，多bloom filter的设计，启动参数可以设定进程所占用内存最大限制，启动后可以使用setmem命令修改该限制大小。

命令列表：

|  |  |  |
| --- | --- | --- |
| add | add key 0 0 value\_lengthexpected\_max\_amount\_of\_elements|false\_positive\_rate | 成功返回STORED失败返回NOT\_STORED |
| set  | set key 0 0 value\_lengthsubkey | 成功返回STORED失败返回NOT\_STORED |
| get | get key|subkey | 存在返回1 |
| stats | **stats** 查看服务器的总体状况**stats blooms**列举所有过滤器的名称和占用内存字节大小**stats bloom key**可以查看名字为key的bloom filter的详细信息 |  |
| try | **try expected\_max\_amount\_of\_elements|false\_positive\_rate****比如 try 100000000|0.0001 表示计算1亿个目标存储数，在误判率万分之一的情况下，需要的内存大小**用来预估过滤器所需的内存大小和hash函数个数 |  |
| setmem | **setmem size(Mbytes)**用来设定当前进程可使用的内存容量，单位是m,比如要设置内存1G，setmem 1024 | 成功返回STORED |

## Hash算法选择

作为bloom\_filter的核心，hash算法的选择必须强调其hash函数的独立性和均匀分布特性，并且运算起来尽可能的快。所以广泛使用的加密算法md5，sha1等，在这里并不是个非常好的选择。

murmurhash是 Austin Appleby于2008年创立的一种非加密hash算法，运算简单高效，抗碰撞，出色的雪崩效应。最新版本是MurMurHash3，支持32位、64位及128位值的产生。

murmurhash在多个开源项目中得到应用，包括libstdc、libmemcached、nginx、hadoop等。

Code source: <http://code.google.com/p/smhasher/>

 Bloom filter需要很多个hash函数来计算hash值，然后置位，但是市场上的开源的64位的hash算法，远远达不到我们需要的个数（10个左右）。

我们采用murmur 计算出一个128位的hash值，然后拆成两个hash，做为源hash值。

然后采用哈佛大学的一个bloom filter的论文，把原本需要几十个hash函数的bloom filter转化成了两个hash值的运算，i是hash函数的个数，需要多少个hash值，就运算这个多少遍。



## 存储、日志设计

此版本中，无服务器日志，不持久化，所以需要客户端自己打日志，系统出现异常，需要客户端跑程序塞数据。

## 运维设计（同memcached）

## 使用详细说明和运维（内存预测、评估等）

### Bloom filter的安装

####  Bloom filter的依赖

Bloom filter使用memcache的网络层，所以依赖于libevent，在<http://libevent.org/> 上下载libevent最新稳定版本。

|  |
| --- |
| wget <https://github.com/downloads/libevent/libevent/libevent-2.0.20-stable.tar.gz>tar zxvf libevent-2.0.20-stable.tar.gzcd libevent-2.0.20-stable./configuremake && make install |

### Bloom filter的安装

bloom fitler的google code的地址，只上传测试稳定版本

从<http://code.google.com/p/mc-bloom-filter/> 下载最新稳定版本

|  |
| --- |
| 1.wget bloom filter的最新稳定版本2.修改Makefile文件，主要是修改libevent到你的目录3.在目录中执行make,生成mc\_bloom\_filter【线上版】 mc\_bloom\_filter\_【调试版】 两个可执行文件，调试版会打很多日志4. nohup ./mc\_bloom\_filter -p12345 -d -uroot -m4000 –p/tmp/mc\_bloom\_filter.pid –l127.0.0.1日志文件就是当前目录的nohup.out文件 |

启动参数的意义

|  |  |  |
| --- | --- | --- |
| 参数 | 是否必须 | 值的含义 |
| p(小p) | 否 | 默认12345 |
| P(大P) | 是 | pid 文件的地址(需要绝对路径) |
| u(小u) | 是 | 用哪个用户启动 |
| m(小m) | 是 | 最大内存限制 |
| d(小d) | 是 | 是否daemon后台运行 |
| l(小l) | 是 | 绑定的ip |
| t | 否 | 只有在多线程版本里面才有这个参数，单线程版本没有这个参数，默认是4 |

Telnet测试运行

|  |
| --- |
| **telnet 127.0.0.1 12345****add test 0 0 13****1000000|0.001****stats blooms****stats bloom test****set test 0 0 3****test\_subkey****get test|test\_subkey** |

php脚本测试

|  |
| --- |
| $mc = new Memcache();$mc -> connect('127.0.0.1','12345');$res = $mc -> add("test\_php","1000000|0.0001");if(!$res){ die(“create error”);}$res = $mc -> set("test\_php","test\_subkey");if(!$res){ die(“can not set”);}var\_dump($mc->get("test\_php|test\_subkey")); |

C语言的测试脚本，在test目录下有测试脚本。

### 内存评估

请用bloom filter的try 命令进行评估

|  |
| --- |
| try 100000000|0.00001 表示计算一亿个目标存储量，误判率在十万分之一下，计算需要的内存大小。输出样例：need\_memory 300000128(Bytes) 286.102(M) use\_function\_num 17 false\_positive\_rate 0.000010 |

# Mc bloom filter 的开发进展

2012-09-18 ~ 2012-09-21 基本代码的搭建；

2012-09-25 ~ 2012-09-29 原型系统开发完成；

2012-10-08 ~ 2012-10-12 Alpha测试、优化、文档完善；

2012-10-13 ~ 2012-10-22 Beta测试、优化、文档完善；

2012-10-23 ~ 2012-11-08 多线程版本的开发、发布