公司黑客松 - 实时发号系统方案

作者：GunShotPanda
原文链接：https://ld246.com/article/1565314772198
来源网站：链滴

背景

- 公司每日新增设备百万级 周新增接近亿级
- 要给每一个设备都发一个唯一id且需要一个合适的偏移量避免漏号和发重
- 每个id对应唯一且最大的offset

ID格式：

UUID+一位类型

场景描述

每天最大十亿级的日志，日活亿级的设备数，其中百万级的新增

比赛要求

业务：

目前ID量90多亿，每天日活2亿左右，每天新增600万左右。
设计实时发号系统。其中id长33位（1magicNumber + 32位MD5），offset整数、依次递增。

【服务的功能性要求】：

- 输入id，输出offset
- QPS = 600 w /(12 * 60 * 60) = 138

参考资料:

1.

https://tech.meituan.com/2017/04/21/mt-leaf.html

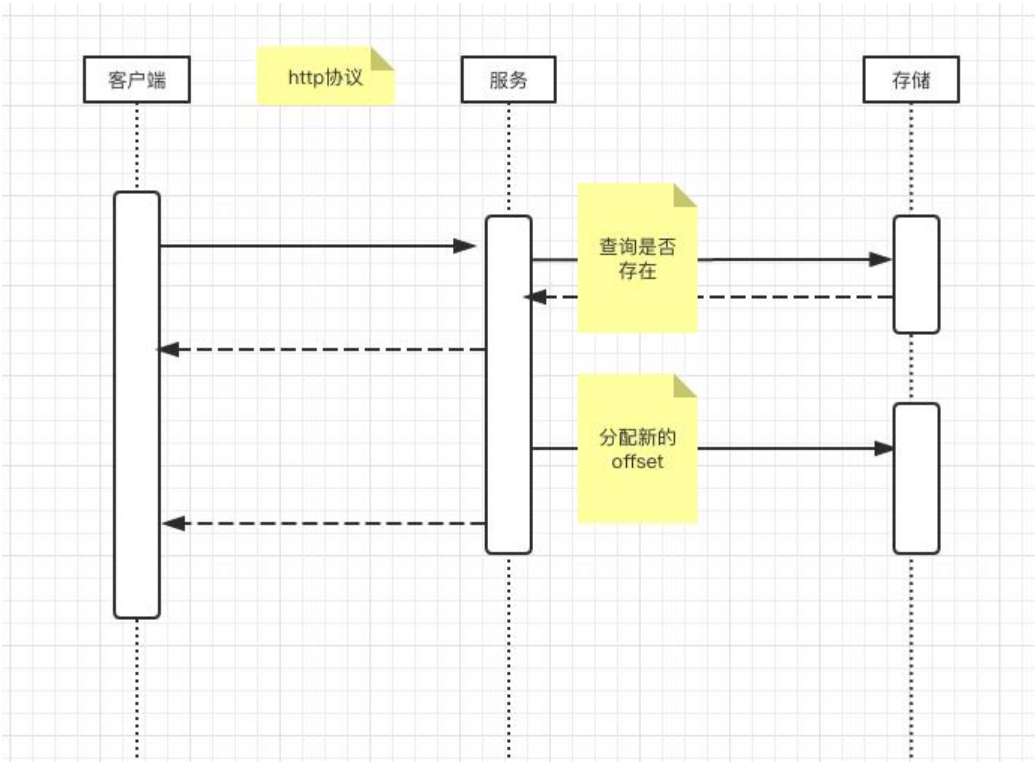
2. EWAHCompressedBitmap -google

3. javaEWAH: https://github.com/lemire/javaewah

4. https://www.chainnews.com/articles/555869220376.htm id查询服务,作为其中一个环节

5. 雪花算法 https://segmentfault.com/a/1190000011282426?utm_source=tag-newest

整体流程:



分工:

模块	负责人	功能性需求	非功能性需求	技术选型	风险点	备注
测试数据生成，验证程序	我	1.测试程序： 2.测试数据： 3.覆盖场景：测试不会分配不同offset 并发不会分配两个offset 4.得出性能指标：		wrk，siege	这种赛制下,演示决定最终的效果(可能)	
服务	大佬A	1.(批量实时)查询offset+生成offset 2.(批量离线)支持将已有offset导入进当前系统 3.(批量离线)将所有tdid导入	负载均衡 可用性 一致性 可扩展 并发不分配两个offset			
db	大佬B	自增id tdid->id映射	记录使用情况,最近使用情况	热点数据内存存储 集群		

技术调研记录:

1.测试数据生成，验证程序

```
import java.io.*;
import java.util.UUID;

public class UUIDGenerator {
    private String getId() {
        UUID uuid = UUID.randomUUID();
        return uuid.toString().replace("-", "");
    }
    public static void main(String[] args) {
        File f = new File("path");
        long startTime = System.currentTimeMillis();
        Writer out = null;
        try {
            int i = 100000000;
            BufferedWriter bw = new BufferedWriter(new FileWriter("path"));
            bw.write(i);
            bw.newLine();
            for (i = 1; i <= 100000000; i++) {
                String uid = "h" + new UUIDGenerator().getId();
                bw.write(uid);
                bw.newLine();
            }
            bw.flush();
            bw.close();
        } catch (IOException e) {
        e.printStackTrace();
        }
        long endTime = System.currentTimeMillis();
        System.out.println("time: " + (endTime - startTime) / 1000 + " s");
    }
}
```

http 性能测试

wrk

https://juejin.im/post/5a59e74f5188257353008fea

lua自定义压测用例，包括数据量，新旧数据，重复数据比例

Siege

一款开源的压力测试工具，可以根据配置对一个WEB站点进行多用户的并发访问，记录每个用户所有求过程的相应时间，并在一定数量的并发访问下重复进行。

Siege官方：http://www.joedog.org/

Siege下载：http://www.joedog.org/pub/siege/siege-latest.tar.gz

Siege解压并安装：

tar -zxvf siege-latest.tar.gz
cd siege-latest/
./configure
make
make install

Siege使用:
siege -c 100 -r 10 -f site.url
-c是并发量，-r是重复次数。
url文件就是一个文本，每行都是一个url，它会从里面随机访问的。

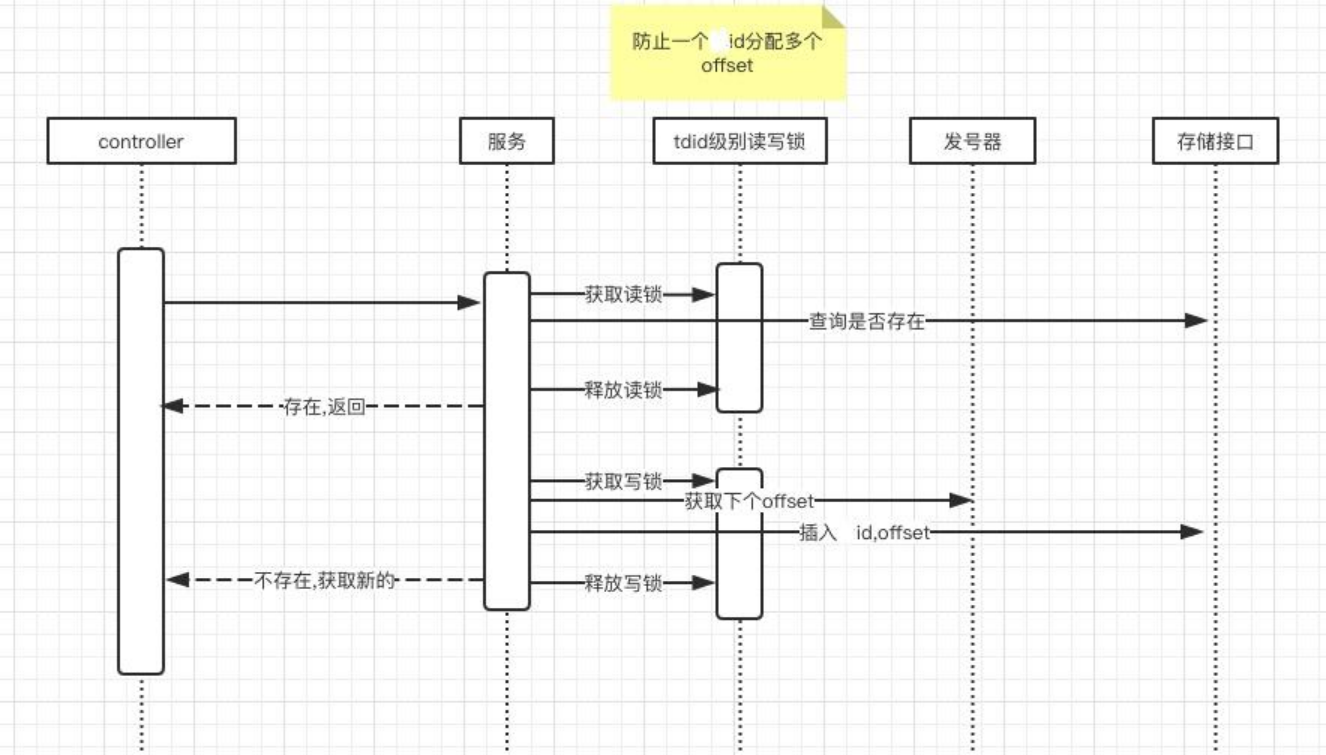
site.url内容:
http://www.qixing318.com/
http://www.zendsns.com/
http://www.qixing.info/

Transactions: 550 hits // 完成550次处理
Availability: 55.00 % // 55.00 % 成功率
Elapsed time: 31.32 secs // 总共用时
Data transferred: 1.15 MB // 共数据传输1.15 MB
Response time: 3.04 secs // 显示网络连接的速度
Transaction rate: 17.56 trans/sec // 均每秒完成 17.56 次处理：表示服务器后
Throughput: 0.04 MB/sec // 平均每秒传送数据
Concurrency: 53.44 // 实际最高并发数
Successful transactions: 433 // 成功处理次数
Failed transactions: 450 // 失败处理次数
Longest transaction: 15.50 // 每次传输所花最长时间
Shortest transaction: 0.42 // 每次传输所花最短时间

2.服务

主流程:

根据id查询旧的offset
存在,直接返回
不存在,分配新的offset
获取新的offset
保存id和offset的映射关系

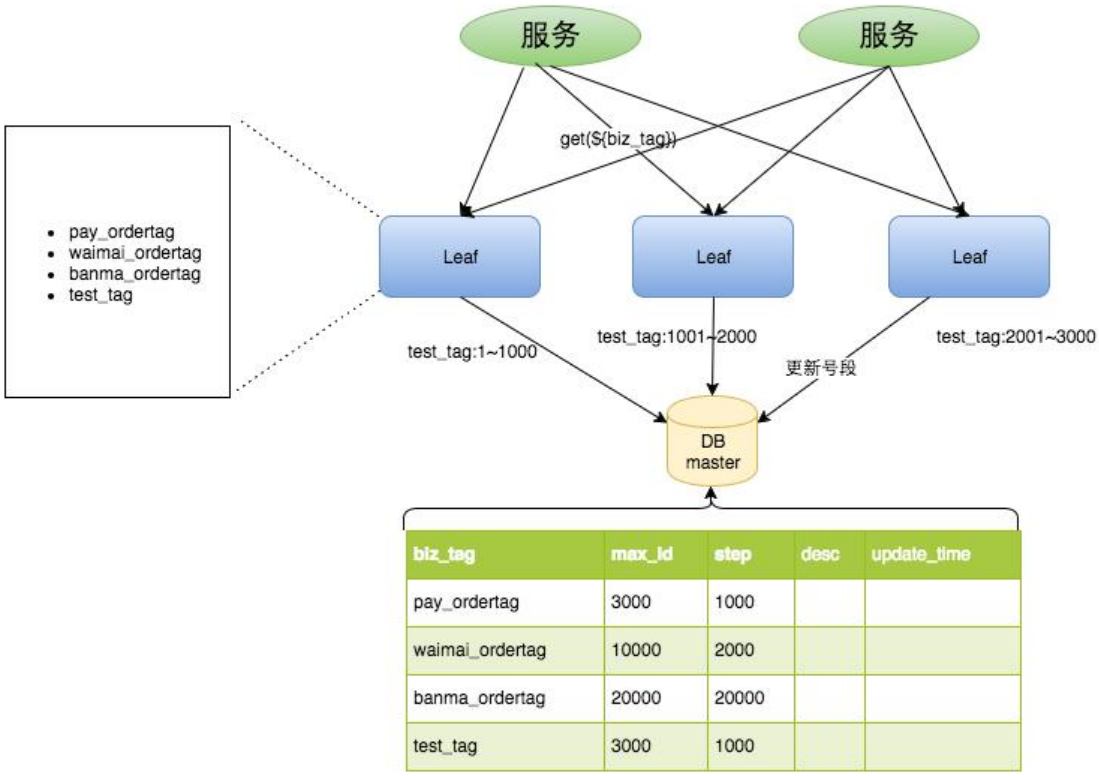


1.发号器:号段-减少db压力

https://tech.meituan.com/2017/04/21/mt-leaf.html

每次获取一个segment(step决定大小)号段的值。用完之后再去数据库获取新的号段，可以大大的减数据库的压力。

原来获取ID每次都需要写数据库，现在只需要把step设置得足够大，比如1000。那么只有当1000个被消耗完了之后才会去重新读写一次数据库。读写数据库的频率从1减小到了1/step，大致架构如下所示：



biz_tag	max_id	step	desc	update_time
pay_ordertag	3000	1000		
waimai_ordertag	10000	2000		
banma_ordertag	20000	20000		
test_tag	3000	1000		

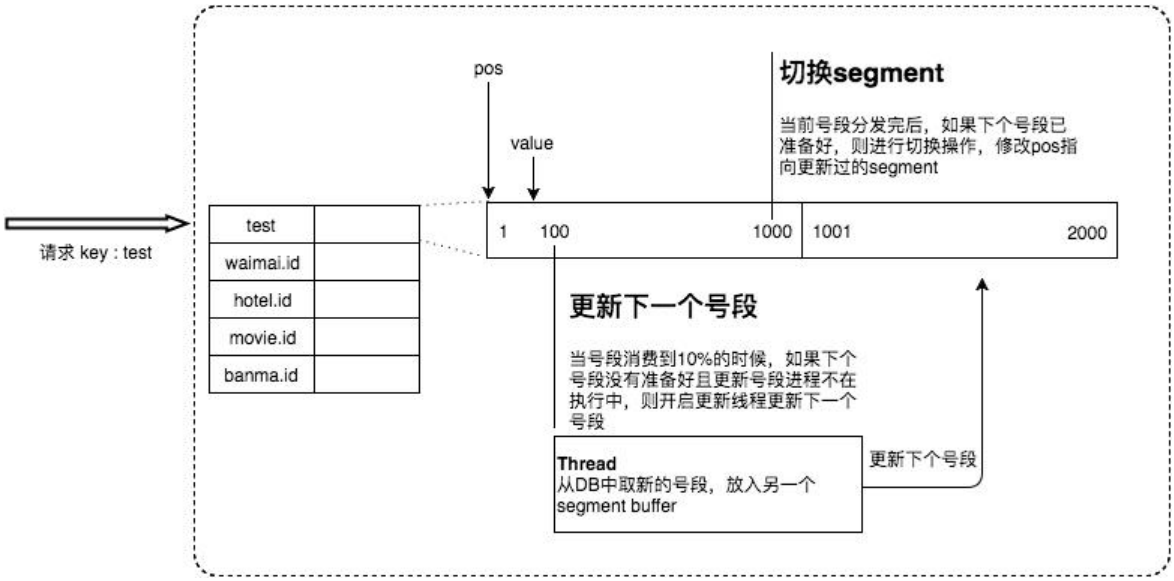
原文链接：公司黑客松 - 实时发号系统方案

2.发号器:双buffer优化

对于第二个缺点，Leaf-segment做了一些优化，简单的说就是：

Leaf 取号段的时机是在号段消耗完的时候进行的，也就意味着号段临界点的ID下发时间取决于下一次DB取回号段的时间，并且在这期间进来的请求也会因为DB号段没有取回来，导致线程阻塞。如果请DB的网络和DB的性能稳定，这种情况对系统的影响是不大的，但是假如取DB的时候网络发生抖动，者DB发生慢查询就会导致整个系统的响应时间变慢。

为此，我们希望DB取号段的过程能够做到无阻塞，不需要在DB取号段的时候阻塞请求线程，即当号消费到某个点时就异步的把下一个号段加载到内存中。而不需要等到号段用尽的时候才去更新号段。样做就可以很大程度上的降低系统的TP999指标。详细实现如下图所示：

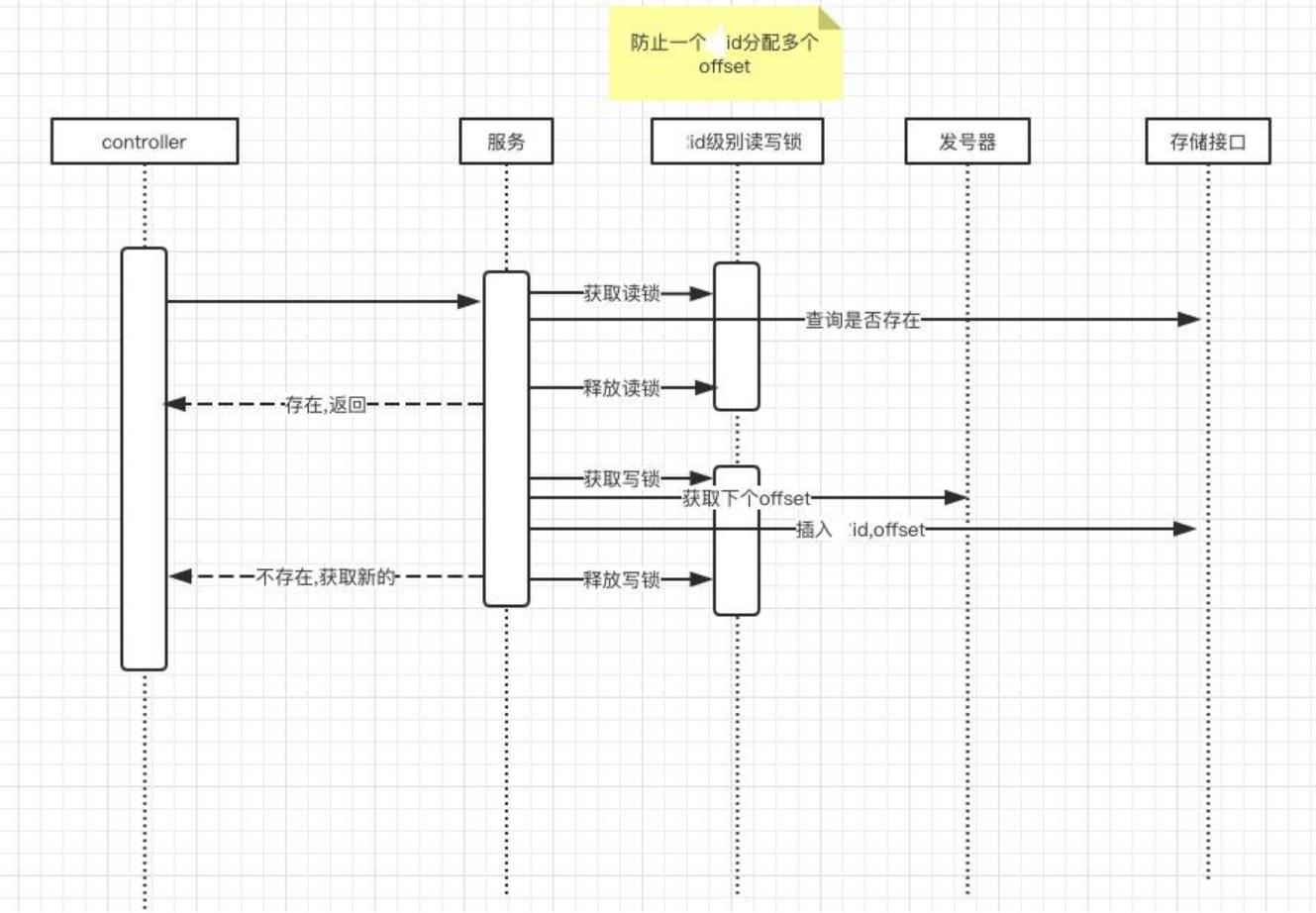


3.Nginx:配置url_hash负载均衡策略

https://www.imooc.com/article/19980

4.读写锁:

其中,Nginx配置url_hash负载均衡策略+tdid级别读写锁,保证数据严格一对一



5.依赖的存储层接口

1. 根据id查询offset
2. 获取当前值,并自增指定步幅(保证原子性)
3. 设置id,offset

3.存储

1. 大规模数据判重：

redis modules插件BoolmFilter https://github.com/RedisBloom/RedisBloom

api: https://oss.redislabs.com/redisbloom/Quick_Start

2. tdid-> offset 映射：2亿存储 10G数据。 60亿 300G数据

ssdb

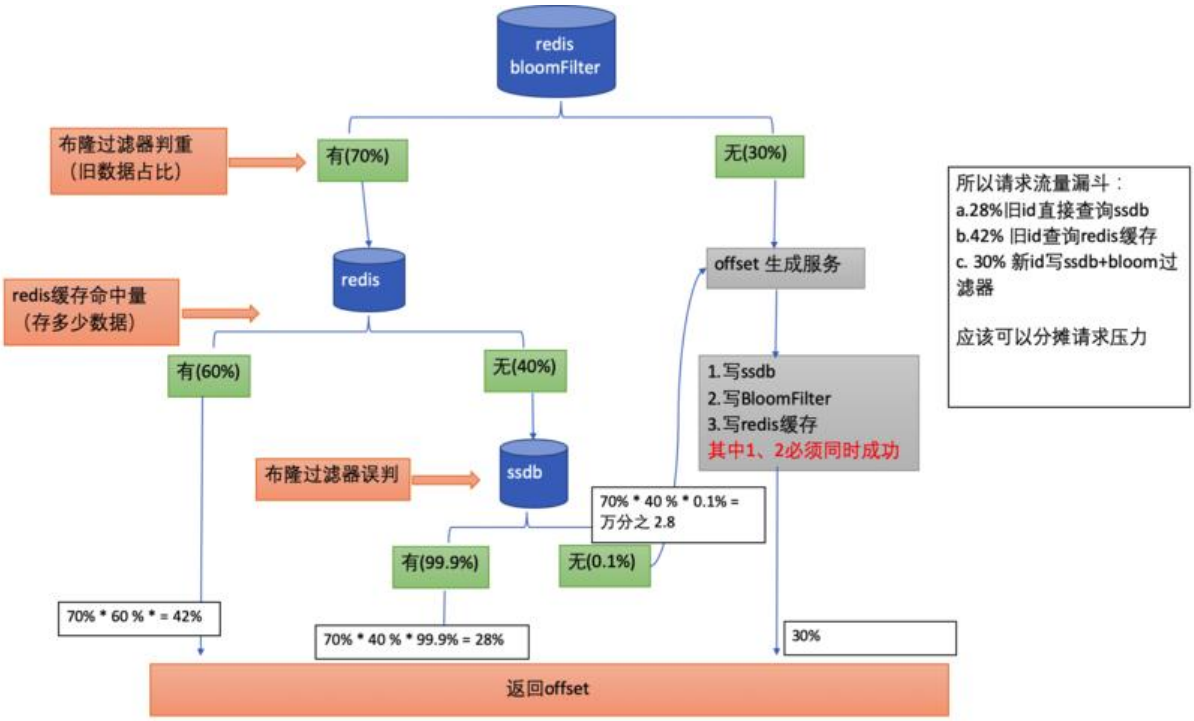
考虑映射关系读写很频繁，计划使用nosql 数据库 ssdb, 是对leveldb存储引擎的redis兼容协议封装其性能可和redis比肩,有待验证效率。而且可以把存储不下的持久化到硬盘，解决Redis容量有限问题。

官网：http://ssdb.io/zh_cn/

api: http://ssdb.io/docs/zh_cn/commands/index.html

别人的读写测试： https://www.cnblogs.com/lulu/p/4231810.html

1. 读写逻辑



测试场景:

每天最大60亿日的日志，日活2亿的设备数，其中600w是新增

100%新数据(发号)

0%数据(查询)

极端情况:并发新增

性能测试

1.bloom Filter

1000w测试， 325M ， bloom配置（容量1000w, 0.001%错误率）,内存占用33M

a.写 395700ms = 395s，插入 QPS = 1000W /400 = 2.5W/s

1亿 4000s = 66.7min， 平均一亿一小时。 够了，每天新增用户才500W-600W，最快半小时插入布过滤器，而且请求不会很集中

b.正向读（存在数据） 378456ms = 378s = 6.3min ， QPS = 1000w/400 = 2.5w/s

c.反向读（不存在数据）402886ms = 400s，QPS = 1000w/400 = 2.5w/s

1000w测试， 325M ， bloom配置（容量1000w, 0.1%错误率）,内存占用11.4M

读写速率和上面类似

1000w测试， 325M ， bloom配置（容量1亿，0.1%错误率）,内存占用63M

1000w测试， 325M ， bloom配置（容量10亿，0.1%错误率）,内存占用1000M

1000000000 =10亿

1000w测试， 325M ， bloom配置（容量20亿，0.1%错误率）,内存占用2000M

20亿已经很大了，再大需要拆分过滤器，取 hash % number = filter index

100亿需要4个过滤器 = 4 * 2 = 8G 内存

结论

1.相同数据，误判率越低，占用内存越大;

2.相同数据，bloomFilter容量越大，占用内存越大，且容量在初始化时已经确定，add数据不会改变;

3.bloomFilter容量有上线，30亿左右，再大的量需要多个过滤器分片了。

4.读写速度基本和容量、错误率无关。

5.读写QPS均为 2.5w/s ，关键优势内存占用很少。

2.ssdb

单线程，读写速度相似。 1000w数据 325M ，内存占用几十M，空间占用100M ,leveldb有压缩

写 120000ms = 1200s = 20 min QPS = 0.83W/s

读 123500ms = 1235s = 20.6 min QPS = 0.8W/s

3.redis

1000w 测试

写 1021980 ms = 1022s QPS = 0.97W/s

正向读 949310 ms =949s QPS =1.05W/s

反向读 935231 ms =935s QPS = 1.07W/s

结论

性能比较ssdb提升不大，缓存作用不明显。

联调性能

测试要点

！！！！每次测试开始必须保证全新环境！！！！！

1.初始化新的布隆过滤器

1. 命令行 BF.RESERVE {bloom_key} {error_rate} {capacity} 误差率是小数，不是百分比

Parameters:¶

- **key**: The key under which the filter is to be found
- **error_rate**: The desired probability for false positives. This should be a decimal value betwe n 0 and 1. For example, for a desired false positive rate of 0.1% (1 in 1000), error_rate should e set to 0.001. The closer this number is to zero, the greater the memory consumption per it m and the more CPU usage per operation.
- **capacity**: The number of entries you intend to add to the filter. Performance will begin to d

grade after adding more items than this number. The actual degradation will depend on how ar the limit has been exceeded. Performance will degrade linearly as the number of entries gr w exponentially.

1. 客户端, 需要maven引入

```
<dependency>
   <groupId>com.redislabs</groupId>
   <artifactId>jrebloom</artifactId>
   <version>1.2.0</version>
</dependency>
```

Client client = new Client("localhost", 6379);

client.createFilter("Filter3", 10_000_000, 0.1);　过滤器名称，容量，误差率（百分比值）

2.ssdb flushdb清空存储

测试场景

初始化:

想ssdb中插入2亿条数据

正常测试:

全新

全旧

保证数据严格一对一:并发新加,并发新增,并检测返回值

可优化的点

双buffer

web容器线程池

连接池

非阻塞式

jvm参数

ssdb参数

setnx http://ssdb.io/docs/zh_cn/commands/setnx.html 替换所有的代码

nginx参数

将long转成base64编码,然后存到redis中

TODOList:

验证url_hash是否正确

验证读取文件的wrk和不读取的wrk差异

压力测试

!!!!使用siege注意
手动打乱顺序
使用siege进行压力测试要注意！
他的随机访问文件中的URL，是会重复访问到同一个url且不会停止。
siege 参数 -c * -r == wc -l nohup.log

脚本可用测试 1000条
siege -c 5 -f test_v1_1000.txt -i
Transactions: 161418 hits
Availability: 100.00 %
Elapsed time: 74.98 secs
Data transferred: 0.77 MB
Response time: 0.00 secs
Transaction rate: 2152.81 trans/sec
Throughput: 0.01 MB/sec
Concurrency: 4.41
Successful transactions: 161418
Failed transactions: 0
Longest transaction: 3.02
Shortest transaction: 0.00

服务自测试场景

14w写库 siege -c 100 -r 1400 -f test_v1_140w_aa
2000w 总量 新增60w 旧数据140w 100并发 请求十次 重复1800w

Transactions: 140000 hits
Availability: 100.00 %
Elapsed time: 11.49 secs
Data transferred: 0.70 MB
Response time: 0.01 secs
Transaction rate: 12184.51 trans/sec
Throughput: 0.06 MB/sec
Concurrency: 93.64
Successful transactions: 140000
Failed transactions: 0
Longest transaction: 0.05
Shortest transaction: 0.00

140w 写库 siege -c 100 -r 14000 -f test_v1_140w.txt

Transactions: **1400000 hits**
Availability: 100.00 %
Elapsed time: **115.59 secs**
Data transferred: 8.29 MB
Response time: 0.01 secs
Transaction rate: ** 12111.77 trans/sec**
Throughput: 0.07 MB/sec
Concurrency: 96.01
Successful transactions: 1400000
Failed transactions: 0
Longest transaction: 0.09
Shortest transaction: 0.00

1000w验证
2000w 总量 新增60w 旧数据140w 重复1800w
1000w 140w旧 60w新 800w重复
siege -c 100 -r 200000 -f test_v1_2000w_shuf.txt
siege -c 200 -r 50000 -f test_v1_1000w_shuf.txt
Transactions: **10000000 hits**
Availability: 100.00 %
Elapsed time: ** 789.79 secs = 13min **
(由此推算，1亿请求130min 2h10min , 2亿请求 260min 4h 20min.)
Data transferred: 61.46 MB
Response time: 0.01 secs
Transaction rate: **12661.59 trans/sec**
Throughput: 0.08 MB/sec
Concurrency: ** 179.90**
Successful transactions: **10000000**
Failed transactions: 0
Longest transaction: 1.04
Shortest transaction: 0.00

v2
cp test_v2_140w.txt test_v2_200w.txt
siege -c 100 -f test_v2_140w.txt -i
tail -f nohup.out | head -n 10
140w 写库
布隆过滤器判重服务与redis请求数量过多会超时 无法完成写库

nohup siege -c 100 -r 14000 -f test_v2_140w.txt &

1000w 验证

nohup siege -c 200 -r 50000 -f test_v2_1000w_shuf.txt &

[2650008, 2650007, 2650012, 2650009, 2650010, 2650013, 2650011, 2650014]

演示数据

28w 写库

40w 5 = 验证 28w旧 + 12w新 +160w重复(12w4 + 28*4)

预计 运行时间6分钟 接口时间2分40s

124 /home/hadoop/data/show_case/show_200w_shuf.txt

写库

nohup siege -c 100 -r 2800 -f show_28w.txt &

验证

nohup siege -c 200 -r 10000 -f show_200w_shuf.txt &

siege -c 200 -r 10000 -f show_200w_shuf.txt

wrk -t4 -c1000 -d30s -T5s --script=get.lua --latency XXXXXXX(请求地址)

最后

文章还没有详细干货化整理，只是把竞赛时的思路记录下来，有什么问题可以评论留言，看到可能就复了哈哈。

感谢我杰哥宇哥，刚进公司有领路人真的是非常幸运的事情。

祝大家工作顺利，生活愉快。