随手科技
SUISHOU TECHNOLOGY

随手记Go实践
高并发积分系统改造

作者：曾龙

随手记 卡牛 随管家

主要内容

1、改造背景

2、业务逻辑优化

3、两种负载均衡策略

4、Go程序优化(map、GC)

5、总结

改造细节

业务逻辑优化

1、合并数据库操作，将原来的5张表操作缩减成4张表操作

2、确立表操作的优先级，如果将积分操作分为三个阶段：
①写入积分记录
②更新用户总积分
③更新缓存
其中阶段①是必须先完成的，但操作②和③可以采用异步方式处理，如此可大大提升系统的响应速度

业务逻辑优化

随手科技

兼容旧版积分操作的挑战：

旧版积分操作对于客户端来说都是同步等待完成的。
那如何做到：
1. 用户可实时获取积分操作结果
2. 高并发下保证业务可用
3. 简单易实现

业务逻辑优化

数据分库分表后，积分写入操作单库高达(1.5-2)w/s,能够
轻松应对日常积分操作需求

极端高并发情况下，系统实现自动降级，通过消息队列进行处理,
达到削峰填谷的目的

降级开关简单实现：

```
select {
   case queue.direct <- v: //入直接操作队列成功
   case <- time.After(10 * time.Millisecond):
      //入直接操作队列超时后转消息队列处理
}
```

两种负载均衡方案

最常用的2种负载均衡方案：

1. 轮询（或带权重）
2. hash

为了做更好的对比，我们同时实现了2种负载均衡方案

两种负载均衡方案

	轮询	user hash
请求分配	最均匀	均匀(和hash函数有关)
水平扩展	容易	略复杂
积分项目缓存依赖	redis集群	本地map
性能	高	极高
请求执行速度(10万次均值)	841μs	317μs
最好值	398μs	109μs

注：最坏值一般发生在写入时候，两者较为接近，都是大于30ms

两种负载均衡方案

为何user hash更快?

与轮询相比，hash方式可以将同一个用户的请求转发到同一台机器上，因此：
1、可以直接用Golang内置数据结构，比如map进行用户数据存储。而轮询方式必须借助第三方的redis集群进行数据存储。

2、轮询方式在进行用户积分操作时，可能需要借助分布式锁进行操作，以确保数据一致性，而采用hash方式则完全不需要。例如：更新用户缓存

hash下如何保障系统高可用?

两种负载均衡方案

土豪的正确做法

买机器

建集群

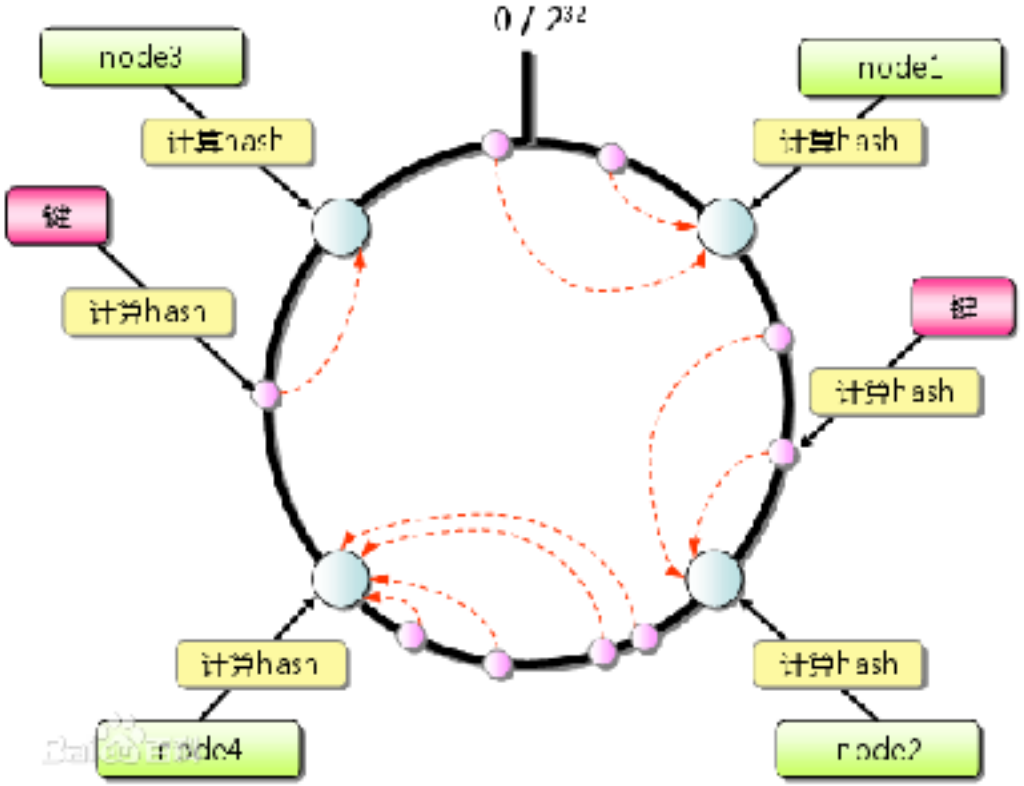
所有服务都做成集群，永远不要出现单机应用，要是有钱，一主N备都不是梦

两种负载均衡方案

节约成本的做法：

采用一致性hash算法

当然，这里有个前提条件，即缓存丢失后可立即从数据库恢复



Go程序优化——map

影响map性能的主要因素

1. hash函数
   例如hash冲突率
2. 装载因子
   决定map扩容阈值
3. 高并发下锁开销
   多读多写下开销巨大

Go程序优化——map

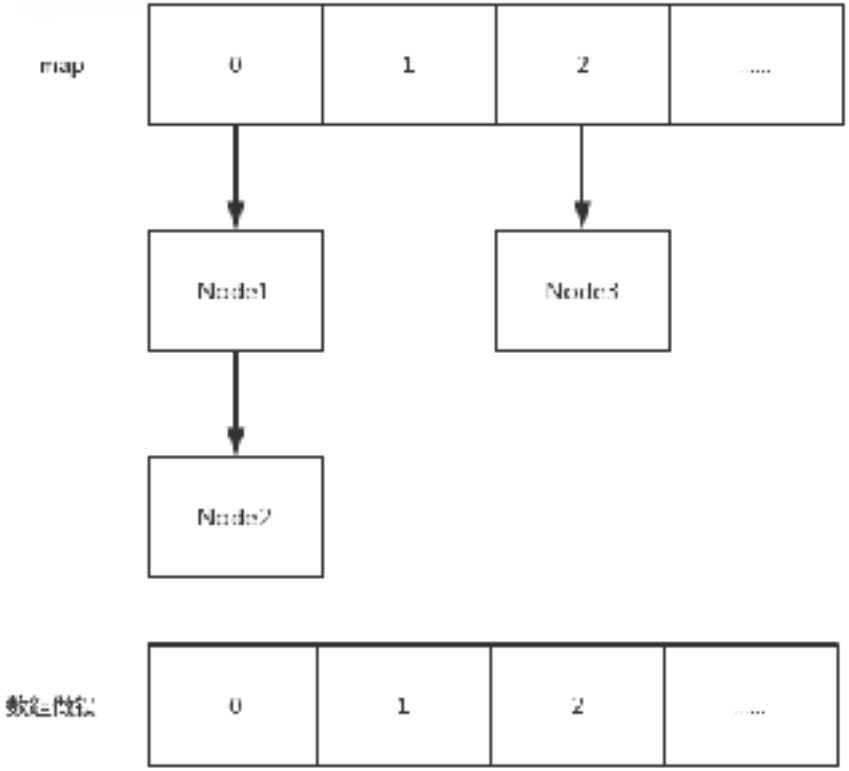
map优化建议：

1. 根据实际需求编写hash函数

2. 放弃装载因子，一次性提供足够大小的map空间，杜绝map扩容操作

3. 空间换时间，减少锁操作，以原子锁替代系统锁，获得更好效果

Go程序优化——map



锁lock,数组所有元素初始值为0

一种少锁map实现：

1、确定map长度

2、数据写入：例如Node2经hash函数得出结果是0，意味着我们将要将数据存储到map[0],步骤如下：

①将做锁的数组lock[0]做原子操作，改为1

②将Node2放到Node1后面Node1->next=Node2

③lock[0]=0，释放锁，完成插入。

3、因为有2的操作，所以读取和删除时完全不用加锁

Go程序优化——map

性能对比

	内置map	自定义map
10协程读1协程写	613 ns/op	83.5 ns/op
1协程读1协程写	306 ns/op	83.2 ns/op

自定义map表现非常稳定，性能也有大幅提升，当然，其他资源例如内存占用大家可以自己去测试

当然，相对于整个程序来说，这点性能提升可以说是微不足道，然而在一些大型系统中，能争一点是一点，主要是我们要把思路放宽，勤于思考。

GC卡顿

Go已经大幅度优化了GC卡顿问题，所以对于99.9%的的应用来说，GC根本不是问题
但对于一些实时性要求极高或者对象生成销毁操作极为频繁的应用来说，GC卡顿也是不得不关注的问题。

Go程序优化——GC

1. 减少对象生成，合并对象，或者使用资源池增加对象的重复利用

2. 利用cgo，用c语言进行内存管理

3. 换个无runtime的语言，例如C

一些旁门左道办法：

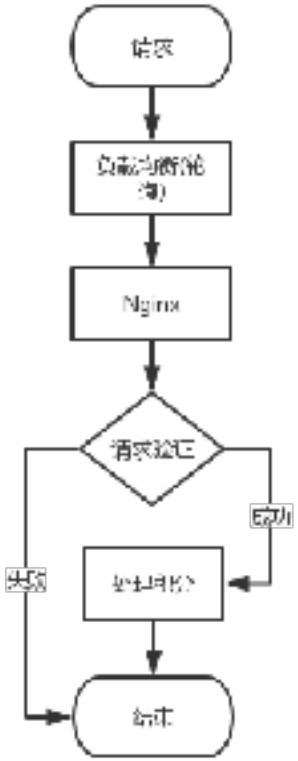
1、设置环境变量GOGC=off关闭垃圾回收，自己手动回收内存
然而，目瞪口呆的我们会发现，这跟C有啥区别，说好的奔驰宝马怎么突然变成手推车了。

2、要不然就是GOGC=200或者更大的数咯，减少GC次数。
然而这种方式虽然可以减少GC的次数，但因为堆积的对象更多了，每次卡顿时间更长了

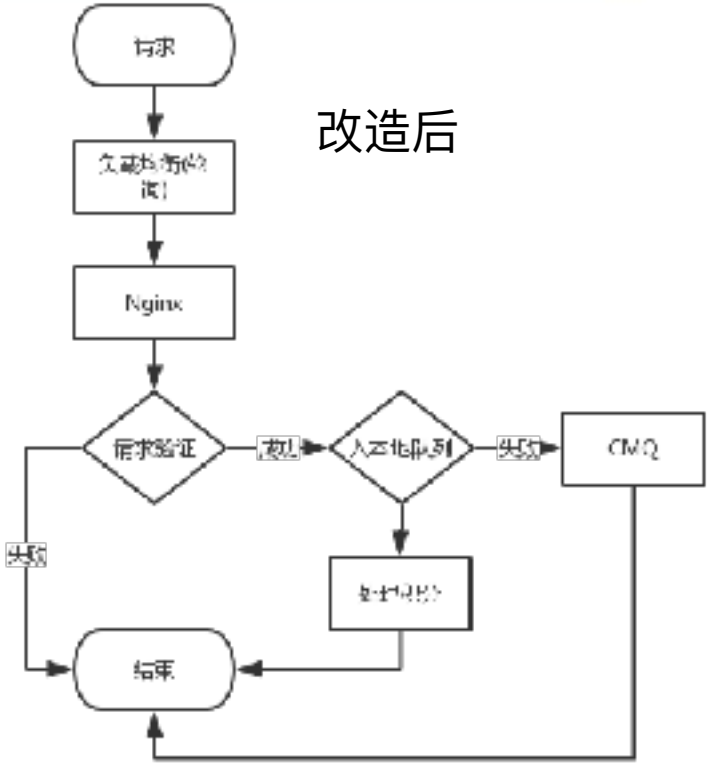
内存碎片问题

如果我们使用cgo进行内存管理，同样也会面对c语言的内存碎片问题，解决方法也挺简单，直接利用第三方内存管理软件即可。
jemalloc是个不错的选择。

流程图



改造前



改造后

总结

随手科技

本次采用Go进行积分系统重构，从各个方面来说都取得了不错的成绩

积分系统整体性能有了 10 倍以上提升

随手科技

THANKS

随手记 卡牛 随管家