author = "gaj"
title = "sql调优经历"
description = ""
date = 2024-11-16T16:02:00+08:00
draft = true
tags = [
]
categories = [
  "随记"
]

SQL调优经历

业务背景（需要更换）

优惠卷活动的状态更新，这里我们有俩个定时任务

- 定时更新优惠卷活动的状态为进行中、结束
- 定时扫表预热活动

数据库

数据表

我们的优惠卷活动和别家电商的优惠卷不太一样，我们的优惠卷分为俩种类型：

- 普通满减、折扣优惠卷
- 分阶段满减、折扣优惠卷（越买越优惠）

所以这里我们的优惠卷**存在一个关联表**，用于关联这种分阶段优惠的优惠阶段

定位问题：优惠卷定时任务耗时长、cpu占用高的解决

数据库（这个活动表不会有这么大的数据量）

64分片，一主一从，分片键是活动id

数据源

- 平台的优惠活动
- 商家的店铺优惠活动

机器配置和垃圾回收器

单台机器用的8C16G（相关配置需要修改）

-Xms8192m -Xmx8192m -XX:MaxMetaspaceSize=1024m -XX:MetaspaceSize=1024m -XX:MaxDirectMemorySize=1966m -XX:+**UseG1GC** -XX:**MaxGCPauseMillis**=200 -XX:**ParallelGCThreads**=8

定时更新活动状态逻辑（需要换个更复杂的场景）

根据活动的状态

- 将待生效的活动根据活动的时间（开始时间、结束时间）更新为进行中
- 将待生效的、进行中的活动根据活动时间更新为结束

问题现象

我们的定时任务间隔一分钟，执行时间很长，40秒，cpu占用80%，虽然前端做了倒计时，但占用率太高不健康。

> 按照这个问题假设估计我们数据量有可能会达到千万级

优化

S1：分析数据源，调大步长缩短任务执行时间

（一次任务中分多步处理，而不是直接全部搜出来到内存，与云岚存在差异）

数据源表里数据非常稀疏

- 程序问题：（落入了别的来源的数据）
- 历史问题：（）

所以为了避免每次处理的数据较少，调大了步长，避免空跑

阶段性结果执行时间减少到了30秒（假设），但cpu顶着100%跑

S2：减少临时对象大小和无效日志，避免ygc

想法：对事务总线负载（mq）到多台机器，但耗费资源

于是我们分析cpu高在哪里，理论上这个任务是**IO型任务**，cpu利用率低，仔细观察发现**机器不断多次ygc。**G1的ygc比1.8的默认回收期更耗资源（G1要兼顾mixgc回收）。我们这的任务其实更适合用默认的垃圾回收器（Parallel）

> G1用于响应优先，默认的垃圾回收器则是吞吐量优先。

那么不断ygc肯定是我们**产生了大量的临时对象导致的**

（1）去掉无效日志

java在序列化的时候会产生比原来的对象占用更多内存的临时变量

（2）减少临时对象占用的内存

代码对象的个数不能减少，于是我们减小了对象的大小，为这个定时任务写了专用的接口，每个接口**只查我们需要的字段**

阶段性结果时间减少到了10秒（假设），cpu降到50%

但每次任务执行，都给数据库带来了**很高的qpm增长，有隐患**

问题待解决

- 减少交互次数
- 降低任务执行时间

仍有很大的优化空间

S3：基于游标查询数据源，数据库分片批量更新，降低交互次数，避免空跑

分析这么大的调用量用在了哪里，发现：活动表的**MaxAutoPk非常大**（10亿，太夸张了这个场景根本不能满足），所以即使调大了步长，交互次数仍然很多，**继续调大步长会导致某个数据密集导致负载不均衡**

（1）使用mysql的游标查询

- 表中要有唯一键，且基于唯一键查询排序（升序）
- 取件查询满足查询条件记录越稀疏越有效

（2）深度分页引起慢sql基于游标查询的sql不会有这个问题，每次直接定位到区间开始

（3）按数据库分片进行批量更新

基于游标查询导致每个步长都查出来比以前更多的数据，**出现了长事务**（这里不太契合场景，可以考虑把这个优化删掉）

由于在批量更新的时候**没有带分片键导致sql分发到了所有分片上**， **实际导致了长事务**

于是我们**按数据库分片对这些活动id进行分组**，保证每个分组对应的单元id落到同一个分片（不太理解，是不是类似redis的hash插槽，让我们优惠卷的三个key落到一个redis结点，这里让对应的ids落到对应的数据库分片）

阶段性结果时间1秒，cpu50%。qpm降了很多

问题

cpu还是较高

S4：异构数据源，做了冗余（当前的场景把关联表冗余进来绝对会更差）

前面提到，分阶段优惠卷的关联关系用了一张关联表维护

我们将整个管理表冗余到了活动表中，通过一种链表关系来维护分阶段的优惠活动，多了一个根活动的id，0时表示普通活动，具体id时表示对应的根活动id。

> 要不还是倒过来把字段冗余出去一张表，但这样也不太好

S5：负载均衡，消除所有风险

任务拆分，交给mq负载到多台机器上（mq重试机制不可靠，需要解决重试频率、识别无效重试、防止重试叠加）

> mq的重试中，失败了只会投回队列，什么时候能再次消费不确定
>
> 1、本地重试机制固定间隔重试，可以应对简单场景（也可以递增间隔重试、基于业务优先级重试）
>
> 2、引入定时任务框架（Quartz、xxljob），但这里就任务套任务了不好
>
> 3、结合分布式调度系统（zookeeper、etcd），利用zk的watcher机制来监控结点状态
>
> 4、最终补偿机制：人工

优化效果汇总

毫秒级，cpu占用增加1%