一、为什么要有缓存

当前Guardian操作分流的依据的是请求URL中的License Key的参数值，但是需要先从业务系统的数据库中获取License Key对应的用户ID，然后发起HTTP请求用户系统获取付费状态。最后，根据付费状态做流量导向。

对于License Key的付费状态，不可能说每次请求去做上述的操作获取更新，那样子会有较大的性能损耗。

结合业务特点，对于付费的状态信息，基本上变更的频率并不是那么频繁。所以对于付费状态信息，需要结合License Key做缓存。当前缓存使用OpenResty提供的Shared DICT。缓存策略是对每笔请求，获取License Key，从缓存中去取付费状态。如果缓存中没有，则远程调用获取，获取到了，缓存1个小时。1个小时后缓存失效，则重新调用后端获取付费状态。

二、存在问题

1. 初期缓存命中率低时的热点问题

应用初期运行的时，因为缓存不存在，所以会出现，大量License key付费状态查询请求。这些请求将会同时落到后端数据库上，可能造成服务器卡顿甚至宕机。请求都会有一定的延迟，由于OpenResty对于阻塞的敏感程度较高，很有可能就会因为超时导致将付费类型变为免费用户的可能。

同时，因为缓存设计的失效时间为1个小时，这个期间，如果链接建立，请求全打在这台机器上，那么基本上更新前无法为付费用户提供高质的服务。

2. 缓存失效策略不严谨问题

如果缓存直接失效，按照一开始的策略，会出现周期性1小时负载问题。也就是说，每过一段时间，缓存就会集体失效，然后并发请求后端服务获取付费状态。如果部署多台，启动间隔较短，那么基本是灾难。

3. 缓存无法实时更新问题

Guardian缓存周期为一个小时，这个期间如果某个用户付费，无法立刻将其的付费状态更新为已经付费。且每个节点的失效时间不一致，N台Guardian节点，理论上全部更新的时间上限可能为N小时。

三、解决方案

3.1 启动缓存加载策略：

鉴于Java后端迟早会被废弃，所以设计Guardian前端启动时直接从Redis里面加载付费信息缓存。

存在问题：

A. Redis存储的数据来源？

初步计划就是OpenResty定时将Shared DICT的数据推送至Redis中。

B. 如何保证数据一致性呢？比如说某台OpenRestyHTTP请求获取License Key对应的付费信息失败，所以它的缓存其实是默认免费，但不一定正确，如果它的数据推送至Redis中，可能会将付费用户变为免费用户。

C. Redis不可用时，是否运行Guardian启动，初始数据来源？

可考虑在Guardian上设计缓存数据展示接口，HTTP调用展示当前**某种**缓存信息，Redis不可用时，从Guardian集群中获取实时的生产缓存数据。此缓存展示接口后续也可以作为运维监控的参考数据，比较每个节点的缓存数据是否一致。

3.2 Shared DICT缓存更新锁设计

Guardian获取不到缓存时，需要对所需的License Key加锁，避免后面的相同请求同时发起多个重复请求更新缓存，保证操作原子性。

存在问题：

A. 加锁是否会阻塞，是否会死锁？

加锁本质上，使用的是lua-resty-lock。一、加锁操作无阻塞，二、锁的实现是基于共享内存的，且创建时总会设置一个过期时间，因此不用担心会发生死锁。

B. 加锁后付费信息远程获取失败，如何处理？缓存还是不缓存？

加锁后的操作流程基本是：

1. 检查某个 Key 的缓存是否命中，如果 MISS，则进入步骤 2。
2. 初始化 `resty.lock` 对象，调用 `lock` 方法将对应的 Key 锁住，检查第一个返回值（即等待锁的时间），如果返回 `nil`，按相应错误处理；反之则进入步骤 3。
3. 再次检查这个 Key 的缓存是否命中，如果依然 MISS，则进入步骤 4；反之，则通过调用 `unlock` 方法释放掉这把锁。
4. 通过远程请求后端查询付费信息，把查询到的结果缓存起来，最后通过调用 `unlock` 方法释放当前 Hold 住的这把锁。

所以，主要的失败在于第四步远程调用，对于此种状态，付费状态随机判定，缓存数据不再是付费状态，而改为**TIME_OUT标识**。由OpenResty的timer去对shared DICT中此种标识的数据做更新操作（更新操作依旧需要锁处理）。每次请求如果获取到的缓存值为TIME_OUT标识，则随机判定付费状态。

3.3 Guardian缓存长更新周期，或者不失效设计

按照业务特点，其实付费信息的变更的周期最小也得是一个月左右（最快也可能要一天）。所以我们的缓存可以直接设置为不失效或者很长的时间。

存在问题：

A. 如果第一次缓存请求获取不到付费信息，缓存的其实是错误信息，不更新的话会导致问题。

按照前面的解决方案，获取付费信息超时的授权状态为超时，会有定时的任务去更新状态，直到获取到真正的付费状态。

B. 如果缓存失效，且无法远程调用后端更新数据，怎么办？

此时，可以考虑使用已经过期的旧Shard DICT缓存，可以结合使用 lua-resty-shcache

C. OpenResty的Shared DICT缓存无法持久化存储，重启就没了，如何保证缓存不直接失效？

参考前面的Redis持久化存储，使用Timer定时同步至Redis即可。（从某种角度而言，把Redis当作了DB。）

3.4 Guardian缓存更新接口

设计Guardian内部调用接口，接受外部的缓存更新推送，指定推送的消息格式。用户付费状态变更时调用此接口更新Guardian的付费信息缓存，同时将更新的数据推送至Redis中存储。

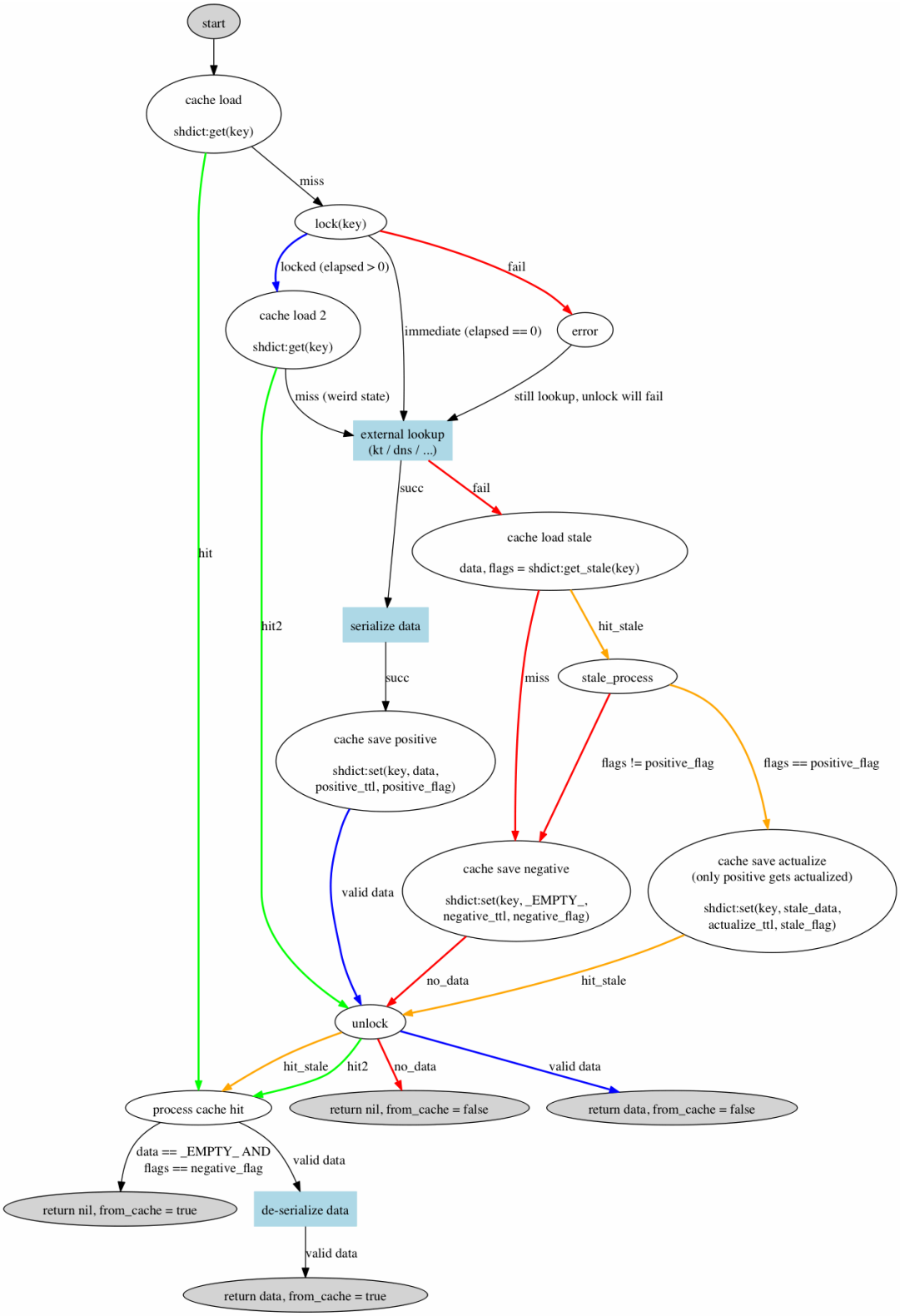
存在问题：

A. Guardian为集群环境下的通知推送问题。

暂时没有更好的解决方案，可能需要一台台去推送更新。结合缓存查询接口和响应报文判断是否推送成功。

三、编码与设计

缓存获取流程：



Like Be the first to like this