WXG技术能力提升

如何实现

ruoyuliu 2021年

目录

| 导语 定时
者在工作中
时器服务。

用的组件，主要用在执行延时通知任务上。本文以笔
对部门最常用的组件快速实现一个业务常用的分布式定
解决方案，希望能够给类似场景提供一些解决思路。

1.什么是

定时器（Tim
的Timer，我
要使用定时器

某一任务的工具（也有周期性反复执行某一任务
延迟队列这一概念关联。 那么在什么场景下我才需

我们先看看以

- 当订单一直
- 如何定期检
- 新创建店铺

的关闭订单，并退还库存?
经退款成功?
如何知道该信息，并发送激活短信?

为了解决以上问题，最简单直接的办法就是定时去扫表。每个业务都要维护一个自己的扫表逻辑。 当业务越来越多时，我们会发现扫表部分的逻辑会非常类似。我们可以考虑将这部分逻辑从具体的业务逻辑里面抽出来，变成一个公共的部分。这个时候定时器就出场了。

2.定时器的本质

一个定时器本质上是这样的一个数据结构：deadline越近的任务拥有越高优先级，提供以下几种基本操作：

1. Add 新增任务
2. Delete 删除任务
3. Run 执行到期的任务/到期通知对应业务处理
4. Update 更新到期时间 (可选)

Run通常有两种工作方式：
1.轮询
每隔一个时间片就去查找哪些任务已经到期；
2.睡眠/唤醒

关于作者

ruoyuliu(刘若愚)
WXG\境外产品中心\境外支付开发组...

作者文章

- 如何实现一个分布式定时器
- 你上传的图片安全吗？记一次隐私图片数据安...
- 腾讯相册小程序&看图小程序后台技术总结
- 空间相册合成视频技术总结
- 空间视频后台总结

收录于

微信境外支付文集
浏览 364 收藏 14

微信&企业微信&政务微信

不停地查找deadline最近的任务，如到期则执行；否则sleep直到其到期。
在sleep期间，如果有任务被Add或Delete，则deadline最近的任务有可能改变，线程会被唤醒并重新进行1的逻辑。

它的设计目标通常包含以下几点要求：

1. 支持任务提交（消息发布）、任务删除、任务通知（消息订阅）等基本功能。
2. 消息传输可靠性：消息进入延迟队列以后，保证至少被消费一次（到期通知保证At-least-once，追求Exactly-once）。
3. 数据可靠性：数据需要持久化，防止丢失。
4. 高可用性：至少得支持多实例部署。挂掉一个实例后，还有后备实例继续提供服务，可横向扩展。
5. 实时性：尽最大努力准时交付信息，允许存在一定的时间误差，误差范围可控。

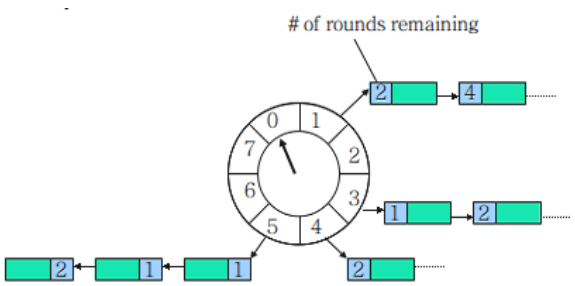
3.数据结构

下面我们谈谈定时器的数据结构。定时器通常与延迟队列密不可分，延时队列是什么？顾名思义它是一种带有延迟功能的消息队列。而延迟队列底层通常可以采用以下几种数据结构之一来实现：

1. 有序链表，这个最直观，最好理解。
2. 堆，应用实例如Java JDK中的DelayQueue、Go内置的定时器等。
3. 时间轮/多级时间轮，应用实例如Linux内核定时器、Netty工具类HashedWheelTimer、Kafka内部定时器等。

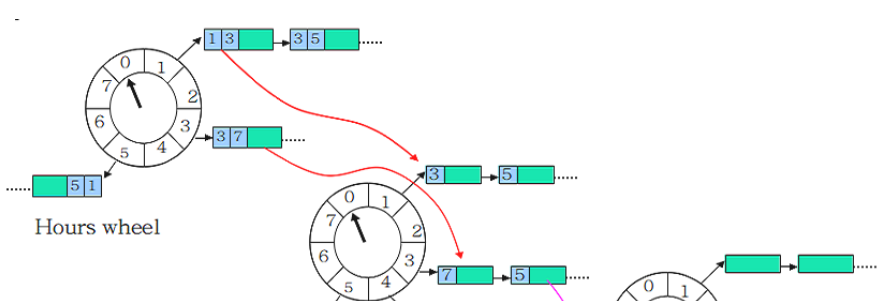
这里重点介绍一下时间轮（TimeWheel）。一个时间轮是一个环形结构，可以想象成时钟，分为很多格子，一个格子代表一段时间（越短Timer精度越高），并用一个List保存在该格子上到期的所有任务，同时一个指针随着时间流逝一格一格转动，并执行对应List中所有到期的任务。任务通过取模决定应该放入哪个格子。示意图如下所示：

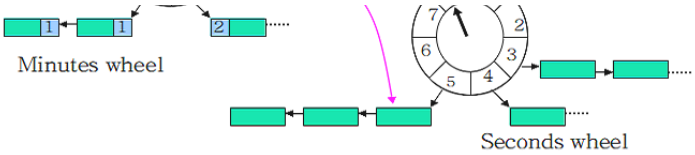
时间轮示意图



如果任务的时间跨度很大，数量也多，传统的单轮时间轮会造成任务的round很大，单个格子的任务List很长，并会维持很长一段时间。这时可将Wheel按时间粒度分级（与水表的思想很像）：

多级时间轮示意图





时间轮是一种比较优雅的实现方式，且如果采用多级时间轮时其效率也是比较高的（具体的实现例子大家可以看看这篇《Go语言中时间轮的实现》）。

4.业界实现方案

业界对于定时器/延时队列的工程实践，则通常基于以下几种方案来实现:

1. 基于Redis ZSet实现。
2. 采用某些自带延时选项的队列实现，如RabbitMQ、Beanstalkd、腾讯TDMQ等。
3. 基于Timing-Wheel时间轮算法实现。

其中《你真的知道怎么实现一个延迟队列吗？》一文详细介绍了具体的实现方式，大家有兴趣可以阅读下。

从任务调度方式上看，以上三种方式都可以归类为以下两种情况：

即按照事件离散化和按照时间离散化两种。

5.方案详述

介绍完定时器的背景知识，接下来看下我们系统的实现。我们先看一下需求背景。在我们组的实际业务中，有延迟任务的需求。一种典型的应用场景是：商户发起扣费请求后，立刻为用户下发扣费前通知，24小时后完成扣费；或者发券给用户，3天后通知用户券过期。基于这种需求背景，我们引出了定时器的开发需求。

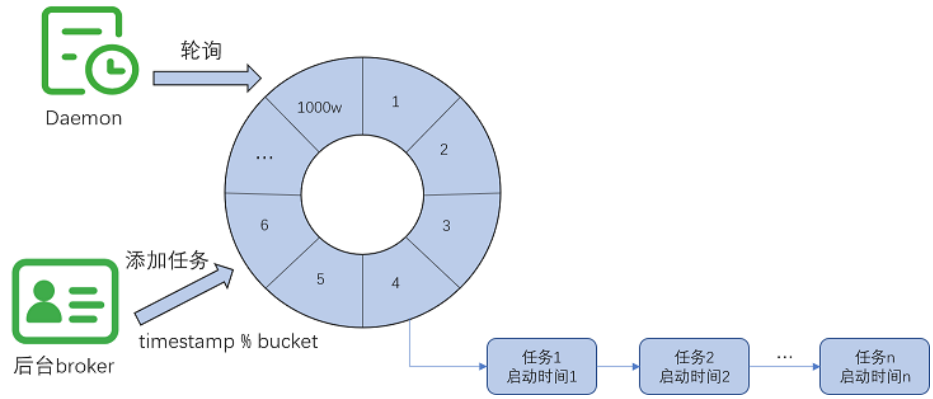
我们首先调研了公司内外的定时器实现，避免重复造轮子。调研了诸如例如公司外部的Quartz、有赞的延时队列等，以及公司内部的PCG tikker、TDMQ等，以及微信支付内部包括营销、代扣、支付分等团队的一些实现方案。最后从可用性、可靠性、易用性、时效性以及代码风格、运维代价等角度考虑，我们决定参考前人的一些优秀的技术方案，并根据我们团队的技术积累和组件情况，设计和实现一套定时器方案。

首先要确定定时器的存储数据结构。这里借鉴了时间轮的思想，基于微信团队最常用的存储组件tablekv进行任务的持久化存储。使用到tablekv的原因是它天然支持按uin分表，分表数可以做到千万级别以上；其次其单表支持的记录数非常高，读写效率也很高，还可以如mysql一样按指定的条件筛选任务。

我们的目标是实现秒级时间戳精度，任务到期只需要单次通知业务方。故我们方案主要的思路是基于tablekv**按任务执行时间分表**，也就是使用使用方指定的start_time（时间戳）作为分表的uin，也即是时间轮bucket。为什么不使用多轮时间轮？主要是因为首先kv支持单表上亿数据， 其二kv分表数可以非常多，例如我们使用1000万个分表需要约115天的间隔才会被哈希分

配到同一分表内。故暂时不需要使用到多轮时间轮。

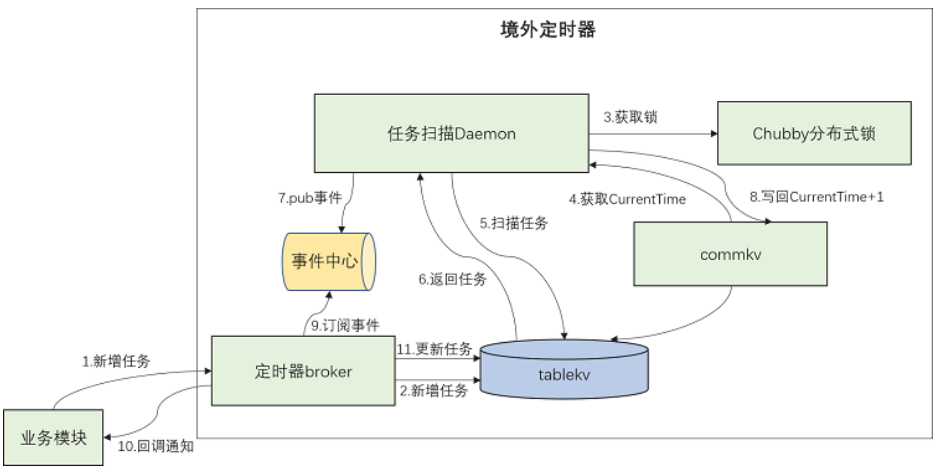
最终我们采用的分表数为1000w，uin=时间戳mod分表数。这里有一个注意点，通过mod分表数进行Key收敛，是为了避免时间戳递增导致的key无限扩张的问题。示例图如下所示：



任务持久化存储之后，我们采用一个Daemon程序执行定期扫表任务，将到期的任务取出，最后将请求中带的业务信息（biz_data添加任务时带来，定时器透传，不关注其具体内容）回调通知业务方。这么一看流程还是很简单的。

这里扫描的流程类似上面讲的时间轮算法，会有一个指针（我们在这里不妨称之为time_pointer）不断向后移动，保证不会漏掉任何一个bucket的任务。这里我们采用的是commkv（可以简单理解为可以按照key-value形式读写的kv，其底层仍是基于tablekv实现）存储CurrentTime，也就是当前处理到的时间戳。每次轮询时Daemon都会通过GetByKey接口获取到CurrentTime，若大于当前机器时间，则sleep一段时间。若小于等于当前机器时间，则取出tablekv中以CurrentTime为uin的分表的TaskList进行处理。本次轮询结束，则CurrentTime加一，再通过SetByKey设置回commkv。这个部分的工作模式我们可以简称为Scheduler。

Scheduler拿到任务后只需要回调通知业务方即可。如果采用同步通知业务方的方式，由于业务方的超时情况是不可控的，则一个任务的投递时间可能会较长，导致拖慢这个时间点的任务整体通知进度。故而这里自然而然想到采用**异步解耦**的方式。即将任务发布至事件中心（微信内部的高可用、高可靠的消息平台，支持事务和非事务消息（详见《微信事件中心 - 高可靠、高可用的事务消息平台》）。由于一个任务的投递到事件中心的时间仅为几十ms，理论上任务量级不大时1s内都可以处理完。此时time_pointer会紧跟当前时间戳。当大量任务需要处理时，需要采用多线程/多协程的方式并发处理，保证任务的准时交付。broker订阅事件中心的消息，接受到消息后由broker回调通知业务方，故broker也充当了Notifier的角色。整体架构图如下所示：

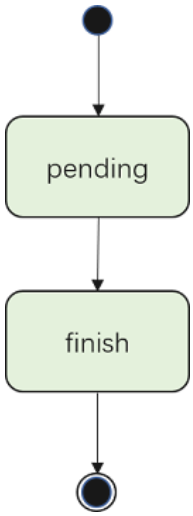


主要模块包括：

任务扫描Daemon：充当Scheduler的角色。扫描所有到期任务，投递到事件中心，让它通知broker，由broker的Notifier通知业务方。

定时器broker：集业务接入、Notifier两者功能于一身。

任务状态机图如下所示，只有两种状态。当任务插入kv成功时即为pending状态，当任务成功被取出并通知业务方成功时即为finish状态。



6.实现细节与难点思考

下面就上面的方案涉及的几个技术细节进行进一步的解释。

6.1 业务隔离

通过biz_type定义不同的业务类型，不同的biz_type可以定义不同的优先级（目前暂未支持），任务中保存biz_type信息。
业务信息（主键为biz_type）采用境外配置中心进行配置管理。方便新业务的接入和配置变更。业务接入时，需要在配置中添加诸如回调通知信息、回调重试次数限制、回调限频等参数。业务隔离的目的在于使各个接入业务不受其他业务的影响，这一点由于目前我们的定时器用于支持本团队内部业务的特点，仅采取对不同的业务执行不同业务限频规则的策略，并未做太多优化工作，就不详述了。

6.2 时间轮空转问题

由于1000w分表，肯定是大部分Bucket为空，时间轮的指针推进存在低效问题。联想到在饭店排号时，常有店员来登记现场尚存的号码，就是因为可以跳过一些号码，加快叫号进度。同理，为了减少这种“空推进”，Kafka引入了DelayQueue，以bucket为单位入队，每当有bucket到期，即queue.poll能拿到结果时，才进行时间的“推进”，减少了线程空转的开销。在这里类似的，我们也可以做一个优化，维护一个有序队列，保存表不为空的时间戳。大家可以思考一下如何实现，具体方案不再详述。

6.3 限频

由于定时器需要写kv，还需要回调通知业务方。因此需要考虑对调用下游服务做限频，保证下游服务不会雪崩。这是一个**分布式限频**的问题（可以参考《单机和分布式场景下，有哪些流控方案？》）。这里使用到的是微信支付的限频组件。保证1.任务插入时不超过定时器管理员配置的频率。 2.Notifier回调通知业务方时不超过业务方申请接入时配置的频率。这里保证了1.kv和事件中心不会压力太大。2.下游业务方不会受到超过其处理能力的请求量的冲击。

6.4 分布式单实例容灾

出于容灾的目的，我们希望Daemon具有容灾能力。换言之若有Daemon实例异常挂起或退出，其他机器的实例进程可以继续执行任务。但同时我们又希望同一时刻只需要一个实例运行，即“分布式单实例”。所以我们完整的需求可以归纳为**“分布式单实例容灾部署”**。

实现这一目标，方式有很多种，例如：

- 接入“调度中心”，由调度中心来负责调度各个机器
- 各节点在执行任务前先分布式抢锁，只有成功占用锁资源的节点才能执行任务
- 各节点通过通信选出“master"来执行逻辑，并通过心跳包持续通信，若“master”掉线，则备机取代成为master继续执行

主要从开发成本，运维支撑两方面来考虑，选取了基于chubby分布式锁的方案来实现单实例容灾部署。这也使得我们真正执行业务逻辑的机器具有随机性（关于Daemon的分布式单实例容灾实现方式参考《境外Daemon库介绍》一文）。

6.5 可靠交付

这是一个核心问题，如何保证任务的通知满足At-least-once的要求?

我们系统主要通过以下两种方式来保证。

1.任务达到时即存入tablekv持久化存储，任务成功通知业务方才设置过期（保留一段时间后删除），故而所有任务都是落地数据，保证事后可以对账。

2.引入可靠事件中心（参考《微信事件中心 - 高可靠、高可用的事务消息平台》）。在这里使用的是事件中心的普通消息，而非事务消息。实质是当做一个高可用性的消息队列。

这里引入消息队列的意义在于：

- 将任务调度和任务执行解耦（调度服务并不需要关心任务执行结果）。
- 异步化，保证调度服务的高效执行，调度服务的执行是以ms为单位。
- 借助消息队列实现任务的可靠消费。

事件中心相比普通的消息队列还具有哪些优点呢?

- 某些消息队列可能丢消息（由其实现机制决定），而事件中心本身底层的分布式架构，使得事件中心保证极高的可用性和可靠性，基本可以忽略丢消息的情况。
- 事件中心支持按照配置的不同事件梯度进行多次重试（回调时间可以配置）。
- 事件中心可以根据自定义业务ID进行消息去重。

事件中心的引入，基本保证了任务从Scheduler到Notifier的可靠性。

当然，最为完备的方式，是增加另一个异步Daemon作为兜底策略，扫出所有超时还未交付的任务进行投递。这里思路较为简单，不再详述。

6.6 及时交付

若同一时间点有大量任务需要处理，如果采用串行发布至事件中心，则仍可能导致任务的回调通知不及时。这里自然而然想到采用多线程/多协程的方式并发处理。在本系统中，我们使用到了微信的BatchTask库，BatchTask是这样一个库，它把每一个需要并发执行的RPC任务封装成一个函数闭包（返回值+执行函数+参数），然后调度协程（BatchTask的底层协程为

libco）去执行这些任务。对于已有的同步函数，可以很方便的通过BatchTask的Api去实现任务的批量执行（详见《微信后台批量任务BatchTask的设计与实现》）。Daemon将发布事件的任务提交到BatchTask创建的线程池+协程池（线程和协程数可以根据参数调整）中，充分利用流水线和并发，可以将任务List处理的整体时延大大缩短，尽最大努力及时通知业务方。

6.7 任务过期删除

从节省存储资源考虑，任务通知业务成功后应当删除。但删除应该是一个异步的过程，因为还需要保留一段时间方便查询日志等。这种情况，通常的实现方式是启动一个Daemon异步删除已完成的任务。我们系统中，是利用了tablekv的自动删除机制，回调通知业务完成后，除了设置任务状态为完成外，同时通过tablekv的update接口设置kv的过期时间为1个月，避免了异步Daemon扫表删除任务，简化了实现。

6.8 其他风险项

1.由于time_pointer的CurrentTime初始值置为首次运行的Daemon实例的机器时间，而每次轮询时都会对比当前Daemon实例的机器时间与CurrentTime的差别，故机器时间出错可能会影响任务的正常调度。这里考虑到现网机器均有时间校正脚本在跑，这个问题基本可以忽略。

2.本系统的架构对事件中心构成了强依赖。定时器的可用性和可靠性依赖于事件中心的可用性和可靠性。虽然目前事件中心的可用性和可靠性都非常高，但如果要考虑所有异常情况，则事件中心的短暂不可用、或者对于订阅者消息出队的延迟和堆积，都是需要正视的问题。一个解决方案是使用MQ做双链路的消息投递，解决对于事件中心单点依赖的问题。

结语

这里的定时器服务目前仅用于支持境外的定时器需求，调用量级尚不大，已可满足业务基本要求。如果要支撑更高的任务量级，还需要做更多的思考和优化。随时欢迎大家和和我交流探讨。

参考文章

异步代扣可靠事件中心定时器优化

有赞延迟队列设计

TDMQ

自研延时队列设计与实现

【技术交流】你真的知道怎么实现一个延迟队列吗?

如何实现延迟队列

QQ提醒——一个大流量订阅推送系统的实现思路

Go语言中时间轮的实现

境外支付延伸阅读

微信境外支付文集

加入我们

境外支付团队在不断追求卓越的路上寻找同路人。

[28605-微信支付境外支付前端开发工程师（深圳）]

[28605-微信支付境外支付后台开发工程师]

笔者在2021.4.22在腾讯技术工程知乎号就这个主题进行了直播分享，PPT见附件。

最后更新于 2021-04-27 12:09

如何实现一个分布式定时器-ruoyuli...若愚.pdf 查看全部（共27页）

附件： 如何实现一个分布式定时器-ruoyuliu刘若愚.pdf 预览

如果觉得我的文章对您有用，请随意赞赏

赏

5人已赞赏

分类： 01 - 后台开发 标签： 分布式(5) 后台(6) 定时器(11) 延迟队列(4) kv(1)

本文专属二维码，扫一扫还能分享朋友圈

想要微信公众号推广本文章？点击获取链接

相关阅读

- 空间视频后台总结
- 空间相册合成视频技术总结
- 腾讯相册小程序&看图小程序后台技术总结
- 你上传的图片安全吗？记一次隐私图片数据安全改造实践

我顶 (89) 收藏 (606)

选中文章内容可快速反馈

分享到 转载 收录 评论 (68) 反馈

大家评论

yuanlinsun 2021-03-22 11:12:34

高手

顶 (1) 回复 (1)

ruoyuliu (楼主) 2021-03-22 11:33:02

多多指教

回复

lorenzgli 2021-03-22 11:30:55

顶！！！

顶 (1) 回复 (1)

ruoyuliu (楼主) 2021-03-22 11:32:48

欢迎指导

回复

andrewmeng 2021-03-22 11:38:29

好文！学习了

顶 (1) 回复 (1)

ruoyuliu (楼主) 2021-03-22 11:39:06

多多指教

回复

markelzhang 2021-03-22 11:42:38

学习

顶 (1) 回复 (1)

ruoyuliu (楼主) 2021-03-22 11:44:20

欢迎markel大佬指导

回复

sauronyan 2021-03-22 11:43:14

学习

顶 (1) 回复 (1)

ruoyuliu (楼主) 2021-03-22 11:43:44

 互相学习

回复

fredxwang 2021-03-22 11:52:18

赞 

顶 (1) 回复 (1)

ruoyuliu (楼主) 2021-03-22 11:55:17

回复

weigangchen 2021-03-22 11:55:02

过期了，但没被处理的定时任务怎么处理?

顶 (1) 回复 (1)

ruoyuliu (楼主) 2021-03-22 12:47:03

这里首先是在业务需求阶段就要评估业务的量级，保证按照系统（包含定时器和事件中心）的处理能力是可以基本按时交付的，如果有拖延，在秒级的延迟下也是可以处理完成的。因此目前系统理论上最坏只会出现秒级延迟，由于时间轮会依次轮转，所有后面过期的任务还是会被调度。

回复

chansema 2021-03-22 12:22:09

学习学习

顶 (1) 回复 (1)

ruoyuliu (楼主) 2021-03-22 12:47:29

 互相学习

回复

santuliu 2021-03-22 12:48:51

强

顶 (1) 回复 (1)

ruoyuliu (楼主) 2021-03-22 12:51:15

 santu大佬多多指教

回复

jazzjia 2021-03-22 12:50:48

666

顶 (1) 回复 (1)

ruoyuliu (楼主) 2021-03-22 13:01:09

jazz大佬

回复

kaseyli 2021-03-22 13:01:56

学习一波~

顶 (1) 回复 (1)

ruoyuliu (楼主) 2021-03-22 13:04:19

互相学习 

回复

beardollye 2021-03-22 13:08:12

赞!

顶 (1) 回复 (1)

ruoyuliu (楼主) 2021-03-22 13:16:05

多多指教

回复

iversonluo 2021-03-22 16:25:16

学习到很多，赞！

顶 回复 (1)

ruoyuliu (楼主) 2021-03-22 16:29:49

多多指教！~

回复

antyan 2021-03-22 17:04:00

兄台为何如此优秀呢

顶 回复 (1)

ruoyuliu (楼主) 2021-03-22 17:05:46

过奖 还有很多需要学习的

回复

raineyan 2021-03-22 17:42:29

愚神出品，必属精品

顶 回复 (1)

ruoyuliu (楼主) 2021-03-22 18:03:25

雨神 见笑啦

回复

hayderchen 2021-03-22 20:03:09

有几个问题想请教下：

1. “使用方指定的start_time（时间戳）作为分表的uin”，所以修改定时任务时间是不是得带上之前的时间?

2. “分布式单实例容灾”中，Daemon 有多台机器，但只有一台机器去扫表，其他机器都不工作，只作为备机吗?

3. 在 Scheduler 扫表的时候，用的应该也是 start/limit 吧？也就是说如果同一秒有很多任务，会产生很多 select，增加扫表耗时，进而导致后续所有任务都拖慢执行。是不是有这种可能存在?

顶 (2) 回复 (2)

sheldonxxie 2021-03-23 01:11:44

我share下个人理解，lz可供参考下：
1 是的；因为每一个需要被执行的任务都有一个主键(或者单据ID)，可以通过这个单据ID得到一个时间，比如说"完成时间"，start_time肯定是在"完成时间"加上一个gap_time；有了单据ID，可以查到"完成时间"，然后通过gap_time的规则运算则可得到start_time。
2 是的；只有一台机器在work其他机器在等待它失败后再进行处理，作者说是通过 http://km.oa.com/group/24938/articles/show/434987?kmref=search&from_page=1&no=2 实现的。
3 从上文看，应该是串行扫实现的：能处理则处理，同时处理下一秒，否则sleep。如果保证串行处理的效率，作者使用了batch的方式，向可靠事件中心投递。

回复

ruoyuliu (楼主) 2021-03-23 10:11:48

@sheldonxxie @hayderchen 1.目前不支持修改任务 2.和sheldon说的一样 3.是的，和hayder说的一样，有这种可能存在，这里的确会使用到start/limit，我们目前的量级尚不大，但首先接入定时器时在业务需求阶段就要评估业务的量级，保证按照系统（包含定时器和事件中心）的处理能力是可以基本按时交付的，如果有拖延，在秒级的延迟下也是可以处理完成的，否则就可能需要考虑扩容的问题；你说的这个所有任务都拖慢也不是一定会发生，比如后面很多空的时间轮，则很快又可以达到一种收敛的状态。

回复

balisyin 2021-03-23 00:54:55

厉害

顶 回复

sheldonxxie 2021-03-23 01:03:00

写的不错，有几个问题请教一下：

1 分表数1000w（115天重复）和500w（57天重复）有啥区别?

数据过期是30天时，他们两者都大于过期天数，是否都满足要求呢?

2 微信支付的限频组件

====

限频组件解决的是不把下游怼爆的问题，怎么解决client端返回限频错误码的重试问题呢？是下面的方案解决吗?

1 插入时，客户端重试

2 回调时，事件中心重试

3 任务达到时即存入tablekv持久化存储，任务成功通知业务方才设置过期

====

上面我看到有回调逻辑，状态机是否应该是只有两个状态，那么回调一直失败怎么处理?

能不能设计成 inserted->process_over->callback_over

4 每次轮询时都会对比当前Daemon实例的机器时间与CurrentTime的差别；

====

在deamon机器时间调整的时候，如果超过1s，有没有可能那1秒任务被遗漏?

有没有为这种失效做额外的设计?

顶 (1) 回复 (7)

ruoyuliu (楼主) 2021-03-23 10:17:50

sheldon思考非常深入，赞！1.确实都满足，500w也可 2.是 3.回调一直失败，目前是会告警，人工排查问题。4.有可能，要看机器时间是落后还是超前，如某一daemon运行过程中机器时间超前，可能会导致任务提前调度。若时间落后，会导致任务延时调度（但由于daemon有多实例影响较为可控）。考虑到自动调整时间的机制，概率较小，目前没有额外设计。

回复

alineliu 2021-03-23 10:39:25

第4点，daemon切换机器执行时，是否可以往前回溯一段时间？事件中心也具备消息去重能力

回复

ruoyuliu (楼主) 2021-03-23 10:46:42

回溯是指?

回复

alineliu 2021-03-23 11:24:12

从 比当前时间小N秒 的时刻开始遍历任务列表，这样整个系统不同机器时间偏差在N秒内都不会遗漏任务

回复

ruoyuliu (楼主) 2021-03-23 13:28:40

@alineliu 的确是一个办法 需要识别切换机器这一行为

回复

sheldonxxie 2021-03-23 15:04:05

@ruoyuliu 谢谢lz耐心解答，我share下我的观点。

第2和第3点是同一个问题：下游失败了怎么处理，目前量少可以人肉处理，后面量大的话可能需要考虑额外的设计了；比如将状态多加一个，将订阅者的处理成功，也作为本流程的一个步骤等等。

第4点是一个异常恢复的问题，需要找到之前的上下文的问题。aline的方案是将重启后向指针回拨一段时间，这个方式的要求下游是幂等的（当然事件中心的订阅者本身就要求幂等）。或者有其他的方案：有一个全局标记，记录最后一次执行的位置，重启时从这个标记接着做，这样就不依赖某一个机器实例的时间了。

所以说方案有多种，在综合对比后，认清每个方案的优缺点，最后选一个代价较小，并且知道有哪些不足，或者待优化的特性就可以了。lz方案设计做的很不错，有很多的方案对比

👍

回复

ruoyuliu (楼主) 2021-03-23 15:09:07

观点都比较合理和中肯 从你的回复中学习到了很多新思路 目前也确实从量级的考虑上避免过度设计的问题 后续还会继续优化

回复

endyxu 2021-03-23 14:37:30

单表（某一秒）任务过大怎么处理呢？看方案是只能在一台机上扫？

顶 回复 (1)

ruoyuliu (楼主) 2021-03-23 14:47:57

分批按照start/limit的方式拉取。看方案是只能在一台机上扫--是的，否则会引起资源竞争的问题。

回复

zipchen 2021-03-23 15:25:51

请教下：实现定时器有很多种数据结构，各有优劣和适用场景，是否有做过比较呢？时间轮用于定时频繁被触发的场景，也就是每个分片都有定时任务。在实际应用中空分片的比率有多少呢？

顶 回复 (1)

ruoyuliu (楼主) 2021-03-23 15:38:58

在参考文章的第一篇《异步代扣可靠事件中心定时器优化》中有调研各种方案优缺点的对比，但不是以三种数据结构为维度的对比，以三种数据结构为维度的对比目前没有详细记录；目前我们接入的业务不多，空分片的比率会稍微有点高，上面有提到优化方案哈。

回复

bearluo 2021-03-23 17:04:42

被大佬cue到我的文档， 高兴ING～ 

顶 回复 (1)

ruoyuliu (楼主) 2021-03-23 17:25:23

bear大佬很高产，向你学习！ 

回复

davinliu 2021-03-23 21:48:23

https://git.code.oa.com/qidian_platform/e/etimer

楼主，我们已经做了。

顶 回复 (2)

davinliu 2021-03-23 21:48:45

https://git.code.oa.com/qidian_platform/e/etimer/blob/master/doc/%E6%9E%B6%E6%9E%84.md 架构图

回复

ruoyuliu (楼主) 2021-03-23 23:17:21

学习一下

回复

jamiegu 2021-04-02 09:21:35

微信基础设施好强

顶 回复 (1)

ruoyuliu (楼主) 2021-04-07 10:51:35

给做基础组件的同事点赞

回复

junkangchen 2021-04-07 11:55:26

请教一下，如果某一秒的任务中，部分投递到事件中心失败的任务是怎么处理的？ 我看文章提到异步Daemon扫出所有未交付的任务，是依赖这个来处理吗？

顶 回复 (1)

ruoyuliu (楼主) 2021-04-07 14:19:59

1.通过多次重试保障失败的任务可以被投递到事件中心；若真的有失败，则最后TimePointer的时间戳不会更新，下一次Daemon启动后仍将处理这一秒的任务，由事件中心保证去重。
2.异步Daemon可以作为兜底策略。保证百分之百无问题。

回复

ddavidzhang 2021-04-27 11:53:19

“分布式单实例容灾部署” 意味着没法水平扩容?

顶 回复 (3)

ruoyuliu (楼主) 2021-04-27 12:03:16

目前我们的业务量级暂时单实例可以支持 如果并发量更大 需要考虑的是“分布式多实例容灾”哈 对不同业务的task进行打散存储 用多个worker并行来处理 。

回复

mingdezuo 2021-06-17 22:45:52

@ruoyuliu 你的单实例如果挂了怎么办? 你的定时任务都是 cron 的还是倒计时类型的?

回复

ruoyuliu (楼主) 2021-06-18 10:13:39

单实例只是同时只有一个实例执行，实际有多实例，也就是文中提到的“多实例容灾”。一个实例挂了，另外一个实例会继续拿到任务，重新执行这一秒的任务。定时任务是指定时间戳执行任务这种类型的。

回复

stannhuang 2021-06-25 17:19:56

666

顶 回复

bondlin 2021-06-28 17:44:34

很荣幸之前写的文章被引用了 😁

顶 回复 (1)

ruoyuliu (楼主) 2021-06-28 18:14:33

😁互相学习

回复

zenlu 2021-06-28 19:36:02

老铁，有代码开源出来吗?

顶 回复 (1)

ruoyuliu (楼主) 2021-06-28 20:41:27

😁实现是基于微信的研发环境哈 可能有一定的依赖项

回复

tengixu 2021-06-29 10:05:32

优秀!

顶 回复 (1)

ruoyuliu (楼主) 2021-06-29 10:26:19

回复

monadliu 2021-06-29 10:46:47

补充一个，定时器也有用红黑树实现的(Nginx).

顶 回复 (1)

ruoyuliu (楼主) 2021-06-29 11:22:22

👍👍学习了

回复

切换到更多功能

发表评论

[922/963/741 ms]