20 | 存储和并发：万人群聊系统设计中的几个难点

2019-10-11 袁武林

即时消息技术剖析与实战 进入课程 >

讲述：袁武林
时长 15:20 大小 14.04M

你好，我是袁武林。

我在场景篇的第 10 讲“自动智能扩缩容：直播互动场景中峰值流量的应对”中，分析了直播互动场景中，容易出现瓶颈的原因主要在于：“直播间人数多，短时间内活跃度高，消息的扇出量巨大”。

那么，对于同样属于多人互动的群聊场景来说，虽然在“群人数”等方面与高热度的直播间相比要少一些，但由于同时开播的直播间数量一般不会太多，所以群在数量上的总体量级相对要大得多，可能上百万个群同时会有消息收发的情况发生。因此，在整体的流量方面，群聊场景的消息扇出也是非常大的。

而且和直播互动场景不太一样的是，直播互动中，热度高的直播活动持续时间都比较短，可以借助上云，来进行短时间的扩容解决，成本方面也比较可控；但群聊的场景，一般是流量总量高，但是峰值没有那么明显，靠扩容解决不太现实。因此，更多地需要我们从架构和设计层面来优化。

今天，我们就一起从架构设计层面，来聊一聊万人群聊系统中的几个难点，以及相应的解决方案。

群聊消息怎么存储?

首先来看一看群聊消息存储的问题。

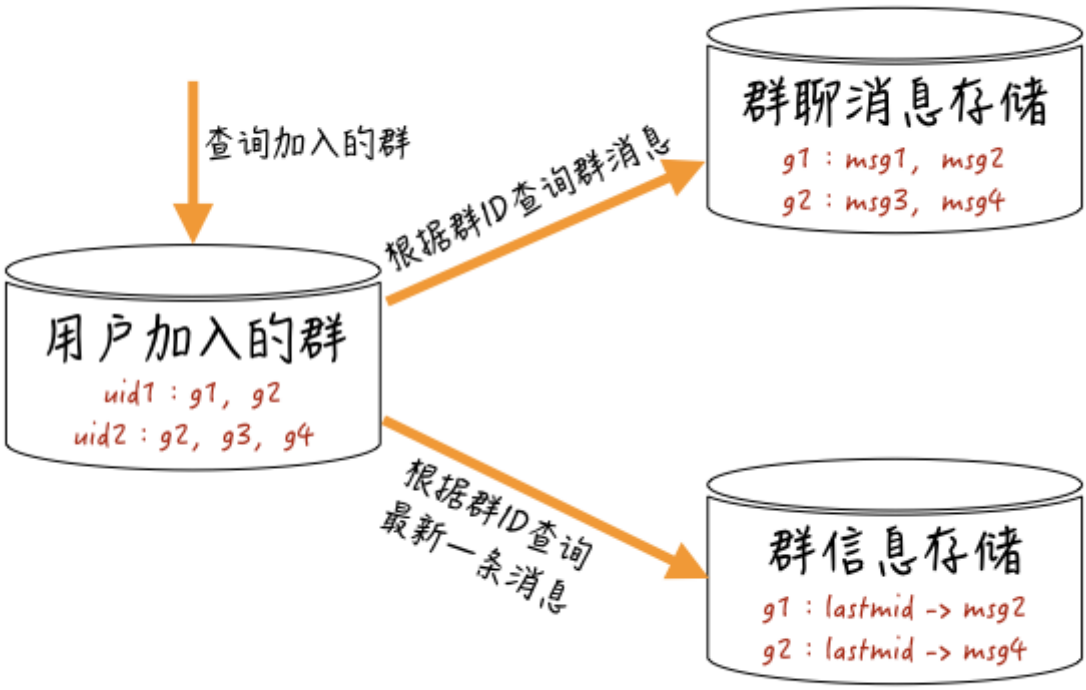
关于点对点聊天场景，我在第 2 课“消息收发架构：为你的 App，加上实时通信功能”中也有讲到：我们在一条消息发出后，会针对消息收发的双方，各自存储一条索引，便于双方进行查询、删除、撤回等操作。

那么，对于群聊消息来说，是不是也需要给群里的每一个用户，都存储一条消息索引呢?

这里需要注意的是：对于点对点聊天来说，针对消息收发双方进行用户维度的索引存储，能便于后续会话维度的消息查看和离线消息的获取，但如果群聊场景也采取这种方式，那么假设一个群有一万个人，就需要针对这一万个人都进行这一条消息的存储，一方面会使写入并发量巨大，另一方面也存在存储浪费的问题。

所以，业界针对群聊消息的存储，一般采取“读扩散”的方式。也就是一条消息只针对群维度存储一次，群里用户需要查询消息时，都通过这个群维度的消息索引来获取。

用户查询群聊消息的大概流程，你可以参考下图：



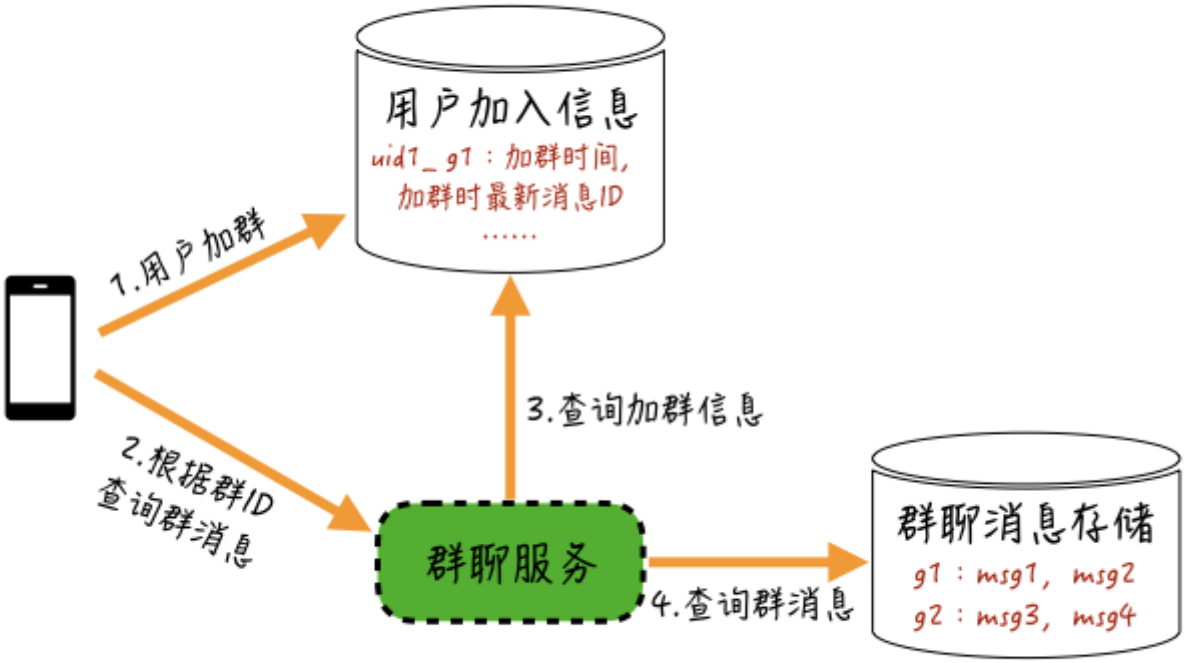
系统先查询这个用户加入的所有群，根据这些群的最新一条消息的 ID（消息 ID 与时间相关），或者最新一条消息的产生时间，来进行“最近联系人”维度的排序，再根据这些群 ID 获取每个群维度存储的消息。

怎么保证新加入群的用户只看到新消息?

群聊用户共用群维度的消息存储，能大幅降低用户维度消息的写入。

但这里有一个问题：如果群消息是共享的，怎么保证新加入群的用户看不到加群前的群聊消息呢?

解决这个问题其实比较简单，你可以采取下图这个方案：



我们只需要在用户加群的时候，记录一个“用户加群的信息”，把用户加群时间、用户加群时该群最新一条消息的 ID 等信息存储起来，当用户查询消息时，根据这些信息来限制查询的消息范围就可以了。

单个用户删除消息怎么办？

除了新加群用户消息查询范围的问题，群消息共享存储方案在实现时，还有一个比较普遍的问题：如果群里的某一个用户删除了这个群里的某条消息，我们应该怎么处理？

首先，由于群消息是共用的，我们肯定不能直接删除群消息索引中的记录。

一个可行的办法是：在用户删除消息的时候，把这条被删除消息加入到当前用户和群维度的一个删除索引中；当用户查询消息时，我们对群维度的所有消息，以及对这个“用户和群维度”的删除索引进行聚合剔除就可以了。

同样的处理，你还可以用在其他一些私有类型的消息中。比如，只有自己能看到的一些系统提示类消息等。

未读数合并变更

解决了群聊消息存储的问题，还有一个由于群聊消息高并发扇出而引起的问题。

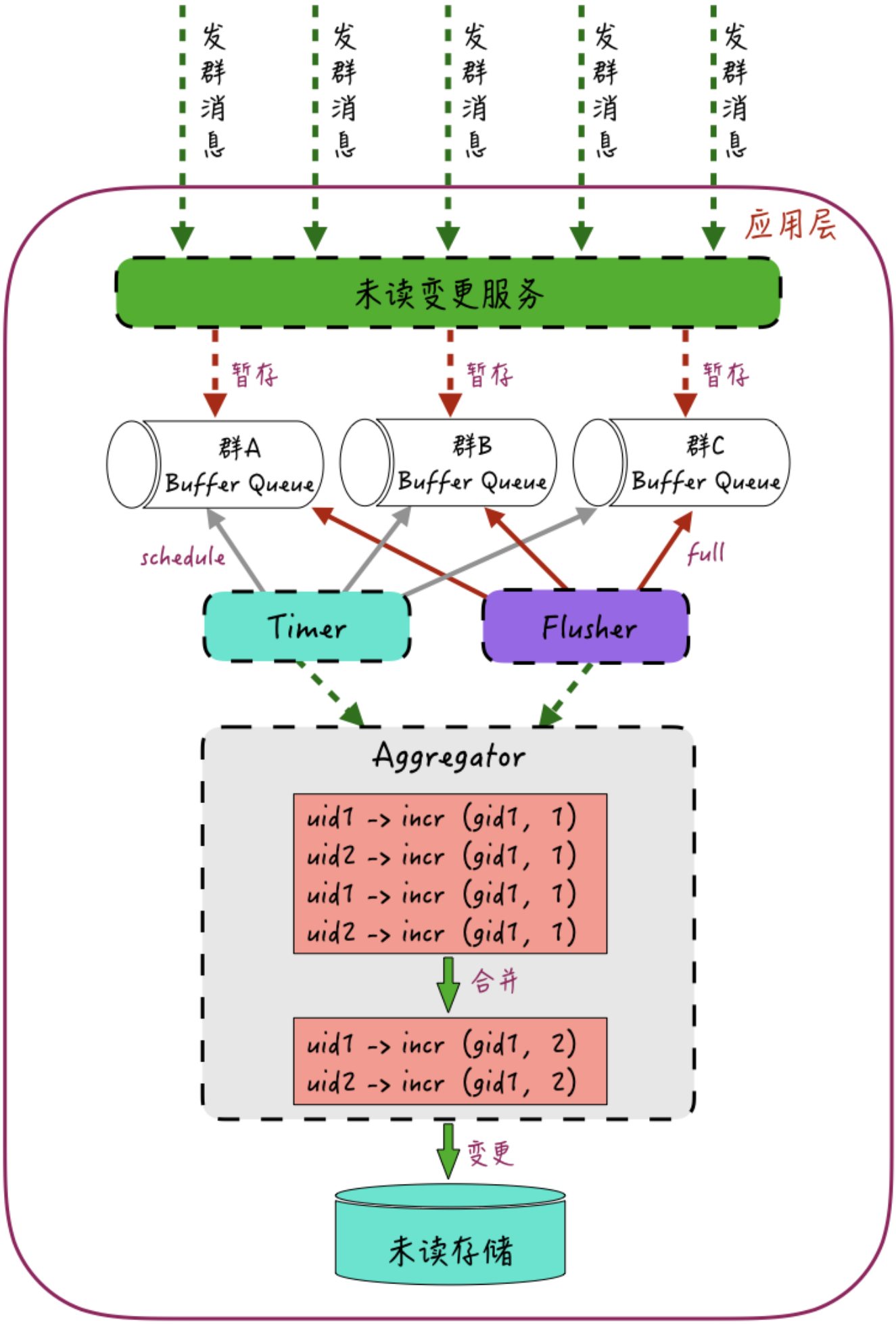
我在“07 | 分布式锁和原子性：你看到的未读消息提醒是真的吗？”这一篇课程中讲到过：针对每一个用户，我们一般会维护两个未读数，用于记录用户在某个群的未读消息数和所有未读数。

也就是说，当群里有人发言时，我们需要对这个群里的每一个人都进行“加未读“操作。因此，对于服务端和未读数存储资源来说，整体并发的压力会随着群人数和发消息频率的增长而成倍上升。

以一个 5000 人的群为例：假设这个群平均每秒有 10 个人发言，那么每秒针对未读资源的变更 QPS 就是 5w；如果有 100 个这样的群，那么对未读资源的变更压力就是 500w，所以整体上需要消耗的资源是非常多的。

解决这个问题的一个可行方案是：在应用层对未读数采取**合并变更**的方式，来降低对存储资源的压力。

合并变更的思路大概如下图：


发群消息
发群消息
发群消息
发群消息
发群消息
应用层
未读变更服务
暂存
暂存
暂存
群A Buffer Queue
群B Buffer Queue
群C Buffer Queue
schedule
full
Timer
Flusher
Aggregator
uid1 -> incr (gid1, 1)
uid2 -> incr (gid1, 1)
uid1 -> incr (gid1, 1)
uid2 -> incr (gid1, 1)
合并
uid1 -> incr (gid1, 2)
uid2 -> incr (gid1, 2)
变更
未读存储

未读变更服务接收群聊的加未读请求，将这些加未读请求按照群 ID 进行归类，并暂存到群 ID 维度的多个“暂存队列”中；这些“暂存队列”的请求会通过一个 Timer 组件和一个 Flusher 组件来负责处理。

Timer 组件负责定时刷新这些队列中的请求，比如，每一秒从这些“暂存队列”取出数据，然后交给 Aggregator 进行合并处理；Flusher 组件则会根据这些“暂存队列”的长度来进行刷新，比如，当队列长度到达 100 时，Flusher 就从队列中取出数据，再交给 Aggregator 来进行合并处理。

所以，Timer 和 Flusher 的触发条件是：这些队列的请求中有任意一个到达，均会进行刷新操作。

提交给 Aggregator 的加未读请求会进行合并操作。比如针对群里的每一个用户，将多个归属于该群的加未读请求合并成一个请求，再提交给底层资源。

如上图所示，群 ID 为 gid1 里的用户 uid1 和 uid2，通过合并操作，由 4 次加未读操作 incr 1 合并成了各自一条的加未读操作 incr 2。

通过这种方式，就将加未读操作 QPS 降低了一半。如果每秒群里发消息的 QPS 是 10 的话，理论上我们通过这种“合并”的方式，能将 QPS 降低到 1/10。

当然，这里需要注意的是：由于加未读操作在应用层的内存中会暂存一定时间，因此会存在一定程度的加未读延迟的问题；而且如果此时服务器掉电或者重启，可能会丢失掉一部分加未读操作。

为了提升“合并变更”操作的合并程度，我们可以通过群 ID 哈希的方式，将某一个群的所有未读变更操作都路由到某一台服务器，这样就能够提升最终合并的效果。

离线 Buffer 只存消息 ID

通过“合并变更”，我们解决了万人群聊系统中，未读数高并发的压力问题。

接下来我们看一下，在离线推送环节中，针对群聊场景还有哪些可优化的点。

我在第 9 课“分布式一致性：让你的消息支持多终端漫游？”中有讲到，为了解决用户离线期间收不到消息的问题，我们会在服务端按照接收用户维度，暂存用户离线期间的消息，

等该用户下次上线时再进行拉取同步。

这里的离线 Buffer 是用户维度的，因此对于群聊中的每一条消息，服务端都会在扇出后进行暂存。

假设是一个 5000 人的群，一条消息可能会暂存 5000 次，这样一方面对离线 Buffer 的压力会比较大，另外针对同一条消息的多次重复暂存，对资源的浪费也是非常大的。

要解决多次暂存导致离线 Buffer 并发压力大的问题，一种方案是可以参考“未读数合并变更”的方式，对群聊离线消息的存储也采用“合并暂存”进行优化，所以这里我就不再细讲了。

另一种解决方案是：我们可以对群聊离线消息的暂存进行限速，必要时可以丢弃一些离线消息的暂存，来保护后端资源。

因为通过“版本号的链表机制”，我们可以在用户上线时发现“离线消息”不完整的问题，然后再从后端消息存储中重新分页获取离线消息，从而可以将一部分写入压力延迟转移到读取压力上来。

不过这里你需要注意的是：这种降级限流方式存在丢失一些操作信令的问题，是有损降级，所以非必要情况下尽量不用。

另外，针对群聊消息重复暂存的问题，我们可以只在离线 Buffer 中暂存“消息 ID”，不暂存消息内容，等到真正下推离线消息的时候，再通过消息 ID 来获取内容进行下推，以此优化群聊消息对离线 Buffer 资源过多占用的情况。

离线消息批量 ACK

在群聊离线消息场景中，还有一个相对并发量比较大的环节就是：离线消息的 ACK 处理。

我在“04 | ACK 机制：如何保证消息的可靠投递？”这节课中讲到，我们会通过 ACK 机制来保证在线消息和离线消息的可靠投递。但是对于群的活跃度较高的情况来说，当用户上线时，服务端针对这个群的离线消息下推量会比较大。

以微博场景中的超大规模的粉丝群为例：本来群内的用户就已经比较活跃了，如果该群隶属的明星突然空降进来，可能会导致大量离线用户被激活，同一时间会触发多个用户的离线消

息下推和这些离线消息的 ACK；针对离线消息接收端的 ACK 回包，服务端需要进行高并发的处理，因而对服务端压力会比较大。

但实际上，由于群聊离线消息的下推发生在用户刚上线时，这个时候的连接刚建立，稳定性比较好，一般消息下推的成功率是比较高的，所以对 ACK 回包处理的及时性其实不需要太高。

因此，一种优化方案是：**针对离线消息接收端进行批量 ACK。**

参照 TCP 的 Delay ACK（延迟确认）机制，我们可以在接收到离线推送的消息后，“等待”一定的时间，如果有其他 ACK 包需要返回，那么可以对这两个回包的 ACK 进行合并，从而降低服务端的处理压力。

需要注意的是：接收端的 Delay ACK，可能会在一定程度上加剧消息重复下推的概率。比如，ACK 由于延迟发出，导致这时的服务端可能会触发超时重传，重复下推消息。

针对这个问题，我们可以通过接收端去重来解决，也并不影响用户的整体体验。

不记录全局的在线状态

群聊场景下的超大消息扇出，除了会加大对离线消息的资源消耗，也会对消息的在线下推造成很大的压力。

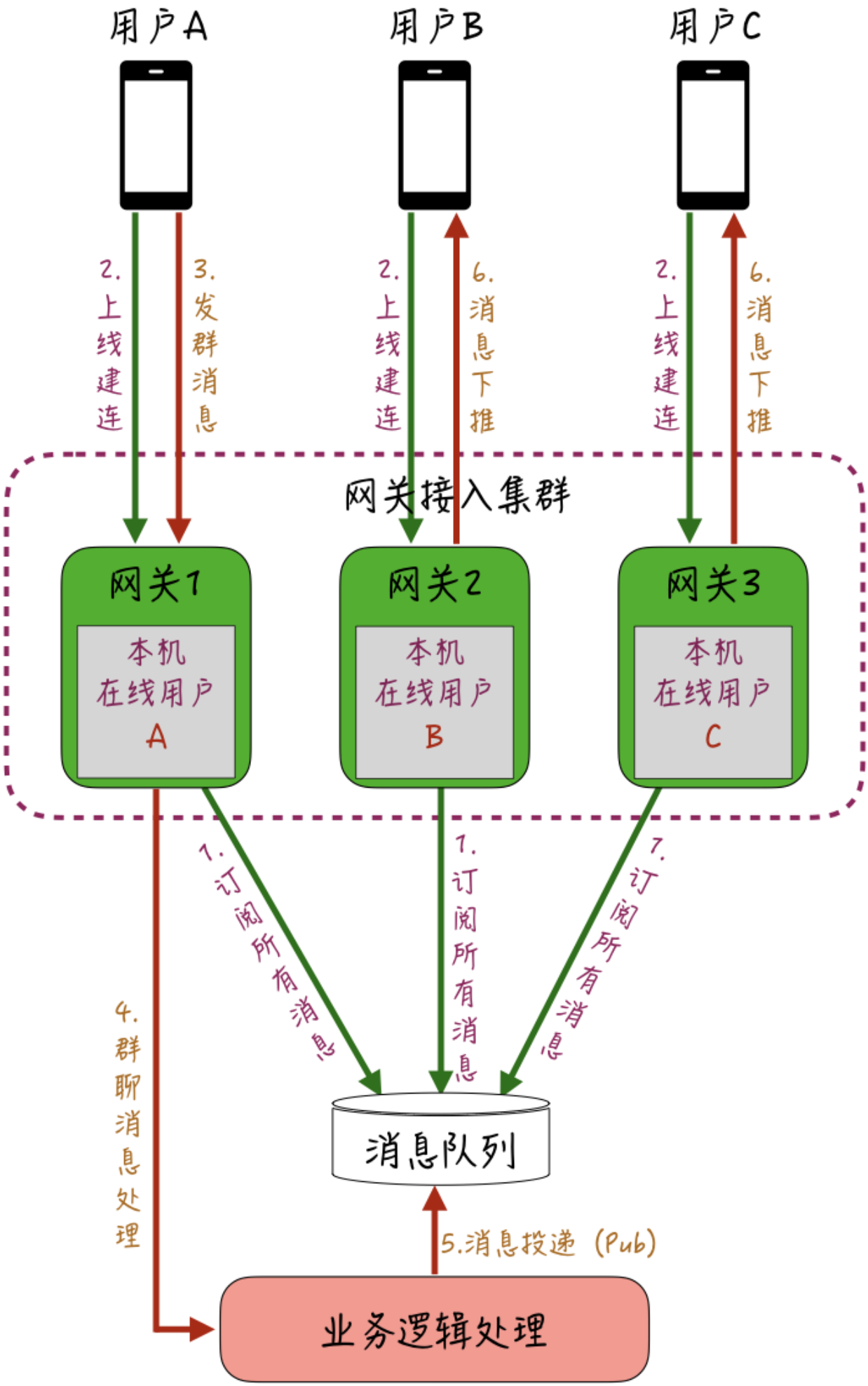
举个例子：在点对点聊天场景中，我们通常会在用户上线时，记录一个“用户连接所在的网关机”的在线状态，而且为了和接入服务器解耦，这个在线状态一般会存储在中央资源中；当服务端需要下推消息时，我们会通过这个“中央的在线状态”来查询接收方所在的接入网关机，然后把消息投递给这台网关机，来进行最终消息的下推。

在群聊场景中，很多实现也会采用类似方式进行在线消息的精准下推，这种方案在群人数较少的时候是没问题的，但是当群成员规模很大时，这种方式就会出现瓶颈。

一个瓶颈在于，用户上线时对“在线状态”的写入操作；另一个瓶颈点在于，服务端有消息下推时，对“在线状态”的高并发查询。

因此，针对万人群聊的场景，我们可以采取类似直播互动中的优化方式，不维护全局的中央“在线状态”，而是让各网关机“自治”，来维护接入到本机的连接和群的映射。你可以

参考下图所示的实现过程：



比如同一个群的用户 A、B、C，分别通过网关机 1、2、3 上线建立长连，处理建连请求时，网关机 1、2、3 会分别在各自的本地内存维护当前登录的用户信息。

上线完成后，用户 A 在群里发了一条消息，业务逻辑处理层会针对这条消息进行处理，查询出当前这条消息所归属群的全部用户信息，假设查询到这个群一共有 3 人，除去发送方用户 A，还有用户 B 和用户 C。

然后业务逻辑处理层把消息扇出到接收人维度，投递到全局的消息队列中；每一台网关机在启动后都会订阅这个全局的 Topic，因此都能获取到这条消息；接着，各网关机查询各自本地维护的“在线用户”的信息，把归属本机的用户的消息，通过长连下推下去。

通过这种方式，消息下推从“全局的远程依赖”变成了“分片的本地内存依赖”，性能上会快很多，避免了服务端维护全局在线状态的资源开销和压力。

小结

今天的课程，我主要是分析了一些在万人群聊场景中比较突出和难解决的问题，并给出了针对性的应对方案。比如以下几种：

- 针对群聊消息的存储，我们可以从点对点的**“写扩散”优化成“读扩散”**，以解决存储写入并发大和资源开销大的问题；
- 针对高热度的群带来的“高并发未读变更”操作，我们可以通过**应用层的“合并变更”**，来缓解未读资源的写入压力；
- 对于离线消息的优化，我们只需要存储消息 ID，避免重复的消息内容存储浪费离线 Buffer 资源，还可以参考 TCP 的 Delay ACK 机制，**在接收方层面进行批量 ACK**，降低服务端的处理并发压力；
- 对于单聊场景中依赖“中央全局的在线状态”，来进行消息下推的架构瓶颈，我们可以在群聊场景中优化成**“网关机本地自治维护”**的方式，以此解决高并发下推时，依赖全局资源容易出现瓶颈的问题，从而提升群聊消息在线下推的性能和稳定性。

针对大规模群聊系统的架构优化，一直是即时消息场景中非常重要和必要的部分，也是体现我们架构能力和功底的环节。

今天课程中介绍的针对万人群聊系统优化的一些应对方案，很多都具备普适性，比如“未读合并变更”的方案，实际上也能应用在很多有写入瓶颈的业务上（如 DB 的写入瓶颈），在

微博的线上业务中，目前也被大范围使用。你也可以看一看，自己的业务中是否也有类似可优化的场景，可以尝试来参考这个方案进行优化。

最后给大家留一个思考题：**点对点消息的在线下推，也适合采用“网关机自治维护本地在线状态”的方式吗？说说你的看法吧。**

以上就是今天课程的内容，欢迎你给我留言，我们可以在留言区一起讨论，感谢你的收听，我们下期再见。

上一篇　19 | 端到端Trace：消息收发链路的监控体系搭建

下一篇　21 | 期末实战：为你的简约版IM系统，加上功能

精选留言 (9)

写留言

clip
2019-10-11

思考题：

看情况考虑。
不好的地方：
所有网关机都要从消息队列消费这条数据，但最终只有一台机器处理，比较浪费处理资源。对这条消息而言本来可以直接下推，结果要多经过了网关机，实时性可能会受到大...
展开

1

clip
2019-10-11

群聊的私有类型消息是不是要采用补齐而不是剔除的逻辑？因为加了一条仅自己可见的消息给其他人都创建不可见索引就得不偿失了。
展开

2 1

zhxh
2019-10-11

如果通过订阅的方式，那么这条消息应该也需要把群成员列表带给网关吧，否则网关怎么筛选，可是万人群成员比较多，会导致这个消息包比较大，如果不带成员列表信息到网关，那么要求用户在和网关建立连接的时候，就要把自己加入的群列表信息带过来，绑定到网关，这样逻辑似乎和网关耦合比较严重，老师能详细解释一下么
展开

1 1

黄海
2019-10-14

请袁老师看一下这样的方案行吗：把各台网关机上的在线群(在线群成员数>0）的群 uid 作为 key，把网关机的消息队列 topic 作为 value，写入 redis set 中，发送群消息时，根据 key 群 uid 从 redis set 中查出群成员分布在哪些网关机上，然后通过消息队列，精准的向这些网关机推送群聊消息
展开

墙角儿的花
2019-10-11

老师 用websocket做长链接通信在网络较好情况下没什么问题 但在弱网下如2g 3g下就会频繁掉线，但是微信却做的这么好，地铁里仍然很稳定，它走的绝对不是websocket，它怎么做到的这么稳定通畅呢？有什么资料可以查看学习吗。
展开

2

我行我素
2019-10-11

老师，想请问下：在用户删除消息的时候，把这条被删除消息加入到当前用户和群维度的一个删除索引中，这一步不是很明白

展开

1

秋日阳光
2019-10-11

万人群聊，系统是维护了万个TCP链接吗?

展开

2

2019-10-11

老师上午好、有些疑惑想跟您确认一下，消息多终端漫游，为解决用户离线期间收不到消息的问题，我们会在服务端按照接收用户维度，暂存用户离线期间的消息，等该用户下次上线时再进行拉取同步。
1、这里的同步，实际上是把服务端的消息同步到客户端，客户端也保存这些消息吧?
2、假如用户重新安装了APP，把客户端保存的数据也清空了，像这种情况下，一般处理...

展开

墙角儿的花
2019-10-11

群聊信息如果不采用收发箱的方式存储，碰见钉钉这种需要记录查看一条群消息哪些成员已阅，哪些未阅，就无能为力了

展开

2