5亿用户如何高效沟通？钉钉首次对外揭秘即时消息服务DTIM

InfoQ 2022-08-09 08:44 发表于北京

作者丨陈万红，张世梁，杨世泉，余秋宇，谈云兵
策划丨褚杏娟

这是钉钉第一次对外揭秘 DTIM(DingTalk IM，钉钉即时消息服务)。我们从设计原理到技术架构、从最底层存储模型到跨地域的单元化，全方位地展现了 DTIM 在实际生产中遇到各种挑战与解决方案，期望为企业级 IM 的建设贡献一臂之力。

钉钉已经有 2100 万 + 组织、5 亿 + 注册用户在使用。DTIM 为钉钉用户提供即时消息服务，用于组织内外的沟通，这些组织包括公司、政府、学校等，规模从几人到百万人不等。DTIM 有着丰富的功能，单聊、各种类型的群聊、消息已读、文字表情、多端同步、动态卡片、专属安全和存储等等。同时钉钉内部很多业务模块，比如文档、钉闪会、Teambition、音视频、考勤、审批等，每个业务都在使用 DTIM，用于实现业务流程通知、运营消息推送、业务信令下发等。每个业务模块对于 DTIM 调用的流量峰值模型各有差别，对可用性要求也不尽相同。DTIM 需要能够面对这些复杂的场景，保持良好的可用性和体验，同时兼顾性能与成本。

通用的即时消息系统对消息发送的成功率、时延、到达率有很高的要求，企业 IM 由于 ToB 的特性，在数据安全可靠、系统可用性、多终端体验、开放定制等多个方面有着极致的要求。构建稳定高效的企业 IM 服务，DTIM 主要面临的挑战是：

- 企业 IM 极致的体验要求对于系统架构设计的挑战，比如数据长期保存可漫游、多端数据同步、动态消息等带来的数据存储效率和成本压力，多端数据同步带来的一致性问题等；

- 极限场景冲击、依赖系统错误带来的可用性问题：比如超大群消息，突发疫情带来的线上办公和线上教学高并发流量，系统需要能够应对流量的冲击，保障高可用；同时在中间件普遍可用性不到 99.99% 的时候，DTIM 服务需要保障核心功能的 99.995% 的可用性；

- 不断扩大的业务规模，对于系统部署架构的挑战：比如持续增长的用户规模，突发事件如席卷全球的疫情，单地域架构已经无法满足业务发展的要求。

DTIM 在系统设计上，为了实现消息收发体验、性能和成本的平衡，设计了高效的读写扩散模型和同步服务，以及定制化的 NoSQL 存储。通过对 DTIM 服务流量的分析，对于大群消息、单账号大量的消息热点以及消息更新热点的场景进行了合并、削峰填谷等处理；同

时核心链路的应用中间件的依赖做容灾处理，实现了单一中间件失败不影响核心消息收发，保障基础的用户体验。

在消息存储过程中，一旦出现存储系统写入异常，系统会回旋缓冲重做，并且在服务恢复时，数据能主动向端上同步。随着用户数不断增长，单一地域已无法承载 DTIM 的容量和容灾需求，DTIM 实现了异地多单元的云原生的弹性架构。在分层上遵从的原则为重云轻端：业务数据计算、存储、同步等复杂操作尽量后移到云端处理，客户端只做终态数据的接收、展示，通过降低客户端业务实现的复杂度，最大化地提升客户端迭代速度，让端上开发可以专注于提升用户的交互体验，所有的功能需求和系统架构都围绕着该原则做设计和扩展。

以下我们将对 DTIM 做更加详细的介绍。在第 1 章，我们介绍了 DTIM 的核心模型设计；第 2 章介绍了针对 IM 特点将计算下沉和与之对应的优化点；在第 3 章中介绍了同步服务，实现 IM 和其他业务的多端同步能力；在第 4 章主要介绍高可用的设计，首先是系统自我防护，之后是系统的弹性扩展能力与异地容灾设计。

模型设计

低延迟、高触达、高可用一直是 DTIM 设计的第一原则，依据这个原则在架构上 DTIM 将系统拆分为三个服务做能力的承载：

- **消息服务：**负责 IM 核心消息模型和开放 API，IM 基础能力包括消息发送、单聊关系维护、群组元信息管理、历史消息拉取、已读状态通知、IM 数据存储以及跨地域的流量转发。

- **同步服务：**负责用户消息数据以及状态数据的端到端同步，通过客户端到服务端长连接通道做实时的数据交互，当钉钉各类设备在线时 IM 及上游各业务通过同步服务做多端的数据同步，保障各端数据和体验一致。

- **通知服务：**负责用户第三方通道维护以及通知功能，当钉钉的自建通道无法将数据同步到端上时，通过三方提供的通知和透传能力做消息推送，保障钉钉消息的及时性和有效性。

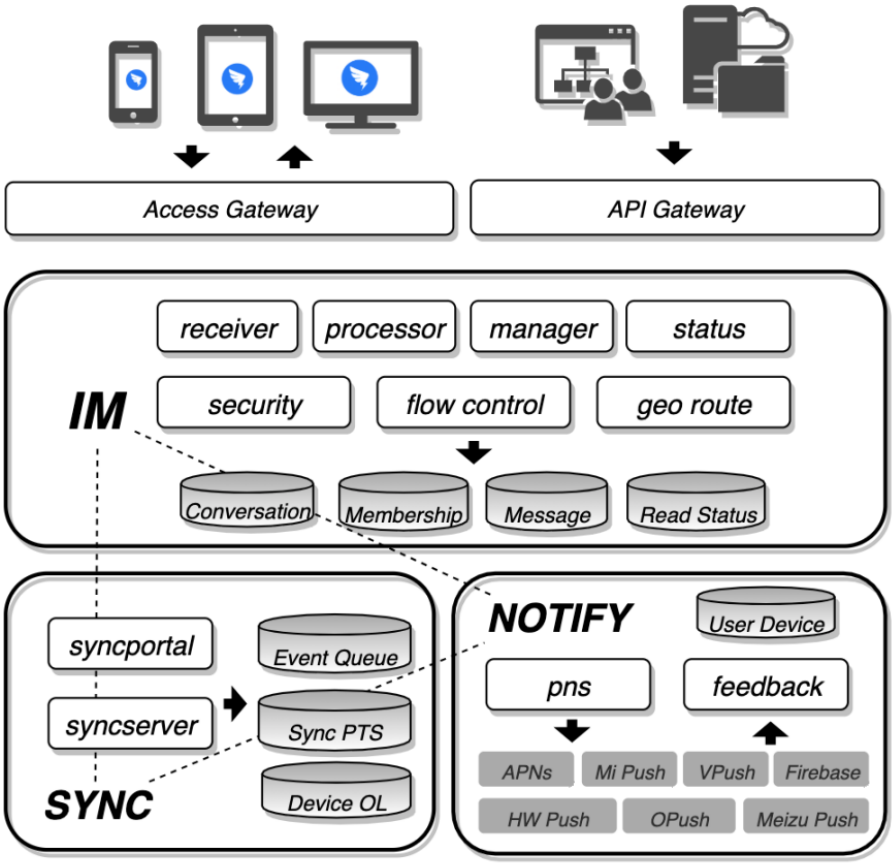

Fig.1:DTIM architecture

同步服务和通知服务除了服务于消息服务，也面向其他钉钉业务比如音视频、直播、Ding、文档等多端 (多设备) 数据同步。上图展示了 DTIM 系统架构，接下来详细介绍消息收发链路。

消息发送：消息发送接口由 Receiver 提供，钉钉统一接入层将用户从客户端发送的消息请求转发到 Receiver 模块，Receiver 校验消息的合法性（文字图片等安全审核、群禁言功能是否开启或者是否触发会话消息限流规则等）以及成员关系的有效性（单聊校验二者聊天、群聊校验发送者在群聊成员列表中），校验通过后为该消息生成一个全局唯一的 MessageId 随消息体以及接收者列表打包成消息数据包投递给异步队列，由下游 Processor 处理。消息投递成功之后，Receiver 返回消息发送成功的回执给客户端。

消息处理：Processor 消费到 IM 发送事件首先做按接收者的地域分布（DTIM 支持跨域部署， Geography，Geo）做消息事件分流，将本域用户的消息做本地存储入库（消息体、接收者维度、已读状态、个人会话列表红点更新），最后将消息体以及本域接收者列表打包为 IM 同步事件通过异步队列转发给同步服务。

消息接收：同步服务按接收者维度写入各自的同步队列，同时查取当前用户设备在线状态，当用户在线时捞取队列中未同步的消息，通过接入层长连接推送到各端。当用户离线时，打包消息数据以及离线用户状态列表为 IM 通知事件，转发给通知服务的 PNS 模块，PNS 查询离线设备做三方厂商通道推送，至此一条消息的推送流程结束。

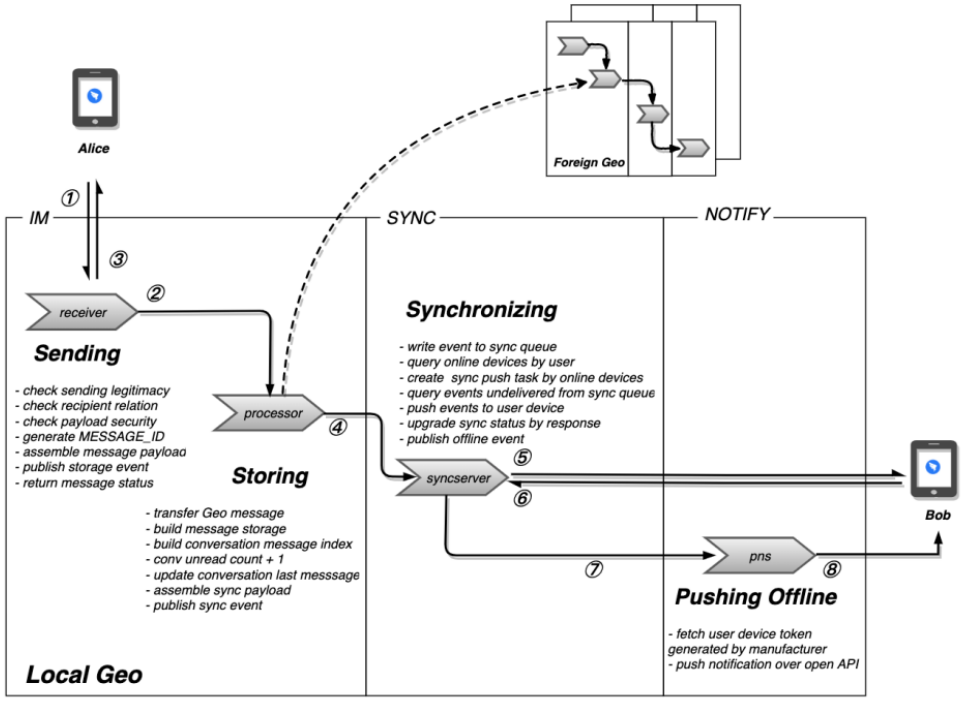

Fig. 2: DTIM message processing architecture

存储模型设计

了解 IM 服务最快的途径就是掌握它的存储模型。业界主流 IM 服务对于消息、会话、会话与消息的组织关系虽然不尽相同，但是归纳起来主要是两种形式：写扩散读聚合、读扩散写聚合，所谓读写扩散其实是定义消息在群组会话中的存储形式，以下图所示：

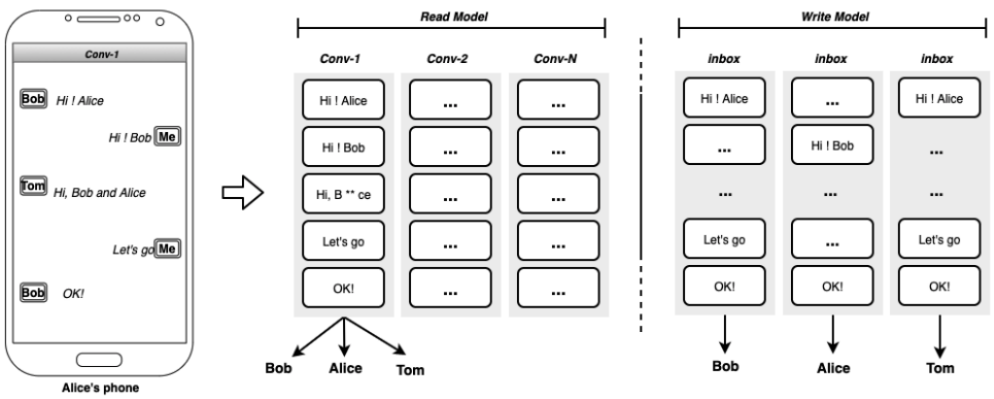

Fig. 3: Read model & Write model

- **读扩散的场景**，消息归属于会话，对应到存储中相当于有张 conversation_message 的表存储着该会话产生的所有消息 (cid->msgid->message，cid 会话 ID、msgid 消息 ID、message 消息)，这样实现的好处是消息入库效率高，只存储会话与消息的绑定关系即可。

- **写扩散的场景**，会话产生的消息投递到类似于个人邮件的收件箱，即 message_inbox 表，存储个人的所有消息（uid->msgid->message， uid 用户 ID、msgid 消息 ID、

message 消息），基于这种实现，会话中的每条消息面向不同的接收者可以呈现出不同状态。

DTIM 对 IM 消息的及时性、前后端存储状态一致性要求异常严格，特别对于历史消息漫游的诉求十分强烈，当前业界 IM 产品对于消息长时间存储和客户端历史消息多端漫游都做得不尽如人意，主要是存储成本过高。因此在产品体验与投入成本之间需要找到一个平衡点。

采用读扩散，在个性化的消息扩展及实现层面有很大的约束。采用写扩散带来的问题也很明显：一个群成员为 N 的会话一旦产生消息就会扩散 N 条消息记录，如果在消息发送和扩散量较少的场景，这样的实现相比于读扩散落地更为简单，存储成本也不是问题，但是 DTIM 会话活跃度超高，一条消息的平均扩散比可以达到 1：30，超大群又是企业 IM 最核心的沟通场景，如果采用完全写扩散所带来存储成本问题势必制约钉钉业务发展。

所以，在 DTIM 的存储实现上，钉钉采取了混合的方案，将读扩散和写扩散针对不同场景做了适配，最终在用户视角系统会做统一合并，如下图所示：

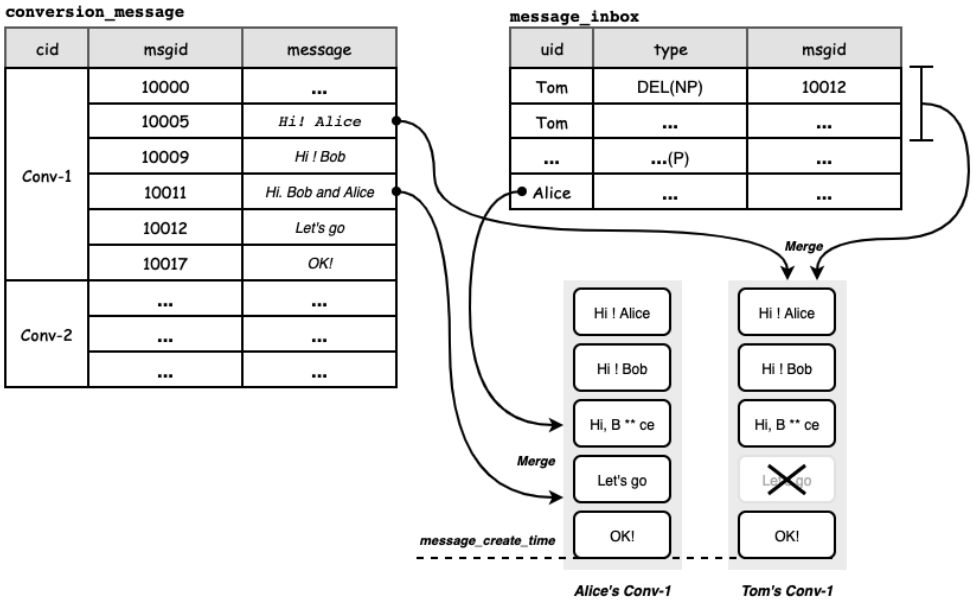

Fig. 4: DTIM Read-Write hybrid model

作为读扩散、写扩散方案的混合形式存在，用户维度的消息分别从 conversation_message 和 message_inbox 表中获取，在应用侧按消息发送时间做排序合并，conversation_message 表中记录了该会话面向所有群成员接收的普通消息 N（Normal），而 message_inbox 表在以下两种场景下被写入：

第一种是定向消息 P（Private，私有消息），群会话中发送的消息指定了接收者范围，那么会直接写入到接收者的 message_inbox 表中，比如红包的领取状态消息只能被红包发送者可见，那么这种消息即被定义为定向消息。

第二种是归属于会话消息的状态转换 NP（Normal to Private，普通消息转私有消息），当会话消息通过某种行为使得某些消息接收者的消息状态发生转换时，该状态会写入到 message_inbox 表中，比如用户在接收侧删除了消息，那么消息的删除状态会写入到

message_inbox 中，在用户拉取时会通过 message_inbox 的删除状态将 conversation_message 中的消息做覆盖，最终用户拉取不到自己已删除的消息。

当用户在客户端发起某个会话的历史消息漫游请求时，服务端根据用户上传的消息截止时间（message_create_time）分别从 conversation_message 表和 message_inbox 表拉取消息列表，在应用层做状态的合并，最终返回给用户合并之后的数据，N、P、NP 三种类型消息在消息个性化处理和存储成本之间取得了很好的平衡。

同步模型

推送模型

用户在会话中发出的消息和消息状态变更等事件是如何同步到端上呢？业界关于消息的同步模型的实现方案大致有三种：客户端拉取、服务端推送、服务端推送位点之后客户端拉取的推拉结合方案。

三种方案各有优劣，在此简短总结：

- 首先，客户端拉取方案的优点是该方案实施简单、研发成本低，是传统的 B/S 架构；劣势是效率低下，拉取间隔控制权在客户端，对于 IM 这种实时的场景，很难设置一个有效的拉取间隔，间隔太短对服务端压力大，间隔太长时效性差。

- 其次，服务端主动推送方案的优点是低延迟、能做到实时，最重要的主动权在服务端；劣势相对拉取方案，如何协调服务端和客户端的处理能力存在问题。

- 最后是推拉结合方案，这个方案整合拉和推的优点，但是方案更复杂，同时会比推的方案多一次 RTT，特别是在移动网络的场景下，不得不面临功耗和推送成功率的问题。

在 DTIM 的场景中，如上文所述，DTIM 相对传统 toC 的场景，有较明显的区别：

第一是对实时性的要求。在企业服务中，比如员工聊天消息、各种系统报警，又比如音视频中的共享画板，无不要求实时事件同步，因此需要一种低延时的同步方案。

第二是弱网接入的能力。在 DTIM 服务的对象中，上千万的企业组织涉及各行各业，从大城市 5G 的高速到偏远的山区弱网，都需要 DTIM 的消息能发送、能触达。对于复杂的网络环境，需要服务端能判断接入环境，并依据不同的环境条件调整同步数据的策略。

第三是功耗可控成本可控。在 DTIM 的企业场景中，消息收发频率比传统的 IM 多出一个数量级，在这么大的消息收发场景怎么保障 DTIM 的功耗可控，特别是移动端的功耗可控，是 DTIM 必须面对的问题。在这种要求下，就需要 DTIM 尽量降低 IO 次数，并基于不同的消息优先级进行合并同步，既能要保障实时性不被破坏，又要做到低功耗。

从以上三点可知，主动推的模型更适合 DTIM 场景，服务端主动推送，首先可以做到极低的延时，保障推送耗时在毫秒级别；其次是服务端能通过用户接入信息判断用户接入环境

好坏，进行对应的分包优化，保障弱网链路下的成功率；最后是主动推送相对于推拉结合来说，可以降低一次 IO，对 DTIM 这种每分钟过亿消息服务来说，能极大的降低设备功耗，同时配合消息优先级合并包的优化，进一步降低端的功耗。

虽说主动推送有诸多优势，但是客户端会离线，甚至客户端处理速度无法跟上服务端的速度，必然导致消息堆积，DTIM 为了协调服务端和客户端处理能力不一致的问题，支持 Rebase 的能力，当服务端消息堆积的条数达到一定阈值，触发 Rebase，客户端会从 DTIM 拉取最新的消息，同时服务端跳过这部分消息从最新的位点开始推送消息。DTIM 称这个同步模型为推优先模型（Preferentially-Push Model，PPM）。

在基于 PPM 的推送方案下，为了保障消息的可靠达到，DTIM 还做一系列优化。

第一，支持消息可重入，服务端可能会针对某条消息做重复推送，客户端需要根据 msgId 做去重处理，避免端上消息重复展示。

第二，支持消息排序，服务端推送消息特别是群比较活跃的场景，某条消息由于推送链路或者端侧网络抖动，推送失败，而新的消息正常推送到端侧，如果端上不做消息排序的话，消息列表就会发生乱序，所以服务端会为每条消息分配一个时间戳，客户端每次进入消息列表就是根据时间戳做排序再投递给 UI 层做消息展示。

第三，支持缺失数据回补，在某个极端情况下客户端群消息事件比群创建事件更早到达端上，此时端上没有群的基本信息消息也就无法展现，所以需要客户端主动向服务端拉取群信息同步到本地，再做消息的透出。

多端数据一致性

多端数据一致性问题一直是多端同步最核心的问题，单个用户可以同时在 PC、Pad 以及 Mobile 登录，消息、会话红点等状态需要在多端保持一致，并且用户更换设备情况下消息可以做全量的回溯。

基于上面的业务诉求以及系统层面面临的诸多挑战，钉钉自研了同步服务来解决一致性问题，同步服务的设计理念和原则如下：

- 统一消息模型抽象，对于 DTIM 服务产生的新消息以及已读事件、会话增删改、多端红点清除等事件统一抽象为同步服务的事件，
- 同步服务不感知事件的类型以及数据序列化方式。同步服务为每个用户的事件分配一个自增的 ID（注：这里非连续递增），确保消息可以根据 ID 做遍历的有序查询。
- 统一同步队列，同步服务为每个用户分配了一个 FIFO 的队列存储，自增 id 和事件串行写入队列；当有事件推送时，服务端根据用户当前各端在线设备状态做增量变更，将增量事件推送到各端。
- 根据设备和网络质量的不同可以做多种分包推送策略，确保消息可以有序、可靠、高效的发送给客户端。

上面介绍了 DTIM 的存储模型以及同步模型的设计与思考，在存储优化中，存储会基于 DTIM 消息特点，进行深度优化，并会对其中原理以及实现细节做深入分析与介绍；在同步机制中，会进一步介绍多端同步机制是如何保障消息必达以及各端消息一致性。

存储优化

DTIM 底层使用了表格存储作为消息系统的核心存储系统，表格存储是一个典型 LSM 存储架构，读写放大是此类系统的典型问题。LSM 系统通过 Major Compaction 来降低数据的 Level 高度，降低读取数据放大的影响。在 DTIM 的场景中，实时消息写入超过百万 TPS，系统需要划归非常多的计算资源用于 Compaction 处理，但是在线消息读取延迟毛刺依旧严重。

在存储的性能分析中，我们发现如下几个特点：

1. 6% 的用户贡献了 50% 左右的消息量，20% 的用户贡献了 74% 的消息量。高频用户产生的消息远多于低频用户，在 Flush MemTable 时，高频用户消息占据了文件的绝大部分。

2. 对于高频的用户，由于其“高频”的原因，当消息进入 DTIM，系统发现用户设备在线（高概率在线），会立即推送消息，因此需要推送的消息大部分在内存的 MemTable 中。

3. 高频用户产生大量的消息，Compaction 耗费了系统大量的计算和 IO 资源。

4. 低频的用户消息通常散落在多个文件当中。

从上面的四个表现来看，我们能得出如下结论：

1. 绝大部分 Compaction 是无效的计算和 IO，由于大量消息写入产生大量的文件，但是高频的用户消息其实已经下推给用户的设备，Compaction 对读加速效果大打折扣。反而会抢占计算资源，同时引起 IO 抖动。

2. 低频用户由于入库消息频率低，在 Flush 之后的文件中占比低；同时用户上线频率低，期间会累计较多的待接收的消息，那么当用户上线时，连续的历史消息高概率散落在多个文件当中，导致遍历读取消息毛刺，严重的有可能读取超时。

为了解决此类问题，我们采用分而治之方法，将高频用户和低频用户的消息区别对待。我们借鉴了 WiscKey KV 分离技术的思想，就是将到达一定阈值的 Value 分离出来，降低这类消息在文件中的占比进而有效的降低写放大的问题。

但是 WiscKey KV 分离仅考虑单 Key 的情况，在 DITM 的场景下，Key 之间的大小差距不大，直接采用这种 KV 分离技术并不能解决以上问题。因此我们在原有 KV 分离的基础上，

改进了 KV 分离，将相同前缀的多个 Key 聚合判断，如果多个 Key 的 Value 超过阈值，那么将这些 Key 的 Value 打包了 value-block 分离出去，写入到 value 文件。

数据显示，上述方法能够在 Minor Compaction 阶段将 MemTable 中 70% 的消息放入 value 文件，大幅降低 key 文件大小，带来更低的写放大；同时，Major Compaction 可以更快降低 key 文件数，使读放大更低。高频用户发送消息更多，因此其数据行更容易被分离到 value 文件。读取时，高频用户一般把最近消息全部读出来，考虑到 DTIM 消息 ID 是递增生成，消息读取的 key 是连续的，同一个 value-block 内部的数据会被顺序读取，基于此，通过 IO 预取技术提前读取 value-block，可以进一步提高性能。对于低频用户，其发送消息较少，K-V 分离不生效，从而减少读取时候 value 文件带来的额外 IO。

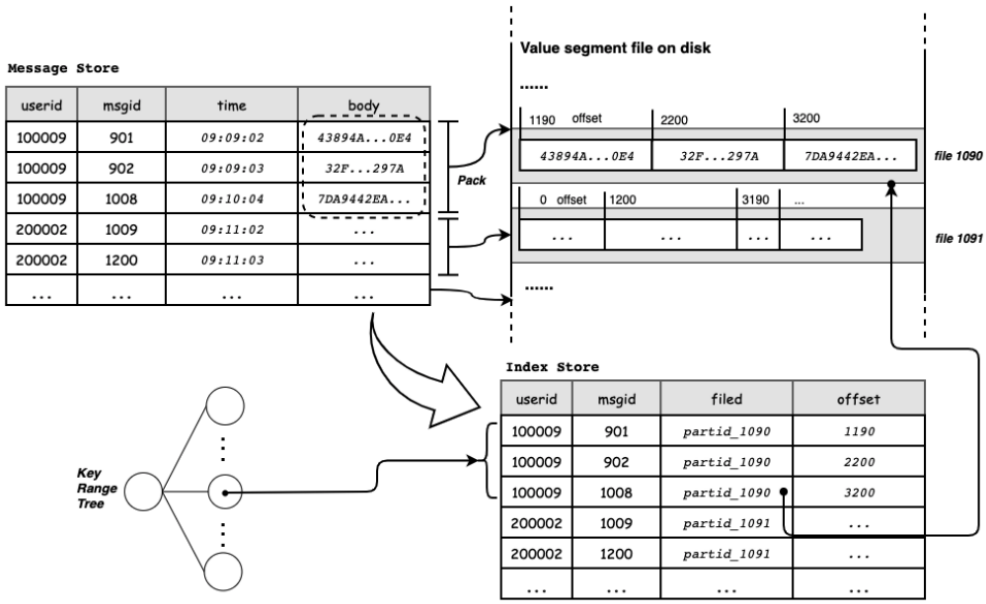

Fig. 5: Key value separation and re-pack

同步机制

DTIM 面向办公场景，和面向普通用户的产品在服务端到客户端的数据同步上最大的区别是消息量巨大、变更事件复杂、对多端同步有着强烈的诉求。DTIM 基于同步服务构建了一套完整同步流程。同步服务是一个服务端到客户端的数据同步服务，是一套统一的数据下行平台，支撑钉钉多个应用服务。

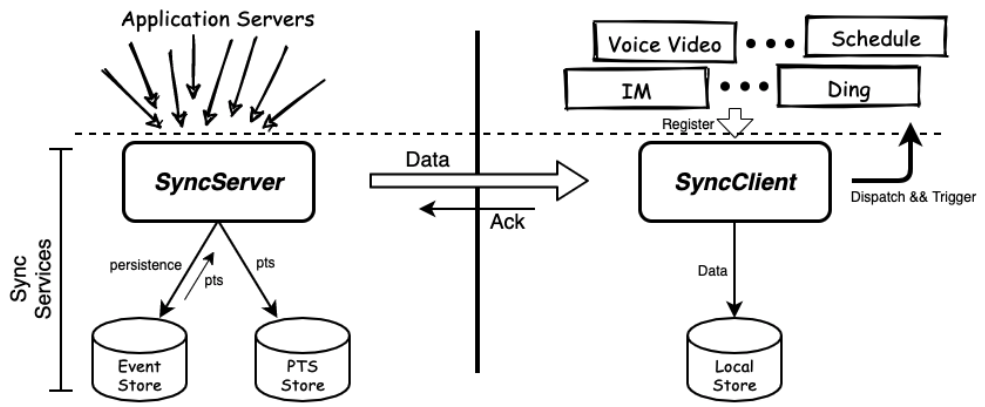

Fig. 6: SyncServices architecture

同步服务是一套多端的数据同步服务，由两部分组成：部署于服务端的同步服务和由客户端 APP 集成的同步服务 SDK。工作原理类似于消息队列，用户 ID 近似消息队列中的 Topic，用户设备近似消息队列中的 Consumer Group，每个用户设备作为一个消费者能够按需获得这个用户一份数据拷贝，实现了多端同步诉求。

当业务需要同步一个变更数据到指定的用户或设备时，业务调用数据同步接口，服务端会将业务需要同步的数据持久化到存储系统中，然后当客户端在线的时候把数据推送给客户端。每一条数据入库时都会原子的生成一个按用户维度单调递增的位点，服务端会按照位点从小到大的顺序把每一条数据都推送至客户端。

客户端应答接收成功后，更新推送数据最大的位点到位点管理存储中，下次推送从这个新的位点开始推送。同步服务 SDK 内部负责接收同步服务数据，接收后写入本地数据库，然后再把数据异步分发到客户端业务模块，业务模块处理成功后删除本地存储对应的内容。

在上文章节中，已经初步介绍同步服务推送模型和多端一致性的考虑，本章主要是介绍 DTIM 是如何做存储设计、在多端同步如何实现数据一致性、最后再介绍服务端消息数据堆积技术方案 Rebase。

事件存储

在同步服务中，采用以用户为中心，将所有要推送给此用户的消息汇聚在一起，并为每个消息分配唯一且递增的 PTS（位点， Point To Sequence），服务端保存每个设备推送的位点。

通过两个用户 Bob 和 Alice，来实际展示消息在存储系统中存储的逻辑形态。例如，Bob 给 Alice 发送了一个消息”Hi! Alice“，Alice 回复了 Bob 消息”Hi! Bob“。

当 Bob 发送第一条消息给 Alice 时，接收方分别是 Bob 和 Alice，系统会在 Bob 和 Alice 的存储区域末尾分别添加一条消息，存储系统在入库成功时，会分别为这两行分配一个唯一且递增的位点（Bob 的位点是 10005，Alice 的位点是 23001）；入库成功之后，触发推送。比如 Bob 的 PC 端上一次下推的位点是 10000，Alice 移动端的推送位点是 23000，在推送流程发起之后，会有两个推送任务，第一是 Bob 的推送任务，推送任务从上一次位点（10000） + 1 开始查询数据，将获取到 10005 位置的”Hi“消息，将此消息推送给 Bob 的设备，推送成功之后，存储推送位点（10005）。Alice 推送流程也是同理。Alice 收到 Bob 消息之后，Alice 回复 Bob，类似上面的流程，入库成功并分配位点（Bob 的位点是 10009，Alice 的位点是 23003）。

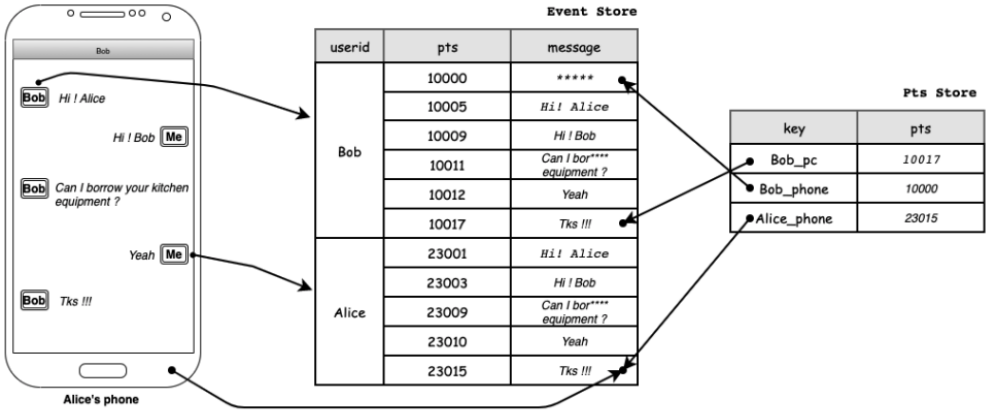

Fig. 7: Storage design of SyncServices

多端同步

多端同步是 DTIM 的典型特点，如何保持多端的数据及时触达和解决一致性是 DTIM 同步服务最大的挑战。上文中已经介绍了同步服务的事件存储模型，将需要推送的消息按照用户聚合。当用户有多个设备时，将设备的位点保存在位点管理系统之中，Key 是用户 + 设备 ID，Value 是上一次推送的位点。如果是设备第一次登录，位点默认为 0。由此可知，每个设备都有单独的位点，数据在服务端只有一份按照用户维度的数据，推送到客户端的消息是服务端对应位点下的快照，从而保障了每个端的数据都是一致的。

比如此时 Bob 登录了手机（该设备之前登录过钉钉），同步服务会获取到设备登录的事件，事件中有此设备上次接收数据的位点（比如 10000），同步服务会从 10000 + 1（位点）开始查询数据，获取到五条消息（10005~10017），将消息推送给此台手机并更新服务端位点。此时，Bob 手机和 PC 上的消息一致，当 Alice 再次发送消息时，同步服务会给 Bob 的两台设备推送消息，始终保持 Bob 两个设备之间消息数据的一致性。

积压处理

正如上文所述，我们采用了推优先的模型下推数据以保障事件的实时性，采用位点管理实现多端同步，但是实际情况却远比上面的情况复杂。最常见的问题就是设备离线重新登录，期间该用户可能会累计大量未接收的消息数据，比如几万条。如果按照上面的方案，服务端在短时间会给客户端推送大量的消息，客户端 CPU 资源极有可能耗尽导致整个设备假死。

其实对于 IM 这种场景来说，几天甚至几小时之前的数据，再推送给用户已经丧失即时消息的意义，反而会消耗客户移动设备的电量，得不偿失。又或者节假日大群中各种活动，都会有大量的消息产生。对于以上情况，同步服务提供 Rebase 的方案，当要推送的消息累计到一定阈值时，同步服务会向客户端发送 Rebase 事件，客户端收到事件之后，会从消息服务中获取到最新的消息（Lastmsg）。这样可以跳过中间大量的消息，当用户需要查看历史消息，可以基于 Lastmsg 向上回溯，即省电也能提升用户体验。

还是以 Bob 为例，Bob 登录了 Pad 设备（一台全新的设备），同步服务收到 Pad 登录的事件，发现登录的位点为 0，查询从 0 开始到当前，已经累计 1 万条消息，累计量大于同步服务的阈值，同步服务发送 Rebase 事件给客户端，客户端从消息服务中获取到最新的一条消息"Tks !!!"，同时客户端从同步服务中获取最新的位点为 10017，并告诉同步服务后续从 10017 这个位置之后开始推送。当 Bob 进入到和 Alice 的会话之后，客户端只要从 Lastmsg 向上回溯几条历史消息填满聊天框即可。

高可用

DTIM 对外提供 99.995% 的可用性 SLA，有上百万的组织将钉钉作为自身数字化办公的基础设施，由于其极广的覆盖面，DTIM 些许抖动都会影响大量企业、机构、学校等组织，进而可能形成社会性事件。因此，DTIM 面临着极大的稳定性挑战。

高可用是 DTIM 的核心能力。对于高可用，需要分两个维度来看，首先是服务自我防护，在遇到流量洪峰、黑客攻击、业务异常时，要有流量管控、容错等能力，保障服务在极端流量场景下还有基本服务的能力。其次是服务扩展能力，比如常见的计算资源的扩展、存储资源的扩展等，资源伴随流量增长和缩减，提供优质的服务能力并与成本取得较好的平衡。

自我防护

DTIM 经常会面临各种突发大流量，比如万人大群红包大战、早高峰打卡提醒、春节除夕红包等等都会在短时间内产生大量的聊天消息，给系统带来很大的冲击，基于此我们采用了多种措施。

首先是流量管控，限流是保护系统最简单有效的方式。DTIM 服务通过各种维度的限流来保护自身以及下游，最重要的是保护下游的存储。在分布式系统中存储都是分片的，最容易出现的是单个分片的热点问题，DTIM 里面有两个维度的数据：用户、会话 (消息属于会话), 分片也是这两个维度，所以限流采用了会话、用户维度的限流，这样既可以保护下游存储单个分区，又可以一定程度上限制整体的流量。要控制系统的整体流量, 前面两个维度还不够，DTIM 还采用了客户端类型、应用 (服务端 IM 上游业务) 两个维度的限流，来避免整体的流量上涨对系统带来的冲击。

其次是削峰平谷。限流简单有效，但是对用户的影响比较大。在 DTIM 服务中有很多消息对于实时性要求不高，比如运营推送等。对于这些场景中的消息可以充分利用消息系统异步性的特点，使用异步消息队列进行削峰平谷，这样一方面减少了对用户的影响，另一方面也减轻对下游系统的瞬时压力。DTIM 可以根据业务类型 (比如运营推送)、消息类型 (比如点赞消息) 等多种维度对消息进行分级，对于低优先级的消息保证在一定时间 (比如 1 个小时) 内处理完成。

最后是热点优化。DTIM 服务中面临着各种热点问题，对于这些热点问题仅仅靠限流是不够的。比如通过钉钉小秘书给大量用户推送升级提醒，由于是一个账号与大量账号建立会话，因此会存在 conversation_inbox 的热点问题，如果通过限速来解决，会导致推送速度过慢、影响业务。对于这类问题需要从架构上来解决。

总的来说，主要是两类问题：大账号和大群导致的热点问题。对于大账户问题，由于 conversation_inbox 采用用户维度做分区，会导致系统账号的请求都落到某个分区，从而导致热点。解决方案为做热点拆分，既将 conversation_inbox 数据合并到 conversation_member 中 (按照会话做分区)，将用户维度的操作转换为会话维度的操作，这样就可以将系统账号的请求打散到所有分区上， 从而实现消除热点。对于大群问题，压力来自大量发消息、消息已读和贴表情互动，大量的接收者带来极大的扩散量。所以我们针对以上三个场景，分而治之。

计算延迟与按需拉取

对于消息发送，一般的消息对于群里面所有人都是一样的，所以可以采用读扩散的方式，即不管多大的群，发一条消息就存储一份。另一方面，由于每个人在每个会话上都有红点数和 Lastmsg, 一般情况下每次发消息都需要更新红点和 Lastmsg，但是在大群场景下会存在大量扩散，对系统带来巨大的压力。我们的解决方案为，对于大群的红点和 Lastmsg，在发消息时不更新，在拉首屏时实时算，由于拉首屏是低频操作且每个人只有一到两个大群，实时计算压力很小，这样高峰期可以减少 99.99 % 的存储操作, 从而解决了大群发消息对 DTIM 带来的冲击。

请求合并

在大群发消息的场景中，如果用户都在线，瞬时就会有大量已读请求，如果每个已读请求都处理，则会产生 M*N(M 消息条数，N 群成员数) 的扩散，这个扩散是十分恐怖的。

DTIM 的解决方案是客户端将一个会话中的多次已读进行合并，一次性发送给服务端，服务端对于每条消息的已读请求进行合并处理，比如 1 分钟的所有请求合并为 1 次请求。在大群中，进行消息点赞时，短时间会对消息产生大量更新，再加上需要扩散到群里面的所有人，系统整体的扩散量十分巨大。我们发现，对于消息多次更新的场景，可以将一段时间里面多次更新合并，可以大大减少扩散量，从实际优化之后的数据来看，高峰期系统的扩散量同比减少 96%。

即使完全做到以上几点，也很难提供当前承诺的 SLA，除了防止自身服务出现问题以外，还必须实现对依赖组件的容灾。我们整体采用了冗余异构存储和异步队列与 RPC 相结合的方案，当任意一类 DTIM 依赖的产品出现问题时， DTIM 都能正常工作，由于篇幅问题，此处不再展开。

水平扩展能力

对于服务的弹性扩展能力，也需要分两个维度来看。首先，服务内部的弹性扩展，比如计算资源的扩展、存储资源的扩展等，是我们通常构建弹性扩展能力关注的重点方向；其次是跨地域维度的扩展，服务能根据自身需要，在其他区域扩展一个服务集群，新的服务集群承接部分流量，在跨地域层面形成一个逻辑统一的分布式服务，这种分布式服务我们称之为单元化。

弹性应用架构

对于 DTIM 的扩展性，因为构建和生长于云上，在弹性扩展能力建设拥有了更多云的特点和选择。对于计算节点，应用具备横向扩展的能力，系统能在短时间之内感知流量突增进而进行快速扩容，对于上文提到的各种活动引起的流量上涨，能做到轻松应对。同时，系统支持定时扩容和缩容，在系统弹性能力和成本之间取得较好的平衡。

对于存储，DTIM 底层选择了可以水平扩展的 Serverless 存储服务，存储服务内部基于读写流量的大小进行动态调度，应用上层完全无感知。对于服务自身的扩展性，我们还实施了不可变基础设施、应用无状态、去单点、松耦合、负载均衡等设计，使 DTIM 构建出了一套高效的弹性应用架构。

地域级扩展：单元化

在应用内部实现了高效弹性之后，伴随着业务流量的增长，单个地域已经无法满足 DTIM 亿级别 DUA 的弹性扩展的需求。由于 DTIM 特点，所有用户都可以在添加好友之后进行聊天，这就意味着不能简单换个地域搭建一套孤岛式的 DTIM。为了解决这种规模下的弹性能力，我们基于云上的多 Region 架构，在一个 Geo 地域内，构建了一套异地多活、逻辑上是一体的弹性架构，我们称之为单元化。下图是 DTIM 的单元化架构。

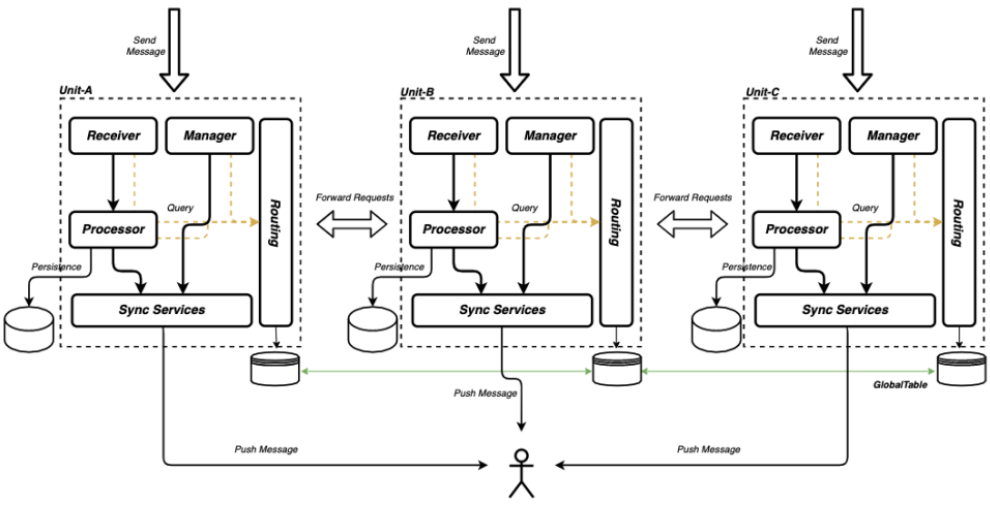

Fig. 8: DTIM Unit group architecture to process message by RoutingService.

对于单元化的弹性扩展架构，其中最核心的内容是流量动态调度、数据单地域的自封闭性和单元整体降级。

动态调度

流量路由决定了数据流向，我们可以依托这个能力，将大群流量调度到新的单元来承接急速增长的业务流量，同时实现流量按照企业维度汇聚，提供就近部署能力，进而提供优质的 RT 服务。

业界现在主流的单元化调度方案主要是基于用户维度的静态路由分段，这种方案算法简单可靠，但是很难实现动态路由调度，一旦用户路由固定，无法调整服务单元。比如在 DTIM 的场景中，企业（用户）规模是随着时间增长、用户业务规模增长之后，单地域无法有效支撑多个大型企业用户时，传统静态方案很难将企业弹性扩展到其他单元，强行迁移会付出极高的运维代价。因此传统的路由方案不具备弹性调度能力。

DTIM 提供一套全局一致性的高可用路由服务系统 (RoutingService)。服务中存储了用户会话所在单元，消息服务基于路由服务，将流量路由到不同的单元。应用更新路由数据之后，随之路由信息也发生变化。与此同时，路由服务发起数据订正事件，将会话的历史消息数据进行迁移，迁移完成之后正式切换路由。路由服务底层依赖存储的 GlobalTable 能力，路由信息更新完成之后，保障跨地域的一致性。

单元自封闭

数据的单元自封闭是将 DTIM 最重要且规模最大的数据：“消息数据”的接收、处理、持久化、推送等过程封闭在当前单元中，解除了对其他单元依赖，进而能高效地扩充单元，实现跨地域级别高效弹性能力。

要做到业务数据在单元内自封闭，最关键是要识别清楚要解决哪种数据的弹性扩展能力。在 DTIM 的场景下，用户 Profile、会话数据、消息数据都是 DTIM 最核心的资产，其中消息数据的规模远超其他数据，弹性扩展能力也是围绕消息数据的处理在建设。怎么将消息按照单元数据合理的划分成为单元自封闭的关键维度。

在 IM 的场景中，消息数据来自于人与人之间的聊天，可以按照人去划分数据，但是如果聊天的两个人在不同的单元之间，消息数据必然要在两个单元拷贝或者冗余，因此按照人划分数据并不是很好的维度。

DTIM 采用了会话维度划分，因为人和会话都是元数据，数据规模有限，消息数据近乎无限，消息归属于会话，会话与会话之间并无交集，消息处理时并没有跨单元的调用。因此，将不同的会话拆分到不同的单元，保障了消息数据仅在一个单元处理和持久化，不会产生跨单元的请求调用，进而实现了单元自封闭。

单元降级

在单元化的架构中，为了支持服务级别的横向扩展能力，多单元是基本形态。单元的异常流量亦或者是服务版本维护的影响都会放大影响面，进而影响 DTIM 整体服务。因此，DTIM 重点打造了单元降级的能力，单一单元失去服务能力之后，DTIM 会将业务流量切换到新的单元，新消息会从正常的单元下推，钉钉客户端在数据渲染时也不会受到故障单元的影响，做到了单元故障切换用户无感知。

总 结

本文通过模型设计、存储优化、同步机制以及高可用等维度，本文全方位地展示了当代企业级 IM 设计的核心。上文是对 DTIM 过去一段时间的技术总结，随着用户数的持续增长，DTIM 也在与时俱进、持续迭代和优化，比如支持条件索引进而实现索引加速和成本可控、实现消息位点的连续累加、实现消息按需拉取和高效的完整性校验、提供多种上下行通道，进一步提升弱网下的成功率和体验等。

今日好文推荐

一晚上累计 292 万人紧盯 Flightradar24 网站，航班跟踪的技术原理是什么?

“台湾省山西刀削面”搜索过多，百度地图宕机；BOSS 直聘即将实行末位淘汰；B 站回应 HR 称核心用户是 Loser | Q 资讯

拿过不合格绩效，但也创造出世界顶级开源项目：梦想用“机器编程”改变工程世界 | 专访 OpenResty 章亦春

开发技能需求变了：经验不是晋升唯一要素，通晓多编程语言的时代已成过去