Titan 在美图推送的落地

导读

美图公司拥有众多的产品以及海量活跃用户，我们的日推消息也达到了上亿次，这对消息推送也提出了较高的要求。2017 年，美图自研推送服务，采用 Redis 作为消息存储，但随着业务接入量增加、数据量快速增加，服务的可维护性变得困难。公司已经搭建了一套新型的数据库 - Titan （已经开源），底层以 PingCAP 的 TIKV 做数据引擎，上层实现 Redis 协议解析，既可以水平弹性扩容，适合海量数据存储管理，性能又可以随水平扩容而提高。

在本文中，先简要的介绍推送现状及难题，后详细的说明 Titan 在全面替换原有的存储的过程、以及在接入过程中遇到的问题和解决方案。

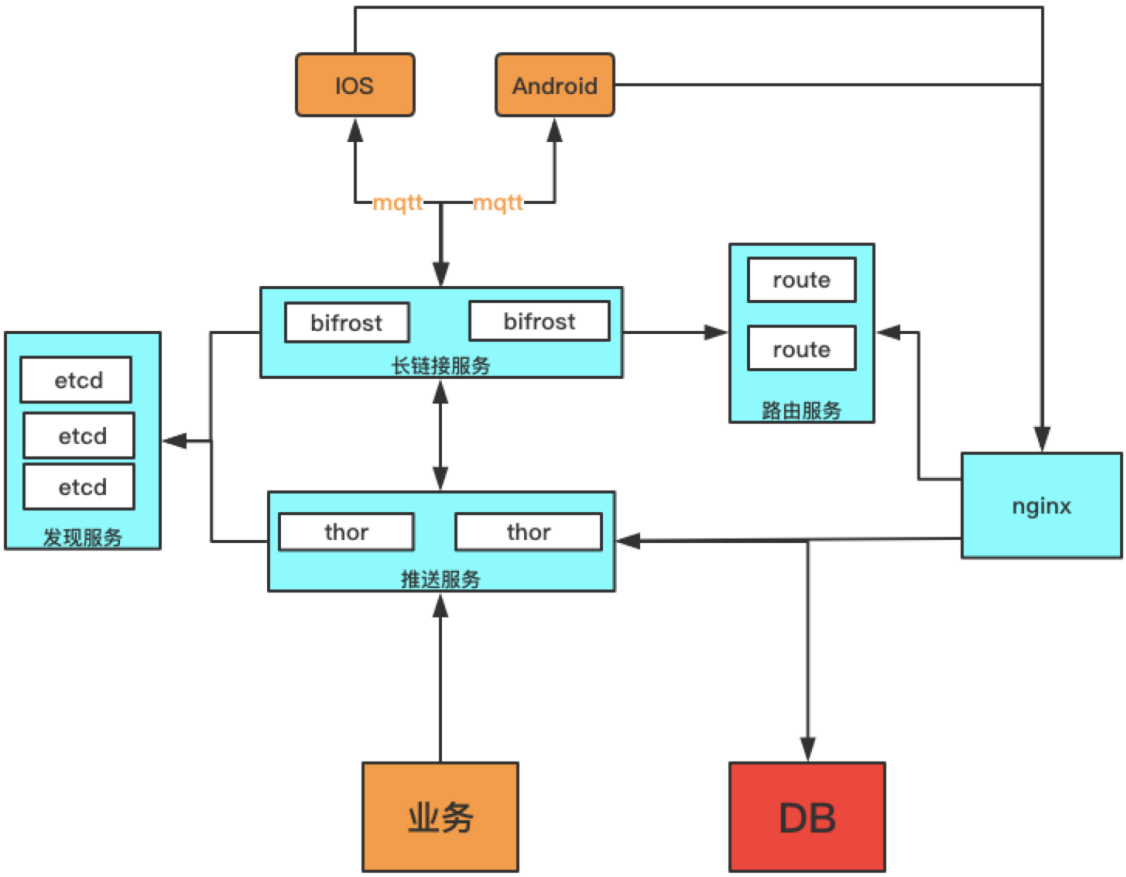
推送现状

2017 年初，美图自研推送服务（Thor），完成 APP 的定向推送、批量推送、离线推送、消息过期、 token 管理等功能。至今已经接入美图全部的 App 中，在线到达率为 99% 以上，消息秒级触达用户，日常服务端消息存储量约为 700G ，最高峰为 1T 。

随着服务接入量级的提升系统的架构模型，存储模型也不断在演进，下面将讲述推送系统目前的架构模型和存储模型以及当前推送服务面临的难题问题。

架构模型

推送服务整体拆分为三个部分：长链接服务（bifrost）、推送服务（thor）、路由分发（route_server)。服务之间串联通过发现服务（etcd）实现。长链接服务负责保持客户端和服务端的链路通畅。推送服务负责管理客户端的 token 信息和存储管理消息。



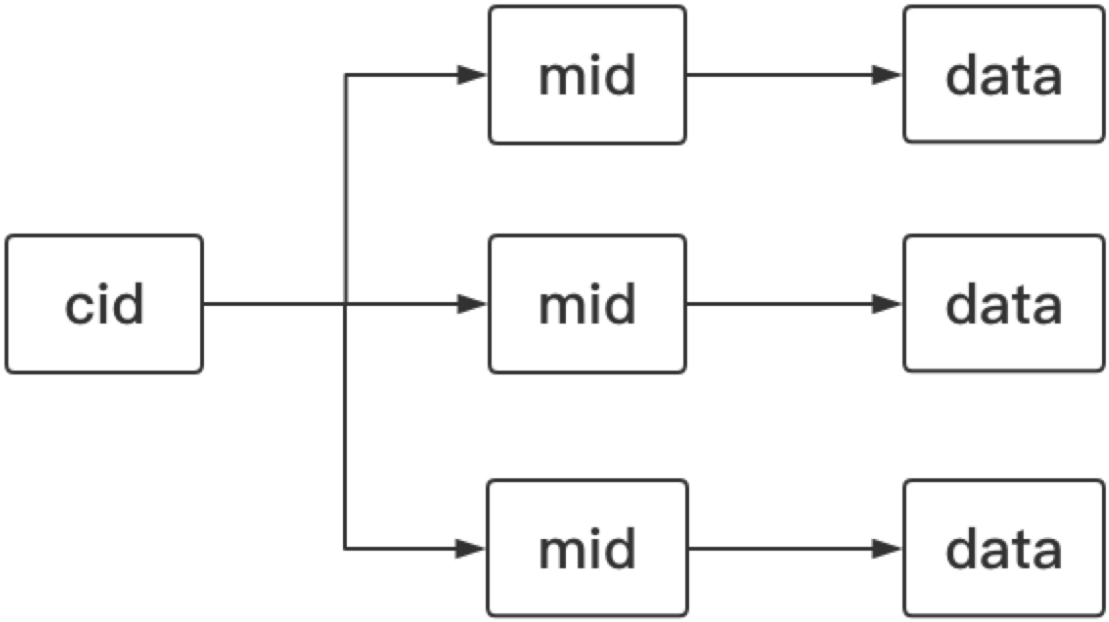
图一：美图推送整体架构图

存储模型

消息存储管理作为推送核心模块，构建一个恰到好处的模型至关重要。在推送业务中消息的操作可以划分为两个维度：

- 站在客户端维度来看，需要保证精准的接受消息和上报回执信息。
- 站在系统维度看，推送系统应该具被消息灵活管理功能，保证在系统异常崩溃、网络隔离等异常场景下消息不丢不乱能力。

综上所述，美图推送将数据模型抽象图如下，每个客户端（cid）拥有唯一的消息下发队列，每个消息会被绑定唯一的标识（mid），服务端通过指定 cid 下发消息，消息前先写入存储后进行消息投递，监测到客户端登陆后服务端查询离线消息重新下发，降低了消息在异常场景损失的概率。通过上报回执消息中的 mid 清除已下发成功消息，保证了队列的消息的有效性。



图二：数据模型图

当下难题

在开发初期选择消息存储的时候，一方面考虑到在推送业务场景中消息留存时间在可控的时间内且数据量较少，Redis 自身支持丰富的数据结构，提供高速的访问能力。另外一方面公司内部对 Redis 使用和管理有丰富的经验。因此选择 Redis 做消息存储，部署方式则采用一主两从客户端做分片写入的集群模式。

随着业务接入量的增加，数据量越来越大，服务的维护性越来越难。其中主要难题如下：

- 频繁的消息过期和删除导致 Redis 的内存碎片比居高不下；
- 单节点数据量增大，持久化耗时加剧，导致服务抖动，短时间不可用；
- 存储扩容导致服务有损；
- 服务成本越来越高。

以上困难，只有替换存储才可以从根本解决这些问题。我们需要选择一种适合海量数据存储、水平弹性扩容、对业务迁移友好的存储。Titan 便是我们的不二人选，接下来要和大家简要介绍下 Titan 。

Titan

Titan 是美图公司研发并开源的 NoSQL，目前交给第三方 DistributedIO 组织维护。DistributedIO 组织由源 Titan 开发团队发起构建。主要适合要求具有分布式事务的大规模数据存储，适用海量数据，少量事务冲突的场景。Titan 划分为两层：下层使用 PingCAP 的 TiKV 数据库做数据持久化； 上层通过解析 Redis 协议将各类数据结构转化为 KV 数据方式存储。

TIKV 是 PingCAP 开源的分布式 Key-Value 存储，它使用 Rust 开发，采用 Raft 一致性协议保证数据的强一致性，以及稳定性，同时通过 Raft 的 Configuration Change 机制实现了系统的可扩展性，提供了支持 ACID 事务的 Transaction API。虽然 PingCAP 也开源一个名字叫的 Titan 的存储引擎，但与本文介绍的 Titan 是不同的，只不过名字一样而已。

Titan 自身是无状态服务，提供 Redis 命令翻译执行功能。在设计上支持在共享一套集群情况下业务数据隔离，遵从 Redis 5.0 开发实现，自身集成 prometheus 监控。 目前 Ttian 已经支持 lists ，strings ，hashes ，sets ，sorted sets 等基础数据结构。自从开源以来，收到了广泛的关注，目前已经收到 700 多的 star 和 50 多次的 fork，在业内存在成功接入并投入线上使用的公司案例，如北京转转精神科技有限公司（转转）已经迁移 800G 的数据到 Titan 中。

存储平滑迁移

现在大家对 Titan 有了基本的了解，但存储的替换不是一蹴而就的事情。Titan 集群的大小，存储迁移平滑过度，数据的迁移，这都是我们的绊脚石。下面将从业务角度出发给出答案。

业务评估及优化

推送服务经常面临高频的消息读写场景，因此对存储的性能要求也是极高的，我们首先要做的就是对 Titan 进行压测。推送服务依赖 Redis 的 Hash 数据结构完成对消息的管理，使用的命令有 hset ，hgetall ，hdel。压测 Titan 在 1 台 sas 盘 CPU 40 核 内存 96 G 机器和 3 台 ssd 盘 CPU 40 核 内存 96 G 机器上，部署采用了 1 个 Titan 12 个 TiKV 实例方式。通过整理压测数据和统计线上监控给出对每个命令的期待的 QPS 如下表。

	hset	hgetall	hdel

压测数据	31 k/s	13k/s	47k
实际需求	150k/s	20k/s	30k/s

表一：业务评估表

通过上图对比分析发现，Titan 按照这个集群配置 hset 和 hgetall 两个命令明显不满足如线上要求。可以通过扩容 Titan 集群的方式提高系统吞吐，但初于对成本的考虑，我们从业务角度尝试优化。

- hset

在 Redis 中消息的保存使用 hset 命令根据客户端 cid 进行 hash 写到固定的 Redis 存储中，需要逐条写入。在 Titan 中我们尝试将 hset 命令从单条的执行改为 batch 操作，经过测试确立方案为每 100 条命令执行一次事务提交， 虽然延迟从 10 ms 增到到 100 ms 以内波动，但优化后性能要求从 150 k/s 将为 20 k/s 且延迟在可以接受的范围内。

- hgetall

在 Redis 中客户端登陆可以通过 hgetall 获取当前所有离线消息重新接收，在调研发现客户端存在少量离线消息或者不存在离线消息。针对这种情况重新对 hgetall 进行压测，hgetall 在这种场景下可以提供 25 k/s QPS 满足需求。

在经过优化之后，Titan 可以满足目前推送线上的基本要求。按照上述配置决定将所有消息存储逐步灰度到 Titan 上。

平滑替换

消息存储作为美图推送核心单元，要保证 7x 24 小时可用，在迁移过程中，如何在 Titan 异常情况下保证服务正常访问，旧数据如何业务无损的同步到新集群是我们面临的两大难题。

在推送业务中消息生命周期都在一周之内，如果采用双写模式，将数据顺序写入 Redis ，Titan 两个集群，读取只在 Redis 集群上。一周后在将读切到 Titan 上，在稳定运行一段时间后，下掉 Redis 集群完成存储集群迁移。其中 Titan 集群在任何时间下发生异常都可以下掉，操作切回 Redis 集群。

问题和解决方法

在美图推送服务接入 Titan 的过程中，我们团队也遇到了不少的问题，比如事务冲突，短时间内 Titan 堆积大量命令，导致雪崩效应等问题。我们在具体实践中也总结了一些解决方法。

事务冲突

现象：在推送的高峰期期间事务冲突频繁，Titan 节点内存升高，TiKV 机器内存耗尽节点 OOM。
原因：Titan 中对相同的 key 写入和删除并发操作会导致事务冲突，消息的写入采用 batch 方式写入，一旦发生事务冲突，这批数据会集体产生滚会、重试操作，短时间内 Titan 会积压大量命令导致内存上升，TiKV 操作事务回滚导致内存耗尽 节点 OOM。
解决方式：舍弃 meta 中数据数量纪录字段，减少单个 key 的操作冲突。通过这种方式仅仅降低了单个 key 的冲突，在 hash 操作中针对单个 felid 的修改仍然存在冲突。但此种优化已经到达业务接受水平。

TiKV OOM

原因：线上采用内存 96G 机器部署 4 个 TiKV 模式，在高峰期队列请求处理队列积压，导致机器内存耗尽。
解决方式：减少 TiKV 配置中 block-cache-size 的大小，降低内存占用。

Raft Store CPU 使用率过高

现象：redis 命令执行存在卡顿，TiKV 监控中 Raft Store CPU 使用率超过 90%。
原因：在 TiKV 中 raftstore 是单线程工作。
解决方案：集群扩容，增加一个服务器，增加一个 4 个 TiKV 节点，提高 Titan 服务的处理能力，从根本上解决了问题。

TiKV channel full

现象：Redis 命令执行超时。
原因：在数据短时间持续大量写入，导致 Raft 热点，直接导致 TiKV 的 Region leader 迁移。
解决方式：通过配置调整 TiKV 的 scheduler-notify-capacity 大小，增加 scheduler 一次获取最大消息个数，降低了问题发生的频率。

总结

在经历半年的尝试后，推送存储整体替换为 Titan，存储也由 16 台 Redis 机器切换为 4 台的 ssd TiKV 专属服务器和 2 台混部 Titan 服务器上，成本节约 60%，可维护性大大提高，现在推送服务已经稳定跑了半年，期间未发生故障。随着 Titan 的数据结构的完善，未来准备推进更多业务接入。

感谢张同学（nioshield@gmail.com）及时实现的 hashes 数据结构。
感谢 PingCAP 公司在迁移过程中对于 TiKV 的技术支持。

Redis 的压测工具：https://github.com/fperf/redis
Titan 项目地址：https://github.com/distributedio/titan
TiKV 项目地址：https://github.com/tikv