微博热点事件背后数据库运维的“功守道”

一、背景&挑战

背景

图1 鹿晗发博截图

正是这条微博动态，让一个平常的国庆假期变得不同平常，微博刚一发出就引爆网络，它将明星CP动态推向了舆论的高潮，并霸占微博热搜榜好几天，也正是因为这个突发的流量，因流量过大一度引发微博服务器瘫痪，而成为吃瓜观众热议的话题之一，相信很多人都有印象吧？微博作为当今中文社交媒体的第一品牌，拥有超过3.76亿的月活用户，也是当前社会热点事件传播的最主要平台，其中包括但不限制于大型活动（如：里约奥运会、十九大等），春晚，明星动态（如：王宝强离婚事件、女排夺冠、乔任梁去世、白百合出轨、TFBOYS生日、鹿晗关晓彤CP等）。而热点事件往往具有不可预见性和突发性，并且伴随着极短时间内流量的数倍增长，甚至更多，有时持续时间较长。如何快速应对突发流量的冲击，确保线上服务的稳定性，是一个非常巨大的挑战和有意义的事情。

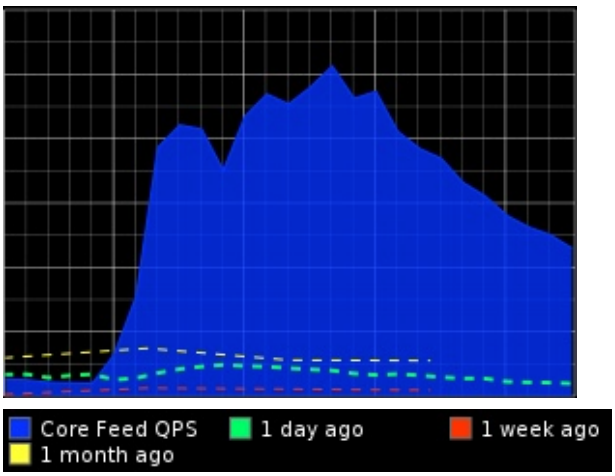

图2 鹿晗关晓彤CP Feed业务量

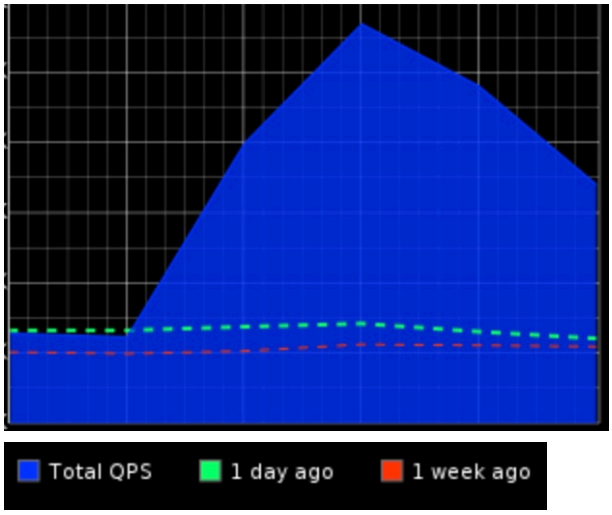

图3 鹿晗关晓彤CP 评论业务量

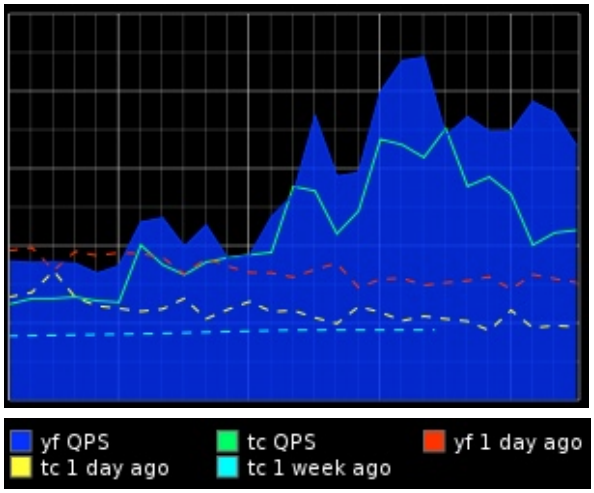

图4 鹿晗关晓彤CP 话题业务量

热点事件的特点

从上述系列流量图中，我们可以看出这两起事件对微博的核心服务如Feed，评论，赞，话题等带来的压力都是成倍的，创造了历史的新高点。仔细分析不难发现，这类事件基本可以分为：发生期、发酵期、暴涨期和缓解期，只是由于天时地利人和等各种因素的综合影响，各个时期的持续时间，流量变化不尽相同，对服务的压力也有所差异，在暴涨期所表现出的不亚于天猫双11、12306抢票的盛况。总体上反映出这类热点事件的特点为：

- 不可预见性和突发性
- 峰值流量大
- 持续时间短
- 参与人数多
- 涉及资源广

面临的挑战

微博研发中心数据库部门主要负责全微博平台的后端资源的托管和运维，主要涉及的后端资源服务包括MySQL、Memcached、Redis、HBase、Memcacheq、Kafka、Pika、PostgreSQL等。为了应对峰值流量和保证用户的良好体验，资源层会面临哪些挑战？又将如何解决呢？

- 面对不可预期的峰值流量，有时恶意的刷站行为，如何应对数倍，甚至数十倍于日常访问量的压力？
- 如何应对瞬时可达几万/秒的发表量？
- 如何做到异地容灾？

- 如何保证缓存的命中率，减少缓存穿透甚至“雪崩”?
- 如何实现缓存的快速动态扩缩容?
- 如何保证服务的低延迟、高可用?
- 如何保证核心系统的稳定性?
- 如何实时监控系统服务状态?
- 等等

二、微博数据库运维体系架构解析

业务架构

微博目前用户基数庞大，DAU和MAU均为数亿。从整个技术体系来看，微博核心总体分为前端和后端平台，端上主要是PC端，移动端，开放平台以及企业开放平台。后端平台主要是Java编写的各种接口层、服务层、中间件层及存储层。除此之外，微博搜索、推荐、广告、大数据平台也是非常核心的产品。从业务角度看，整体架构图如下:

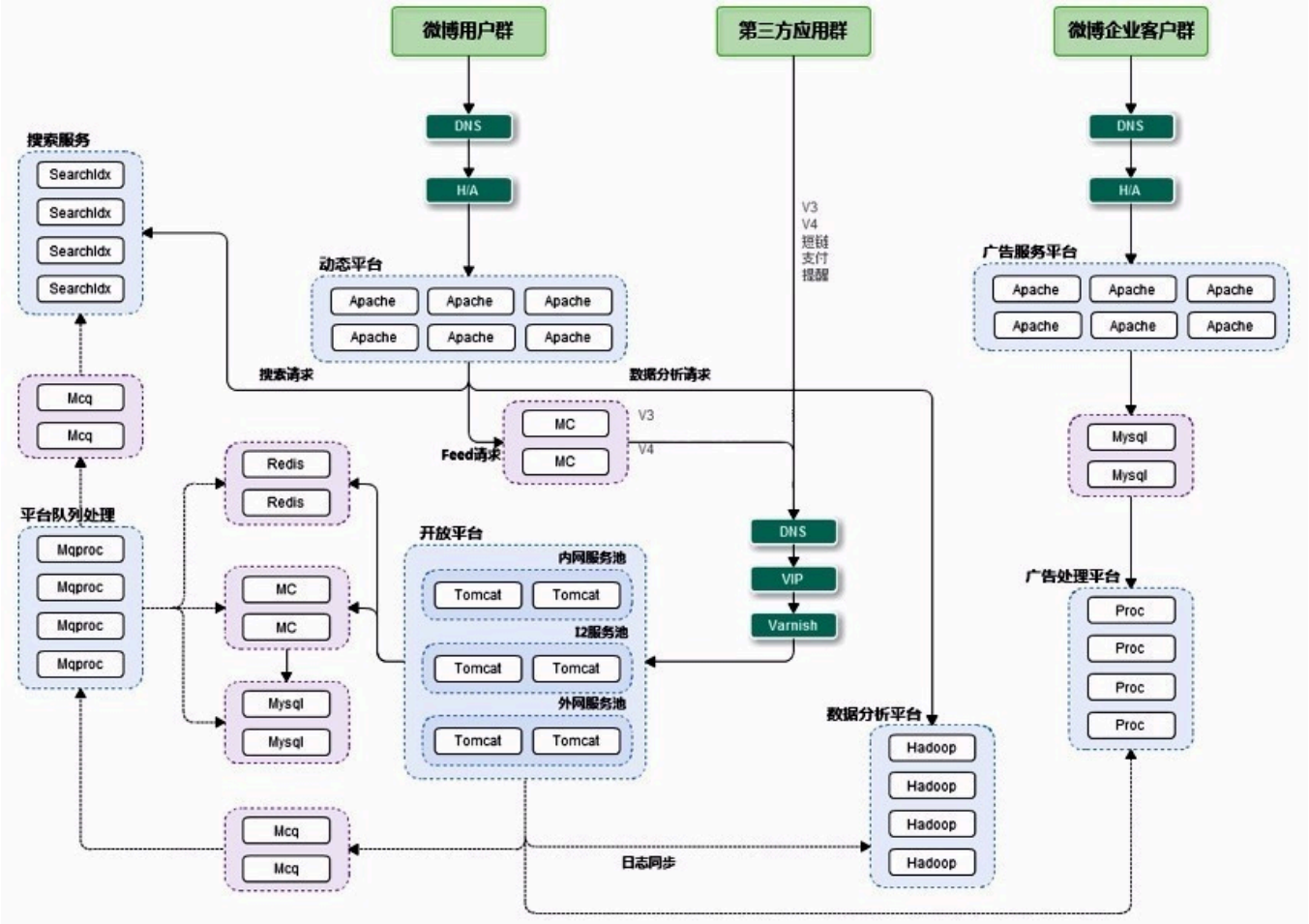

图5 业务架构图

这里以微博平台的业务为例，微博平台的服务部署架构如下:

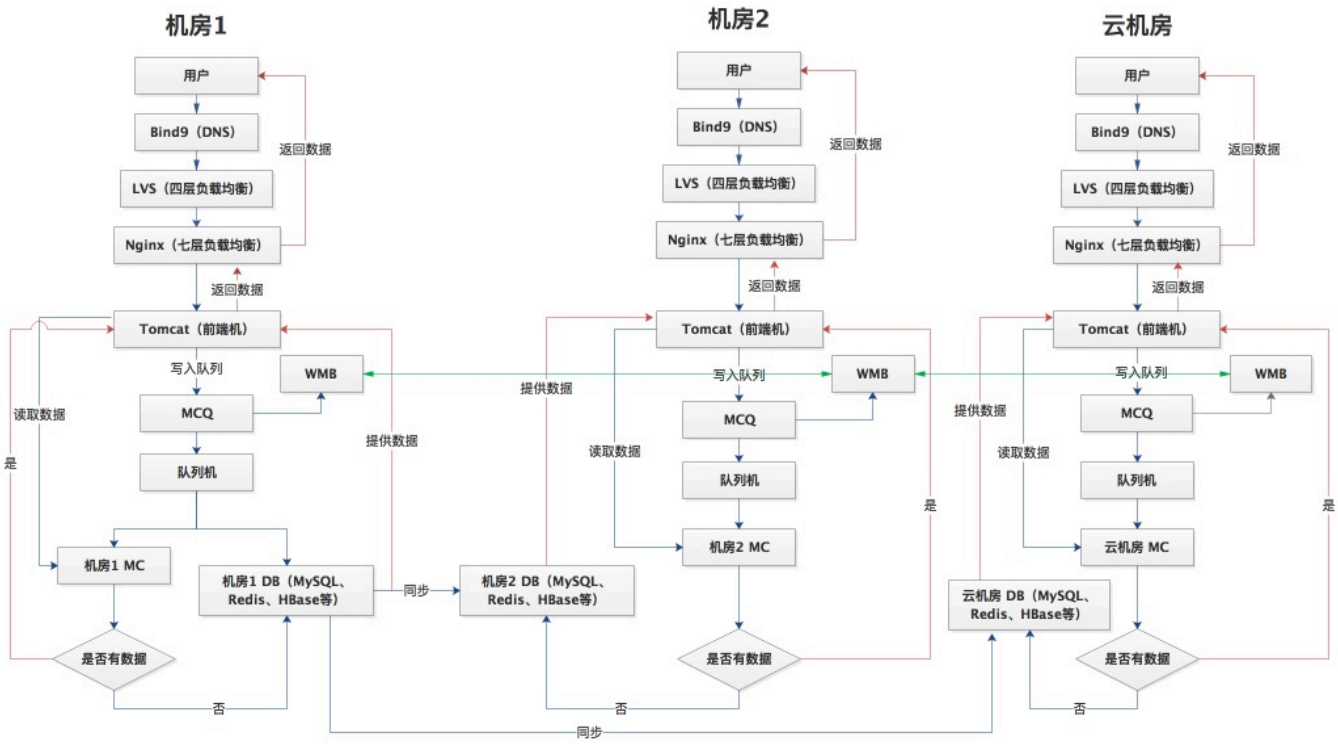

图6 平台服务部署架构图

就平台业务而言，后端的资源每天承载着千亿次的写请求，万亿次的读请求，如MySQL在流量极端峰值的情况下承载着几十万，甚至上百万的QPS，面对这么大体量的业务压力，对于后端资源的稳定性和RTT也提出了极高的要求，对业务部门承诺的SLA必须达到3个9，部分业务甚至4个9，且平均响应时间不能高于50ms。因此面临的挑战不言而喻。为了达到这一目标，需要有一个完善的，稳定可靠的，健壮的数据库运维体系来提供支撑和管理全平台25个分支下600多个产品线的业务运维。

数据库运维体系架构

图7 数据库运维体系架构图

整个运维体系由三大部分组成：UI层、应用服务层、基础服务层。UI层是各种管理平台的Dashboard和功能入口以及提供基础的Restful API等。应用服务层是数据库运维过程中所用到的各种功能模块集合，为RD和DBA提供最全面的数据库管理功能，这部分是整个运维体系中的核心组成部分，也是整个体系中最庞大和最复杂的部分。基础服务层是整个运维体系中最基础的依赖层，为数据库运维体系提供底层的基础服务，这里需要特别提出是，热点事件应对过程中的弹性扩缩容就是依赖这一层的DCP系统和Docker 镜像，结合私有云和公有云提供的ECS虚拟机来完成。

MySQL高可用架构

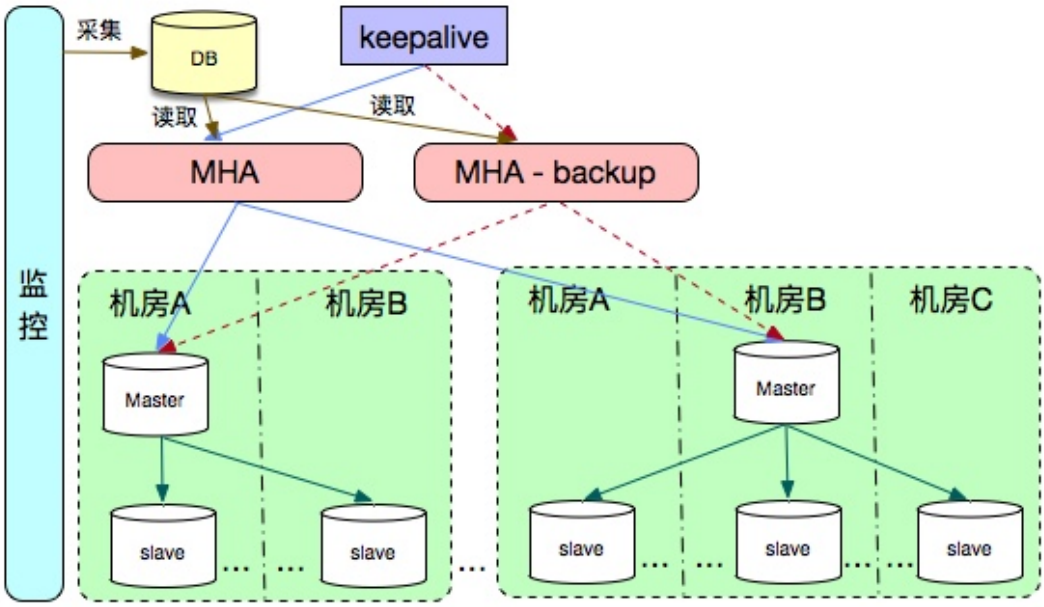

图8 MySQL高可用架构图

MHA（Master High Availability）目前在MySQL高可用方面是一个相对成熟的解决方案，它由日本DeNA公司youshimaton（现就职于Facebook公司）开发，是一套优秀的作为MySQL高可用性环境下故障切换和主从提升的高可用软件。在MySQL故障切换过程中，MHA能做到在0~30秒之内自动完成数据库的故障切换操作，并且在进行故障切换的过程中，MHA能在最大程度上保证数据的一致性，以达到真正意义上的高可用。微博MySQL高可用也采用基于MHA方案实现，这里MHA是在youshimaton开发的MHA 思想上自研而成。整个架构由三部分组成：监控、MHA和MySQL 集群（主从实例）组成。为了实现MySQL的高可用和灾备：

1. 在部署拓扑结构上，我们把MySQL的从库实例部署在不同的数据中心，给业务提供的是域名方式访问后端的DB资源，跨数据中心的同步带来的另一个好处是业务结构清晰和同机房前端访问同机房的后端资源，而不必跨机房访问，同时也方便架构上做流量切换的操作。
2. 我们会为被监控的MySQL主库添加一个白名单，白名单内的端口，在启用监控的条件下，能够实现在主库服务器宕机，所有从库实例都不可连主库，长时间服务器网络故障的条件下，自动完成主从切换。其中，主库check阶段，首先用从管理机上用python自带的MySQL库去连接mysql测试，如果连接失败，再检查主机层面连通性以及进程存活情况，如果失败，再考虑从所有从库去检查主库情况,优先先检查一台从库，这样做主要是为了防止误切。
3. 在MHA自身的高可用上，我们也是基于keepalive做了高可用的方案，保证MHA的可用性。

多级缓存架构

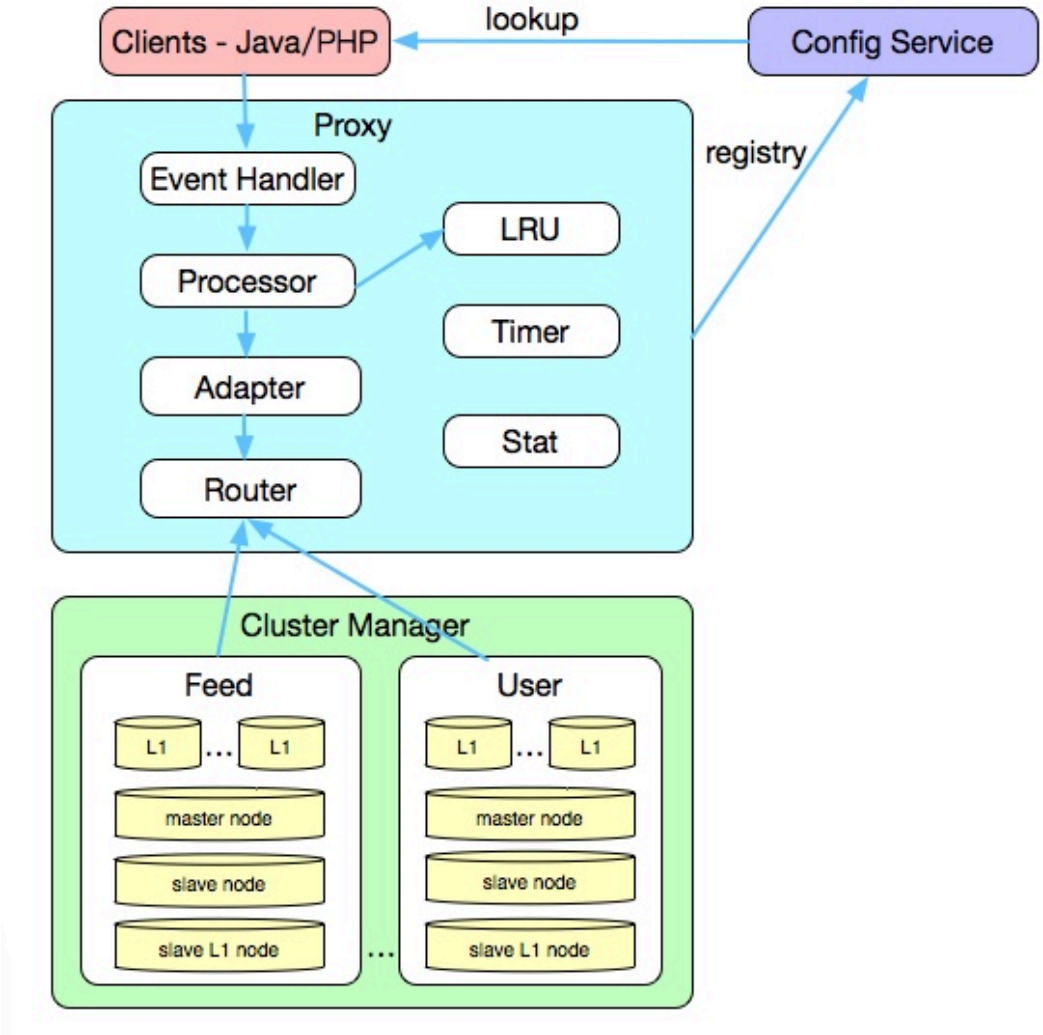

图9 多级缓存架构图

系统由几个模块组成：

- Config Service

  这一模块是基于现有微博的配置服务中心，它主要是管理静态配置和动态命名服务的一个远程服务，能够在配置发生变更的时候实时通知监听的config client。

- Proxy

  这一模块是作为独立的应用对外提供代理服务，用来接收来自业务端的请求，并基于路由规则转发到后端的Cache资源，它本身是无状态的节点。它包含了如下部分：

  - 异步事件处理(Event Handler): 用来管理连接、接收数据请求、回写响应。
  - Processor: 用来对请求的数据进行解析和处理。
  - Adapter：用来对底层协议进行适配，比如支持MC协议，Redis协议。
  - Router: 用来对请求进行路由分发，分发到对应的Cache资源池，进而隔离不同业务。
  - LRU Cache: 用来优化性能，缓解因为经过proxy多一跳(网络请求)而带来的性能弱化。
  - Timer: 用来执行一些后端的任务，包含对底层Cache资源健康状态的探测等。
  - Stat：proxy节点信息状态统计。

  Proxy启动后会去从Config Service加载后端Cache资源的配置列表进行初始化，并接收Config Service的配置变更的实时通知。

- Cluster Manager

  这一模块是作为实际数据缓存的管理模块，通过多层结构来满足服务的高可用。 首先解释一下各层的作用：

  图10 多级缓存中各角色含义

  其中，master是主缓存节点，slave是备份节点，当master缓存失效或实例挂掉后，数据还能够从slave节点获取，避免穿透到后端DB资源，L1主要用来抗住热点的访问，它的容量一般比master要小，平台的业务通常部署多组L1，方便进行水平扩容以支撑更高的吞吐。

- Client客户端

  这一模块主要是提供给业务开发方使用的Client(sdk包)，对外屏蔽掉了所有细节，只提供了最简单的get/set/delete等协议接口，从而简化了业务开发方的使用。

  应用启动时，Client基于namespace从Config Service中获取相应的proxy节点列表，并建立与后端proxy的连接。正常一个协议处理，比如set命令，client会基于负载均衡策略挑选当前最小负载的proxy节点，发起set请求，并接收proxy的响应返回给业务调用端。

  Client会识别Config Service推送的proxy节点变更的情况重建proxy连接列表，同时client端也会做一些容灾，在proxy节点出现问题的时候，把proxy进行摘除，并定期探测是否恢复。

目前微博平台部分业务子系统的Cache服务已经迁移到了CacheService之上，它在实际的运行过程中也取得了良好的性能表现，目前整个集群在线上每天支撑着超过千万级的QPS，平均响应耗时低于1ms。

它本身具备了以下特性：

- 高可用

  所有的数据写入请求，Cache Service会把数据双写到ha的节点，这样，在master缓存失效或挂掉的时候，会从slave读取数据，从而防止节点fail的时候给后端资源(DB等)带来过大的压力。

- 服务的水平扩展

  Cache Service proxy节点本身是无状态的，在proxy集群存在性能问题的时候，能够简单的通过增减节点来伸缩容。而对于后端的Cache资源，通过增减L1层的Cache资源组，来分摊对于

master的请求压力。这样多数热点数据的请求都会落L1层，而L1层可以方便的通过增减Cache资源组来进行伸缩容。

- 实时的运维变更

  通过整合内部的Config Service系统，能够在秒级别做到资源的扩容、节点的替换等相关的运维变更。目前这块主要结合DCP系统，利用docker 镜像以及虚拟机资源完成弹性的扩缩容。

- 跨机房特性：

  微博系统会进行多机房部署，跨机房的服务器网络时延和丢包率要远高于同机房，比如微博广州机房到北京机房需要40ms以上的时延。Cache Service进行了跨机房部署，对于Cache的查询请求会采用就近访问的原则，对于Cache的更新请求支持多机房的同步更新。

异步消息队列更新机制

互联网应用有个显著特点，就是读多写少。针对读多有很多成熟的解决方案，比如可以通过cache 来缓存热数据来降低数据库的压力等方式来解决。而对于写多的情况，由于数据库本身写入性能瓶颈，相对较难解决。我们知道春晚发祝福，或娱乐热点的时候发评论，点赞等场景下，会有大量的并发写入，微博这边为了解决这类问题，引入了“异步消息队列更新机制”。从微博平台的服务部署架构图中也可以看到，当用户发微博或评论等时不是直接去更新缓存和DB，而是先写入到MCQ消息队列中，再通过队列机处理程序读取消息队列中的消息，再写入到数据库和缓存中。那么，如何保证消息队列的读写性能，以及如何保证队列机处理程序的性能，是系统的关键所在。

- 按消息大小设置双重队列，保证写入速度

众所周知，16年中旬之前微博的最大长度不超过140个字，现在放开了这个限制。但是大部分用户的实际发表的微博长度都比较小，为了提高写入队列的速度，我们针对不同长度的微博消息，写入不同大小的消息队列，比如以512字节为分界线，大于512字节的写入长队列，小于512字节的写入短队列，其中短队列的单机写入性能要远远高于长队列。实际在线结果表明，短队列的QPS 在万/s 级别，长队列的QPS 在千/s 级别，而95%的微博消息长度均小于512字节。这种优化，大大提高了微博消息的写入和读取性能。

- 堵塞队列，压测队列的承载能力和队列机极限处理能力

为了验证队列机处理程序的极限处理能力，我们在业务低峰时期，对线上队列机进行了实际的压测，具体方法如下：通过开关控制，使队列机处理程序停止读取消息，从而堵塞消息队列，使堆积的消息分别达到10万，20万，30万，60万，100万，然后再打开开关，使队列机重新开始处理消息，整个过程类似于大坝蓄水，然后开闸泄洪，可想而知，瞬间涌来的消息对队列机将产生极大的压力。通过分析日志，来查找队列机处理程序最慢的地方，也就是瓶颈所在。通过两次实际的压测模拟，出乎意料的是，我们发现系统在极限压力下，首先达到瓶颈的并非是数据库写入，而是缓存更新。因此，为了提高极限压力下，队列机处理程序的吞吐量，我们对一部分缓存更新进行了优化。

另一方面，通过压测，也可以帮助我们发现队列的承载能力和处理能力，有效的帮助我们找到队列的短板，方便我们进行优化和扩容，通常情况下，核心队列都会保证3到4倍的冗余。

三、应对热点事件的手段和策略

扩容

这是DBA常采用的手段。扩容通常分为水平扩容和垂直扩容：

- 垂直扩容

垂直扩容，可以理解为保证架构不变的前提下，通过增加硬件投入即可实现的扩容方式。对于缓存来说，扩容是很容易，很方便，很快捷实现的，而对于DB来说，通常体积比较大，扩容起来很不方便，周期时间很长，尤其是应对流量的情况下，通常优先扩容缓存来抗热点，减少对后端DB的访问。而对于DB的扩容，我们会不定期的对核心DB资源进行评估，随着业务的不断发展，当超过一定的水位线的时候，为了保证DB的可用性，需要保留3到4倍的冗余，当不满足时，便需要扩容。因为热点总是不期而来，往往扩容缓存并没有那么的及时，这时候保证足够的冗余，有助于预防DB被打挂，保障服务的稳定性，显得很重要。

- 水平扩容

水平扩容，可以理解为通过调整架构，扩展资源的承载能力。比如：对于Redis服务来说，原本一个业务是4主4从，在业务量并发不高的情况下，完全满足业务需求，但是随着业务量上涨后，Redis的写性能变差，响应时间变慢，此时，我们需要对该业务进行水平扩展，将业务架构调整为8主8从或者更多，使得部署更多的主库还抗写入量。

降级

对于一个高可用的服务，很重要的一个设计就是降级开关，其目的是抛弃非重要服务，保障主要或核心业务能够正常提供服务。对于后端资源，包括**MySQL**、**MC**、**Redis**、队列等而言，并不能保证时时刻刻都是可用的，一旦出现问题时，降级策略就能派上用场。那么如何降级？降级的标准是什么呢？这里拿话题页举个例子。


0B/s
0B/s
16:54
#领航中国#
社会榜TOP4
阅读7.8亿 讨论30.5万 粉丝8250
主持人：新华视点
主页
精华
视频
相册
回复时间排序
排序
导语：习近平和十九大，你所关注的，都在这里！
置顶
新华社
XINHUA NEWS
新华视点
4小时前 来自 微博 weibo.com
#领航中国##十九大时间# 习近平说，全党要关心和爱护青年，为他们实现人生出彩搭建舞台。广大青年要坚定理想信念，志存高远，脚踏实地，勇做时代的弄潮儿，在实现中国梦的生动实践中放飞青春梦想，在为人民利益的不懈奋斗中书写人生华章。
已关注
发微博
聊天室
更多内容

图11 话题首页

图12　点击“导语”进入的二级页面

- 抛弃非核心大量运算业务，减轻自身计算服务压力。

任何一个系统，都包含核心系统和非核心系统。在出现异常的情况下，弃车保帅，只保障核心系统的稳定性也是可以接受的。对于上面的这个话题来说，话题首页是核心业务，话题二级页面是非核心业务。当话题**DB**资源出现瓶颈时，我们可以优先让**RD**同学降级二级页，释放**DB**的连接资源和减少查询请求，保障话题首页能够有更多的资源可用。

- 抛弃非核心模块，减轻自身资源压力。

同样，对于同一个业务，也要区分核心逻辑和非核心逻辑。对于话题首页的这个资源，新华视点发布的Feed流属于核心逻辑，而头像下方的 “社会榜TOP4” 榜单则属于非核心逻辑。毫无疑问，为了减负，非核心逻辑必要时是可以被降级的。

- 抛弃高耦合逻辑，避免依赖服务性能下降导致雪崩。

对于微博复杂的架构体系中，各种依赖和接口调用，错综复杂。就话题首页的资源来说，“阅读**7.8**亿　讨论**30.5**万　粉丝**8250**” 的计数就是调用微博平台的接口获取**Redis**计数器资源的结果的。对于热点事件而言，尤其是对于这种全平台都会可能会调用的资源来说，当时的访问量是极大的，响应时间肯定会比平时差，如果话题首页因为要取该资源而超时，导致话题“白页”，那不是想要的结果，所以需要抛弃这类高耦合逻辑。对于后端资源来说，当某个资源，比如某台**MCQ**服务器宕机，严重影响**V4**启动，拖累整体的动态扩容的时候，业务方就会考虑暂时**503**这台服务器实例资源，保证整个扩容流程的顺畅和速度。同时业务方也会根据请求的后端资源使用特点的不同，有时在降级前会使用一些**fast fail** 或 **fast over**的策略来保障服务的稳定性和可用性。

切流量

对于一个大型的系统来说，这也是一个常用的手段，通常可用从DNS、LVS、H5、HaProxy（四层负载均衡），Nginx（七层负载均衡）等层面上来实现流量的切换。常见的使用场景有：

1. 某机房业务流量达到瓶颈，影响业务的请求耗时和用户体验的时候，需要将到这个机房的请求量，一部分切到其他的机房，使得流量相对均衡一点。
2. 机房搬迁，机房断网、断电，也需要进行切流量操作，保证服务的可用性。
3. 业务架构调整、链路压测，比如将晚高峰的流量，从一个机房切到另一个机房，压测另一个机房的资源承载能力。
4. 线路带宽打满，由于某条专线的承载能力达到上限，不得不将流量从该线路切到其他的线路。春晚前，弹性扩缩容的需求很多，经常有很多业务方同时进行扩容，因为需要下载镜像，迁移数据，下发配置等，专线很容易成为瓶颈，故需要准备一个备用专线或提前提升专线的容量。
5. 运营商故障，虽然现在一般的机房都具备BGP的能力，但是有时候还是需要从业务层面去解决。

限流

从字面意思可以看出，限流的目的就是为了防止恶意请求（如刷站），恶意攻击，或者防止流量超出系统的峰值。处理的原则就是限制流量穿透到后端的资源，保障资源的可用性。举个例子：比如下图的QPS异常现象，请求流量超出平时晚高峰的好几倍，严重影响了服务的稳定性和后端DB的承载能力，为了保障DB资源的可用性，经排查是属于通过user_timeline接口的恶意刷站行为，此时，我们要做的就是封杀该接口，限制流量穿透到DB层，封杀后，效果很明显。

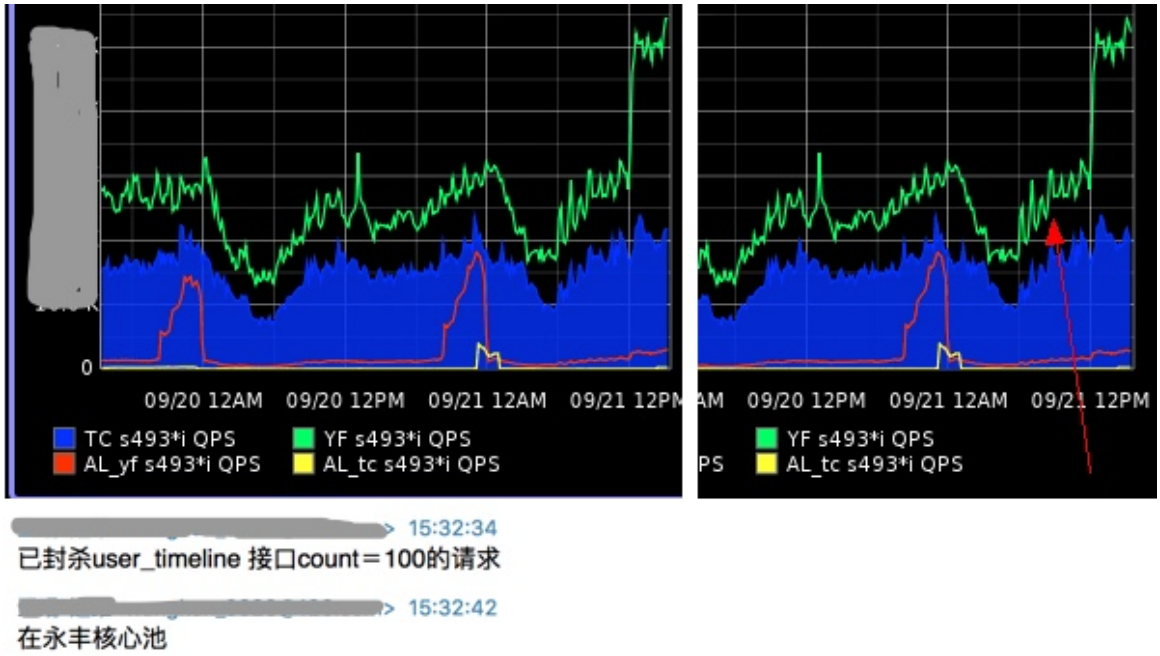

图13 刷站时QPS图

限流（封杀）后的效果如下图所示：

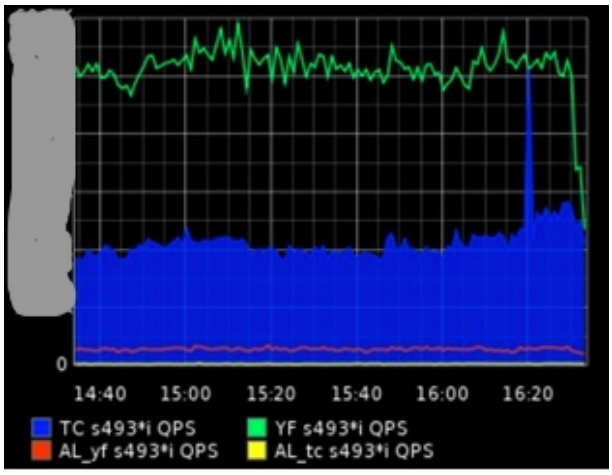

图14 限流效果图

除了封杀接口之外，其他一些可借鉴的思路是：

1. 对于恶意IP：可以使用nginx deny进行屏蔽或采用iptables进行限制。
2. 对于穿透到后端DB的请求流量：可以考虑用Nginx的limit模块处理或者Redis Lua脚本实现或者启用下面的过载保护策略。
3. 对于恶意请求流量：只访问到cache层或直接封杀。

- 过载保护

这个策略是针对DB资源设计的，就是当请求超过DB的承载范围时，启动的一个自我保护机制。比如，在某个明星出轨的时候，有用户恶意刷评论列表，导致评论的数据库从库资源出现大量的延迟，严重影响服务的性能，为了防止DB资源因过载而不可用，通过和业务方沟通后，果断启动了过载保护机制，从而保证了评论服务的稳定性。下图就是应用了过载保护机制后的效果。开源的工具可以使用percona公司开源的pt-kill工具实现。

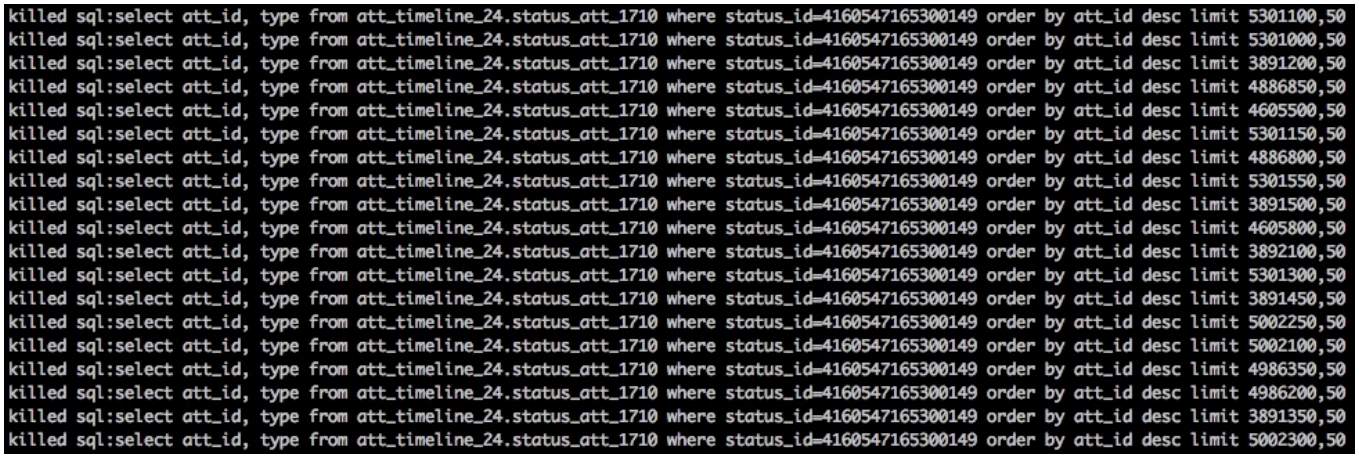

图15 过载保护效果图

链路压测

压测的目的是找出资源或者链路上的瓶颈点或者验证技术选型的可行性或测试优化的效果，这也是一个常态化的过程，一般分为两种场景：

- 找瓶颈点

这个通常是配合业务方共同完成，如上面的异步消息队列更新机制中提到的一样，通过切流量，堵塞队列，压测队列的承载能力和队列机极限处理能力，找出资源的瓶颈点，方便作为进行优化和扩容的依据，有时候也作为设置水位线的一个标准。

- 验证技术选型的可行性或测试优化效果

有时候为了引入新的技术，需要进行性能/压力测试，或者有时候为了验证某项优化的效果，然而压测很难模拟真实的线上环境场景，这时候在保证服务稳定可控的情况下，通过tcpcopy的方式引入线上的真实流量加以验证。

水位线&预警机制

这是一个快速有效的帮助了解线上业务的变化趋势，以及快速定位业务的实时运行情况，如果超过水位线，会触发预警机制，通过这种手段可以辅助DBA做出相应的策略判断，比如：扩容，限流，过载保护等。

四、需要完善和改进的地方或未来的规划

智能扩缩容

当前很多时候，尤其是当有热点的时候，DBA都是通过申请资源，部署服务，然后上线，完成一系列的手动操作完成扩容，这显然会浪费很多的人力，效率也不高。未来，可以通过设置水位线，增加对热key的预判，以及带宽的阈值，当超过警戒位后，通过开发的自动化工具来智能的进行MC的动态扩缩容。

故障自愈系统

故障无处不在，因为任何的硬件都有使用寿命，都有折损率。现在每天都可能有服务器宕机，服务器异常重启，服务器系统readonly，内存损坏，raid卡损坏，CPU损坏，主板损坏，磁盘损坏，电源模块损坏等等，各种硬件的故障都有可能会导致服务的异常，那么如何应对这些故障，保证服务器的高可用？是一个普遍面临的问题。微博这边有很多服务器都过保，它们就像一个“雷”一样无处不在，但是这些故障也是有轻有重，有紧急有非紧急的，根据具体的部署的业务不同而不同。服务器数量之多，每天报警几千条，全凭人工处理很难覆盖全面，这就急需结合监控和故障采集样本开发一套自愈系统，完成基础的，甚至大部分类型的故障修复工作。比如：自动报修。

移动办公

移动互联网时代，移动办公已经不是什么时髦的事情，很多的工作都能直接在手机或智能终端设备上完成。虽然目前有部分工作，比如工单审核，登录服务器执行命令等等都可以在手机上实现，但是还有很多的工作没有办法完成，比如查看监控，手动扩容等复杂的操作，需要后续的完善。

服务自助化

服务自助化，需要依赖完善的基础服务和健壮稳定的自动化平台，以及充足的冗余资源和完备的标准服务流程和规范才能完成。这是一个长期的过程，目前微博已经在SQL审核，Online DDL方面能够提供自助式服务，但是还有很多方面需要完善和提高。比如：资源申请，资源下线，资源扩容等等。