MGW——美团点评高性能四层负载均衡

王伟 · 2017-01-05 18:43

本文整理自美团点评技术沙龙第14期：美团背后的故事－你不知道的美团云。

美团点评技术沙龙由美团点评技术团队主办，每月一期。每期沙龙邀请美团点评及其他互联网公司的技术专家分享来自一线的实践经验，覆盖各主要技术领域。

目前沙龙会分别在北京、上海和厦门等地举行，要参加下一次最新沙龙活动？赶快关注微信公众号“美团点评技术团队”。

本期沙龙包括三场讲座：美团云Docker平台、美团云对象存储系统、美团四层负载均衡网关MGW。其他几场讲座的图文实录会陆续发表，请继续关注。

前言

在高速发展的移动互联网时代，负载均衡有着举足轻重的地位，它是应用流量的入口，对应用的可靠性和性能起着决定性的作用，因此负载均衡需要满足高性能、高可靠两个特点。MGW是美团点评自研的一款四层负载均衡，主要用于替代原有环境的四层负载均衡LVS，目前处理着美团点评数十Gbps的流量、上千万的并发连接。本文主要介绍MGW是如何实现高性能、高可靠的。

什么是负载均衡?

互联网早期，业务流量比较小并且业务逻辑比较简单，单台服务器便可以满足基本的需求；但随着互联网的发展，业务流量越来越大并且业务逻辑也越来越复杂，单台机器的性能问题以及单点问题凸显了出来，因此需要多台机器来进行性能的水平扩展以及避免单点故障。但是要如何将不同的用户的流量分发到不同的服务器上面呢?

早期的方法是使用DNS做负载，通过给客户端解析不同的IP地址，让客户端的流量直接到达各个服务器。但是这种方法有一个很大的缺点就是延时性问题，在做出调度策略改变以后，由于DNS各级节点的缓存并不会及时的在客户端生效，而且DNS负载的调度策略比较简单，无法满足业务需求，因此就出现了负载均衡。

客户端的流量首先会到达负载均衡服务器，由负载均衡服务器通过一定的调度算法将流量分发到不同的应用服务器上面，同时负载均衡服务器也会对应用服务器做周期性的健康检查，当发现故障节点时便动态的将节点从应用服务器集群中剔除，以此来保证应用的高可用。

负载均衡又分为四层负载均衡和七层负载均衡。四层负载均衡工作在OSI模型的传输层，主要工作是转发，它在接收到客户端的流量以后通过修改数据包的地址信息将流量转发到应用服务器。

七层负载均衡工作在OSI模型的应用层，因为它需要解析应用层流量，所以七层负载均衡在接到客户端的流量以后，还需要一个完整的TCP/IP协议栈。七层负载均衡会与客户端建立一条完整的连接并将应用层的请求流量解析出来，再按照调度算法选择一个应用服务器，并与应用服务器建立另外一条连接将请求发送过去，因此七层负载均衡的主要工作就是代理。

既然四层负载均衡做的主要工作是转发，那就存在一个转发模式的问题，目前主要有四层转发模式：DR模式、NAT模式、TUNNEL模式、FULLNAT模式。

DR模式也叫作三角传输，通过修改数据包的目的MAC地址来让流量经过二层转发到达应用服务器，这样应用服务器就可以直接将应答发给应用服务器，性能比较好。由于这种模式需要依赖二层转发，因此它要求负载均衡服务器和应用服务器必须在一个二层可达的环境内，并且需要在应用服务器上配置VIP。

NAT模式通过修改数据包的目的IP地址，让流量到达应用服务器，这样做的好处是数据包的目的IP就是应用服务器的IP，因此不需要再在应用服务器上配置VIP了。缺点是由于这种模式修改了目的IP地址，这样如果应用服务器

直接将应答包发给客户端的话，其源IP是应用服务器的IP，客户端就不会正常接收这个应答，因此我们需要让流量继续回到负载均衡，负载均衡将应答包的源IP改回VIP再发到客户端，这样才可以保证正常通信，所以NAT模式要求负载均衡需要以网关的形式存在于网络中。

TUNNEL模式的优缺点和DR是一样的，并且TUNNEL模式要求应用服务器必须支持TUNNEL功能。

FULLNAT模式是在NAT模式的基础上做一次源地址转换（即SNAT），做SNAT的好处是可以让应答流量经过正常的三层路由回到负载均衡上，这样负载均衡就不需要以网关的形式存在于网络中了，对网络环境要求比较低，缺点是由于做了SNAT，应用服务器会丢失客户端的真实IP地址。

下面详细介绍一下FULLNAT模式。首先负载均衡上需要存在一个localip池，在做SNAT时的源IP就是从localip池中选择的。当客户端流量到达负载均衡设备以后，负载均衡会根据调度策略在应用服务器池中选择一个应用服务器，然后将数据包的目的IP改为应用服务器的IP。同时从localip池中选择一个localip将数据包的源IP改为localip，这样应用服务器在应答时，目的IP是localip，而localip是真实存在于负载均衡上的IP地址，因此可以经过正常的三层路由到达负载均衡。由于FULLNAT比NAT模式多做了一次SNAT，并且SNAT中有选端口的操作，因此其性能要逊色于NAT模式，但是由于其较强的网络环境适应性，我们选择了FULLNAT作为MGW的转发模式。

为什么选择自研四层负载均衡?

选择自研四层负载均衡的原因主要有两个：第一个是考虑到硬件负载均衡成本比较高；第二个，随着美团点评业务流量越来越大，LVS出现了性能瓶颈以及运维成本的上升问题。

硬件负载均衡成本问题

1. 硬件成本：中低端硬件负载均衡价格在数十万，高端的上百万，价格非常昂贵。当我们需要组成一个高可用集群时，需要数台机器，成本异常高。
2. 人力成本：硬件负载均衡功能比较强大，配置比较灵活，这也导致在维护上，我们需要一些经过专业培训的人员，就增加了人力成本。

3. 时间成本：当使用的过程中遇到bug或者新需求需要厂商提供新版本的时候，我们需要经过繁琐的流程向厂商上报，然后厂商再发布新版本供我们升级，时间周期非常长，在高速发展的互联网行业，这种周期是无法接受的。

LVS的性能问题

最初美团点评使用的是LVS+Nginx组成的负载均衡结构，LVS做四层负载均衡，Nginx做七层负载均衡，但是随着美团点评流量的高速增长（几个月内无论新建连接数还是吞吐量都有三倍的增长），LVS故障频发，性能上出现瓶颈，因此我们自研了一款高性能、高可靠的四层负载均衡MGW来替换LVS。

MGW如何实现高性能

下面通过对比LVS的一些性能瓶颈来介绍MGW是如何实现高性能的。

中断问题以及协议栈路径性能过长问题

中断是影响LVS性能最重要的一个因素，假如我们一秒需要处理600万的数据包，每6个数据包产生一个硬件中断的话，那一秒就会产生100万个硬件中断，每一次产生硬件中断都会打断正在进行密集计算的负载均衡程序，中间产生大量的cache miss，对性能的影响异常大。

同时由于LVS是基于内核netfilter开发的一个应用程序，而netfilter是运行在内核协议栈的一个钩子框架。这就意味着当数据包到达LVS时，已经经过了一段很长的协议栈处理，但是这段处理对于LVS来说都不是必需的，这也造成了一部分不必要的性能损耗。

针对这两个问题，解决方法是使用轮询模式的驱动以及做kernel bypass，而DPDK提供的用户态PMD驱动恰好可以解决这两个问题。DPDK在设计时使用了大量硬件相关特性比如numa、 memory channel、 DDIO等，对性能优化非常大，同时提供了比较多网络方面的库，可以大大减小开发难度，提高开发效率。因此选择DPDK作为MGW的开发框架。

锁

由于内核是一个比较通用的应用程序，因此它并没有对一些特定场景做一些定制设计，这就导致一些公共的数据结构需要锁的保护。下面介绍一下出现锁的原因和MGW的解决方法。

首先介绍一下RSS（Receive Side Scaling），RSS是一个通过数据包的元组信息将数据包散列到不同网卡队列的功能，这时候不同的CPU再去对应的网卡队列读取数据进行处理，就可以充分利用CPU资源。之前介绍MGW使用FULLNAT的模式，FULLNAT会将数据包的元组信息全部改变，这样同一个连接，请求和应答方向的数据包有可能会被RSS散列到不同的网卡队列中，在不同的网卡队列也就意味着在被不同的CPU进行处理，这时候在访问session结构的时候就需要对这个结构进行加锁保护。

解决这个问题的方法有两种，一种就是在做SNAT选端口的时候，通过选择一个端口lport0让RSS(cip0, cport0, vip0, vport0) = RSS(dip0, dport0, lip0, lport0)相等；另外一种方法就是我们为每个CPU分配一个localip，在做SNAT选IP的时候，不同的CPU选择自己的localip，等应答回来以后，再通过lip和CPU的映射关系，将指定目的IP的数据包送到指定队列上。

由于第二种方法恰好可以被网卡的flow director特性支持，因此我们选择了第二种方法来去掉session结构的锁。

flow director可以根据一定策略将指定的数据包送到指定网卡队列，其在网卡中的优先级要比RSS高，因此我们在做初始化的时候就为每个CPU分配一个localip，比如为cpu0分配lip0，为cpu1分配lip1，为cpu2分配lip2，为cpu3分配lip3。 当一个请求包（cip0, cport0, vip0, vport0）到达负载均衡后，被RSS散列到了队列0上，这时这个包被cpu0处理。cpu0在对其做fullnat时，选择cpu0自己的localip lip0，然后将数据包（lip0, lport0, dip0, dport0）发到应用服务器，在应用服务器应答后，应答数据包（dip0, dport0, lip0, lport0）被发到了负载均衡服务器。此时我们就可以在flow director下一条将目的IP为lip0的数据包送到队列0的规则，这样应答数据包就会被送到队列0让cpu0处理。这时候CPU在对同一个连接两个方向的数据包进行处理的时候就是完全串行的一个操作，也就不要再对session结构进行加锁保护了。

上下文切换

在设计时，希望控制平面与数据平面完全分离，数据平面专心做自己的处理，不被任事件打断。因此将CPU分成两组，一组用作数据平面一组用做控制平面。同时，对数据平面的CPU进行CPU隔离，这样控制平面的进程就不会调度到数据平面的这组CPU上面了；对数据平面的线程进行CPU绑定，这样就可以让每个数据线程独占一个CPU。 其他的控制平面的程序比如Linux kernel、 SSH等都跑在控制平面的这组CPU上。

MGW如何做到高可靠

下面从MGW集群、MGW单机以及应用服务器层这三个层介绍MGW如何在每一层实现高可靠。

集群的高可靠

MGW使用OSPF+ECMP的模式组成集群，通过ECMP将数据包散列到集群中各个节点上，再通过OSPF保证单台机器故障以后将这台机器的路由动态的剔除出去，这样ecmp就不会再给这台机器分发流量，也就做到了动态的failover。

传统的ecmp算法有一个很严重的问题，当集群中节点数量发生变化以后，会导致大部分流量的路径发生改变，发生改变的流量到达其他MGW节点上时是找不到自己的session结构的，这就会导致大量的连接出现异常，对业务影响很大，并且当我们在对集群做升级操作时会将每个节点都进行一次下线操作，这样就加重了这个问题的影响。

一种解决方式是使用支持一致性hash的交换机，这样在节点发生变化的时候，只有发生变化的节点上面的连接会有影响，其他连接都会保持正常，但是支持这种算法的交换机比较少，并且也没有完全实现高可用，因此我们做了集群间的session同步功能。

集群中每个节点都会全量的将自己的session同步出去，使集群中每个节点都维护一份全局的session表，因此无论节点变化以后流量的路径以任何形式改变，这些流量都可以找到自己的session结构，也就是说可以被正常的转发，这样就可以在集群中节点数量发生变化时保证所有连接正常。

在设计的过程中主要考虑了两个问题：第一个是故障切换，第二个是故障恢复以及扩容。

故障切换

在故障切换的问题上，我们希望在机器故障以后，交换机可以立刻将流量切到其他机器上，因为流量不切走，意味着到达这台机器流量会被全部丢掉，产生大量丢包。经过调研测试发现，当交换机侧全部使用物理接口并且服务器侧对接口进行断电时，交换机会瞬间将流量切换到其他机器上。通过一个100ms发两个包的测试（客户端和服务端各发一个），这种操作方法是0丢包的。

由于故障切换主要依赖于交换机的感知，当服务器上出现一些异常，交换机感知不到时，交换机就无法进行故障切换操作，因此需要一个健康自检程序，每半秒进行一次健康自检，当发现服务器存在异常时就对服务器执行网口断电操作，从而让流量立刻切走。

故障切换主要依赖于网口断电操作并且网卡驱动是跑在主程序里面的，当主程序挂掉以后，就无法再对网口执行断电操作了，因此为了解决这个问题，主进程会捕获异常信号，当发现异常时就对网卡进行断电操作，在断电操作结束以后再继续将信号发给系统进行处理。

经过以上设计，MGW可以做到升级操作0丢包，主程序故障0丢包，其他异常（网线等）会有一个最长500ms的丢包，因为这种异常需要靠自检程序去检测，而自检程序的周期是500ms。

故障恢复与扩容

无论是在进行故障恢复还是扩容操作，都会导致集群节点数量发生变化，这样也就会导致流量路径发生变化。当变化的流量到达集群中原有的节点时，因为原有节点都维护着一个全局的session表，因此这些流量是可以被正常转发的；但是如果流量到达了新机器上，这个机器是没有全局session表的，那么这部分流量就会全部被丢弃。为了解决这个问题，MGW在上线以后会经历一个预上线的中间状态，在这个状态上，MGW不会让交换机感知到自己上线了，这样交换机也就不会把流量切过来。首先MGW会对集群中其他节

点发送一个批量同步的请求，其他节点收到请求以后会将自己的session全量的同步到新上线的节点上，新上线节点在收到全部session以后才会让交换机感知到自己上线，这时交换机再将流量切过来就可以正常被转发出去了。

在这个过程中主要存在两点问题。
第一个问题是，由于集群中并没有一个主控节点来维护一个全局的状态，如果request报丢失或者session同步的数据丢失的话，那新上线节点就没办法维护一个全局的session状态。但是考虑到所有节点都维护着一个全局的session表，因此所有节点拥有的session数量都是相同的，那么就可以在所有节点每次做完批量同步以后发送一个finish消息，finish消息中带着自己拥有的session数量。当新上线节点收到finish消息以后，便会以自己的session数量与finish中的数量做对比。当达到数量要求以后，新上线节点就控制自己进行上线操作。否则在等待一定的超时时间以后，重新进行一次批量同步操作，直到达到要求为止。

另外一个问题是在进行批量同步操作时，如果出现了新建连接，那么新建连接就不会通过批量同步同步到新上线的机器上。如果新建连接特别多，就会导致新上线机器一直达不到要求。因此，需要保证处于预上线状态的机器能接收到增量同步数据，因为新建连接可以通过增量同步同步出来。通过增量同步和批量同步就可以保证新上线机器可以最终获得一个全局的session表。

单机高可靠

在单机高可靠方面，MGW做了一个自动化测试平台，自动化平台通过连通性和配置的正确性来判断一个测试用例是否执行成功，失败的测试用例平台可以通过邮件通知测试人员。在每次新功能迭代结束以后，都会将新功能的测试用例加到自动化平台里面，这样在每次上线之前都进行一次自动化测试，可以大大避免改动引发的问题。

在之前，每次上线之前都需要进行一次手动的回归测试，回归测试非常耗时并且很容易遗漏用例，但是为了避免改动引发新问题又不得不做，有了自动化测试平台以后，大大提高了回归测试的效率和可靠性。

RS可靠性

节点平滑下线

在RS可靠性方面，MGW提供了节点平滑下线功能，主要是为了解决当用户需要对RS进行升级操作时，如果直接将需要升级的RS下线，那这个RS上存在的所有连接都会失败，影响到业务。此时如果调用MGW的平滑下线功能，MGW就可以保证此RS已有连接正常工作，但不会往上面调度新的连接。当所有已有连接结束以后，MGW会上报一个结束的状态，用户就可以根据这个结束的状态对RS进行升级操作，升级后再调用上线接口让这个RS器进行正常的服务。如果用户平台支持自动化应用部署，那就可以通过接入云平台使用平滑下线功能，实现完全自动化且对业务无影响的升级操作。

一致性源IP Hash调度器

源IP Hash调度器主要是保证相同的客户端的连接被调度到相同应用服务器上，也就是说建立一个客户端与应用服务器一对一的映射关系。普通的源IP Hash调度器在应用服务器发生变化以后会导致映射关系发生改变，会对业务造成影响。

因此我们开发了一致性源IP Hash调度器，保证在应用服务器集群发生变化时，只有发生变化的应用服务器与客户端的映射关系发生改变，其他都是不变的。

为了保证流量的均衡，首先在hash环上分配固定数量的虚拟节点，然后将虚拟机节点均衡的重分布到物理节点上，重分布算法需要保证两点：

1. 在物理节点发生变化时，只有少数虚拟节点映射关系发生变化，也就是要保证一致性Hash的基本原则。
2. 因为MGW是以集群的形式存在的，当多个应用服务器发生上线下线操作时，反馈到不同的MGW节点上就有可能会出现顺序不一致的问题，因此无论不同的MGW节点产生何种应用服务器上下线顺序，都需要保证最终的映射关系一致，因为如果不一致就导致相同客户端的连接会被不同的MGW节点调度到不同的应用服务器上，也就违背了源IP Hash调度器的原则。

综合以上两点，Google Maglev负载均衡的一致性Hash算法是一个很好的例子，在paper中有详细的介绍，这里就不过多讨论了。

总结

经过美团点评以及美团云的流量验证，MGW无论在传统网络环境还是overlay的大二层环境下都有出色的性能和稳定性表现。在业务场景方面涵盖数据库业务，千万级别的长连接业务，嵌入式业务，存储业务以及酒店、外卖、团购等Web应用业务。在业务需求快速变化的环境下，MGW在不断完善自身功能，在各种业务场景下都有良好的表现。 在未来的一段时间内，MGW除了会完善四层的功能需求外，也会考虑向七层方向发展。

参考资料

1. DPDK.
2. LVS.
3. Eisenbud D E, Yi C, Contavalli C, et al. Maglev: A Fast and Reliable Software Network Load Balancer.