负载均衡

February 24, 2013

高性能高可用性的系统一般会有多层次的负载均衡(High-performance systems may use multiple layers of load balancing)。负责均衡的实现机制有硬件和软件两种。下面我们以top down的方式一层层介绍下去。

总体来说负责均衡分为无状态服务负载均衡，以及有状态的数据负载均衡两类。前者（比如web server，app Server）由于是无状态的，所以backen-server是对等的，可以路由到任何一台机器，路由的策略一般是业务无关的（灰度除外），比如backen-server的负载情况。而对于有状态的数据负载均衡（比如cache，MQ或者DB），对等是基本不可能的，因为数据存放多份带来的写压力以及一致性问题往往使得读操作那点负载均衡得不偿失（特别是事务型的数据操作）。而且对等数据元写操作无法实现负载均衡。所以一般是对于数据类的backen-server，采用的往往是sharding策略。这样，负载均衡就必须是业务路由了。

说明 关于有状态的服务资源扩容和负载均衡没有那么简单，需要单独写一篇文章来介绍。简单来说就是：

1. 能不sharding就不要sharding。sharding主要是为了解决写入瓶颈，而不是读取瓶颈。读性能完全可以通过 Cache + Read Only Slave + Search 来解决。
2. 虽然采用sharding来进行负载均衡，但是相对于无状态的对等服务的线性拓展的方便程度，为了避免分片的单点问题，一般来说每个分片还是会有一个或者多个slave作为backup的。

无状态服务负载均衡

1. 前端web服务器负载均衡

一般的做法是：

首先是使用DNS做第一层的负载均衡(DNS based load balancing cluster)，同一个域名可能映射到不同的服务器（IP地址）。一般采用简单Round-robinDNS策略。也有采用GSLB做全局负载均衡的（参见下面补充说明部分）。

其次域名一般不会直接映射到具体的某个web服务器，而是先到一个负载均衡服务器。负载均衡服务器根据实现方式不同一般有两种：

1. 硬件负载均衡：如F5。
2. 软件负载均衡：如HAProxy，LVS，等。

   软件负载均衡服务器也有不同的实现机制：

   1. 有用户空间的负载均衡程序

      如HAProxy, nginx，等。这种一般是建立在TCP/IP协议之上的负载均衡，所以也称

为application-level load balancing，或者称为Layer-7 switching。 缺点就是性能比较差，但是功能会比较多，除了负载均衡，还能作Web服务器，可以Cache，可以压缩等等。 用户空间的负载均衡程序一般有两种实现：1. event-driven, single-process model；或者Multi-process or multi-threaded models。

2. 还有编译进入内核的负载均衡程序

   如LVS，这种一般涉及到修改IP层package，所以也称为IP-based load balancers，或者Layer-4 switching）。

2. 后端应用服务器负责均衡

后端应用服务器之间的通讯一般都是服务调用的方式，一般还会跨语言。所以也称为**组件负载平衡**(CLB, Component Load Balancing)。

现在业界普遍的做法就是采用配置中心+客户端负载均衡的方式实现服务路由和负载均衡。使用`trift`或者`protocol buffer`来实现跨语言交互（主要是序列化以及桩代码生成）。

为了避免每次都是去配置中心查询服务注册信息，一般还会使用本地agent作为cache（本地配置中心，类似于git的分布式结构）。agent可以定时拉取中心的修改，中心也可以下发到指定机器的agent中。agent还会应用上报的调用情况（主要是请求成功率和请求延时这两个关键指标）调整负载均衡策略：

- 负载均衡：以请求成功率和请求延时这两个关键指标进行动态权重计算，动态均衡各个被调服务器的负载，达到较好的整体服务质量；
- 故障容错：迅速自动屏蔽错误率高或有故障的机器，并进行适时探测，待故障恢复后自动恢复；
- 过载保护：实现对单台机器或者整个模块机器的过载保护能力，防止雪崩现象。

有状态的数据负载均衡

1. 数据库负载均衡

数据库如果把SQL解析与存储IO操作分开来看的话，那么可以将SQL解析部分看成无状态服务，可以采用前面讨论的负载均衡方案进行负载均衡。然后共享存储。不过由于DB的瓶颈其实在于存储的IO性能（一个具体例子就是B2B的FileServer）。所以关键还是存储的负载均衡。

当我们说负载均衡的时候其实我们就是考虑scale out而不是scale up了。DB的scale out方式一般就是sharding了。一般来说先根据业务进行垂直切分，比如将detail表单独切分出来。如果记录还是太大了，就用hash(key)进行水平切分。hash的key值一般是业务相关的，比如卖家id，要保证不会有跨key的查询，否则查询效率低下。

sharding对业务其实是有影响的，但是可以通过中间件透明化。让应用看起来好像还是访问一个数据源头，当然由于key是业务相关的，而且是routing的依据，所以每次SQL都需要带上key进行hash路由。所以其实也不是特别透明。

阿里以前使用的是一个叫做Cobar的中间件。貌似已经开源了。也有用HAProxy做LB的。不过不是sharding。 1. USING HAPROXY FOR MYSQL FAILOVER AND REDUNDANCY 2. A more stable MySQL with HAProxy

另外，对于事务要求不是很高的业务，可以考虑NoSQL解决方案，比如HBase可以实现自动扩

容和负载均衡。

2. Cache

Cache本质上与DB没有区别，可以看做是关系型数据库的后端存储子系统——内存中的key-value记录存储，当然也有支持持久化的，如memcachedb和redis。一般采用一致性哈希(Consistent hashing)算法进行mapping。

cache扩容分为两种不同的方式。

1. 写入能力扩容：通过re-sharding扩容shard分片的数量来提升写入能力
2. 读取能力扩容：线性增加sharding-group的数量，读请求分散到不同的group，实现读请求的load balance与带宽的均匀分布。这里面存在一个矛盾，group的数量越多，则写入操作时，需要更新的组就越多，反过来会影响写入的性能与复杂度，因此在设计扩容方案时要权衡这两点。

3. 文件系统（FileServer）

像图片、音乐、视频这样的二进制文件一般不会存储在DB中，而是存放在文件系统中。

文件系统跟DB很相似，都是落在磁盘上。但是当文件数多了之后，基于操作系统自身的文件系统在性能一般都很差。因为一次文件存取需要多次IO才能进行。具体参见笔者以前的文章[海量图片存储思考]。

解决方案一般是两步走：

1. 合并文件元数据。最简单的做法就是放在DB中。
2. 分布文件数据。一般是异步进行，而且可以控制需要replicate多少台。

现成解决方案——分布式文件系统

1. mogilefs
2. GFS

实际企业级应用实例

Alibaba B2B

前端DNS做全局负载均衡。DNS后面是F5做硬件负载均衡。apache+jboss做web和app server（部署在一起，使用mod_jk做通讯）。使用内部服务化框架dubbo作为服务负载均衡（配置中心+客户端负载均衡，不支持多语言）。使用memcached作为分布式cache。使用MySQL sharding进行DB scale out。使用FileServer+NFS共享存储作为文件服务器。

Tencent ECC

前端GSLB+DNS做全局负载均衡。GSLB+DNS后面是LVS做内核级别的软负载均衡。nginx+tomcat做web和app server（分开部署）。使用内部服务化框架IDL作为服务负载均衡（配置中心+本地agent缓存+客户端负载均衡，支持多语言）。使用memcached作为分布式cache。使用MySQL sharding进行DB scale out。使用内部分布式文件系统TFS存储海量文件。

一些补充和说明

LVS

LVS(Linux Virtual Servers)是目前最好的IP-based load balancers。它的优点是编译进内核，速度非常快。缺点就是平台和内核版本相关，目前只支持linux。配合Keepalived来监控后台servers的情况，效果非常好。

Linux Virtual Server (LVS) Project

Load Balancing The UK National JANET Web Cache Service——Using Linux Virtual Servers

DNS

1. 域名解析：是指完成域名向IP的映射。域名解析的初衷是为了用易于记忆的字符串替代IP地址。目前，它不仅能够完成其本职任务，同时也是互联网服务运营中的有力工具，GSLB即是基于域名解析的全局负载均衡解决方案。
2. RR：Resource Rocord的首字母缩写，意为资源记录。在域名解析过程中，一跳完整的记录就称作一条资源记录，如`www 300 IN A 119.147.15.17`。RR通常有几个字段构成，具体可以参考《DNS与BIND》一书。
3. A记录：用于指定域名对应的IP地址，如`www 300 IN A 119.147.15.17`即为一条A记录。
4. CNAME记录：别名记录，可以给同一个IP或域名指定多个不同的名字，如`www.xxx.com. 600 IN CNAME www.xxx.forward.yy.com.`即为一条CNAME记录。
5. MX记录：邮箱路由记录，可以将某个域名下的域名服务器指向到指定的邮箱服务器，如`xx.com 43200 IN MX 10 mail.xx.com.`。
6. TXT记录：文本记录，用户进行域名的辅助信息说明。但现在TXT类型记录有了越来越多的扩展应用，如用于记录SPF信息，domain-key信息等。
7. TTL：Time To Live的首字母缩写，意为生存时间，代表DNS记录在DNS服务器上缓存的时间，通常以秒为单位，允许的值为0-2^{32}；如`www 300 IN A 119.147.15.17`中的第二个字段就是TTL，说明该RR记录在DNS服务器中会缓存300，之后就会被丢弃。

GSLB

全局负载均衡（GSLB, Global Server Load Balance）的目的是实现在广域网（包括互联网）上不同地域的服务器间的流量调配，保证使用最佳的离自己最近的客户服务器服务，从而确保访问质量。它对服务器和链路进行综合判断来决定由哪个地点的服务器来提供服务，实现异地服务器群服务质量的保证。

GSLB目前主要有基于DNS实现、基于重定向实现、基于路由协议实现三种方式。

对于全局负载均衡而言，为了执行就近性判断，通常可以采用两种方式，一种是静态的配置，例如根据静态的IP地址配置表进行IP地址到CDN节点的映射。另一种方式是动态的检测，例如实时地让CDN节点探测到目标IP的距离（可以采用RRT，Hops作为度量单位），然后比较探测结果进行负载均衡。当然，静态和动态的方式也可以综合起来使用。

TX的GSLB平台使用基于DNS系统的实现。他的粒度一直能细到各个运营商，各个省份，也就是我们可以配置不同的运营商返回不同的ip，不同的省返回不同的ip的目录。

1. 无线下的分运营商访问：由于手机上网的局限性，在速度本来就慢的情况，不跨网访问的重要性就突显出来了，而这个可以通过gslb来搞定，让不同的运营商用户到相应运营商ip去。

2. 用户的就近接入：这里是主要是分省给不同的ip，比如华东用户去上海访问相应资源，华南去深圳或者汕头，东北到天津等，这里的的重要性主要体现在cdn的应用上，如果一些大的资源，应用软件，用户头像等。这里不是基于RTT之类的动态检测方式，而是通过维持一个静态的IP地址库(注意，不是用户的IP， 而是LocalDNS的IP)，通过映射快速查找。
3. 运维对业务的控制加强：业务运维可以方便的通过gslb来控制自己域名，如果设置短ttl使更改更快生效，比如可以把一个省的用户定向到测试服务器来达到测试的目的，还可以在网络或者机房有故障的时候快速把受影响的用户通过dns转移到没有问题的机房，把影响降到最低。

当然GSLB调度也存在一些问题，主要是如下两个缺陷：

1. 域名解析有TTL时间，如果发生故障，GSLB配置屏蔽，生效至少需要一个TTL时间。
2. 调度精度依赖LocalDNS，比如一个在北京的用户配置使用了深圳的dns服务器（特别是配置8.8.8.8，你们懂的。），GSLB系统会误认为这个北京用户在深圳，就会把这个用户调度到深圳的服务器上。又比如某些运营商的dns服务器建设较少，只集中在几个省份，这样一来GSLB的调度能力就大大折扣。

负载均衡的三角策略

负载均衡涉及到三个角色：client，balancer和backend-server。 请求路径非常明确：client==>balancer==>backend-server 但是根据负载均衡实现机制的不同，返回路径有两种：1. backend-server==>balancer==>client 2. backend-server==>client

对于第一种方式，一般是使用一种称之为NAT(http://www.linux-vs.org/VS-NAT.html)的路由策略。所有应用层的load balancer(如HAProxy，Nginx，等)也只能是这种方式。由于返回需要再次经过LB，所以笔者也称之为`store-and-forward` Routing方式。

对于第二种，由于返回的时候backend-server直接绕过load balancer，说明以下几点：

1. backend-server有真实的外部IP地址，也就是说backend-server也是在一个LAN/WAN网络里。而不是一个私有网络。
2. backend-server必须有client的IP信息，否则它没法发送给client，也就是说balancer发给它的package中除了包含它自己的IP信息（IP协议要求）还必须包含client的IP信息。

解决这个问题的方案就是IP tunneling，也称为IP encapsulation，其实是一个非常简单协议，就是为了解决发送端IP地址与返回地址不是同一个的问题，要保留两种，那么简单再增加一个IP头部就是了。不过要求balancer和backend-servers都支持IP分装协议(IP encapsulation protocol)。 具体的实现机制和原理参见LVS的这篇文档：Virtual Server via IP Tunneling。

另一种做法就是直接路由方式(Direct Routing)：就是load balancer和backend-server处于同样的地位：双IP，其中一个IP都是配置同样的VIP，处于同一个局域网内。当请求过来的时候，balancer直接将package转发给挑选到的backend-server，因为两者都有同样的VIP，所以就不需要修改IP头或者做IP tunnele了。具体做法参见：Virtual Server via Direct Routing。

返回消息不经过load balancer能够带来极大的性能提高，因为请求消息一般小而快，而返回消息一般比较大而慢。让backend-server直接返回给client，极大的解放了load balancer。

关于 LVS提供的三种routing方式，这篇文章介绍的非常详细：Load Balancing The UK National JANET Web Cache Service——Using Linux Virtual Servers

疑问：load balancer本身不会成为瓶颈和单点吗？

由于load balancer基本就是一个forward逻辑，所以很快，吞吐量非常高。如果采用backend-server直接返回client的模式，那么不需要改写response信息，吞吐量会进一步提高。再配合NDS负载均衡，一般不会有问题。

对于单点问题，一般的作法是配对一个backup，与active server做heart beat。如果有状态的话，一般是共享存储，或者在backup server之间做数据同步。如果当前的活动server宕掉了，那么backup Server通过程序自动接管IP地址，从而将服务迅速切换到backup server。除非有复杂的状态问题，一般不需要人工接入。也可以使用ZoopKeeper进行master选举，实现自动切换。

关于HA具体做法可以参考如下文章：

1. The heartbeat+mon+coda solution
2. The heartbeat+ldirectord solution

在这些负载均衡服务器会配置需要负载均衡的实际web服务器，一般是nginx或者apache。负载均衡的策略一般是最简单也是最有效的随机或者轮循策略。而事实上，由于现在大部分网站都是动态内容，这些web服务器往往也是一个有cache和压缩等功能的load balancer，为后台的应用服务器（tomcat、JBoss等）做负载均衡。

ECC L5负载均衡组件的功能职责

L5的功能特征如下：

- 名字服务：以SID（由模块ID和命令字ID组成）为关键字，通过SID取得真正的IP和端口地址，使得IP和端口配置对调用者透明，运维变更配置更方便；
- 负载均衡：以请求成功率和请求延时这两个关键指标进行动态权重计算，动态均衡各个被调服务器的负载，达到较好的整体服务质量；
- 故障容错：迅速自动屏蔽错误率高或有故障的机器，并进行适时探测，待故障恢复后自动恢复；
- 过载保护：实现对单台机器或者整个模块机器的过载保护能力，防止雪崩现象。

另外，为了提高性能和避免单点，一般都有本地agent作为缓存。

主要核心算法：

1. 负载均衡算法: 以请求成功率和请求延时构成的动态权重，结合预先配置的静态权重，最后构成了被调节点的调度权重，然后以周期性建立的分配模型分配给各个主调节点；另外还需要考虑机器伸缩的时候，负载的变化需要平滑过度的问题。
2. 门限收缩算法: 考虑到系统运行是一个动态过程，任何静态的最大阀值最小阀值限制都是不科学的，所以需要根据延时和成功率情况对阀值进行动态伸缩，这样才能更好的契合业务的需要。
3. 宕机探测算法: 由于机器临时性故障难免，通过成功率的逐渐下降可以预知宕机风险，并及时排除；另外需要以极低的代价，及时探测被点节点，并自动恢复负载。

参考文章

1. Why DNS Based Global Server Load Balancing (GSLB) Doesn’t Work 关于GSLB和DNS非常详尽的资料，虽然有点老，但是并没有过时。
2. Why DNS Based GSLB Doesn't Work, Part II
3. CDN的四大关键技术 CDN与GSLB有很大关系。

Related Posts

- 24 Jul 2015 » Tomcat调优
- 22 Jul 2015 » 记一次MySQL主从同步错误处理
- 03 Jul 2015 » Metric监控系统