**摘要**：对于一个大型网站来说，负载均衡是永恒的话题。随着硬件技术的迅猛发展，越来越多的负载均衡硬件设备涌现 出来，如F5 BIG-IP、Citrix NetScaler、Radware等等，虽然可以解决问题，但其高昂的价格却往往令人望而却步，因此负载均衡软件仍然是大部分公司的不二之选。 nginx作为webserver的后起之秀，其优秀的反向代理功能和灵活的负载均衡策略受到了业界广泛的关注。本文将以工业生产为背景，从设计实现和具 体应用等方面详细介绍nginx负载均衡策略。

**关键字**：nginx 负载均衡 反向代理

 负载均衡技术在现有网络结构之上提供了一种廉价、有效、透明的方法，来扩展网络设备和服务器的带宽、增加吞吐量、加强网络数据处理能力、提高网络的灵活性和可用性。它有两方面的含义：首先，大量的并发访问或数据流量分担到多台节点设备上分别处理，减少用户等待响应的时间；其次，单个重负载的运算分担到多台节点设备上做并行处理，每个节点设备处理结束后，将结果汇总，返回给用户，系统处理能力得到大幅度提高

http://stblog.baidu-tech.com/wp-includes/js/tinymce/plugins/wordpress/img/trans.gif

**1.前言**

随着互联网信息的爆炸性增长，负载均衡（load balance）已经不再是一个很陌生的话题，顾名思义，负载均衡即是将负载分摊到不同的服务单元，既保证服务的可用性，又保证响应足够快，给用户很好的 体验。快速增长的访问量和数据流量催生了各式各样的负载均衡产品，很多专业的负载均衡硬件提供了很好的功能，但却价格不菲，这使得负载均衡软件大受欢 迎，nginx就是其中的一个。

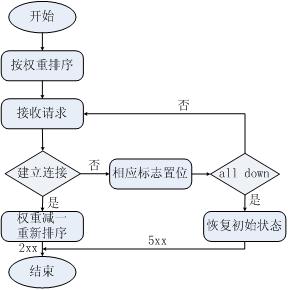
nginx第一个公开版本发布于2004年，2011年发布了1.0版本。它的特点是稳定性高、功能强大、资源消耗低，从其目前的市场占有而 言，nginx大有与apache抢市场的势头。其中不得不提到的一个特性就是其负载均衡功能，这也成了很多公司选择它的主要原因。本文将从源码的角度介 绍nginx的内置负载均衡策略和扩展负载均衡策略，以实际的工业生产为案例，对比各负载均衡策略，为nginx使用者提供参考。

**2.   源码剖析**

nginx的负载均衡策略可以划分为两大类：内置策略和扩展策略。内置策略包含加权轮询和ip hash，在默认情况下这两种策略会编译进nginx内核，只需在nginx配置中指明参数即可。扩展策略有很多，如fair、通用hash、 consistent hash等，默认不编译进nginx内核。由于在nginx版本升级中负载均衡的代码没有本质性的变化，因此下面将以nginx1.0.15稳定版为例， 从源码角度分析各个策略。

**2.1.           加权轮询（weighted round robin）**

轮询的原理很简单，首先我们介绍一下轮询的基本流程。如下是处理一次请求的流程图：

[](http://stblog.baidu-tech.com/wp-content/uploads/wp-display-data.php?filename=13433621041.JPG&type=image%2Fjpeg&width=288&height=290)

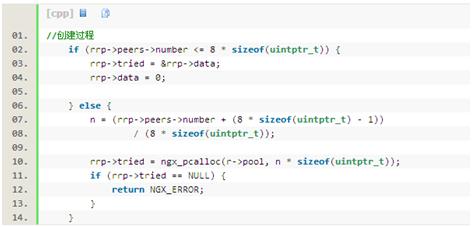
图中有两点需要注意，第一，如果可以把加权轮询算法分为先深搜索和先广搜索，那么nginx采用的是先深搜索算法，即将首先将请求都分给高权重的机 器，直到该机器的权值降到了比其他机器低，才开始将请求分给下一个高权重的机器；第二，当所有后端机器都down掉时，nginx会立即将所有机器的标志 位清成初始状态，以避免造成所有的机器都处在timeout的状态，从而导致整个前端被夯住。

接下来看下源码。nginx源码的目录结构很清晰，加权轮询所在路径为nginx-1.0.15/src/http /ngx\_http\_upstream\_round\_robin.[c|h]，在源码的基础上，针对重要的、不易理解的地方我加了注释。首先看下 ngx\_http\_upstream\_round\_robin.h中的重要声明：

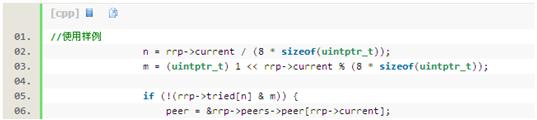
[](http://stblog.baidu-tech.com/wp-content/uploads/wp-display-data.php?filename=13433621331.JPG&type=image%2Fjpeg&width=462&height=297)

从变量命名中，我们就可以大致猜出其作用。其中，current\_weight和weight的区别主要是前者为权重排序的值，随着处理请求会动态的变化，后者是配置值，用于恢复初始状态。

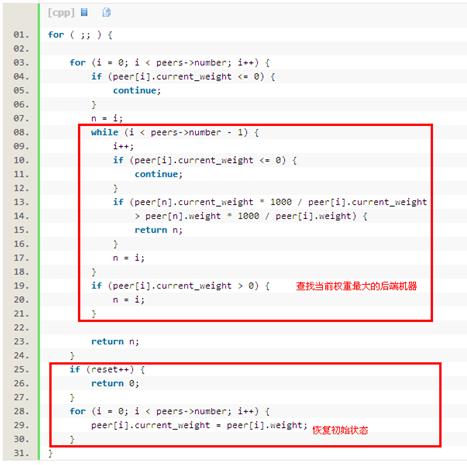
接下来看下轮询的创建过程，代码如下图所示。

[](http://stblog.baidu-tech.com/wp-content/uploads/wp-display-data.php?filename=13433621691.JPG&type=image%2Fjpeg&width=472&height=227)

这里有个tried变量需要做些说明。tried中记录了服务器当前是否被尝试连接过。他是一个位图。如果服务器数量小于32，则只需在一个int中即可记录下所有服务器状态。如果服务器数量大于32，则需在内存池中申请内存来存储。对该位图数组的使用可参考如下代码：

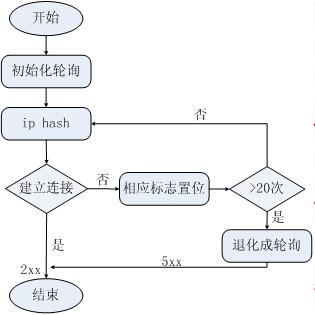
[](http://stblog.baidu-tech.com/wp-content/uploads/wp-display-data.php?filename=13433622181.JPG&type=image%2Fjpeg&width=537&height=120)

最后是实际的策略代码，逻辑很简单，代码实现也只有30行，直接上代码。

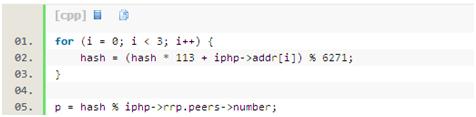
[](http://stblog.baidu-tech.com/wp-content/uploads/wp-display-data.php?filename=13433622551.JPG&type=image%2Fjpeg&width=467&height=465)

**2.2.           ip hash**

ip hash是nginx内置的另一个负载均衡的策略，流程和轮询很类似，只是其中的算法和具体的策略有些变化，如下图所示：

[](http://stblog.baidu-tech.com/wp-content/uploads/wp-display-data.php?filename=13433676191.JPG&type=image%2Fjpeg&width=315&height=315)

ip hash算法的核心实现如下图：

[](http://stblog.baidu-tech.com/wp-content/uploads/wp-display-data.php?filename=13433693801.JPG&type=image%2Fjpeg&width=476&height=117)

从代码中可以看出，hash值既与ip有关又与后端机器的数量有关。经过测试，上述算法可以连续产生1045个互异的value，这是该算法的硬限 制。对此nginx使用了保护机制，当经过20次hash仍然找不到可用的机器时，算法退化成轮询。因此，从本质上说，ip hash算法是一种变相的轮询算法，如果两个ip的初始hash值恰好相同，那么来自这两个ip的请求将永远落在同一台服务器上，这为均衡性埋下了很深的 隐患。

**2.3.           fair**

fair策略是扩展策略，默认不被编译进nginx内核。其原理是根据后端服务器的响应时间判断负载情况，从中选出负载最轻的机器进行分流。这种策略具有很强的自适应性，但是实际的网络环境往往不是那么简单，因此要慎用。

**2.4.           通用hash、一致性hash**

这两种也是扩展策略，在具体的实现上有些差别，通用hash比较简单，可以以nginx内置的变量为key进行hash，一致性hash采用了nginx内置的一致性hash环，可以支持memcache。

**3.   对比测试**

本测试主要为了对比各个策略的均衡性、一致性、容灾性等，从而分析出其中的差异性，并据此给出各自的适用场景。为了能够全面、客观的测试nginx 的负载均衡策略，我们采用了两个测试工具、在不同场景下做测试，以此来降低环境对测试结果造成的影响。首先简单介绍测试工具、测试网络拓扑和基本的测试流 程。

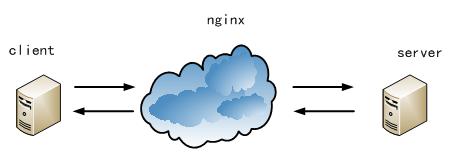
**3.1.           测试工具**

**3.1.1  easyABC**

easyABC是公司内部开发的性能测试工具，采用epool模型实现，简单易上手，可以模拟GET/POST请求，极限情况下可以提供上万的压 力，在公司内部得到了广泛的使用。由于被测试对象为反向代理服务器，因此需要在其后端搭建桩服务器，这里用nginx作为桩webserver，提供最基 本的静态文件服务。

**3.1.2  polygraph**

polygraph是一款免费的性能测试工具，以对缓存服务、代理、交换机等方面的测试见长。它有规范的配置语言PGL（Polygraph Language），为软件提供了强大的灵活性。其工作原理如下图所示：

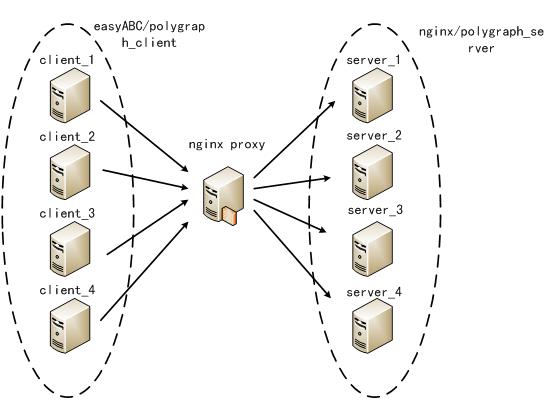
[](http://stblog.baidu-tech.com/wp-content/uploads/wp-display-data.php?filename=13433701121.JPG&type=image%2Fjpeg&width=452&height=157)

polygraph提供client端和server端，将测试目标nginx放在二者之间，三者之间的网络交互均走http协议，只需配置 ip+port即可。client端可以配置虚拟robot的个数以及每个robot发请求的速率，并向代理服务器发起随机的静态文件请求，server 端将按照请求的url生成随机大小的静态文件做响应。这也是选用这个测试软件的一个**主要原因**：可以产生随机的url作为nginx各种hash策略的key。

另外，polygraph还提供了日志分析工具，功能比较丰富，感兴趣的同学可以参考附录中的相关材料。

**3.2.           测试环境**

本测试运行在5台物理机上，其中被测对象单独搭在一台8核机器上，另外四台4核机器分别搭建了easyABC、webserver桩和polygraph，如下图所示：

[](http://stblog.baidu-tech.com/wp-content/uploads/wp-display-data.php?filename=13433701701.JPG&type=image%2Fjpeg&width=554&height=401)

**3.3.           测试方案**

首先介绍下关键的测试指标：

**均衡性**：是否能够将请求均匀的发送给后端

**一致性**：同一个key的请求，是否能落到同一台机器

**容灾性**：当部分后端机器挂掉时，是否能够正常工作

以上述指标为指导，我们针对如下四个测试场景分别用easyABC和polygraph进行测试：

场景1      server\_\*均正常提供服务；

场景2      server\_4挂掉，其他正常；

场景3      server\_3、server\_4挂掉，其他正常；

场景4      server\_\*均恢复正常服务。

上述四个场景将按照时间顺序进行，每个场景将建立在上一个场景基础上，被测试对象无需做任何操作，以最大程度模拟实际情况。另外，考虑到测试工具自 身的特点，在easyabc上的测试压力在17000左右，polygraph上的测试压力在4000左右。以上测试均保证被测试对象可以正常工作，且无 任何notice级别以上（alert/error/warn）的日志出现，在每个场景中记录下server\_\*的qps用于最后的策略分析。

**3.4.           测试结果**

表1和图1是轮询策略在两种测试工具下的负载情况。对比在两种测试工具下的测试结果会发现，结果完全一致，因此可以排除测试工具的影响。从图表中可以看出，轮询策略对于均衡性和容灾性都可以做到很好的满足。(点击图片查看大图)

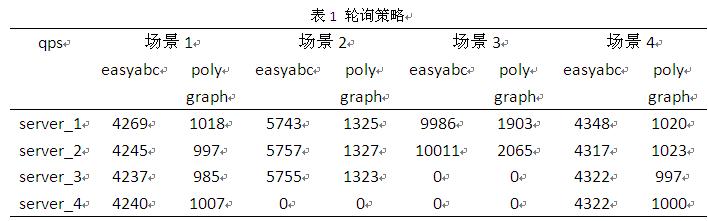
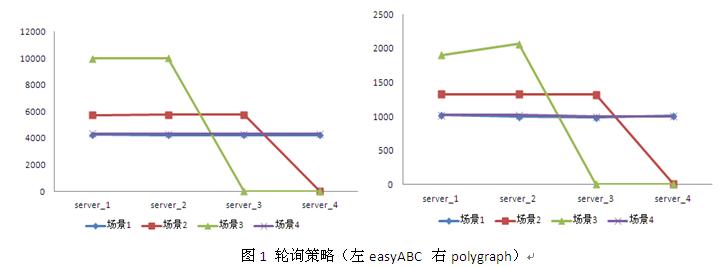
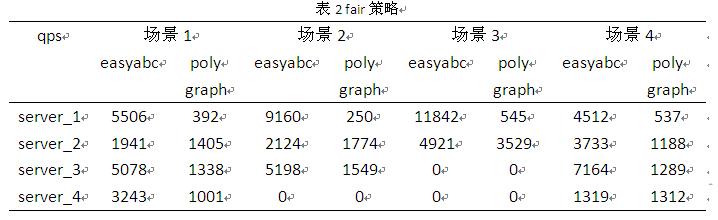
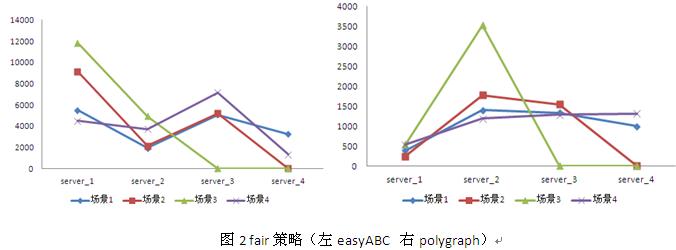
[](http://stblog.baidu-tech.com/wp-content/uploads/wp-display-data.php?filename=13433702311.JPG&type=image%2Fjpeg&width=708&height=221)[](http://stblog.baidu-tech.com/wp-content/uploads/wp-display-data.php?filename=13433702611.JPG&type=image%2Fjpeg&width=718&height=268)

表2和图2是fair策略在两种测试工具下的负载情况。fair策略受环境影响非常大，在排除了测试工具的干扰之后，结果仍然有非常大的抖动。从直 观上讲，这完全不满足均衡性。但是从另一个角度出发，恰恰是由于这种自适应性确保了在复杂的网络环境中能够物尽所用。因此，在应用到工业生产中之前，需要 在具体的环境中做好测试工作。(点击图片查看大图)

[](http://stblog.baidu-tech.com/wp-content/uploads/wp-display-data.php?filename=13433705291.JPG&type=image%2Fjpeg&width=718&height=220)[](http://stblog.baidu-tech.com/wp-content/uploads/wp-display-data.php?filename=13433705761.JPG&type=image%2Fjpeg&width=676&height=250)

以下图表是各种hash策略，所不同的仅仅是hash key或者是具体的算法实现，因此一起做对比。实际测试中发现，通用hash和一致性hash均存在一个问题：当某台后端的机器挂掉时，原有落到这台机器 上的流量会丢失，但是在ip hash中就不存在这样的问题。正如上文中对ip hash源码的分析，当ip hash失效时，会退化为轮询策略，因此不会有丢失流量的情况。从这个层面上说，ip hash也可以看成是轮询的升级版。(点击图片查看大图)

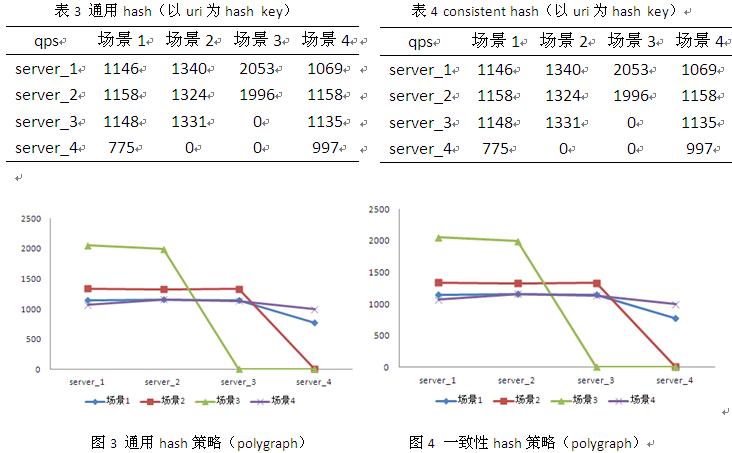
[](http://stblog.baidu-tech.com/wp-content/uploads/wp-display-data.php?filename=13433706331.JPG&type=image%2Fjpeg&width=732&height=453)

图5为ip hash策略，ip hash是nginx内置策略，可以看做是前两种策略的特例：以来源ip为key。由于测试工具不便于模拟海量ip下的请求，因此这里截取线上实际的情况加以分析，如下图所示：

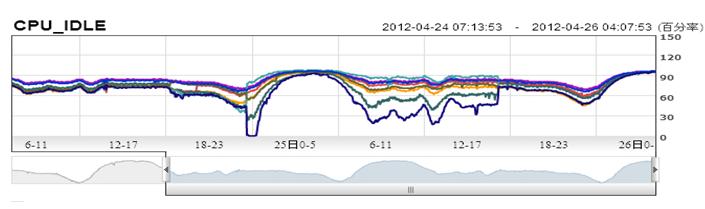
[](http://stblog.baidu-tech.com/wp-content/uploads/wp-display-data.php?filename=13433706851.JPG&type=image%2Fjpeg&width=705&height=209)

图5 ip hash策略

图中前1/3使用轮询策略，中间段使用ip hash策略，后1/3仍然是轮询策略。可以明显的看出，ip hash的均衡性存在着很大的问题。原因并不难分析，在实际的网络环境中，有大量的高校出口路由器ip、企业出口路由器ip等网络节点，这些节点带来的流 量往往是普通用户的成百上千倍，而ip hash策略恰恰是按照ip来划分流量，因此造成上述后果也就自然而然了。

**4.   总结与展望**

通过实际的对比测试，我们对nginx各个负载均衡策略进行了验证。下面从均衡性、一致性、容灾性以及适用场景等角度对比各种策略。(点击图片查看大图)

[](http://stblog.baidu-tech.com/wp-content/uploads/wp-display-data.php?filename=13433707471.JPG&type=image%2Fjpeg&width=716&height=168)

以上从源码和实际的测试数据角度分析说明了nginx负载均衡的策略，并给出了各种策略适合的应用场景。通过本文的分析不难发现，无论哪种策略都不 是万金油，在具体的场景下应该选择哪种策略一定程度上依赖于使用者对这些策略的熟悉程度。希望本文的分析和测试数据能够对读者有所帮助，更希望有越来越 多、越来越好的负载均衡策略产出。