**高速负载均衡器设计，保证每个连接的一致性**

**挑战：**

大型服务提供商使用负载平衡器每秒向数千台服务器发送数百万个传入连接。

**负载平衡器有两个基本但关键的要求：跨服务器的传入连接的均匀负载分布和每个连接的一致性（PCC），**

**Pcc就是**即使在服务器和负载平衡器的数量发生变化的情况下，也能够将属于同一连接的包映射到同一服务器。

**背景：**

数据中心运营商为每个运营的服务分配一个**虚拟IP（VIP）地址**。

DC中的每个VIP都与一组提供该服务的服务器相关联。

每个服务器都有**一个直接IP（DIP）地址**，它唯一地标识DC中的服务器。

LB接收某个VIP的传入连接并选择一个服务器来提供请求的服务。

**解决方案：**

CHEETAH，一个负载平衡系统，**它支持任意负载平衡机制，并在不牺牲性能的情况下保证PCC。设计CHEETAH支持任何可实现的负载平衡机制PCC而不违反PCC**

该系统具有以下**特性**：

•动态性 -- LBs和服务器的数量可根据实际负载而增减；

•每个连接一致性（PCC）-- 将属于同一连接的包转发到同一服务器上；

•统一的负载分配 -- 通过支持高效利用服务器的高级负载平衡机制；

•高效的数据包处理 --- LB对通信延迟的影响最小

•弹性 -- 客户端很难用虚假流量“阻塞”LB和服务器。

**CHEETAH将连接映射的信息存储到连接本身**

CHEETAH通过将**有关连接的信息编码到cookie**中，并将其**添加到连接的所有数据包**中

**CHEETAH的两种不同实现，无状态版本和有状态版本**

无状态LBs（不存储任何每个连接状态）

有状态LBs（存储有关正在进行的连接的一些信息）

**3.1无状态负载均衡器。**

**将一个不透明的偏移量编码到cookie中**

目标是在支持任意负载平衡机制并保证PCC的同时，实现与当今生产就绪的无状态LBs相同的弹性水平。

假设采用单层LB体系结构

AllServers表将服务器标识符映射到服务器的DIP

VIPToServers表将每个VIP映射到运行该VIP的服务器集

AllServers表基本上是静态的，因为它包含针对DC网络中的每个服务器的条目。只有当服务器部署到DC（ datacenters）中/从DC中删除时，AllServers表才会更新。

当运行某项服务的服务器数量增加/减少时，VIPToServers表被修改，这是**处理VIP当前需求变化**的一种更常见的操作。

**过程：**

当LB接收到连接的第一个包时，它从VIPToServers表中提取运行该服务的服务器集，根据任何预先配置的负载平衡机制选择其中的一个服务器，并转发该包。

对于从服务器接收到的每个包，LB为这个连接在cookie中编码一个服务器映射的“不透明”标识符。

为此，CHEETAH使用salt S（客户机未知）计算连接标识符的哈希值，用服务器的标识符对其进行XOR运算，并将XOR的输出作为cookie添加到**包头**中。salt S对所有连接都是一样的。

当LB接收到属于该连接的任何后续包时，它从包报头提取cookie，用salt S计算连接标识符的散列，用cookie对散列的输出进行XOR，并将XOR的输出用作服务器的标识符。然后，LB在AllServers表中查找服务器的DIP。

**无状态的CHEETAH保证PCC。**CHEETAH依靠两个主要的设计思想来避免连接中断**:**

**(i)将保存连接及其服务器之间映射所需的状态转移到连接的包头中;**

**(ii)仅对连接的第一个包使用更动态的VIPToServers表。**

**3.2 状态负载均衡器。**

它改进了当前有状态LBs的包处理性能，并提出了一个LB体系结构，在性能和弹性方面进行了不同的权衡。

有状态LB可以跟踪每个单独连接的行为，并支持复杂的网络功能，如速率限制器、NAT、重量级检测，以及重新路由到专用的清理设备

**在包头中编码表索引**

我们存储一组m ConnTable表，**这些表保存每个连接的统计信息和DIP映射**。

还使用相同数量的ConnStack索引堆栈，每个堆栈将未使用的条目存储在其相应的ConnTable中。

每当需要安装新的连接状态时，CHEETAH会**从ConnStack弹出一个索引**，并将其作为cookie的一部分**合并到包的头**中。它还将选定的服务器和带有salt S的连接标识符的散列存储到相应的表条目中。这个散列值允许LB过滤出干扰合法流量的恶意尝试。每当属于现有连接的包到达LB时，CHEETAH从cookie中提取索引并使用它快速执行ConnTable中的查找。请注意，连接的插入、修改和删除可以完全在数据平面中以恒定的时间执行。

**实验：**

**1.包处理分析。**

**无状态CHEETAH的数据包处理成本最小。**无状态CHEETAH与最优化的、基于硬件的散列机制消耗的CPU周期数几乎相同，并且比有状态方法消耗的CPU周期要少得多。

**Stateful CHEETAH的性能优于基于buckoo哈希的LBs**。

**2．负荷不平衡分析。**

与性能最好的基于哈希的机制（即哈希RSS）相比，CHEETAH将第99个百分位FCT降低了2-3倍。

**3.PCC违规分析。**

图9示出了随着时间的推移，中断请求与已完成请求之间的百分比。

一些连接在十几秒后被认为是断开的，因为客户端在引发错误之前继续发送重传。与

Beamer相比，Cheetah不仅实现了AWRR，但也不会断开任何连接。