网络速率限制以及 Guava 的 RateLimiter

在互联网网络中，当网络发生拥塞（congestion）时，交换机将开始丢弃数据包。这可能导致数据重发（retransmissions）、数据包查询（query packets），这些操作将进一步导致网络的拥塞。为了防止网络拥塞（network congestion），需限制流出网络的流量，使流量以比较均匀的速度向外发送。

主要有两种限流算法：漏桶算法（Leaky Bucket）和令牌桶算法（Token Bucket）

漏桶算法（Leaky Bucket）

漏桶算法是网络世界中流量整形（Traffic Shaping）或速率限制（Rate Limiting）时经常使用的一种算法，它的主要目的是控制数据注入到网络的速率，平滑网络上的突发流量。漏桶算法提供了一种机制，通过它，突发流量可以被整形以便为网络提供一个稳定的流量。

漏桶算法思路很简单，水（数据或者请求）先进入到漏桶里，漏桶以一定的速度出水，当水流入速度过大会直接溢出，可以看出漏桶算法能强行限制数据的传输速率。

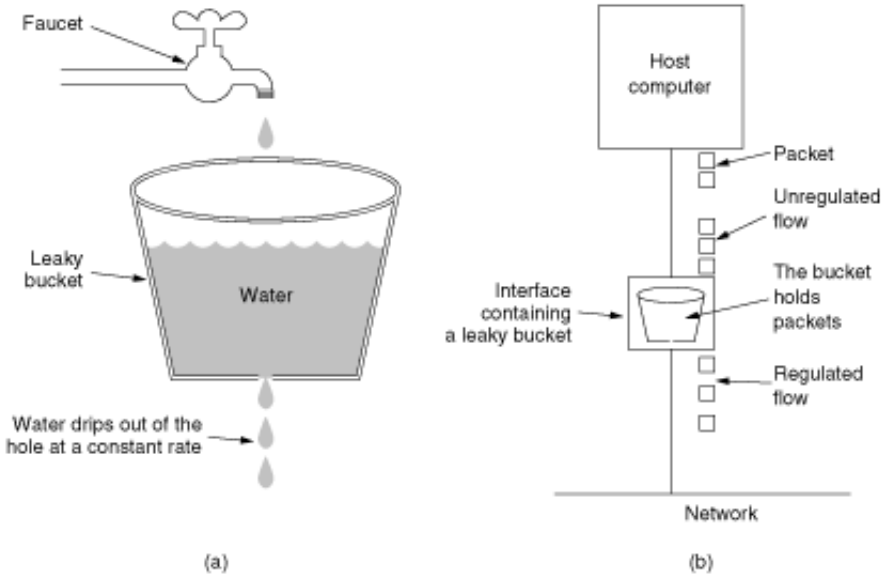

Fig. 5-24. (a) A leaky bucket with water. (b) A leaky bucket with packets.

如果想及时了解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公共帐号：iteblog_hadoop

令牌桶算法（Token Bucket）

令牌桶算法的思想是：大小固定的令牌桶可自行以恒定的速率源源不断地产生令牌，并将产生的令牌放入桶中。如果令牌不被消耗，或者被消耗的速度小于产生的速度，令牌就会不断地增多，直到把桶填满。后面再产生的令牌就会从桶中溢出。最后桶中可以保存的最大令牌数永远不会超过桶的大小。

传送到令牌桶的数据包需要消耗令牌。不同大小的数据包，消耗的令牌数量不一样。令牌桶这种控制机制基于令牌桶中是否存在令牌来指示什么时候可以发送流量。令牌桶中的每一个令牌都代表一个字节。如果令牌桶中存在令牌，则允许发送流量；而如果令牌桶中不存在令牌，则不允许发送流量。因此，如果突发门限被合理地配置并且令牌桶中有足够的令牌，那么流量就可以以峰值速率发送。

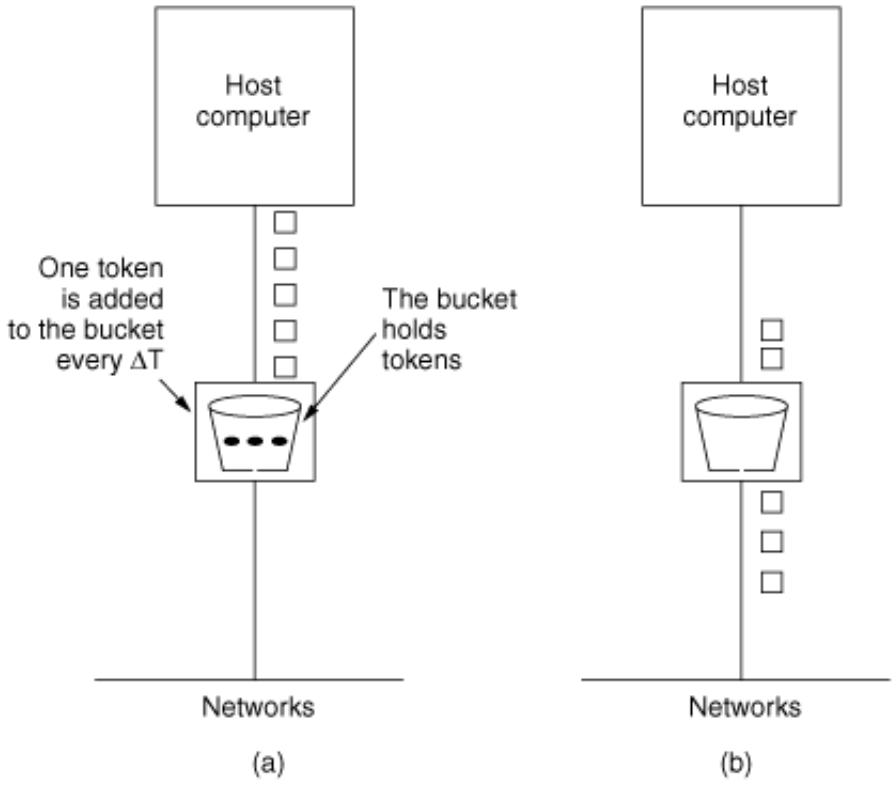

Fig. 5-26. The token bucket algorithm. (a) Before. (b) After.

漏桶算法和令牌桶算法的区别

漏桶算法与令牌桶算法在表面看起来类似，很容易将两者混淆。但事实上，这两者具有截然不同

的特性，且为不同的目的而使用。漏桶算法与令牌桶算法的区别在于：

- 漏桶算法能够强行限制数据的传输速率。
- 令牌桶算法能够在限制数据的平均传输速率的同时还允许某种程度的突发传输。

Guava 中的 RateLimiter

流量限速最初来源于计算机网络，但是随着当今互联网技术的迅速发展，流量限速也被用于实际的项目工程中。比如双 11 网站限流等。

Guava 中的 RateLimiter 就是使用上面的令牌桶算法来进行限流的，RateLimiter 会按照一定的频率往桶里扔令牌，线程拿到令牌才能执行，比如你希望自己的应用程序 QPS 不要超过 800，那么 RateLimiter 设置 800 的速率后，就会每秒往桶里扔 800 个令牌。在我们项目中使用这个也非常的简单，如下：

```
package com.iteblog.www

import java.util.concurrent.{Callable, Executors, TimeUnit}

import com.google.common.util.concurrent.{MoreExecutors, RateLimiter}

object IteblogRateLimitTest {
  class Task(index: Int) extends Callable[Integer] {
    override def call: Integer = {
      println("Execute Task - " + index)
      TimeUnit.SECONDS.sleep(1)
      0
    }
  }

  def main(args: Array[String]): Unit = {
    val executorService = MoreExecutors.listeningDecorator(Executors.newCachedThreadPool)

    val limiter = RateLimiter.create(2.0)

    (0 until 10).foreach { i =>
      limiter.acquire()
      val listenableFuture = executorService.submit(new Task(i))
    }
    executorService.shutdown()

  }
}
```

上面示例限制每秒的令牌数为2，所以可以限制每秒的 QPS 为2。limiter.acquire() 就是用于获取令牌的。