架构必备： Rate limiting 的作用和常见方式

架构 2015-08-21 23:06:57 发布

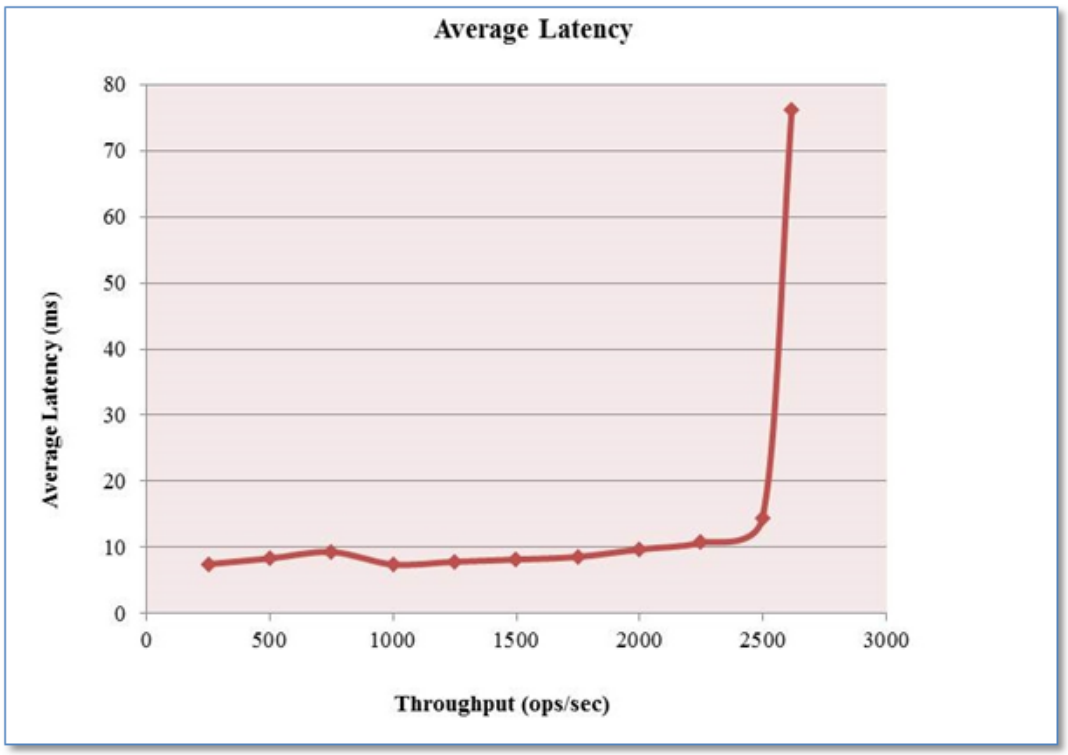
原文 http://blog.eood.cn/rate-limiting

Rate limiting 在 Web 架构中非常重要，是互联网架构可靠性保证重要的一个方面。

从最终用户访问安全的角度看，设想有人想暴力碰撞网站的用户密码；或者有人攻击某个很耗费资源的接口；或者有人想从某个接口大量抓取数据。大部分 人都知道应该增加 Rate limiting，做请求频率限制。从安全角度，这个可能也是大部分能想到，但不一定去做的薄弱环节。

从整个架构的稳定性角度看，一般 SOA 架构的每个接口的有限资源的情况下，所能提供的单位时间服务能力是有限的。假如超过服务能力，一般会造成整个接口服务停顿，或者应用 Crash，或者带来连锁反应，将延迟传递给服务调用方造成整个系统的服务能力丧失。有必要在服务能力超限的情况下 Fail Fast。

另外，根据排队论，由于 API 接口服务具有延迟随着请求量提升迅速提升的特点，为了保证 SLA 的低延迟，需要控制单位时间的请求量。这也是 Little’s law 所说的。



架构必备：Rate limiting 的作用和常见方式

还有，公开 API 接口服务，Rate limiting 应该是一个必备的功能，否则公开的接口不知道哪一天就会被服务调用方有意无意的打垮。

所以，提供资源能够支撑的服务，将过载请求快速抛弃对整个系统架构的稳定性非常重要。这就要求在应用层实现 Rate limiting 限制。

常见的 Rate limiting 的实现方式

Proxy 层的实现，针对部分 URL 或者 API 接口进行访问频率限制

Nginx 模块

```
limit_req_zone $binary_remote_addr zone=one:10m rate=1r/s;

server {
    location /search/ {
        limit_req zone=one burst=5;
    }
```

详细参见： ngx_http_limit_req_module

Haproxy 提供的功能

详细参见： Haproxy Rate limit 模块

Java、Scala JVM 系应用层实现

Google Guava 提供了一个 RateLimiter 实现。使用方式简单明了，在自己的应用中简单封装即可，放到 HTTP 服务或者其他逻辑接口调用的前端。

```
final RateLimiter rateLimiter = RateLimiter.create(2.0); // rate is "2 permits per second"
  void submitTasks(List<Runnable> tasks, Executor executor) {
    for (Runnable task : tasks) {
      rateLimiter.acquire(); // may wait
      executor.execute(task);
    }
  }
```

详细参见： Google Guava RateLimiter

基于 Redis 功能的实现

这个在 Redis 官方文档有非常详细的实现。一般适用于所有类型的应用，比如 PHP、Python 等等。Redis 的实现方式可以支持分布式服务的访问频率的集中控制。Redis 的频率限制实现方式还适用于在应用中无法状态保存状态的场景。

参见： Redis INCR rate limiter