软件架构与中间件

涂志莹
tzy_hit@hit.edu.cn

哈尔滨工业大学

软件架构与中间件

Software Architecture and Middleware

第3章
计算层的软件架构技术

第3章 计算层的软件架构技术

3.1 计算层的软件架构技术挑战

3.2 分布式编程模型

3.3 负载均衡

3.4 消息队列

3.5 分布式服务框架

3.3 负载均衡

极端负载

2018订单峰值：49.1万笔每秒

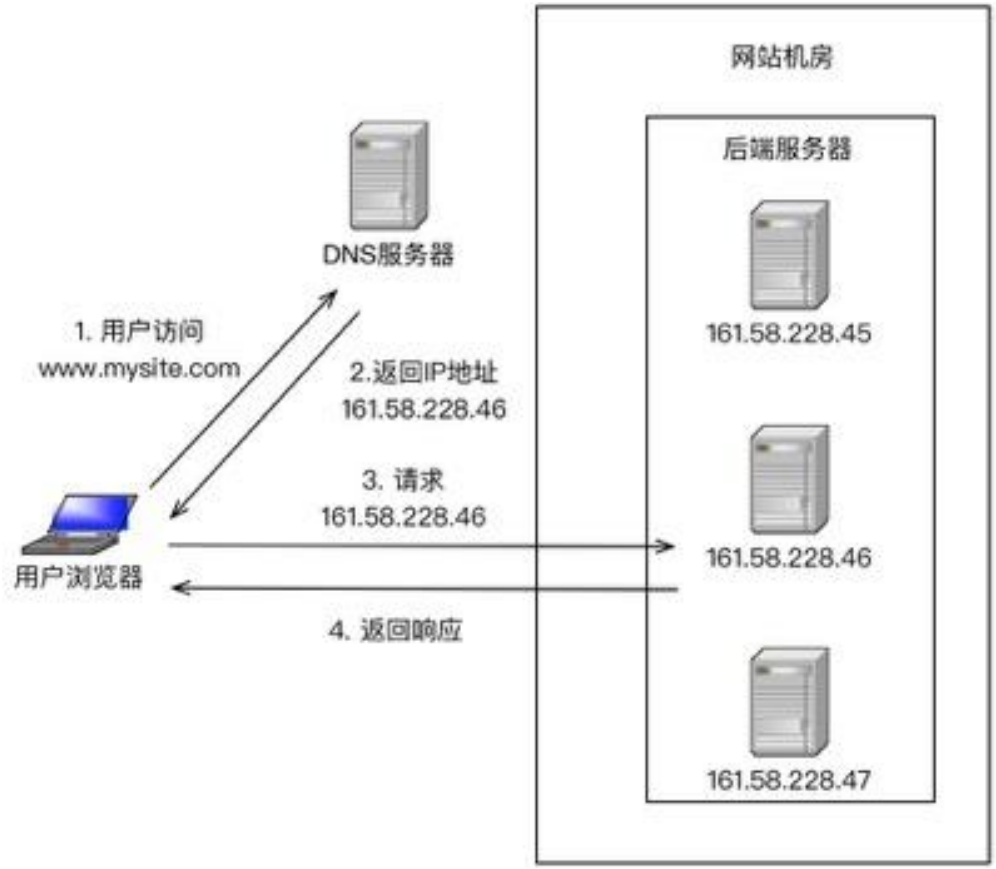
2017收发峰值：76万个每秒

负载均衡

- 单服务器的性能天花板
  - 无论如何优化，无论采用多好的硬件，总会有一个上限
  - 当无法满足业务需求时，就需要设计高性能集群来提升系统整体的处理性能。
- 高性能集群的复杂性
  - 主要体现在需要增加一个任务分配器（负载均衡器）
  - 以及为任务选择一个合适的任务分配算法
- 常见的负载均衡器包括 3 种
  - DNS 负载均衡
  - 硬件负载均衡
  - 软件负载均衡

DNS 负载均衡

- DNS 是最简单也是最常见的负载均衡方式
  - 一般用来实现地理级别的均衡。



DNS 负载均衡

- 优点：
  - 简单、成本低：交给 DNS 服务器处理，无须自己开发、维护负载均衡设备。
  - 就近访问，提升访问速度：DNS 解析时可以根据请求来源 IP，解析成距离用户最近的服务器地址，可以加快访问速度，改善性能。
- 缺点：
  - 更新不及时：DNS 缓存的时间比较长，修改 DNS 配置后，还是有很多用户会继续访问修改前的 IP，这样的访问会失败，达不到负载均衡的目的，并且也影响用户正常使用业务。
  - 扩展性差：DNS 负载均衡的控制权在域名商那里，无法根据业务特点针对其做更多的定制化功能和扩展特性。
  - 分配策略比较简单：DNS 负载均衡支持的算法少；不能区分服务器的差异（不能根据系统与服务的状态来判断负载）；也无法感知后端服务器的状态。

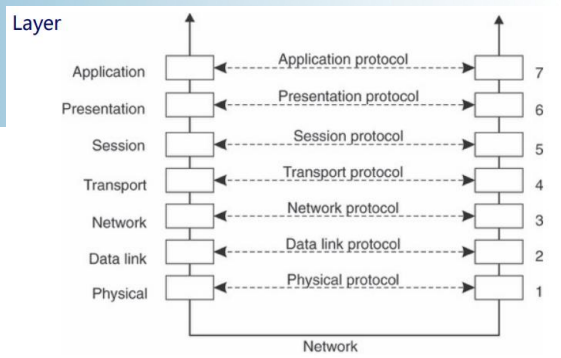
硬件负载均衡

- 硬件负载均衡通过单独的硬件设备来实现负载均衡功能
  - 这类设备和路由器、交换机类似
- 目前业界典型的硬件负载均衡设备有两款：
  - F5 和 A10
  - 性能强劲、功能强大，但价格都不便宜

硬件负载均衡

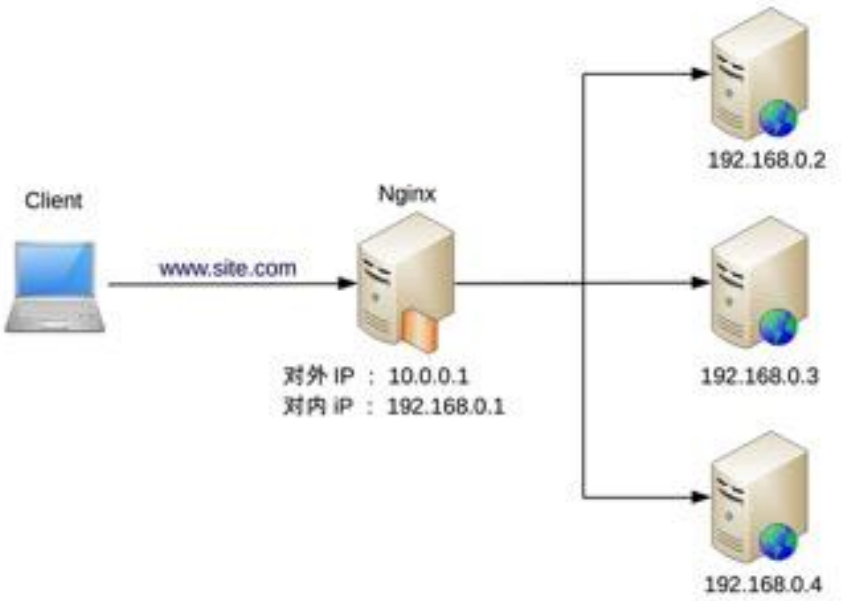
- 优点：
  - 功能强大：全面支持各层级的负载均衡，支持全面的负载均衡算法，支持全局负载均衡。
  - 性能强大：对比一下，软件负载均衡支持到 10 万级并发已经很厉害了，硬件负载均衡可以支持 100 万以上的并发。
  - 稳定性高：商用硬件负载均衡，经过了良好的严格测试，经过大规模使用，稳定性高。
  - 支持安全防护：硬件均衡设备除具备负载均衡功能外，还具备防火墙、防DDoS 攻击等安全功能。
- 缺点：
  - 价格昂贵：最普通的一台 F5 价格在20万-70万之间。
  - 扩展能力差：硬件设备，可以根据业务进行配置，但无法进行扩展和定制。

软件负载均衡



- 软件负载均衡通过负载均衡软件来实现负载均衡功能
  - 常见的有 Nginx 和 LVS
    - Nginx 是 7 层负载均衡，LVS 是 Linux 内核的 4 层负载均衡
    - 4 层和 7 层的区别就在于协议和灵活性
    - Nginx 支持 HTTP、E-mail 协议
    - 而 LVS 是 4 层负载均衡，和协议无关，几乎所有应用都可以做，例如，聊天、数据库等。
- 软件和硬件负载均衡方法的最主要区别就在于性能
  - 硬件负载均衡性能远远高于软件负载均衡性能。
    - Ngxin 性能是万级，一般的 Linux 服务器上装一个 Nginx 大概能到 5 万 / 秒；LVS 的性能是十万级，据说可达到 80 万 / 秒；
    - 而 F5 性能是百万级，从 200 万 / 秒到 800 万 / 秒都有。

软件负载均衡



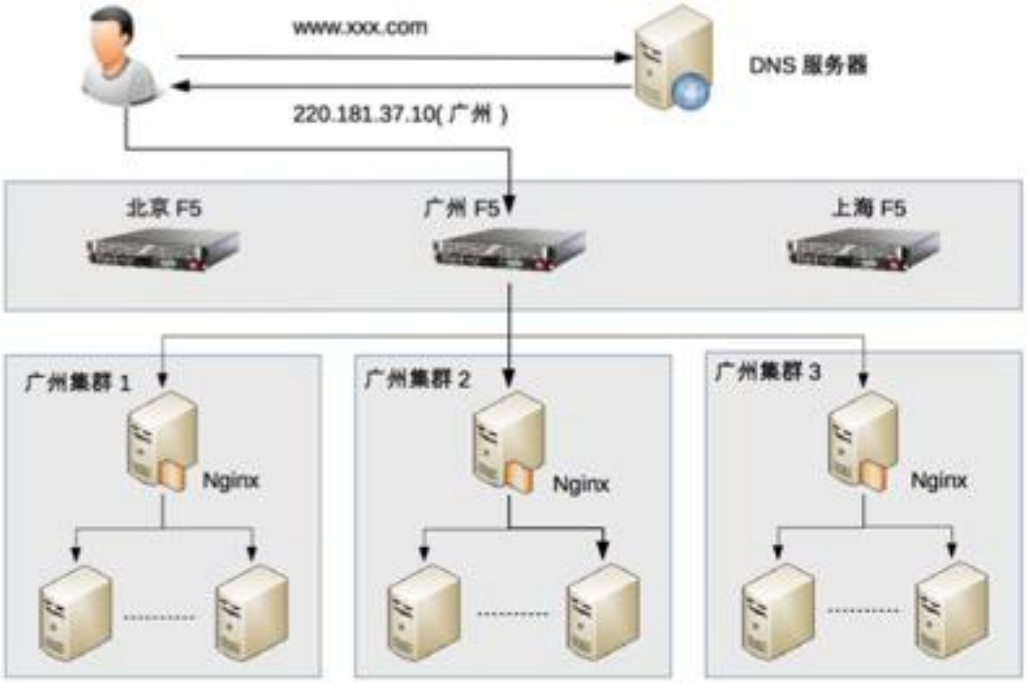
- 除了使用开源的系统进行负载均衡，如果业务比较特殊，也可能基于开源系统进行定制（例如，Nginx 插件），甚至进行自研

软件负载均衡

- 优点：
  - 简单：无论是部署还是维护都比较简单。
  - 便宜：只要买个 Linux 服务器，装上软件即可。
  - 灵活：4 层和 7 层负载均衡可以根据业务进行选择；也可以根据业务进行比较方便的扩展，例如，可以通过 Nginx 的插件来实现业务的定制化功能。
- 与硬件负载均衡相比的缺点：
  - 性能一般：一个 Nginx 大约能支撑 5 万并发。
  - 功能没有硬件负载均衡那么强大。
  - 一般不具备防火墙和防 DDoS 攻击等安全功能。

负载均衡典型架构

- 地理级别负载均衡：当用户访问时，DNS 会根据用户的地理位置来决定返回哪个机房的 IP，
- 集群级别负载均衡：F5 收到请求后，进行集群级别的负载均衡
- 机器级别的负载均衡：Nginx 收到用户请求后，将用户请求发送给集群里面的某台服务器



负载均衡算法

- 根据算法期望达到的目的，大体上可以分为下面几类
  - 任务数平分类：负载均衡系统将收到的任务平均分配给服务器进行处理，这里的“平均”可以是绝对数量的平均，也可以是比例或者权重上的平均。
  - 负载均衡类：负载均衡系统根据服务器的负载来进行分配，用连接数、I/O 使用率、网卡吞吐量等来衡量系统的压力。
  - 性能最优类：负载均衡系统根据服务器的响应时间来进行任务分配，优先将新任务分配给响应最快的服务器。
  - Hash 类：负载均衡系统根据任务中的某些关键信息进行 Hash 运算，将相同 Hash 值的请求分配到同一台服务器上。常见的有源地址 Hash、目标地址 Hash、session id hash、用户 id hash 等。

轮询(Round-Robin)

- 负载均衡系统收到请求后，按照顺序轮流分配到服务器上。
- 轮询是最简单的一个策略，无须关注服务器本身的状态，例如：
  - 某个服务器当前因为触发了程序 bug 进入了死循环导致 CPU 负载很高，负载均衡系统是不感知的，还是会继续将请求源源不断地发送给它。
  - 集群中有新的机器是 32 核的，老的机器是 16 核的，负载均衡系统也是不关注的，新老机器分配的任务数是一样的。
- “简单”是轮询算法的优点，也是它的缺点。

加权轮询

- 负载均衡系统根据服务器权重进行任务分配
  - 这里的权重一般是根据硬件配置进行静态配置的
- 加权轮询是轮询的一种特殊形式
  - 其主要目的就是为了解决不同服务器处理能力有差异的问题
  - 解决了轮询算法中无法根据服务器的配置差异进行任务分配的问题
  - 但同样存在无法根据服务器的状态差异进行任务分配的问题。

负载最低优先

- 负载均衡系统将任务分配给当前负载最低的服务器
  - LVS，可以以“连接数”来判断服务器的状态，服务器连接数越大，表明服务器压力越大
  - Nginx，可以以“HTTP 请求数”来判断服务器状态（Nginx 内置的负载均衡算法不支持这种方式，需要进行扩展）。
- 如果自己开发负载均衡系统，可以根据业务特点来选择指标衡量系统压力
  - 如果是 CPU 密集型，可以以“CPU 负载”来衡量系统压力；
  - 如果是 I/O 密集型，可以以“I/O 负载”来衡量系统压力。

负载最低优先

- 负载最低优先算法
  - 解决了轮询算法中无法感知服务器状态的问题，由此带来的代价是复杂度要增加很多
  - CPU 负载最低优先的算法要求以某种方式收集每个服务器的 CPU 负载
    - 不同业务最优的时间间隔是不一样的，时间间隔太短容易造成频繁波动，时间间隔太长又可能造成峰值来临时响应缓慢。
- 负载最低优先算法基本上能够比较完美地解决轮询算法的缺点
  - 可以感知服务器当前的运行状态
  - 其代价是复杂度大幅上升
  - 看起来很美好，但实际应用场景反而不如轮询（包括加权轮询）

性能最优类

- 性能最优优先类算法则是站在客户端的角度来进行分配的
  - 优先将任务分配给处理速度最快的服务器，通过这种方式达到最快响应客户端的目的
  - 负载最低优先类算法是站在服务器的角度来进行分配的
- 性能最优优先类算法本质上也是感知了服务器的状态
  - 只是通过响应时间这个外部标准来衡量服务器状态而已
  - 复杂度很高，主要体现在：
    - 负载均衡系统需要收集和分析每个服务器每个任务的响应时间，在大量任务处理的场景下，这种收集和统计本身也会消耗较多的性能
- 工业上使用采样，并调优采样率

Hash 类

- 负载均衡系统根据任务中的某些关键信息进行 Hash 运算
  - 将相同 Hash 值的请求分配到同一台服务器上，这样做的目的主要是为了满足特定的业务需求。
- 源地址 Hash
  - 将来源于同一个源 IP 地址的任务分配给同一个服务器进行处理，适合于存在事务、会话的业务。
  - 例如，当我们通过浏览器登录网上银行时，会生成一个会话信息，这个会话是临时的，关闭浏览器后就失效。网上银行后台无须持久化会话信息，只需要在某台服务器上临时保存这个会话就可以了，但需要保证用户在会话存在期间，每次都能访问到同一个服务器，这种业务场景就可以用源地址 Hash 来实现。
- ID Hash
  - 将某个 ID 标识的业务分配到同一个服务器中进行处理，这里的 ID 一般是临时性数据的 ID（如 session id）。例如，上述的网上银行登录的例子，用 session id hash 同样可以实现同一个会话期，用户每次都是访问到同一台服务器的目的。

第3章

计算层的软件架构技术

Thanks for listening

涂志莹

哈尔滨工业大学计算机学院

企业与服务计算研究中心