图解：从单个服务器扩展到百万用户的系统

原创：Wolfram Hempel 码农翻身 1周前

作者 | Wolfram Hempel
翻译 | Join

你开发了一个网站（例如网上商店、社交网站或者其他任何东西），之后你把它发布到了网上，网站运行良好，每天有几百的访问量，能快速地相响应用户的请求。

但是有一天，不知道什么原因，你的网站出名了！

每分每秒都有成千上万的用户蜂拥而至，你的网站变得越来越慢……

对你来讲，这是个好消息，但是对你的Web应用来说这是个坏消息。因为现在它需要扩展了，你的应用需要为全球用户提供7*24不宕机服务。

如何进行扩展？

几年前，我讨论过水平扩展与垂直扩展。简而言之， 垂直扩展意味着在性能更

强的计算机上运行同样的服务，而水平扩展是并行地运行多个服务。

如今，几乎没有人说垂直扩展了。原因很简单：

- 随着计算机性能的增长，其价格会成倍增长
- 单台计算机的性能是有上限的，不可能无限制地垂直扩展
- 多核CPU意味着即使是单台计算机也可以并行的。那么，为什么不一开始就并行化呢？

现在我们水平扩展服务。需要哪些步骤呢？

1.单台服务器 + 数据库

上图可能是你后端服务最初的样子。有一个执行业务逻辑的应用服务器（Application Server）和保存数据的数据库。

看上去很不错。但是这样的配置，满足更高要求的唯一方法是在性能更强的计算机上运行，这点不是很好。

2. 增加一个反向代理

成为大规模服务架构的第一步是添加反向代理。类似于酒店大堂的接待处。

你也可以让客人直接去他们的客房。但是实际上，你需要一个中间人他去检查是否允许客人进入， 如果客房没有开放，得有人告诉客人，而不是让客人处于尴尬的境地。这些事情正是反向代理需要做的。

通常，代理是一个接收和转发请求的过程。正常情况下，「正向代理」代理的对象是客户端，「反向代理」代理的对象是服务端，它完成这些功能：

- 健康检查功能，确保我们的服务器是一直处于运行状态的
- 路由转发功能，把请求转发到正确的服务路径上
- 认证功能,确保用户有权限访问后端服务器
- 防火墙功能，确保用户只能访问允许使用的网络部分等等

3.引入负载均衡器

大多数反向代理还有另外一个功能：他们也可以充当负载均衡器。

负载均衡器是个简单概念，想象下有一百个用户在一分钟之内在你的网店里付款。遗憾的是，你的付款服务器在一分钟内只能处理50笔付款。这怎么办呢？同时运行两个付款服务器就行了。

负载均衡器的功能就是把付款请求分发到两台付款服务器上。用户1往左，用户2往右，用户3再往左。。。以此类推。

如果一次有500个用户需要立刻付款，这该怎么解决呢？确切地说，你可以扩展到十台付款服务器，之后让负载均衡器分发请求到这十台服务器上。

4.扩展数据库

负载均衡器的使用使得我们可以在多个服务器之间分配负载。但是你发现问题了吗？尽管我们可以用成百上千台服务器处理请求，但是他们都是用同一个数据库存储和检索数据。

那么，我们不能以同样的方式来扩展数据库吗？很遗憾，这里有个一致性的问题。

系统使用的所有服务需要就他们使用的数据达成一致。数据不一致会导致各种问题，如订单被多次处理，从一个余额只有100元的账户中扣除两笔90元的付款等等.....那么我们在扩展数据库的时候如何确保一致性呢？

我们需要做的第一件事是把数据库分成多个部分。一部分专门负责接收并存储数据，其他部分负责检索数据。这个方案有时称为主从模式或者单实例写多副本读。这里假设是从数据库读的频率高于写的频率。这个方案的好处是保证了一致性，因为数据只能被单实例写入，之后把写入数据同步到其他部分即可。缺点是我们仍然只有一个写数据库实例。

这对于中小型的Web应用来说没问题， 但是像Facebook这样的则不会这样做了。我们会在第九节中研究扩展数据库的步骤。

5.微服务

到目前为止，我们的付款、订单、库存、用户管理等等这些功能都在一台服务器上。

这也不是坏事，单个服务器同时意味着更低的复杂性。随着规模的增加，事情会变得复杂和低效：

- 开发团队随着应用的发展而增长。但是随着越来越多的开发人员工作在同一台服务器上，发生冲突的可能性很大。

- 仅有一台服务器，意味着每当我们发布新版本时，必须要等所有工作完成后才能发布。当一个团队想快速地发布而另外一个团队只完成了一半工作的时候，这种互相依赖性很危险。

对于这些问题的解决方案是一个新的架构范式：微服务， 它已经在开发人员中掀起了风暴。

- 每个服务都可以单独扩展，更好地适应需求
- 开发团队之间相互独立，每个团队都负责自己的微服务生命周期(创建，部署，更新等)
- 每个微服务都有自己的资源，比如数据库，进一步缓解了第4节中的问题。

6.缓存和内容分发网络(CDN)

有什么方式能使服务更高效？网络应用的很大一部由静态资源构成,如图片、CSS样式文件、JavaScript脚本以及一些针对特定产品提前渲染好的页面等等。

我们使用缓存而不是对每个请求都重新处理，缓存用于记住最后一次的结果并交由其他服务或者客户端，这样就不用每次都请求后端服务了。

缓存的加强版叫内容分发网络（Content Delivery Network），遍布全球的大量缓存。 这使得用户可以从物理上靠近他们的地方来获取网页内容，而不是每次都把数据从源头搬到用户那里。

7.消息队列

你去过游乐园吗？你是否走到售票柜台去买票？也许不是这样，可能是排队等候。政府机构、邮局、游乐园入口都属于并行概念的例子，多个售票亭同时售票，但似乎也永远不足以为每个人立即服务，于是队列形成了。

队列同样也是用于大型Web应用。每分钟都有成千上万的图片上传到Instagram、Facebook每个图片都需要处理，调整大小，分析与打标签，这些都是耗时的处理过程。

因此，不要让用户等到完成所有步骤，图片接收服务只需要做以下三件事：

- 存储原始的、未处理的图片
- 向用户确认图片已经上传
- 创建一个待办的任务

这个待办事项列表中的任务可以被其他任意数量服务接收，每个服务完成其中一个任务，直到所有的待办事项完成。管理这些“待办事项列表”的称为消息队

列。使用这样的队列有许多优点：

- 解耦了任务和处理过程。有时需要处理大量的图片，有时很少。有时有大量服务可用，有时很少可用。简单地把任务添加到待办事项而不是直接处理它们，这确保了系统保持响应并且任务也不会丢失。

- 可以按需扩展。启动大量的服务比较耗时，所以当有大量用户上传图片时再去启动服务，这已经太晚了。我们把任务添加到队列中，我们可以推迟提供额外的处理能力。

好了，如果按照我们上面的所有步骤操作下来，我们的系统已经做好提供大流量服务的准备了。但是如果还想提供更大量的，该怎么做呢？还有一些可以做：

8.分片，分片，还是分片

什么是分片？好吧，深呼吸一下，准备好了吗？我们看下定义：

"Sharding is a technique of parallelizing an application's stacks by separating them into multiple units, each responsible for a certain key or namespace"

哎呦..... 分片究竟是是什么意思呢？

其实也很简单：Facebook上需要为20亿用户提供个人资料，可以把你的应用架构分解为 26个mini-Facebook，用户名如果以A开头，会被mini-facebook A处理， 用户名如果以B开头，会被mini-facebook B来处理……

分片不一定按字母顺序，根据业务需要，你可以基于任何数量的因素，比如位置、使用频率(特权用户被路由到好的硬件)等等。你可以根据需要以这种方式切分服务器、数据库或其他方面。

9. 对负载均衡器进行负载均衡

到目前为止，我们一直使用一个负载均衡器，即使你购买的一些功能强悍(且其价格极其昂贵)的硬件负载均衡器，但是他们可以处理的请求量也存在硬件限制。

幸运地是，我们可以有一个全球性、分散且稳定的层，用于在请求达到负载均衡器之前对请求负载均衡。最棒的地方是免费，这是域名系统或简称DNS。DNS

将域名(如arcentry.com)映射到IP，143.204.47.77。DNS允许我们为域名指定多个IP，每个IP都会解析到不同的负载均衡器。

你看，扩展Web应用确实需要考虑很多东西，感谢你和我们一起待了这么久。我希望这篇文章能给你一些有用的东西。但是如果你做任何IT领域相关的工作，你在阅读本文的时候，可能有个问题一直萦绕在你的脑海："云服务是怎样的呢？"

Cloud Computing / Serverless

但是云服务如何呢？确实，它是上面许多问题最有效的解决方案。

你无需解决这些难题。相反，这些难题留给了云厂商，他们为我们提供一个系统，可以根据需求进行扩展，而不用担心错综复杂的问题。

例如。Arcentry网站不会执行上述讨论的任何操作(除了数据库的读写分离)，而只是把这些难题留给Amazon Web Service Lambda函数处理了，用户省去了烦恼。

但是,并不是说你使用了云服务以后(如 Amazon Web Service Lambda)，所有的问题都解决了,它随之而来的是一系列挑战和权衡。请继续关注本系列的下一篇文章，
了解更多关于"the cloud for newbs and non-techies".

原文链接：
https://arcentry.com/blog/scaling-webapps-for-newbs-and-non-techies/