作者：牛浩帆  
链接：https://www.zhihu.com/question/20657269/answer/15763722  
来源：知乎  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

[只是大框架介绍，实际使用中的不容易注意的细节太多了，需要经验的积累，才能运用娴熟]

以下的架构都是在假设已经优化过linux内核的情况下进行

**初级篇：（单机模式）**

假设配置：（Dual core 2.0GHz,4GB ram,SSD）

**基础框架**：apache(PHP) + Mysql / IIS + MSSQL  
（最基础框架，处理一般访问请求）

**进阶1**：替换Apache为Nginx，并在数据库前加上cache层【**数据库的速度是最大的瓶颈**】  
Nginx(PHP) + Memcache + Mysql  
（此时已经具备处理小型访问量的能力）

**进阶2**：随着访问量的上涨，最先面临的问题就来了：CGI无法匹配上Nginx的高IO性能，这时候可以通过写扩展来替代脚本程序来提升性能，C扩展是个好办法，但是大家更喜欢用简单的脚本语言完成任务，Taobao团队开源了一个Nginx\_lua模块，可以用lua写Nginx扩展，这时候可处理的并发已经超越进阶1 一个档次了。  
Nginx(nginx\_lua or C) + Memcache + Mysql  
（此时处理个同时在线三四千人没有问题了）

**进阶3**：随着用户的增多，Mysql的写入速度成了又一大瓶颈，读取有memcache做缓存，但写入是直接面对Mysql，性能受到了很大阻碍，这时候，要在Nginx和Mysql中间加入一层写缓存，队列系统就出场了,就以RabbitMQ为例，所有写入操作全部丢到这只兔子的胃里面，然后屁股后面写个接应程序，一条条的拉出来再写入mysql。而RabbitMQ的写入效率是Mysql的N倍，此时架构的处理能力又上一阶层。  
|----write------>RabbitMQ--------  
Nginx（lua or c）----- |--------->Mysql  
|----read------>Memcache--------

(此时的并发吞吐能力已经可以处理万人左右在线)

**中级篇：（分而治之）**

此时我们在单机优化上已经算是达到极限，接下来就要集群来显示作用了。

**数据库篇：** 数据库总是在整个环节中是吞吐能力最弱的，最常见的方法就是sharding。  
sharding可以按多种方法来分，没有定式，看情况。可以按用户ID区段分，按读写分等等，可用参考软件：mysql proxy（工作原理类似lvs）

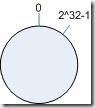
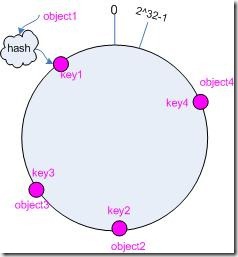
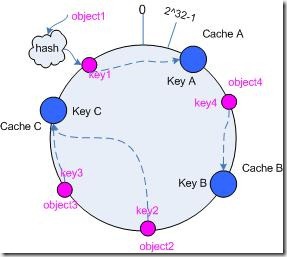
**缓存篇：**memcache一般采用的是构建memcache pool，将缓存分散到多台memcache节点上，如何将缓存数据均匀分散在各节点，一般采用将各节点顺序编号，然后hash取余对应到各个节点上去。这样可以做到比较均匀的分散，但是有一个致命点就是，如果节点数增加或减少，将会带来几乎80%的数据迁移，解决方案我们在高级篇再提。

**WEB服务器篇：** web服务器集群的建设，最常见的就是lvs方式(memcache pool同样可以如此组建)，lvs的核心就是调度节点，调度节点负责将流量通过算法分散到各个节点上，因调度所耗资源很少，所以可以产生很高的吞吐率，后台节点数量可以任意增删，但此法弊病就是如果调度节点挂了，则整个集群都挂了，解决方案我们在高级篇提。  
方法2:参见 [HAProxy - The Reliable, High Performance TCP/HTTP Load Balancer](https://link.zhihu.com/?target=http://haproxy.1wt.eu/" \t "_blank)

**高级篇：（高可用性+高可扩展性的集群）**

**单点调度故障解决：**  
集群的好处显而易见，但是有一个弊端就是单节点进行调度，如果节点出现故障，则整个集群全部都无法服务，对此的解决方案，我们使用**keepalived**来解决。[Keepalived for Linux](https://link.zhihu.com/?target=http://www.keepalived.org/" \t "_blank)  
keepalived是基于VRRP协议（[VRRP协议介绍](https://link.zhihu.com/?target=http://blog.chinaunix.net/uid-127037-id-2919520.html" \t "_blank)）的，请一定先了解VRRP协议后再进行配置。  
keepalived可以把多台设备虚拟出一个IP，并自动在故障节点与备用节点之间实现failover切换。这样我们配置两台货多台lvs调度节点，然后配置好keepalived就可以做到lvs调度节点出现故障后，自动切换到备用调度节点。（同样适用于mysql）

**memcache集群扩展解决：**  
memcache因为我们一般采用的都是hash后除以节点数取余，然后分配到对应节点上，如果节点数出现变化，以前的缓存数据将基本都不能命中。  
解决方法：consistent hashing 简介：[一致性哈希](https://link.zhihu.com/?target=http://baike.baidu.com/view/1588037.htm" \t "_blank)

consistent hashing大概的思路就是，把hash后的值保证在 0 ~ (2^32)-1 的数值上，然后把这一连串数字对应映射到一个想象的圆上。 &lt;img src="https://pic2.zhimg.com/c3cd8e6cf787c82d6f7e011e882ce27d\_b.jpg" data-rawwidth="95" data-rawheight="108" class="content\_image" width="95"&gt;  
把要存储的各个值hash后，放到圆上，如图  
&lt;img src="https://pic2.zhimg.com/77a5474913b2017a9c64e877940ca625\_b.jpg" data-rawwidth="238" data-rawheight="257" class="content\_image" width="238"&gt;  
然后把cache节点也用同样的hash方法，映射到圆上，然后每个刚才hash过的value顺时针寻找离自己最近的节点，这个节点就是存储它的节点。  
&lt;img src="https://pic2.zhimg.com/cf3a58c791488b89d6bda4f6c8c43d55\_b.jpg" data-rawwidth="287" data-rawheight="257" class="content\_image" width="287"&gt;

为了提高存储的平衡性，算法还可以加入虚拟节点的概念，即每个实际cache节点，会在圆上对应N个虚拟的节点，这样可以提高算法的命中率，更加平衡。

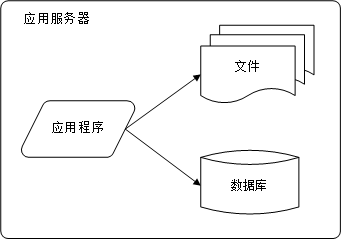
consistent hashing原理：[Consistent hashing and random trees](https://link.zhihu.com/?target=http://dl.acm.org/citation.cfm?id=258660" \t "_blank)

作者：xlzd  
链接：https://www.zhihu.com/question/20657269/answer/101795180  
来源：知乎  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

**非原创，整理自： [大型网站技术架构 (豆瓣)](https://link.zhihu.com/?target=https://book.douban.com/subject/25723064/" \t "_blank)**

**1. 初始阶段的网站架构**

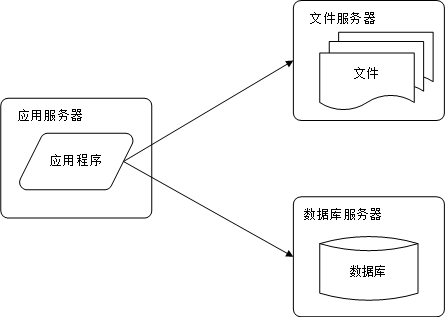
一般来讲，大型网站都是从小型网站发展而来，一开始的架构都比较简单，随着业务复杂和用户量的激增，才开始做很多架构上的改进。当它还是小型网站的时候，没有太多访客，一般来讲只需要一台服务器就够了，这时应用程序、数据库、文件等所有资源都在一台服务器上，网站架构如下图所示：

<img src="https://pic2.zhimg.com/b1fb995ac418a4bb50bdb976d52a6c31\_b.png" data-rawwidth="341" data-rawheight="239" class="content\_image" width="341">  
  
**2. 应用服务和数据服务分离**

随着网站业务的发展和用户量的增加，一台服务器就无法再满足需求了。大量用户访问导致访问速度越来越慢，而逐渐增加的数据也会导致存储空间不足。这时就需要将应用和数据分离，应用和数据分离后整个网站使用 3 台服务器：应用服务器、文件服务器和数据库服务器。这 3 台服务器对硬件资源的要求各不相同：

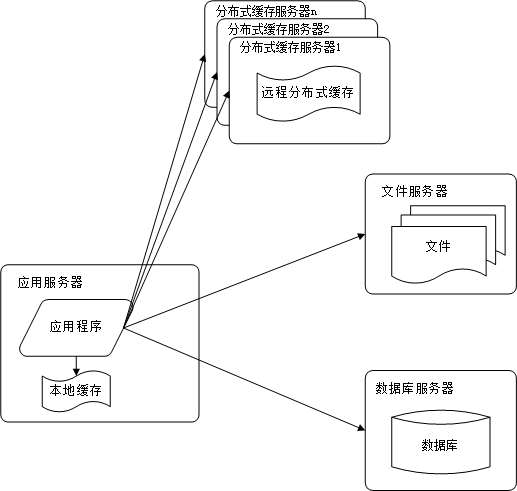
* 应用服务器业务逻辑，需要强大的CPU
* 数据库服务器对磁盘读写操作很多，需要更快的磁盘和更大的内存
* 文件服务器存储用户上传的文件，因此需要更大的磁盘空间

此时，网站系统的架构如下图所示：

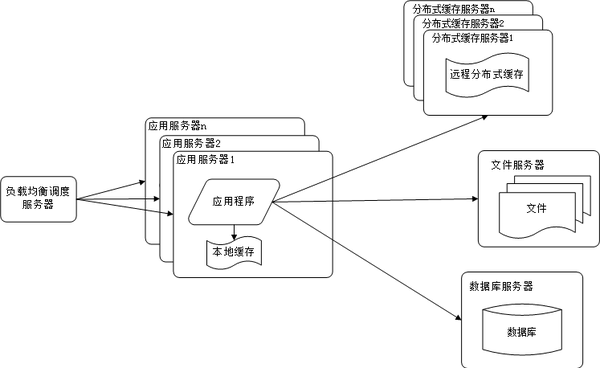
<img src="https://pic1.zhimg.com/f6d05e8fafd86940614503fdb53dd608\_b.png" data-rawwidth="445" data-rawheight="317" class="origin\_image zh-lightbox-thumb" width="445" data-original="https://pic1.zhimg.com/f6d05e8fafd86940614503fdb53dd608\_r.png">  
  
**3. 使用缓存改善网站性能**

随着用户再增加，网站又会一次面临挑战：数据库压力太大导致整站访问效率再此下降，用户体验受到影响。一个网站，往往 80% 的业务访问集中在 20% 的数据上，比如微博请求量最多的肯定是那些千万级粉丝的大 V 的微博，而几乎没有人关注的你的首页，除了自己想起来之外根本不会被打开。既然大部分业务访问集中在一小部分数据上，那就把这一小部分数据先提前缓存在内存中，而不是每次都去数据库读取，这样就可以减少数据库的访问压力，从而提高整个网站的访问速度。

网站使用的缓存一般分为缓存到应用服务器或者缓存在专门的分布式缓存服务器。缓存到应用服务器自己的访问速度快很多，但是受自身内存限制，往往不太适用。远程分布式缓存使用一个集群专门负责缓存服务，当内存不够还可以轻松得动态扩容。

<img src="https://pic4.zhimg.com/3f7742ea34208549d12c8cbf9fa85bab\_b.png" data-rawwidth="517" data-rawheight="491" class="origin\_image zh-lightbox-thumb" width="517" data-original="https://pic4.zhimg.com/3f7742ea34208549d12c8cbf9fa85bab\_r.png">  
  
  
**4. 使用应用服务器集群改善网站的并发处理能力**

使用缓存后，数据访问压力得到了缓解，但是单一应用服务器能够处理的请求连接有限，在网站访问高峰期，应用服务器就成了整个网站的效率瓶颈。使用分布式集群是网站解决高并发、海量数据问题的常用手段。当一台服务器的处理能力和存储空间不足时，不要尝试去更换更强大的服务器，对大型网站而言，多么强大的服务器，都满足不了网站持续增长的业务需求。这种情况下，更恰当的做法是增加一台服务器分担原有服务器的访问及存储压力。 对网站架构而言，只要能通过增加一台服务器的方式改善负载压力，就可以以同样的方式持续增加服务器不断改善系统性能，从而实现系统的可伸缩性。应用服务器实现集群是网站可伸缩架构设计中较为简单成熟的一种，如下图所示：

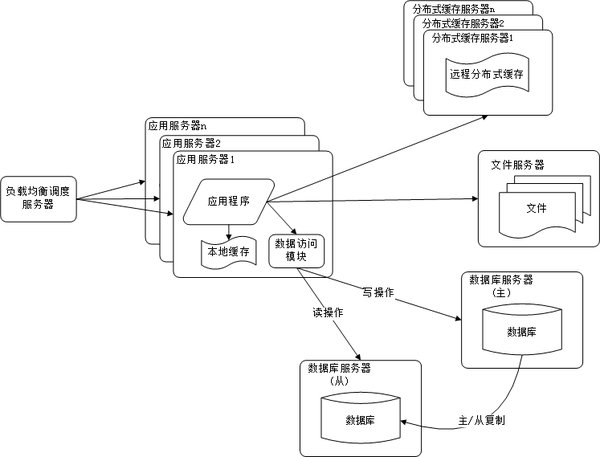
<img src="https://pic3.zhimg.com/758b0d2283f0c8fe32059ef718aa9a1e\_b.png" data-rawwidth="747" data-rawheight="459" class="origin\_image zh-lightbox-thumb" width="747" data-original="https://pic3.zhimg.com/758b0d2283f0c8fe32059ef718aa9a1e\_r.png">

通过负载均衡调度服务器，可以将来自用户浏览器的访问请求分发到应用服务器集群中的任何一台服务器上，如果有更多用户，就在集群中加入更多的应用服务器，使应用服务器的压力不再成为整个网站的瓶颈。

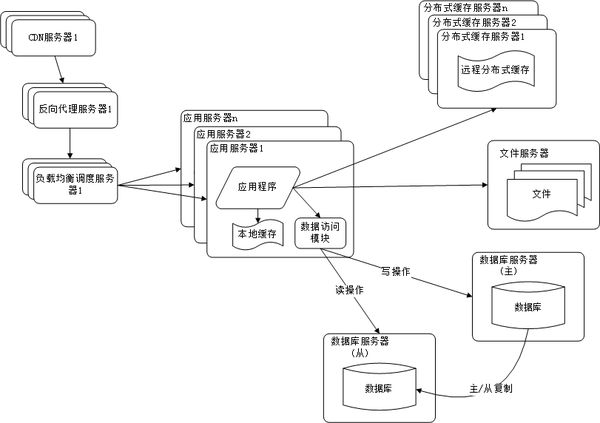
**5. 数据库读写分离**

网站在使用缓存后，使对大部分数据读操作访问都可以不通过数据库就能完成，但是仍有一部分读操作（缓存访问不命中、缓存过期）和全部的写操作都需要访问数据库，在网站的用户达到一定规模后，数据库因为负载压力过高而成为网站的瓶颈。 目前大部分的主流数据库都提供主从热备功能，通过配置两台数据库主从关系，可以将一台数据库服务器的数据更新同步到另一台服务器上。网站利用数据库的这一功能，实现数据库读写分离，从而改善数据库负载压力。如下图所示：

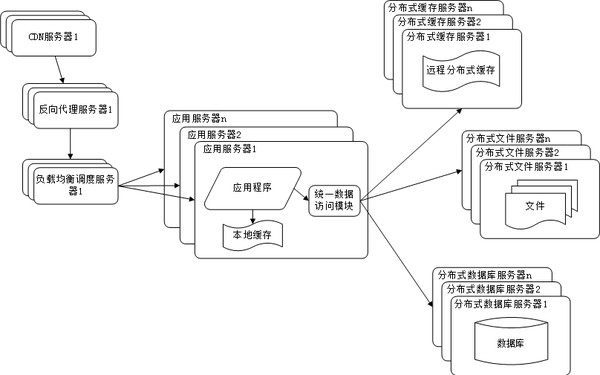
应用服务器在写数据的时候，访问主数据库，主数据库通过主从复制机制将数据更新同步到从数据库，这样当应用服务器读数据的时候，就可以通过从数据库获得数据。为了便于应用程序访问读写分离后的数据库，通常在应用服务器端使用专门的数据访问模块，使数据库读写分离对应用透明。

<img src="https://pic4.zhimg.com/3611799b5b84326062b86f7e8383b5e3\_b.png" data-rawwidth="747" data-rawheight="569" class="origin\_image zh-lightbox-thumb" width="747" data-original="https://pic4.zhimg.com/3611799b5b84326062b86f7e8383b5e3\_r.png">  
  
**6. 使用反向代理和 CDN 加速网站响应**

随着网站业务不断发展，用户规模越来越大，由于中国复杂的网络环境，不同地区的用户访问网站时，速度差别也极大。有研究表明，网站访问延迟和用户流失率正相关，网站访问越慢，用户越容易失去耐心而离开。为了提供更好的用户体验，留住用户，网站需要加速网站访问速度。主要手段有使用 CDN 和反向代理。如下图所示：

<img src="https://pic4.zhimg.com/cf613de59351dc6209fc23c722351adf\_b.png" data-rawwidth="811" data-rawheight="572" class="origin\_image zh-lightbox-thumb" width="811" data-original="https://pic4.zhimg.com/cf613de59351dc6209fc23c722351adf\_r.png">  
 **7. 使用分布式文件系统和分布式数据库系统**

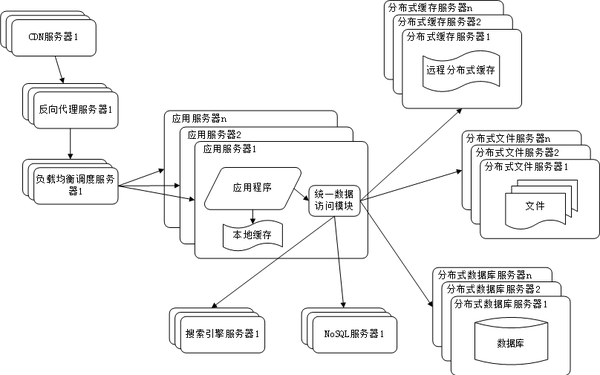
任何强大的单一服务器都满足不了大型网站持续增长的业务需求。数据库经过读写分离后，从一台服务器拆分成两台服务器，但是随着网站业务的发展依然不能满足需求，这时需要使用分布式数据库。文件系统也一样，需要使用分布式文件系统。如下图所示：

<img src="https://pic2.zhimg.com/b41911410fd25fe17e53b8d99ee2808d\_b.png" data-rawwidth="808" data-rawheight="506" class="origin\_image zh-lightbox-thumb" width="808" data-original="https://pic2.zhimg.com/b41911410fd25fe17e53b8d99ee2808d\_r.png">

分布式数据库是网站数据库拆分的最后手段，只有在单表数据规模非常庞大的时候才使用。不到不得已时，网站更常用的数据库拆分手段是业务分库，将不同业务的数据部署在不同的物理服务器上。

**8. 使用 NoSQL 和搜索引擎**

随着网站业务越来越复杂，对数据存储和检索的需求也越来越复杂，网站需要采用一些非关系数据库技术如 NoSQL 和非数据库查询技术如搜索引擎。如下图所示：

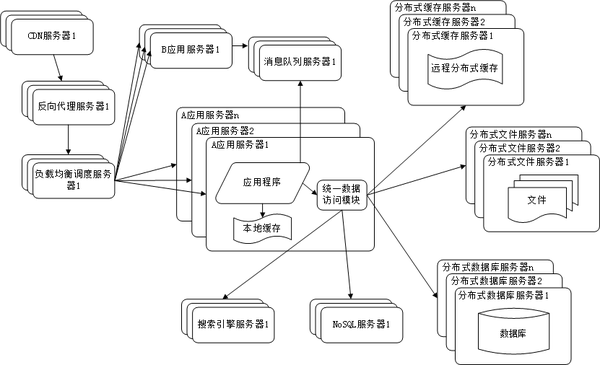
<img src="https://pic3.zhimg.com/25f90e03351f541671aea86a488ae322\_b.png" data-rawwidth="808" data-rawheight="506" class="origin\_image zh-lightbox-thumb" width="808" data-original="https://pic3.zhimg.com/25f90e03351f541671aea86a488ae322\_r.png">

NoSQL 和搜索引擎都是源自互联网的技术手段，对可伸缩的分布式特性具有更好的支持。应用服务器则通过一个统一数据访问模块访问各种数据，减轻应用程序管理诸多数据源的麻烦。

**9. 业务拆分**

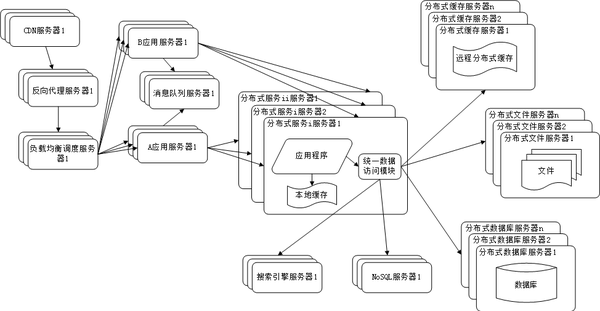
大型网站为了应对日益复杂的业务场景，通过使用分而治之的手段将整个网站业务分成不同的产品线。如大型购物交易网站都会将首页、商铺、订单、买家、卖家等拆分成不同的产品线，分归不同的业务团队负责。

具体到技术上，也会根据产品线划分，将一个网站拆分成许多不同的应用，每个应用独立部署。应用之间可以通过一个超链接建立关系（在首页上的导航链接每个都指向不同的应用地址），也可以通过消息队列进行数据分发，当然最多的还是通过访问同一个数据存储系统来构成一个关联的完整系统，如下图所示：

<img src="https://pic2.zhimg.com/cc6bd78d9d214d5e0fe6283b5ac2a3e5\_b.png" data-rawwidth="831" data-rawheight="506" class="origin\_image zh-lightbox-thumb" width="831" data-original="https://pic2.zhimg.com/cc6bd78d9d214d5e0fe6283b5ac2a3e5\_r.png">  
  
**10. 分布式服务**

随着业务拆分越来越小，存储系统越来越庞大，应用系统的整体复杂度呈指数级增加，部署维护越来越困难。由于所有应用要和所有数据库系统连接，在数万台服务器规模的网站中，这些连接的数目是服务器规模的平方，导致数据库连接资源不足，拒绝服务。

既然每一个应用系统都需要执行许多相同的业务操作，比如用户管理、商品管理等，那么可以将这些共用的业务提取出来，独立部署。由这些可复用的业务连接数据库，提供共用业务服务，而应用系统只需要管理用户界面，通过分布式服务调用共用业务服务完成具体业务操作。如下图所示：

<img src="https://pic1.zhimg.com/694fb3f64b999a481b9b1166e6c09600\_b.png" data-rawwidth="974" data-rawheight="506" class="origin\_image zh-lightbox-thumb" width="974" data-original="https://pic1.zhimg.com/694fb3f64b999a481b9b1166e6c09600\_r.png">

大型网站的架构演化到这里，基本上大多数的技术问题都可以得以解决了。