构建一个可伸缩,高性能,高可用的分布式互联网应用

1. 应用无状态(淘宝session框架)

对于服务器程序来说，有个基本假设，即服务器是基于状态请求，还是基于无状态请求。根据这个假设，可以将服务器划分为状态服务器和无状态服务器

状态服务器

如果是状态化请求，那么服务端一般需要保存请求的相关信息，**每个请求可以默认地使用以前的请求信息**。

状态服务器具有以下特点：

- 保存客户请求的数据（状态）
- 服务端容易对客户状态进行管理
- 服务端并不要求每次客户请求都携带额外的状态数据

无状态服务器

无状态服务器处理的客户信息必须全部来自于**请求所携带的信息以及其他服务器自身所保存的、并且可以被所有请求所使用的公共信息**。

无状态服务器具有以下特点：

- 并**不保存客户请求的数据**（状态）
- 客户在请求时**需要携带额外的状态数据**
- 无状态服务器更加健壮，重启服务器不会丢失状态信息，这使得维护和扩容更加简单

无状态的服务器程序，最著名的就是WEB服务器。

每次HTTP请求和以前请求没有直接关联。

为了跟踪客户请求的状态信息，请求中加入COOKIE。

COOKIE的存在，是无状态化向状态化过渡的一种手段。

应用的状态如何管理：

集群:实现了负载均衡,有失效恢复failover

应用的无状态性的重要性：

集群中的状态恢复也有其缺点，那就是严重影响了系统的伸缩性，系统不能通过增加更多的机器来达到良好的水平伸缩，因为集群节点间session的 通信会随着节点的增多而开销增大，因此要想做到应用本身的伸缩性，我们需要保证应用的无状态性，这样集群中的各个节点来说都是相同的，从而使得系统更好的水平伸缩

实现无状态性：

将状态保存到了cookie里面，这样就使得应用节点本身不需要保存任何状态信息，这样在系统用户变多的时候，就可以通过增加更多的应用节点来达到水平扩展的目的.但是采用客户端cookie的方式来保存状态也会遇到限制，比如每个cookie一般不能超过4K的大小，同时很多浏览器都限制一个站点最多保存20个cookie.淘 宝cookie框架采用的是“多值cookie”， 就是一个组合键对应多个cookie的 值，这样不仅可以防止cookie数量超过20， 同时还节省了cookie存储有效信息的空间，因为默认每个cookie都会有大约50个字节的元信息来描述cookie。

其实集中式session管理也可以完成,说具体点就是多个无状态的应用节点连接一个session服务器,session服务器将session保存到缓存中,session服务器后端再配有底层持久性数据源,比如数据库,文件系统等等。

2.有效使用缓存(Tair)

浏览器缓存，反向代理缓存，页面缓存，局部页面缓存，对象缓存等等都是缓存应用的场 景

缓存的分类：

缓存根据与应用程序的远近程度不同可以分为：**local cache** 和 **remote cache**。 一般系统中**要么采用local cache，要么采用remote cache**,两者混合使用的话对 于local cache和remote cache的数据一致性处理会变 大比较麻烦.

大部分情况下,我们所说到的缓存都是**读缓存**,缓存还有另外一个类型:**写缓存**.

在什么情况下使用缓存:

对 于一些读写比不高，同时对数据安全性需求不高的数据，我们可以将其缓存起来从而减少对底层数据库的访问,比如 统计商品的访问次数,统 计API的 调用量等等,可 以采用先写内存缓存然后延迟持久化到数据库，这样可以大大减少对数据库的写压力。

具体举例:

以店铺线的系统为例，在用户浏览店铺的时候，比如店铺介绍，店铺交流区页面，店铺服务条款页面，店铺试衣间页面，以及店铺内搜索界面这些界面更新不是非 常频繁，因此适合放到缓存中，这样可以大大减低DB的负载。另外宝贝详情页面相对也更新比较 少，因此也适合放到缓存中来减低DB负载。

3.应用拆分(HSF)

一个运用在从小变到大的过程中,一步一步实现拆分.

在什么情况下需要去拆分:

随着系统用户的不断增加，系统的访问压力越来越多，同时随着系统发展，为了满足用户 的需求，原有的系统需要增加新的功能进来，系统变得越来越复杂的时候，我们会发现系统变得越来越难维护，难扩展，同时系统伸缩性和可用性也会受到影响.

如何拆分:

将原来的系统根据一定的标准，比如业务相关性等分为不同的子系统， 不同的系统负责不同的功能

对单独的子系统进行扩展和维护，从而提高系统的扩展性和可维护性，同时系统的水平伸缩性scale out大 大的提升了

因为可以有针对性的对压力大的子系统进行水平扩展而不会影响到其它的子系统，而不会像拆分以前，每次系统压力变大的时候，我们都需要对整 个大系统进行伸缩，而这样的成本是比较大的

另外经过切分，子系统与子系统之间的耦合减低了，当某个子系统暂时不可用的时候，整体系统还是可用的，从而整 体系统的可用性也大大增强了。

因此一个大型的互联网应用，肯定是要经过拆分，因为只有拆分了，系统的扩展性，维护性,伸缩性，可用性才会变的更好

如何拆分系统:

垂直拆分和水平拆分

水平方向上，按照功能分为交易，评价，用户，商品等系统

垂直方向上，划分为业务系统，核心业务 系统以及以及基础服务，这样以来，各个系统都可以独立维护和独立的进行水平伸缩，比如交易系统可以在不影响其它系统的情况下独立的进行水平伸缩以及功能扩 展。

拆分带来的问题:

子系统通信问题,最值得关注的问题是**系统之间的依赖关系**,因为系统多了,系统的依赖关系就变得复杂,此时就需要更好的去关注拆分标准

能否将一些**有依赖的系统进行垂直化**，使得这些系统的功能尽量的垂直，

同时**一定要注意系统之间的循环依赖**，如果出现循环依赖一定要小心，因为这可能导致系统连锁启动失败

4.数据库的拆分

互联网应用除了应用级别的拆分以外，还有另外一个很重要的层面就是存储如何拆分的。因此这个主题主要涉及到如何对存储系统，通常就是所说的RDBMS进 行拆分

拆分的前提:

用户多了,数据 库读取压力太大了 ==>>> 该**读写分离**了,配置一个server为master节点，然后配几个salve节点，这样以来通过读写分离，使得读取数据的压力分摊到了不同的salve节点上面;

但是一段时间之后,master负载过高了,这时候就需要我们进行**垂直分区**(就是**分库**);比如将商品信息，用户信息，交易信息**分别存储到不同的数据库中**，同时还可以针对商品信息的库采用master，salve模式

各个按照功能拆分的数据库写压力被分担到了不同的server上面

但是还没完,随着用户的增加,某些表中的数据会变的异常庞大,这时候无论是读取数据还是写入数据,对数据库都是一个耗费精力的事情,因此此时就需要**水平分区---就是分表,或者说sharding;**

总结:数据库是系统中最不容易scale out的一层,

一个大型的互联网 应用必然会经过一个从单一DB server,到Master/salve,再到垂直分区（分 库），然后再到水平分区（分表，sharding）的过程，而在这个过程中，Master/salve 以 及垂直分区相对比较容易，对应用的影响也不是很大

但是分表会引起一些棘手的问题，比如不能跨越多个分区join查 询数据，如何平衡各个shards的 负载等等，这个时候就需要一个通用的DAL框架来屏蔽底层数据存储对应用逻辑的影响，使得底层数据的访问对应用透明化。

淘宝根据自 己的业务特点也开发了自己的TDDL框架，此框架主要解决了分库分表对应用的透明化以及异构数据库之间的数据复制。

5. 异步通信(notify)

异步通信,"**消息中间件**",采用异步通信这其实也是**关系到系统的伸缩性**,以及最大化对各个子系统进行解耦;

使用异步通信的场合:

说 到异步通信，我们需要关注的一点是这里的**异步一定是根据业务特点来的**，一定是针对业务的异步，通常**适合异步的场合是一些松耦合的通信场合**，

对于本身业务上**关联度比较大的业务系统之间，我们还是要采用同步通信比较靠谱**

异步通信的好处:

首先:同步通信在提高系统整体的伸缩性时,必须同时对子系统进行伸缩,这就影响了整个系统进行scale out,

其次:同步调用还会影响到可用性,因为在同步通信时,两者要同时可用才能保证系统的流畅,

再次:异步通信可以大大提高系统的响应时间,使得每个请求的响应时间变短,从而提高用户体验,

因此:异步在提高系统的伸缩性以及可用性的同时,也大大的增强了请求的响应时间(请求的总体时间也许下不会变少)

异步通信的具体使用:

我们就以淘宝的业务来看看异步在淘宝的具体应用。**交易系统**会与很多其它的业务系统交互

如果在一次交易过程中采用同步调用的话，这就要求要想交易成功，必须依赖的所有系统都可用，

而如果采用异步通信以后，交易系 统借助于消息中间件Notify和 其它的系统进行了解耦，这样以来当其它的系统不可用的时候，也不会影响到某此交易，从而提高了系统的可用性。

6. 非结构化数据存储 (TFS,NOSQL)

在一个大型的互联网应用当中，我们会发现并不是所有的数据都是结构化的.

哪些信息和文件不需要保存在RDBMS中:

配置文件,一个用户的动态,一次交易的快照等信息，这些信息一般不 适合保存到RDBMS中， 它们**更符合一种Key-value的 结构**

数据量非常的大，但是实时性要求不高，此时这些数据也需要通过另外的一种存储方式进行存储，

另外一**些静态文件**，比如各个商品的图 片，商品描述等信息，这些信息因为比较大，**放入RDBMS会引起读取性能问题**，从而影响到其它 的数据读取性能，因此这些信息也需要和其它信息分开存储，而一般的互联网应用系统**都会选择把这些信息保存到分布式文件系统中**

分析非结构化数据存储(NOSQL)的产生原因：

根据**CAP**理论，**一致性**，**可用性**和**分区容错性** 3者 不能同时满足，**最多只能同时满足两个**

传统的关系型数据采用了ACID的事务策略,而ACID事务策略更加讲究的是一种**高一致性而降低了可用性的需求**;

但是互联网应用往往对可用性的要求要略高于一致性的需求，这个时候我们就需要避免采用数据的ACID事务策略，转而采用BASE事 务策略，BASE事 务策略是**基本可用性**，**事务软状态**以及**最终一致性**的缩写

通过BASE事务策略，我们可以通过最终一致性来提 升系统的可用性，这也是目前很多NOSQL产品所采用的策略,其非常适合一些非结构化的数据，比如key-value形 式的数据存储，**并且这些产品有个很好的优点就是水平伸缩性**。

7.监控、预警系统

真理：

对于大型网站来说,唯一可靠的就是系统的各个部分是不可靠的.

监控的重要性：

因为一个大型的分布式系统中势必会涉及到各种各样的设备，数量越来越大,出现错误的概率也会变大，因此我们需要时时刻刻监控系统的状态，而监控也有粒度的粗细之分.

粒度粗一点的话，我们需要对整个应用系统进行监控.

比如目前的系统网络流量是多少，内存利用率是多少，IO，CPU的 负载是多少，服务的访问压力是多少，服务的响应时间是多少等这一系列的监控

细粒度一点的话，我们就需对比如应用中的某个功能

某个URL的 访问量是多，每个页面的PV是 多少，页面每天占用的带宽是多少，页面渲染时间是多少，静态资源比如图片每天占用的带宽是多少等等进行进一步细粒度的监控。

因此一个监控系统就变得必不可 少了。

有了监控系统以后，更重要的是要和预警系统结合起来

比如当某个页面访问量增多的时候，系统能自动预警，

某台Server的CPU和 内存占用率突然变大的时候，系统也能自动预警，

当并发请求丢失严重的时候，系统也能自动预警等等，

这样一来通过监控系统和预警系统的结合可以使得我们能快 速响应系统出现的问题，提高系统的稳定性和可用性。

8.配置统一管理

一个大型的分布式应用，一般都是有很多节点构成的

如果每次一个新的节点加入都要更改其它节点的配置，或者每次删除一个节点也要更改配置的话，这样不仅不利于系统的维护 和管理，同时也更加容易引入错误。

另外很多时候集群中的很多系统的配置都是一样的，如果不进行统一的配置管理，就需要在所有的系统上维护一份配置，这样会造成配置的管理维护很麻烦

而通过一个统一的配置管理可以使得这些问题得到很好的解决，当有新的节点加入或者删除的时候，配置管理系统可以通知各个节点更新配置，从而达到所有节点的配置一致性，这样既方便也不会出错。