**淘宝网采用什么技术架构来实现网站高负载的**

时间:2010-09-15

 时间过得很快，来淘宝已经两个月了，在这两个月的时间里，自己也感受颇深。下面就结合淘宝目前的一些底层技术框架以及自己的一些感触来说说如何构建一个可 伸缩，高性能，高可用性的分布式互联网应用。

**一 应用无状态（淘宝session框架）**

       俗话说，一个系 统的伸缩性的好坏取决于应用的状态如何管理。为什么这么说呢？咱们试想一下，假如我们在session中保存了大量与客户端的状态信 息的话，那么当保存状态信息的server宕机的时候，我们怎么办？通常来说，我们都是通过集群来解决这个问题，而通常 所说的集群，不仅有负载均衡，更重要的是要有失效恢复failover,比如tomcat采 用的集群节点广播复制，jboss采 用的配对复制等session状 态复制策略，但是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，从而是的系统更好的 水平伸缩。

        OK， 上面说了无状态的重要性，那么具体如何实现无状态呢？此时一个session框架就会发挥作用了。幸运的是淘 宝已经具有了此类框架。淘宝的session框架采用的是client cookie实现，主要将状态 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。

        除 了淘宝目前的session框 架的实现方式以外，其实集中式session管理来完成，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。

**二 有效使用缓存（Tair）**

        做 互联网应用的兄弟应该都清楚，缓存对于一个互联网应用是多么的重要，从浏览器缓存，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。

        一 般来说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.

        在 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.  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。

        OK， 我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。

**三 应用拆分（HSF）**

首 先，在说明应用拆分之前，我们先来回顾一下一个系统从小变大的过程中遇到的一些问题，通过这些问题我们会发现拆分对于构建一个大型系统是如何的重要。

系 统刚上线初期，用户数并不多，所有的逻辑也许都是放在一个系统中的，所有逻辑跑到一个进程或者一个应用当中，这个时候因为比较用户少，系统访问量低，因此 将全部的逻辑都放在一个应用未尝不可。但是，兄弟们都清楚，好景不长，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。那 么这个时候我们如何解决这些问题呢？明智的办法就是拆分（这也算是一种解耦），我们需要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，这样切分以后，我们可以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，同时我们系统的水平伸缩性scale out大 大的提升了，因为我们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。但是拆分也给系 统带来了问题，就是子系统之间如何通信的问题，而具体的通信方式有哪些呢？一般有同步通信和异步通信，这里我们首先来说下同步通信，下面的主题“消息系 统”会说到异步通信。既然需要通信，这个时候一个高性能的远程调用框架就显得非常总要啦，因此咱们淘宝也有了自己的HSF框 架。

上 面所说的都是拆分的好处，但是拆分以后必然的也会带来新的问题，除了刚才说的子系统通信问题外，最值得关注的问题就是系统之间的依赖关系，因为系统多了， 系统的依赖关系就会变得复杂，此时就需要更好的去关注拆分标准，比如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，这也是目前淘宝正 在做的系统垂直化，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。

OK， 既然明白了拆分的重要性，我们看看随着淘宝的发展，淘宝本身是如何拆分系统的。

首 先我们来看以下这个图：

从 上面的图可以看出淘宝系统的一个演变过程，在这个演变的过程中，我们所说的拆分就出现V2.2和V3.0之 间。在V2.2版 本中，淘宝几乎所有的逻辑都放在(Denali)系统中，这样导致的问题就是系统扩展和修改非常麻烦，并且更加致命的是随 着淘宝业务量的增加，如果按照V2.2的架构已经没有办法支撑以后淘宝的快速发展，因此大家决定对整个系统进行拆分，最 终V3.0版 本的淘宝系统架构图如下：

从 上图可以看出V3.0版 本的系统对整个系统进行了水平和垂直两个方向的拆分，水平方向上，按照功能分为交易，评价，用户，商品等系统，同样垂直方向上，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。

 从上面可以看出，一个大型系统要想变得可维 护，可扩展，可伸缩，我们必须的对它进行拆分，拆分必然也带来系统之间如何通信以及系统之间依赖管理等问题，关于通信方面，淘宝目前独立开发了自己的高性 能服务框架HSF， 此框架主要解决了淘宝目前所有子系统之间的同步和异步通信（目前HSF主要用于同步场合，FutureTask方 式的调用场景还比较少）。至于系统间的依赖管理，目前淘宝还做的不够好，这应该也是我们以后努力解决的问题。

**四 数据库拆分（TDDL）**

        在 前面“应用拆分”主题中，我们提到了一个大型互联网应用需要进行良好的拆分，而那里我们仅仅说了”应用级别”的拆 分，其实我们的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。

好 了，确定了这个小节的主题之后，我们回顾一下，一个互联网应用从小变大的过程中遇到的一些问题，通过遇到的问题来引出我们拆分RDBMS的 重要性。

系 统刚开始的时候，因为系统刚上线，用户不多，那个时候，所有的数据都放在了同一个数据库中，这个时候因为用户少压力小，一个数据库完全可以应付的了，但是 随着运营那些哥们辛苦的呐喊和拼命的推广以后，突然有一天发现，oh,god,用户数量突然变多了起来，随之而 来的就是数据库这哥们受不了，它终于在某一天大家都和惬意的时候挂掉啦。此时，咱们搞技术的哥们，就去看看究竟是啥原因，我们查了查以后，发现原来是数据 库读取压力太大了，此时咱们都清楚是到了读写分离的时候，这个时候我们会配置一个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，系统终于又恢复了正常，开 始正常运行了。但是好景还是不长，有一天我们发现master这哥们撑不住了，它负载老高了，汗 流浃背，随时都有翘掉的风险，这个时候就需要咱们垂直分区啦（也就是所谓的分库），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，OK， 通过分库以后，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，这样数据库的压力终于有恢复 到正常状态。但是是不是这样，我们就可以高枕无忧了呢？NO,这个NO， 不是我说的，是前辈们通过经验总结出来的，随着用户量的不断增加，你会发现系统中的某些表会变的异常庞大，比如好友关系表，店铺的参数配置表等，这个时候 无论是写入还是读取这些表的数据，对数据库来说都是一个很耗费精力的事情，因此此时就需要我们进行“水平分区”了（这就是俗话说的分表，或者说sharding）.

OK,上 面说了一大堆，无非就是告诉大家一个事实“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。

拿 淘宝目前的情况来说，淘宝目前也正在从昂贵的高端存储（小型机+ORACLE）切换到MYSQL,切 换到MYSQL以 后，势必会遇到垂直分区（分库）以及水平分区（Sharding）的问题，因此目前淘宝根据自 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

**五 异步通信（Notify）**

在”远 程调用框架”的 介绍中,我 们说了一个大型的系统为了扩展性和伸缩性方面的需求,肯定是要进行拆分,但是 拆分了以后,子 系统之间如何通信就成了我们首要的问题,在”远程调用框架”小节 中,我 们说了同步通信在一个大型分布式系统中的应用,那么这一小节我们就来说说异步通信.好了,既 然说到了异步通信,那 么”消 息中间件”就 要登场了,采 用异步通信这其实也是关系到系统的伸缩性,以及最大化的对各个子系统进行解耦.

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，而对于本身业务 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。

OK,那 么下一步我们说说异步能给系统带来什么样子的好处。首先我们想想，假如系统有A和B两个 子系统构成，假如A和B是 同步通信的话，那么要想使得系统整体伸缩性提高必须同时对A和B进行 伸缩，这就影响了对整个系统进行scale out.其次，同步调用还会影响到可用性，从数学推理的角度来说，A同 步调用B， 如果A可 用，那么B可 用，逆否命题就是如果B不 可用，那么A也 不可用，这将大大影响到系统可用性，再次，系统之间异步通信以后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 提高了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（当然了，请求的总体处理时间也许不会变少）。

下 面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，这就要求要向交易成功，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。

最后，关于异步方面的讨论，我可以 推荐大家一些资源：

 1 . [J2EE meets web2.0](http://www.ibm.com/developerworks/web/library/wa-aj-web2jee/?S_CMP=cn-a-wa&S_TACT=105AGX52)

 2. [Ebay架构特点(HPTS 2009)](http://www.jdon.com/jivejdon/thread/37753)

**六 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，比如一些配置文件，一个用户对应的动态，以及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，另外还有一类数据，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，因此淘宝目前也开发了自己的 分布式文件系统TFS，TFS目 前限制了文件大小为2M， 适合于一些小于2M数 据的存放。

        随 着互联网的发展，业界从08年 下半年开始逐渐流行了一个概念就是NOSQL。我们都知道根据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，我们传统的关系数据采用了ACID的事务策略，而ACID的 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，包括facebook 的cassandra,apache hbase,google bigtable等，这些产品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。目前淘宝也在研究和使用一些成熟的NOSQL产品。

**七 监控、预警系统**

       对于大型的系统 来说，唯一可靠的就是系统的各个部分是不可靠。

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，比如网络交换机，普通PC机，各种型号的网卡，硬盘，内存等等，而这 些东东都在数量非常多的时候，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，而细粒度一点的话，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。

前 面说了一个监控系统的重要性，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，这样以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。

**八 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，就需要再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。