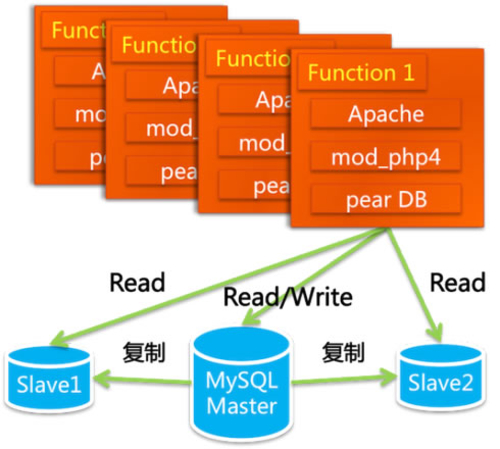
淘宝体系架构：淘宝网站技术架构

1. **LAMP（Linux+Apache+Mysql+php）**

拆分成一个主库、两个从库，读写分离



上述方案随着访问量和数据量的飞速上涨，问题很快出现。

1）数据库锁表，MySQL当时是第4版的，使用默认的存储引擎MyISAM → 读数据时会锁表，尤其是主库往从库写数据时会对主库产生大量的读操作，使主库性能急剧下降。

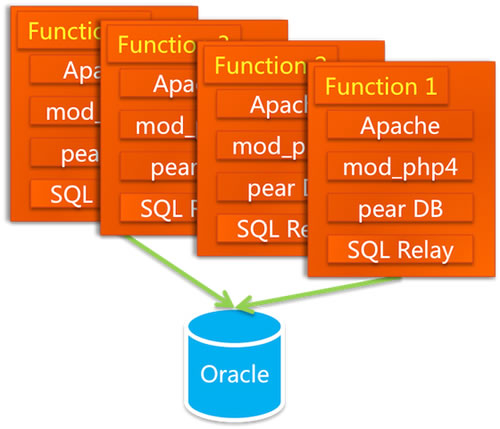
高访问量时，数据库支撑不住。 (Oracle在写数据时会有行锁，读数据时没有)

2）数据容量和安全性问题，MySQL当时不稳定。

**二、ORACLE本地后台DB+连接池**

更换数据库为ORACLE，但更换数据库不是只换个库就可以的，访问方式，SQL 语法都要跟着变，最重要的一点是，Oracle 并发访问能力之所以如此强大，有一个关键性的设计——连接池。但对于 PHP 语言来说它是放在 Apache 上的，每一个请求都会对数据库产生一个连接，它没有连接池这种功能(Java 语言有 Servlet 容器，可以存放连接池)。

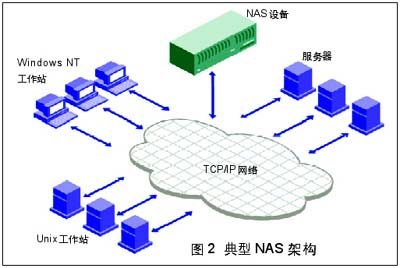
开发了一个PHP的连接池，一个开源的连接池代理服务 SQLRelay。



三、**ORACLE+NAS+RAC+连接池**

数据一开始是放在本地的，DBA 们对 Oracle 做调优的工作，也对 SQL 进行调优。后来数据量变大了，本地存储不行了。买了 NAS(NetworkAttached Storage：网络附属存储)，NetApp 的 NAS 存储作为了数据库的存储设备，加上 Oracle RAC(Real Application Clusters，实时应用集群)来实现负载均衡。

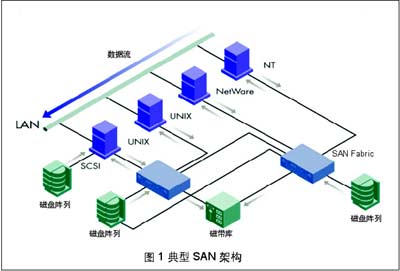
NAS(NetworkAttached Storage：网络附属存储)， NAS类似于FTP，主要应用于文件共享。适用于存取为只读方式、数据库小、存取量低、不适合处理大规模块数据、而且会造成网络拥塞。



四、**ORACLE+SAN+RAC+连接池**

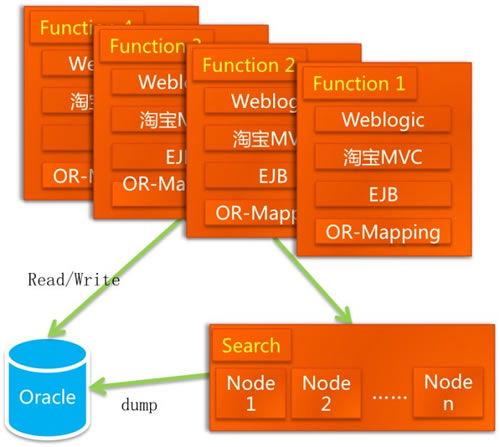
后来采购了 Dell 和 EMC 合作的 SAN 低端存储，性能一下子提升了 10 几倍。

SAN：存储区域网。子网中的设备可以从主网卸载流量。通常SAN由RAID阵列连接光纤通道组成，SAN和服务器和客户机的数据通信通过SCSI命令而非TCP/IP，数据处理是“块级”



QLRelay 之后就噩梦不断，这个代理服务经常会死锁，如同之前的 MySQL 死锁一样。

**五、大刀阔斧修改**



语言：PHP->JAVA

MVC 框架：WebX(<http://www.openwebx.org/docs/Webx3_Guide_Book.html>)

控制层：EJB

持久层： iBATIS

另外为了缓解数据库的压力，商品查询和店铺查询放在搜索引擎上面

之后还买了小型机，在小型机上跑ORACLE。

**六、增加DBRoute**

一台 Oracle 的处理能力有限：连接池有数量限制，查询速度跟容量成反比。数据量上亿、查询量上亿时，到达极限。

↓

多用几个Oracle 数据库，即”分库分表“。

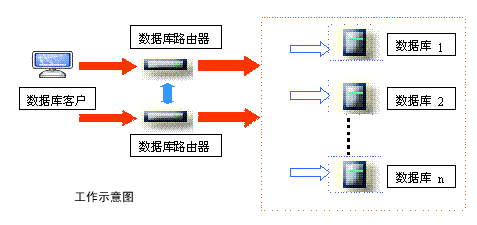
↓

用户信息按ID分到两个数据库中 (DB1/DB2)，商品信息和卖家也分到两个对应的数据库中，商品类目等通用信息放在第三个库中 (DBcommon)。

买家的操作（关键字查询、分页、按时间排序）

↓

数据库路由程序框架DBRoute（自行开发）。

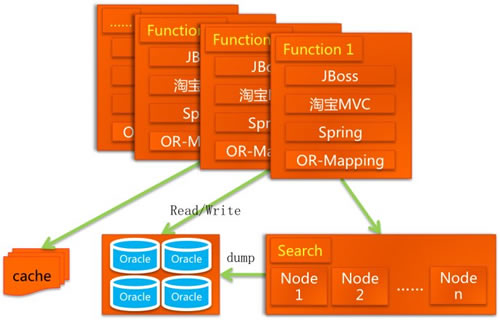


利用 Spring 的反射(IoC)模式替代了 EJB 的工厂模式，精简了很多代码。

七、添加缓存+CDN(内容分发到网络)

庞大的访问量和数据量继续给数据和存储带来压力，通过缓存（一个基于 Berkeley DB 的开源的缓存系统，用于缓存变化较少的只读信息）和 CDN(内容分发网络)提升性能。

八、采用开源的JBoss



**九、TFS（Taobao File System）**

考虑两点原因：查询的负载；存储有限。不能让所有的商品一直存放在主库中。

↓

起初，商品由卖家选择7天或14天有效期，到期之后就下架，须重新发布上架，之后变成新的商品(ID改变)。

↓

商品之前的好评没有和新商品ID关联。

↓

拆分商品和交易：一个商家的一种商品有唯一的ID（不论上下架与否）。

↓

卖家改价格、库存，已成交的信息怎么处理?

↓

快照文件：买家每交易一次都记录下商品的快照信息。

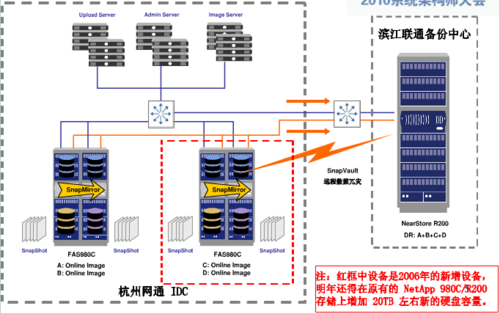
↓

内容增多，存储成本增加。

淘宝网要存储很多文件：一个商品几张图片，每一张图片要生成几张规格不同的缩略图，商品有描述信息，还有交易快照。

2010年，淘宝网有286亿个图片文件，图片的访问流量占整体流量的90%以上。图片平均大小为17.45KB，小于8K的图片占图片总数61%，占系统容量的11%。

2007年之前的图片存储架构如下图：



对于大量高并发访问的系统，困难在于大规模的小文件存储与读取（磁头要频繁的寻道和换道，容易导致较长的读取延时）。

局限性：

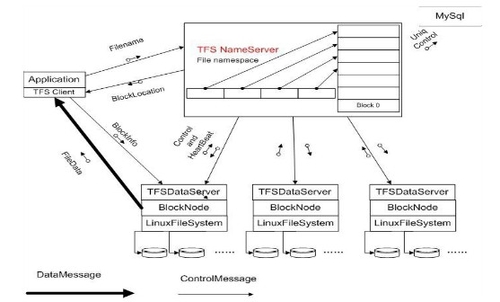
1）商用存储系统没有对小文件存储和读取结合应用需求进行有针对性的优化。

2）网络存储设备容量有限，文件数量大，系统扩容成本高（10T容量需要几百万）。

3）整个系统所连接的服务器也越来越多，网络连接数已经到达了网络存储设备的极限。

这么多的图片数据、这么大的访问流量，在读取上容易带来较长的延时。在大量高并发访问量的情况下，简直就是系统的噩梦。怎么办?

TFS架构图：



在这个架构中：

　　• 每个Data Server运行在一台普通的Linux主机上

　　• 以block文件的形式存放数据文件(一般64M一个block)

　　• block存多份保证数据安全

　　• 利用ext3文件系统存放数据文件

　　• 磁盘raid5做数据冗余

　　• 文件名内置元数据信息，用户自己保存TFS文件名与实际文件的对照关系–使得元数据量特别小。

TFS优化策略：

文件名内置元数据信息，在图片的保存文件名上暗藏了一些元数据信息（图片的大小、时间、访问频次等信息），抛弃了传统的目录树结构，减少开销。

**十、OceanBase（海量数据库）**