

大型 WEB 网站架构深入分析

1、HTML 静态化

其实大家都知道，效率最高、消耗最小的就是纯静态化的 html 页面，所以我们尽可能使我们的网站上的页面采用静态页面来实现，这个最简单的方法其实也是最有效的方法。但是对于大量内容并且频繁更新的网站，我们无法全部手动去挨个实现，于是出现了我们常见的信息发布系统 CMS，像我们常访问的各个门户站点的新闻频道，甚至他们的其他频道，都是通过信息发布系统来管理和实现的，信息发布系统可以实现最简单的信息录入自动生成静态页面，还能具备频道管理、权限管理、自动抓取等功能，对于一个大型网站来说，拥有一套高效、可管理的 CMS 是必不可少的。

除了门户和信息发布类型的网站，对于交互性要求很高的社区类型网站来说，尽可能的静态化也是提高性能的必要手段，将社区内的帖子、文章进行实时的静态化，有更新的时候再重新静态化也是大量使用的策略，像 Mop 的大杂烩就是使用了这样的策略，网易社区等也是如此。

同时，html 静态化也是某些缓存策略使用的手段，对于系统中频繁使用数据库查询但是内容更新很小的应用，可以考虑使用 html 静态化来实现，比如论坛中论坛的公用设置信息，这些信息目前的主流论坛都可以进行后台管理并且存储再数据库中，这些信息其实大量被前台程序调用，但是更新频率很小，可以考虑将这部分内容进行后台更新的时候进行静态化，这样避免了大量的数据库访问请求。

2、图片服务器分离

大家知道，对于 Web 服务器来说，不管是 Apache、IIS 还是其他容器，图片是最消耗资源的，于是我们有必要将图片与页面进行分离，这是基本上大型网站都会采用的策略，他们都有独立的图片服务器，甚至很多台图片服务器。这样的架构可以降低提供页面访问请求的服务器系统压力，并且可以保证系统不会因为图片问题而崩溃，在应用服务器和图片服务器上，可以进行不同的配置优化，比如 apache 在配置 ContentType 的时候可以尽量少支持，尽可能少的 LoadModule，保证更高的系统消耗和执行效率。

3、数据库集群和库表散列

大型网站都有复杂的应用，这些应用必须使用数据库，那么在面对大量访问的时候，数据库的瓶颈很快就能显现出来，这时一台数据库将很快无法满足应用，于是我们需要使用数据库集群或者库表散列。

在数据库集群方面，很多数据库都有自己的解决方案，Oracle、Sybase 等都有很好的方案，常用的 MySQL 提供的 Master/Slave 也是类似的方案，您使用了什么样的 DB，就参考相应的解决方案来实施即可。

上面提到的数据库集群由于在架构、成本、扩张性方面都会受到所采用 DB 类型的限制，于是我们需要从应用程序的角度来考虑改善系统架构，库表散列是常用并且最有效的解决方案。我们在应用程序中安装业务和应用或者功能模块将数据库进行分离，不同的模块对应不同的数据库或者表，再按照一定的策略对某个页面或者功能进行更小的数据库散列，比如用户表，按照用户 ID 进行表散列，这样就能够低成本的提升系统的性能并且有很好的扩展性。sohu 的论坛就是采用了这样的架构，将论坛的用户、设置、帖子等信息进行数据库分离，然后对帖子、用户按照板块和 ID 进行散列数据库和表，最终可以在配置文件中简单的配置便能让系统随时增加一台低成本的数据库进来补充系统性能。

4、缓存

缓存一词搞技术的都接触过，很多地方用到缓存。网站架构和网站开发中的缓存也是非常重要。这里先讲述最基本的两种缓存。高级和分布式的缓存在后面讲述。

架构方面的缓存，对 Apache 比较熟悉的人都能知道 Apache 提供了自己的缓存模块，也可以使用外加的 Squid 模块进行缓存，这两种方式均可以有效的提高 Apache 的访问响应能力。

网站程序开发方面的缓存，Linux 上提供的 Memory Cache 是常用的缓存接口，可以在 web 开发中使用，比如用 Java 开发的时候就可以调用 MemoryCache 对一些数据进行缓存和通讯共享，一些大型社区使用了这样的架构。另外，在使用 web 语言开发的时候，各种语言基本都有自己的缓存模块和方法，PHP 有 Pear 的 Cache 模块，Java 就更多了，.net 不是很熟悉，相信也肯定有。

5、镜像

镜像是大型网站常采用的提高性能和数据安全性的方式，镜像的技术可以解决不同网络接入商和地域带来的用户访问速度差异，比如 ChinaNet 和 EduNet 之间的差异就促使了很多网站在教育网内搭建镜像站点，数据进行定时更新或者实时更新。在镜像的细节技术方面，这里不阐述太深，有很多专业的现成的解决架构和产品可选。也有廉价的通过软件实现的思路，比如 Linux 上的 rsync 等工具。

6、负载均衡

负载均衡将是大型网站解决高负荷访问和大量并发请求采用的终极解决办法。

负载均衡技术发展了多年，有很多专业的服务提供商和产品可以选择，我个人接触过一些解决方法，其中有两个架构可以给大家做参考。

7、硬件四层交换

第四层交换使用第三层和第四层信息包的报头信息，根据应用区间识别业务流，将整个区间段的业务流分配到合适的应用服务器进行处理。第四层交换功能就象是虚 IP，指向物理服务器。它传输的业务服从的协议多种多样，有 HTTP、FTP、NFS、Telnet 或其他协议。这些业务在物理服务器基础上，需要复杂的载量平衡算法。在 IP 世界，业务类型由终端 TCP 或 UDP 端口地址来决定，在第四层交换中的应用区间则由源端和终端 IP 地址、TCP 和 UDP 端口共同决定。

在硬件四层交换产品领域，有一些知名的产品可以选择，比如 Alteon、F5 等，这些产品很昂贵，但是物有所值，能够提供非常优秀的性能和很灵活的管理能力。Yahoo 中国当初接近 2000 台服务器使用了三四台 Alteon 就搞定了。

8、软件四层交换

大家知道了硬件四层交换机的原理后，基于 OSI 模型来实现的软件四层交换也就应运而生，这样的解决方案实现的原理一致，不过性能稍差。但是满足一定量的压力还是游刃有余的，有人说软件实现方式其实更灵活，处理能力完全看你配置的熟悉能力。

软件四层交换我们可以使用 Linux 上常用的 LVS 来解决，LVS 就是 Linux Virtual Server，他提供了基于心跳线 heartbeat 的实时灾难应对解决方案，提高系统的鲁棒性，同时可供了灵活的虚拟 VIP 配置和管理功能，可以同时满足多种应用需求，这对于分布式的系统来说必不可少。

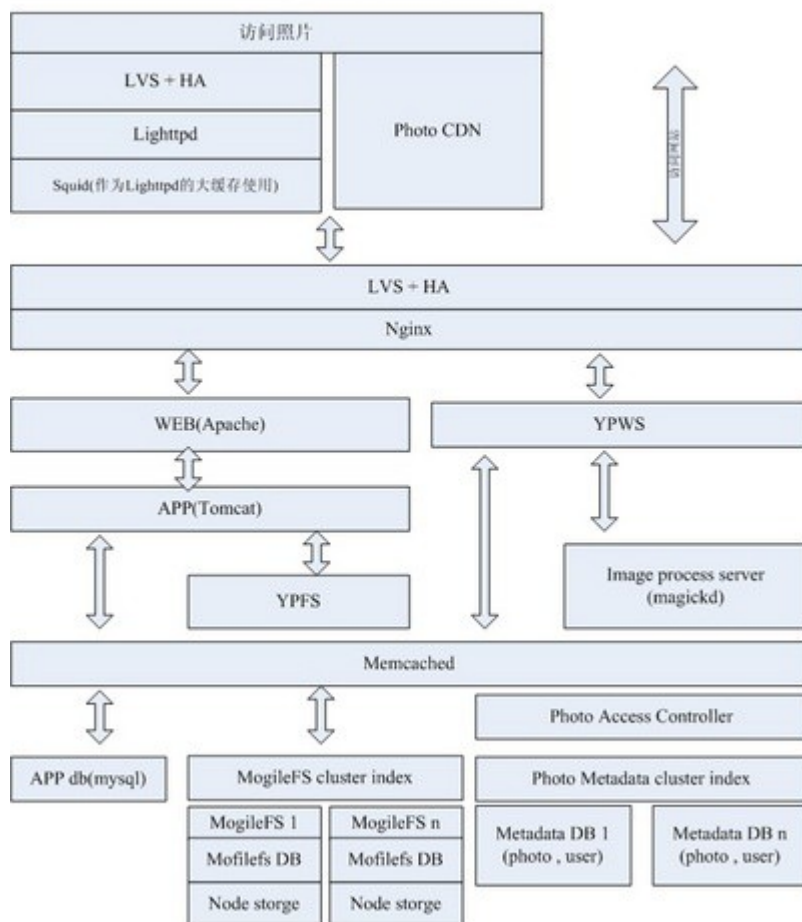
一个典型的使用负载均衡的策略就是，在软件或者硬件四层交换的基础上搭建 squid 集群，这种思路在很多大型网站包括搜索引擎上被采用，这样的架构低成本、高性能还有很强的扩张性，随时往架构里面增减节点都非常容易。这样的架构我准备空了专门详细整理一下和大家探讨。

对于大型网站来说，前面提到的每个方法可能都会被同时使用到，我这里介绍得比较浅显，具体实现过程中很多细节还需要大家慢慢熟悉和体会，有时一个很小的 squid 参数或者 apache 参数设置，对于系统性能的影响就会很大，希望大家一起讨论，达到抛砖引玉之效。

用 squid 做 web cache server，而 apache 在 squid 的后面提供真正的 web 服务。当然使用这样的架构必须要保证主页上大部分都是静态页面。这就需要程序员的配合将页面在反馈给客户端之前将页面全部转换成静态页面。

基本看出 sina 和 sohu 对于频道等栏目都用了相同的技术，即 squid 来监听这些 IP 的 80 端口，而真正的 web server 来监听另外一个端口。从用户的感觉上来说不会有任何的区别，而相对于将 web server 直接和客户端连在一起的方式，这样的方式明显的节省的带宽和服务器。用户访问的速度感觉也会更快。

Yupoo 网站架构



http://www.dbanotes.net/arch/yupoo_arch.html

带宽：4000M/S (参考)

服务器数量：60 台左右

Web 服务器：Lighttpd, Apache, nginx

应用服务器：Tomcat

其他：Python, Java, MogileFS、ImageMagick 等

关于 Squid 与 Tomcat

Squid 与 Tomcat 似乎在 Web 2.0 站点的架构中较少看到。我首先是对 Squid 有点疑问，对此阿华的解釋是"目前暂时还没找到效率比 Squid 高的缓存系统，原来命中率的确很差，后来在 Squid 前又装了层 Lighttpd, 基于 url 做 hash, 同一个图片始终会到同一台 squid 去，所以命中率彻底提高了"

对于应用服务器层的 Tomcat，现在 Yupoo! 技术人员也在逐渐用其他轻量级的东西替代，而 YPWS/YPFS 现在已经用 Python 进行开发了。

名次解释：

YPWS--Yupoo Web Server YPWS 是用 Python 开发的一个小型 Web 服务器，提供基本的 Web 服务外，可以增加针对用户、图片、外链网站显示的逻辑判断，可以安装于任何有空闲资源的服务器中，遇到性能瓶颈时方便横向扩展。

YPFS--Yupoo File System 与 YPWS 类似，YPFS 也是基于这个 Web 服务器上开发的图片上传服务器。

【Updated: 有网友留言质疑 Python 的效率，Yupoo 老大刘平阳在 del.icio.us 上写到 "YPWS 用 Python 自己

写的，每台机器每秒可以处理 294 个请求，现在压力几乎都在 10% 以下”】

图片处理层

接下来的 Image Process Server 负责处理用户上传的图片。使用的软件包也是 ImageMagick，在上次存储升级的同时，对于锐化的比率也调整过了(我个人感觉，效果的确好了很多)。“Magickd”是图像处理的一个远程接口服务，可以安装在任何有空闲 CPU 资源的机器上，类似 Memcached 的服务方式。

我们知道 Flickr 的缩略图功能原来是用 ImageMagick 软件包的，后来被雅虎收购后出于版权原因而不用了(?)；EXIF 与 IPTC Flicke 是用 Perl 抽取的，我是非常建议 Yupoo! 针对 EXIF 做些文章，这也是潜在产生受益的一个重点。

图片存储层

原来 Yupoo! 的存储采用了磁盘阵列柜，基于 NFS 方式的，随着数据量的增大，“Yupoo! 开发部从 07 年 6 月份就开始着手研究一套大容量的、能满足 Yupoo! 今后发展需要的、安全可靠的存储系统”，看来 Yupoo! 系统比较有信心，也是满怀期待的，毕竟这要支撑以 TB 计算的海量图片的存储和管理。我们知道，一张图片除了原图外，还有不同尺寸的，这些图片统一存储在 MogileFS 中。

对于其他部分，常见的 Web 2.0 网站必须软件都能看到，如 MySQL、Memcached、Lighttpd 等。Yupoo! 一方面采用不少相对比较成熟的开源软件，一方面也在自行开发定制适合自己的架构组件。这也是一个 Web 2.0 公司所必需要走的一个途径。

非常感谢一下 Yupoo! 阿华对于技术信息的分享，技术是共通的。下一个能爆料是哪家?

--EOF--

lighttpd+squid 这套缓存是放在另外一个机房作为 cdn 的一个节点使用的，图中没描绘清楚，给大家带来不便了。

squid 前端用 lighttpd 没用 nginx，主要是用了这么久，没出啥大问题，所以就没想其他的了。

URL Hash 的扩展性的确不好，能做的就是轻易去增减服务器，我们目前是 5 台服务器做一组 hash。

我们现在用 Python 写的 Web Server，在效率方面，我可以给个测试数据，根据目前的访问日志模拟访问测试的结果是 1 台 ypws, 平均每秒处理 294 个请求(加载所有的逻辑判断)。

在可靠性上，还没没具体的数据，目前运行 1 个多月还没有任何异常。

lvs 每个节点上都装 nginx，主要是为了反向代理及处理静态内容，不过 apache 已显得不是那么必需，准备逐渐去掉。

我们处理图片都是即时的，我们目前半数以上的服务器都装了 magickd 服务，用来分担图片处理请求。

http://www.dbanotes.net/review/tailrank_arch.html

每天数以千万计的 Blog 内容中，实时的热点是什么? [Tailrank](#) 这个 Web 2.0 Startup 致力于回答这个问题。专门爆料网站架构的 [Todd Hoff](#) 对 [Kevin Burton](#) 进行了采访。于是我们能了解一下 [Tailrank 架构](#) 的一些信息。每小时索引 2400 万的 Blog 与 Feed，内容处理能力为 160-200Mbps，IO 写入大约在 10-15MBps。每个月要处理 52T 之多的原始数据。Tailrank 所用的爬虫现在已经成为一个独立产品：[spinn3r](#)。

服务器硬件

目前大约 15 台服务器，CPU 是 64 位的 Opteron。每台主机上挂两个 SATA 盘，做 RAID 0。据我所知，国内很多 Web 2.0 公司也用的是类似的方式，SATA 盘容量大，低廉价格，堪称不二之选。操作系统用的是 Debian Linux。Web 服务器用 Apache 2.0，Squid 做反向代理服务器。

数据库

Tailrank 用 MySQL 数据库，联邦数据库形式。存储引擎用 InnoDB，数据量 500GB。Kevin Burton 也指出了 MySQL 5 在修了一些多核模式下互斥锁的问题([This Bug?](#))。到数据库的 JDBC 驱动连接池用 [lbpool](#) 做负载均衡。MySQL Slave 或者 Master 的复制用 [MySQLSlaveSync](#) 来轻松完成。不过即使这样，还要花费 20% 的时间来折腾 DB。

其他开放的软件

任何一套系统都离不开合适的 Profiling 工具，Tailrank 也不例外，针对 Java 程序的 Benchmark 用 [Benchmark4j](#)。Log 工具用 [Log5j](#)(不是 Log4j)。Tailrank 所用的大部分工具都是开放的。

Tailrank 的一个比较大的竞争对手是 [Techmeme](#)，虽然二者暂时看面向内容的侧重点有所不同。其实，最大的对手还是自己，当需要挖掘的信息量越来越大，如果精准并及时的呈现给用户内容的成本会越来越高。从现在来看，Tailrank 离预期目标还差的很远。期待罗马早日建成

<http://hideto.javaeye.com/blog/129726>

[YouTube 架构学习](#)

关键字: YouTube

原文: [YouTube Architecture](#)

YouTube 发展迅速，每天超过 1 亿的视频点击量，但只有很少人在维护站点和确保伸缩性。

平台

Apache

Python

Linux(SuSe)

MySQL

psyco，一个动态的 Python 到 C 的编译器

lighttpd 代替 Apache 做视频查看

状态

支持每天超过 1 亿的视频点击量

成立于 2005 年 2 月


于 2006 年 3 月达到每天 3 千万的视频点击量

于 2006 年 7 月达到每天 1 亿的视频点击量

2 个系统管理员，2 个伸缩性软件架构师

2 个软件开发工程师，2 个网络工程师，1 个 DBA

处理飞速增长的流量

Java 代码 

```
1. while (true)
2. {
3.     identify_and_fix_bottlenecks();
4.     drink();
5.     sleep();
6.     notice_new_bottleneck();
7. }
```

每天运行该循环多次

Web 服务器

1，NetScaler 用于负载均衡和静态内容缓存

2，使用 mod_fast_cgi 运行 Apache

- 3, 使用一个 Python 应用服务器来处理请求的路由
- 4, 应用服务器与多个数据库和其他信息源交互来获取数据和格式化 html 页面
- 5, 一般可以通过添加更多的机器来在 Web 层提高伸缩性
- 6, Python 的 Web 层代码通常不是性能瓶颈, 大部分时间阻塞在 RPC
- 7, Python 允许快速而灵活的开发和部署
- 8, 通常每个页面服务少于 100 毫秒的时间
- 9, 使用 psyco(一个类似于 JIT 编译器的动态的 Python 到 C 的编译器)来优化内部循环
- 10, 对于像加密等密集型 CPU 活动, 使用 C 扩展
- 11, 对于一些开销昂贵的块使用预先生成并缓存的 html
- 12, 数据库里使用行级缓存
- 13, 缓存完整的 Python 对象
- 14, 有些数据被计算出来并发送给各个程序, 所以这些值缓存在本地内存中。这是个使用不当的策略。应用服务器里最快的缓存将预先计算的值发送给所有服务器也花不了多少时间。只需弄一个代理来监听更改, 预计算, 然后发送。

视频服务

- 1, 花费包括带宽, 硬件和能源消耗
- 2, 每个视频由一个迷你集群来 host, 每个视频被超过一台机器持有
- 3, 使用一个集群意味着:
 - 更多的硬盘来持有内容意味着更快的速度
 - failover。如果一台机器出故障了, 另外的机器可以继续服务
 - 在线备份
- 4, 使用 lighttpd 作为 Web 服务器来提供视频服务:
 - Apache 开销太大
 - 使用 epoll 来等待多个 fds
 - 从单进程配置转变为多进程配置来处理更多的连接
- 5, 大部分流行的内容移到 CDN:
 - CDN 在多个地方备份内容, 这样内容离用户更近的机会就会更高
 - CDN 机器经常内存不足, 因为内容太流行以致很少有内容进出内存的颠簸
- 6, 不太流行的内容(每天 1-20 浏览次数)在许多 colo 站点使用 YouTube 服务器
 - 长尾效应。一个视频可以有多个播放, 但是许多视频正在播放。随机硬盘块被访问
 - 在这种情况下缓存不会很好, 所以花钱在更多的缓存上可能没太大意义。
 - 调节 RAID 控制并注意其他低级问题
 - 调节每台机器上的内存, 不要太多也不要太少

视频服务关键点

- 1, 保持简单和廉价
- 2, 保持简单网络路径, 在内容和用户间不要有太多设备
- 3, 使用常用硬件, 昂贵的硬件很难找到帮助文档
- 4, 使用简单而常见的工具, 使用构建在 Linux 里或之上的大部分工具
- 5, 很好的处理随机查找(SATA, tweaks)

缩略图服务

- 1, 做到高效令人惊奇的难

2, 每个视频大概 4 张缩略图, 所以缩略图比视频多很多

3, 缩略图仅仅 host 在几个机器上

4, 持有一些小东西所遇到的问题:

- OS 级别的大量的硬盘查找和 inode 和页面缓存问题

- 单目录文件限制, 特别是 Ext3, 后来移到多分层的结构。内核 2.6 的最近改进可能让 Ext3 允许大目录, 但在一个文件系统里存储大量文件不是个好主意

- 每秒大量的请求, 因为 Web 页面可能在页面上显示 60 个缩略图

- 在这种高负载下 Apache 表现的非常糟糕

- 在 Apache 前端使用 squid, 这种方式工作了一段时间, 但是由于负载继续增加而以失败告终。它让每秒 300 个请求变为 20 个

- 尝试使用 lighttpd 但是由于使用单线程它陷于困境。遇到多进程的问题, 因为它们各自保持自己单独的缓存

- 如此多的图片以致一台新机器只能接管 24 小时

- 重启机器需要 6-10 小时来缓存

5, 为了解决所有这些问题 YouTube 开始使用 Google 的 BigTable, 一个分布式数据存储:

- 避免小文件问题, 因为它将文件收集到一起

- 快, 错误容忍

- 更低的延迟, 因为它使用分布式多级缓存, 该缓存与多个不同 collocation 站点工作

- 更多信息参考 [Google Architecture](#), [GoogleTalk Architecture](#) 和 [BigTable](#)

数据库

1, 早期

- 使用 MySQL 来存储元数据, 如用户, tags 和描述

- 使用一整个 10 硬盘的 RAID 10 来存储数据

- 依赖于信用卡所以 YouTube 租用硬件

- YouTube 经过一个常见的革命: 单服务器, 然后单 master 和多 read slaves, 然后数据库分区, 然后 sharding 方式

- 痛苦与备份延迟。master 数据库是多线程的并且运行在一个大机器上所以它可以处理许多工作, slaves 是单线程的并且通常运行在小一些的服务器上并且备份是异步的, 所以 slaves 会远远落后于 master

- 更新引起缓存失效, 硬盘的慢 I/O 导致慢备份

- 使用备份架构需要花费大量的 money 来获得增加的写性能

- YouTube 的一个解决方案是通过把数据分成两个集群来将传输分出优先次序: 一个视频查看池和一个一般的集群

2, 后期

- 数据库分区

- 分成 shards, 不同的用户指定到不同的 shards

- 扩散读写

- 更好的缓存位置意味着更少的 IO

- 导致硬件减少 30%

- 备份延迟降低到 0

- 现在可以任意提升数据库的伸缩性

数据中心策略

1, 依赖于信用卡, 所以最初只能使用受管主机提供商

- 2, 受管主机提供商不能提供伸缩性, 不能控制硬件或使用良好的网络协议
- 3, YouTube 改为使用 colocation arrangement。现在 YouTube 可以自定义所有东西并且协定自己的契约
- 4, 使用 5 到 6 个数据中心加 CDN
- 5, 视频来自任意的数据中心, 不是最近的匹配或其他什么。如果一个视频足够流行则移到 CDN
- 6, 依赖于视频带宽而不是真正的延迟。可以来自任何 colo
- 7, 图片延迟很严重, 特别是当一个页面有 60 张图片时
- 8, 使用 BigTable 将图片备份到不同的数据中心, 代码查看谁是最近的

学到的东西

- 1, Stall for time。创造性和风险性的技巧让你在短期内解决问题而同时你会发现长期的解决方案
- 2, Prioritize。找出你的服务中核心的东西并对你的资源分出优先级别
- 3, Pick your battles。别怕将你的核心服务分出去。YouTube 使用 CDN 来分布它们最流行的内容。创建自己的网络将花费太多时间和太多 money
- 4, Keep it simple! 简单允许你更快的重新架构来回应问题
- 5, Shard。Sharding 帮助隔离存储, CPU, 内存和 IO, 不仅仅是获得更多的写性能
- 6, Constant iteration on bottlenecks :
 - 软件: DB, 缓存
 - OS: 硬盘 I/O
 - 硬件: 内存, RAID
- 7, You succeed as a team。拥有一个跨越条律的了解整个系统并知道系统内部是什么样的团队, 如安装打印机, 安装机器, 安装网络等等的人。With a good team all things are possible。

<http://hideto.javaeye.com/blog/130815>

[Google 架构学习](#)

关键字: Google

原文: [Google Architecture](#)

Google 是伸缩性的王者。Google 一直的目标就是构建高性能高伸缩性的基础组织来支持它们的产品。

平台

Linux

大量语言: Python, Java, C++

状态

在 2006 年大约有 450,000 台廉价服务器

在 2005 年 Google 索引了 80 亿 Web 页面, 现在没有人知道数目

目前在 Google 有超过 200 个 GFS 集群。一个集群可以有 1000 或者甚至 5000 台机器。成千上万的机器从运行着 5000000000000000 字节存储的 GFS 集群获取数据, 集群总的读写吞吐量可以达到每秒 40 兆字节

目前在 Google 有 6000 个 MapReduce 程序, 而且每个月都写成百个新程序

BigTable 伸缩存储几十亿的 URL, 几百千兆的卫星图片和几亿用户的参数选择

堆栈

Google 形象化它们的基础组织为三层架构:

- 1, 产品: 搜索, 广告, email, 地图, 视频, 聊天, 博客

- 2, 分布式系统基础组织: GFS, MapReduce 和 BigTable
- 3, 计算平台: 一群不同的数据中心里的机器
- 4, 确保公司里的人们部署起来开销很小
- 5, 花费更多的钱在避免丢失日志数据的硬件上, 其他类型的数据则花费较少

可信赖的存储机制 GFS(Google File System)

- 1, 可信赖的伸缩性存储是任何程序的核心需求。GFS 就是 Google 的核心存储平台
- 2, Google File System - 大型分布式结构化日志文件系统, Google 在里面扔了大量的数据
- 3, 为什么构建 GFS 而不是利用已有的东西? 因为可以自己控制一切并且这个平台与别的不一样, Google 需要:
 - 跨数据中心的高可靠性
 - 成千上万的网络节点的伸缩性
 - 大读写带宽的需求
 - 支持大块的数据, 可能为上千兆字节
 - 高效的跨节点操作分发来减少瓶颈
- 4, 系统有 Master 和 Chunk 服务器
 - Master 服务器在不同的数据文件里保持元数据。数据以 64MB 为单位存储在文件系统中。客户端与 Master 服务器交流来在文件上做元数据操作并且找到包含用户需要数据的那些 Chunk 服务器
 - Chunk 服务器在硬盘上存储实际数据。每个 Chunk 服务器跨越 3 个不同的 Chunk 服务器备份以创建冗余来避免服务器崩溃。一旦被 Master 服务器指明, 客户端程序就会直接从 Chunk 服务器读取文件
- 6, 一个上线的新程序可以使用已有的 GFS 集群或者可以制作自己的 GFS 集群
- 7, 关键点在于有足够的基础组织来让人们对自己的程序有所选择, GFS 可以调整来适应个别程序的需求

使用 MapReduce 来处理数据

- 1, 现在你已经有了一个很好的存储系统, 你该怎样处理如此多的数据呢? 比如你有许多 TB 的数据存储在 1000 台机器上。数据库不能伸缩或者伸缩到这种级别花费极大, 这就是 MapReduce 出现的原因
- 2, MapReduce 是一个处理和生成大量数据集的编程模型和相关实现。用户指定一个 map 方法来处理一个键/值对来生成一个中间的键/值对, 还有一个 reduce 方法来合并所有关联到同样的中间键的中间值。许多真实世界的任务都可以使用这种模型来表现。以这种风格来写的程序会自动并行的在一个大量机器的集群里运行。运行时系统照顾输入数据划分、程序在机器集之间执行的调度、机器失败处理和必需的内部机器交流等细节。这允许程序员没有多少并行和分布式系统的经验就可以很容易使用一个大型分布式系统资源
- 3, 为什么使用 MapReduce?
 - 跨越大量机器分割任务的好方式
 - 处理机器失败
 - 可以与不同类型的程序工作, 例如搜索和广告。几乎任何程序都有 map 和 reduce 类型的操作。你可以预先计算有用的数据、查询字数统计、对 TB 的数据排序等等
- 4, MapReduce 系统有三种不同类型的服务器
 - Master 服务器分配用户任务到 Map 和 Reduce 服务器。它也跟踪任务的状态
 - Map 服务器接收用户输入并在其基础上处理 map 操作。结果写入中间文件
 - Reduce 服务器接收 Map 服务器产生的中间文件并在其基础上处理 reduce 操作
- 5, 例如, 你想在所有 Web 页面里的字数。你将存储在 GFS 里的所有页面抛入 MapReduce。这将在成千上万台机器上同时进行并且所有的调整、工作调度、失败处理和数据传输将自动完成

-步骤类似于：GFS -> Map -> Shuffle -> Reduction -> Store Results back into GFS

-在 MapReduce 里一个 map 操作将一些数据映射到另一个中，产生一个键值对，在我们的例子里就是字和字数

-Shuffling 操作聚集键类型

-Reduction 操作计算所有键值对的综合并产生最终的结果

6，Google 索引操作管道有大约 20 个不同的 map 和 reduction。

7，程序可以非常小，如 20 到 50 行代码

8，一个问题是掉队者。掉队者是一个比其他程序慢的计算，它阻塞了其他程序。掉队者可能因为缓慢的 IO 或者临时的 CPU 不能使用而发生。解决方案是运行多个同样的计算并且当一个完成后杀死所有其他的

9，数据在 Map 和 Reduce 服务器之间传输时被压缩了。这可以节省带宽和 I/O。

在 BigTable 里存储结构化数据

1，BigTable 是一个大伸缩性、错误容忍、自管理的系统，它包含千千万兆的内存和 1000000000000000 的存储。它可以每秒钟处理百万的读写

2，BigTable 是一个构建于 GFS 之上的分布式哈希机制。它不是关系型数据库。它不支持 join 或者 SQL 类型查询

3，它提供查询机制来通过键访问结构化数据。GFS 存储存储不透明的数据而许多程序需求有结构化数据

4，商业数据库不能达到这种级别的伸缩性并且不能在成千上万台机器上工作

5，通过控制它们自己的低级存储系统 Google 得到更多的控制权来改进它们的系统。例如，如果它们想让跨数据中心的操作更简单这个特性，它们可以内建它

6，系统运行时机器可以自由的增删而整个系统保持工作

7，每个数据条目存储在一个格子里，它可以通过一个行 key 和列 key 或者时间戳来访问

8，每一行存储在一个或多个 tablet 中。一个 tablet 是一个 64KB 块的数据序列并且格式为 SSTable

9，BigTable 有三种类型的服务器：

-Master 服务器分配 tablet 服务器，它跟踪 tablet 在哪里并且如果需要则重新分配任务

-Tablet 服务器为 tablet 处理读写请求。当 tablet 超过大小限制(通常是 100MB-200MB)时它们拆开 tablet。当一个 Tablet 服务器失败时，则 100 个 Tablet 服务器各自挑选一个新的 tablet 然后系统恢复。

-Lock 服务器形成一个分布式锁服务。像打开一个 tablet 来写、Master 调整和访问控制检查等都需要互斥

10，一个 locality 组可以用来在物理上将相关的数据存储在一起来得到更好的 locality 选择

11，tablet 尽可能的缓存在 RAM 里

硬件

1，当你有很多机器时你怎样组织它们来使得使用和花费有效？

2，使用非常廉价的硬件

3，A 1,000-fold computer power increase can be had for a 33 times lower cost if you use a failure-prone infrastructure rather than an infrastructure built on highly reliable components. You must build reliability on top of unreliability for this strategy to work.

4，Linux，in-house rack design，PC 主板，低端存储

5，Price per wattage on performance basis isn't getting better. Have huge power and cooling issues

6，使用一些 collocation 和 Google 自己的数据中心

其他

1，迅速更改而不是等待 QA

2，库是构建程序的卓越方式

- 3, 一些程序作为服务提供
- 4, 一个基础组织处理程序的版本, 这样它们可以发布而不用害怕会破坏什么东西

Google 将来的方向

- 1, 支持地理位置分布的集群
- 2, 为所有数据创建一个单独的全局名字空间。当前的数据由集群分离
- 3, 更多和更好的自动化数据迁移和计算
- 4, 解决当使用网络划分来做广阔区域的备份时的一致性(例如保持服务即使一个集群离线维护或由于一些损耗问题)

学到的东西

- 1, 基础组织是有竞争性的优势。特别是对 Google 而言。Google 可以很快很廉价的推出新服务, 并且伸缩性其他人很难达到。许多公司采取完全不同的方式。许多公司认为基础组织开销太大。Google 认为自己是一个系统工程公司, 这是一个新的看待软件构建的方式
- 2, 跨越多个数据中心仍然是一个未解决的问题。大部分网站都是一个或者最多两个数据中心。我们不得不承认怎样在一些数据中心之间完整的分布网站是很需要技巧的
- 3, 如果你自己没有时间从零开始重新构建所有这些基础组织你可以看看 [Hadoop](#)。Hadoop 是这里很多同样的主意的一个开源实现
- 4, 平台的一个优点是初级开发人员可以在平台的基础上快速并且放心的创建健全的程序。如果每个项目都需要发明同样的分布式基础组织的轮子, 那么你将陷入困境因为知道怎样完成这项工作的人相对较少
- 5, 协同工作不一直是掷骰子。通过让系统中的所有部分一起工作则一个部分的改进将帮助所有的部分。改进文件系统则每个人从中受益而且是透明的。如果每个项目使用不同的文件系统则在整个堆栈中享受不到持续增加的改进
- 6, 构建自管理系统让你没必要让系统关机。这允许你更容易在服务器之间平衡资源、动态添加更大的容量、让机器离线和优雅的处理升级
- 7, 创建可进化的基础组织, 并行的执行消耗时间的操作并采取较好的方案
- 8, 不要忽略学院。学院有许多没有转变为产品的好主意。Most of what Google has done has prior art, just not prior large scale deployment.
- 9, 考虑压缩。当你有许多 CPU 而 IO 有限时压缩是一个好的选择。

<http://blog.daviesliu.net/2006/09/09/010620/>

Lighttpd+Squid+Apache 搭建高效率 Web 服务器

架构原理

[Apache](#) 通常是开源界的首选 Web 服务器, 因为它的强大和可靠, 已经具有了品牌效应, 可以适用于绝大部分的应用场合。但是它的强大有时候却显得笨重, 配置文件得让人望而生畏, 高并发情况下效率不太高。而轻量级的 Web 服务器 [Lighttpd](#) 却是后起之秀, 其静态文件的响应能力远高于 Apache, 据说是 Apache 的 2-3 倍。Lighttpd 的高性能和易用性, 足以打动我们, 在它能够胜任的领域, 尽量用它。Lighttpd 对 PHP 的支持也很好, 还可以通过 Fastcgi 方式支持其他的语言, 比如 Python。

毕竟 Lighttpd 是轻量级的服务器, 功能上不能跟 Apache 比, 某些应用无法胜任。比如 Lighttpd 还不支持缓存, 而现在的绝大部分站点都是用程序生成动态内容, 没有缓存的话即使程序的效率再高也很难满足大访问量的需求, 而且让程序不停的去做同一件事情也实在没有意义。首先, Web 程序是需要做缓存处理的, 即把反复使用的数据做缓存。即使这样也还不够, 单单是启动 Web 处理程序的代价就不少, 缓存最后生成的静态页面是必不可少的。而做这个是 [Squid](#) 的强项, 它本是做代理的, 支持高效的缓存, 可以用来给站点做反向代理加速。把 Squid 放在 Apache 或者 Lighttpd 的前端来缓存 Web 服务器生成的动态内

容，而 Web 应用程序只需要适当地设置页面实效时间即可。

即使是大部分内容动态生成的网站，仍免不了会有一些静态元素，比如图片、JS 脚本、CSS 等等，将 Squid 放在 Apache 或者 Lighttpd 前端后，反而会使性能下降，毕竟处理 HTTP 请求是 Web 服务器的强项。而且已经存在于文件系统中的静态内容再在 Squid 中缓存一下，浪费内存和硬盘空间。因此可以考虑将 Lighttpd 再放在 Squid 的前面，构成 Lighttpd+Squid+Apache 的一条处理链，Lighttpd 在最前面，专门用来处理静态内容的请求，把动态内容请求通过 proxy 模块转发给 Squid，如果 Squid 中有该请求的内容且没有过期，则直接返回给 Lighttpd。新请求或者过期的页面请求交由 Apache 中 Web 程序来处理。经过 Lighttpd 和 Squid 的两级过滤，Apache 需要处理的请求将大大减少，减少了 Web 应用程序的压力。同时这样的构架，便于把不同的处理分散到多台计算机上进行，由 Lighttpd 在前面统一把关。

在这种架构下，每一级都是可以单独优化的，比如 Lighttpd 可以采用异步 IO 方式，Squid 可以启用内存来缓存，Apache 可以启用 MPM 等，并且每一级都可以使用多台机器来均衡负载，伸缩性很好。