

COPYRIGHT © 2011, POSSIBILITY OUTPOST. ALL RIGHTS RESERVED.

→

- RECENT POSTS
- [A Super Short on the Youporn Stack - 300K QPS and 100 Million Page Views Per Day](#)
  - [A Short on the Pinterest Stack for Handling 3+ Million Users](#)
  - [Sponsored Post: Percona](#)

|  |   |
|--|---|
| advertise  | Live,<br>AiCache,<br>Next<br>Big<br>Sound,<br>ElasticHosts,<br>Red<br>5<br>Studios,<br>Logic<br>Monitor,<br>New<br>Relic,<br>AppDynamics,<br>CloudSigma,<br>ManageEngine,<br>Site24x7   |
| <ul style="list-style-type: none"><li>• Login</li><li>• Register</li><li>• Useful Products</li><li>• Useful Papers</li><li>• Useful Strategies</li><li>• Useful Blogs</li><li>• Useful Books</li><li>• Useful Conferences</li><li>• Book Store</li></ul> | <ul style="list-style-type: none"><li>• Tumblr Architecture</li></ul>   |
| <ul style="list-style-type: none"><li>• High Scalability RSS</li><li>• High Scalability Comments RSS</li></ul>   | <ul style="list-style-type: none"><li>• 15 Billion Page Views a Month and Harder to Scale than Twitter</li><li>• Stuff The Internet Says On Scalability For February 10, 2012</li><li>• Hypertable Routs HBase in Performance Test -- HBase Overwhelmed by Garbage Collection</li><li>• The Design of 99designs - A</li></ul> |

Clean  
Tens  
of  
Millions  
Pageviews  
Architecture  
• Stuff  
The  
Internet  
Says  
On  
Scalability  
For  
February  
3,  
2012  
• The  
Data-  
Scope  
Project  
-  
6PB  
storage,  
500GBytes/sec  
sequential  
IO,  
20M  
IOPS,  
130TFlops  
• Performance  
in  
the  
Cloud:  
Business  
Jitter  
is  
Bad

- ALL TIME  
FAVORITES
- [YouTube Architecture](#)
  - [Plenty Of Fish Architecture](#)
  - [Google Architecture](#)
  - [How Twitter Stores 250 Million Tweets A Day Using MySQL](#)
  - [Scaling Twitter: Making](#)

Twitter  
10000  
Percent  
Faster

- [Flickr Architecture](#)
- [Amazon Architecture](#)
- [How I Learned to Stop Worrying and Love Using a Lot of Disk Space to Scale](#)
- [Stack Overflow Architecture](#)
- [Facebook](#)
- [An Unorthodox Approach to Database Design : The Coming of the Shard](#)
- [Building Super Scalable Systems: Blade Runner Meets Autonomic Computing in the Ambient Cloud](#)
- [Are Cloud Based Memory Architectures the Next Big Thing?](#)
- [Latency is Everywhere and it Costs You Sales -](#)

- Howto  
Crush  
it
- How  
will  
memristors  
change  
everything?
- DataSift  
Architecture:  
Realtime  
Datamining  
At  
120,000  
Tweets  
Per  
Second
- Useful  
Scalability  
Blogs
- Scaling  
Traffic:  
People  
Pod  
Pool of  
On  
Demand  
Self  
Driving  
Robotic  
Cars  
who  
Automatically  
Refuel  
from  
Cheap Solar
- VoltDB  
Decapitates  
Six SQL  
Urban  
Myths  
And  
Delivers  
Internet  
Scale  
OLTP  
In The  
Process
- The  
Canonical  
Cloud  
Architecture
- Justin.Tv's  
Live  
Video  
Broadcasting  
Architecture
- Why  
Are  
Facebook,

Digg,  
And  
Twitter  
So  
Hard  
To  
Scale?

■ What  
The  
Heck  
Are  
You  
Actually  
Using  
NoSQL  
For?

■ Playfish's  
Social  
Gaming  
Architecture  
- 50  
Million  
Monthly  
Users  
And  
Growing

■ The  
Updated  
Big List  
Of  
Articles  
On The  
Amazon  
Outage

Wednesday  
Mar122008  
YouTube Architecture

WEDNESDAY, MARCH 12, 2008 AT 3:54 PM

**Update 2: YouTube Reaches One Billion Views Per Day.** *That's at least 11,574 views per second, 694,444 views per minute, and 41,666,667 views per hour.*

**Update: YouTube: The Platform.** YouTube adds a new rich set of APIs in order to become your video platform leader--all for free. Upload, edit, watch, search, and comment on video from your own site without visiting YouTube. Compose your site internally from APIs because you'll need to expose them later anyway.

YouTube grew incredibly fast, to over 100 million video views per day, with only a handful of people responsible for scaling the site. How did they manage to deliver all that video to all those users? And how have they evolved since being acquired by Google?

Information Sources

- 1. [Google Video](#)

Platform

1. Apache
2. Python
3. Linux (SuSe)
4. MySQL
5. psyco, a dynamic python->C compiler
6. lighttpd for video instead of Apache

## What's Inside?

### The Stats

1. Supports the delivery of over 100 million videos per day.
2. Founded 2/2005
3. 3/2006 30 million video views/day
4. 7/2006 100 million video views/day
5. 2 sysadmins, 2 scalability software architects
6. 2 feature developers, 2 network engineers, 1 DBA

### Recipe For Handling Rapid Growth

```
while (true)
{
    identify_and_fix_bottlenecks();
    drink();
    sleep();
    notice_new_bottleneck();
}
```

This loop runs many times a day.

### Web Servers

1. NetScaler is used for load balancing and caching static content.
2. Run Apache with mod\_fast\_cgi.
3. Requests are routed for handling by a Python application server.
4. Application server talks to various databases and other informations sources to get all the data and formats the html page.
5. Can usually scale web tier by adding more machines.
6. The Python web code is usually NOT the bottleneck, it spends most of its time blocked on RPCs.
7. Python allows rapid flexible development and deployment. This is critical given the competition they face.
8. Usually less than 100 ms page service times.
9. Use psyco, a dynamic python->C compiler that uses a JIT compiler approach to optimize inner loops.
10. For high CPU intensive activities like encryption, they use C extensions.
11. Some pre-generated cached HTML for expensive to render blocks.
12. Row level caching in the database.
13. Fully formed Python objects are cached.
14. Some data are calculated and sent to each application so the values are cached in local memory. This is an underused strategy. The fastest cache is in your application server and it doesn't take much time to send precalculated data to all your servers. Just have an agent that watches for changes, precalculates, and

sends.

## Video Serving

- Costs include bandwidth, hardware, and power consumption.
- Each video hosted by a mini-cluster. Each video is served by more than one machine.
- Using a cluster means:
  - More disks serving content which means more speed.
  - Headroom. If a machine goes down others can take over.
  - There are online backups.
- Servers use the `lighttpd` web server for video:
  - Apache had too much overhead.
  - Uses `epoll` to wait on multiple `fds`.
  - Switched from single process to multiple process configuration to handle more connections.
- Most popular content is moved to a CDN (content delivery network):
  - CDNs replicate content in multiple places. There's a better chance of content being closer to the user, with fewer hops, and content will run over a more friendly network.
  - CDN machines mostly serve out of memory because the content is so popular there's little thrashing of content into and out of memory.
- Less popular content (1-20 views per day) uses YouTube servers in various colo sites.
  - There's a long tail effect. A video may have a few plays, but lots of videos are being played. Random disks blocks are being accessed.
  - Caching doesn't do a lot of good in this scenario, so spending money on more cache may not make sense. This is a very interesting point. If you have a long tail product caching won't always be your performance savior.
  - Tune RAID controller and pay attention to other lower level issues to help.
  - Tune memory on each machine so there's not too much and not too little.

## Serving Video Key Points

1. Keep it simple and cheap.
2. Keep a simple network path. Not too many devices between content and users. Routers, switches, and other appliances may not be able to keep up with so much load.
3. Use commodity hardware. More expensive hardware gets the more expensive everything else gets too (support contracts). You are also less likely find help on the net.
4. Use simple common tools. They use most tools build into Linux and layer on top of those.
5. Handle random seeks well (SATA, tweaks).

## Serving Thumbnails

- Surprisingly difficult to do efficiently.
- There are a like 4 thumbnails for each video so there are a lot more thumbnails than videos.
- Thumbnails are hosted on just a few machines.
- Saw problems associated with serving a lot of small objects:
  - Lots of disk seeks and problems with inode caches and page



caches at OS level.

- Ran into per directory file limit. Ext3 in particular. Moved to a more hierarchical structure. Recent improvements in the 2.6 kernel may improve Ext3 large directory handling up to **100 times**, yet storing lots of files in a file system is still not a good idea.

- A high number of requests/sec as web pages can display 60 thumbnails on page.

- Under such high loads Apache performed badly.

- Used squid (reverse proxy) in front of Apache. This worked for a while, but as load increased performance eventually decreased. Went from 300 requests/second to 20.

- Tried using lighttpd but with a single threaded it stalled. Run into problems with multiprocesses mode because they would each keep a separate cache.

- With so many images setting up a new machine took over 24 hours.

- Rebooting machine took 6-10 hours for cache to warm up to not go to disk.

- To solve all their problems they started using Google's BigTable, a distributed data store:

- Avoids small file problem because it clumps files together.

- Fast, fault tolerant. Assumes its working on an unreliable network.

- Lower latency because it uses a distributed multilevel cache.

This cache works across different collocation sites.

- For more information on BigTable take a look at [Google Architecture](#), [GoogleTalk Architecture](#), and [BigTable](#).

## Databases

### 1. The Early Years

- Use MySQL to store meta data like users, tags, and descriptions.

- Served data off a monolithic RAID 10 Volume with 10 disks.

- Living off credit cards so they leased hardware. When they needed more hardware to handle load it took a few days to order and get delivered.

- They went through a common evolution: single server, went to a single master with multiple read slaves, then partitioned the database, and then settled on a sharding approach.

- Suffered from replica lag. The master is multi-threaded and runs on a large machine so it can handle a lot of work. Slaves are single threaded and usually run on lesser machines and replication is asynchronous, so the slaves can lag significantly behind the master.

- Updates cause cache misses which goes to disk where slow I/O causes slow replication.

- Using a replicating architecture you need to spend a lot of money for incremental bits of write performance.

- One of their solutions was prioritize traffic by splitting the data into two clusters: a video watch pool and a general cluster. The idea is that people want to watch video so that function should get the most resources. The social networking features of YouTube are less important so they can be routed to a less capable cluster.

### 2. The later years:

- Went to database partitioning.

- Split into shards with users assigned to different shards.
- Spreads writes and reads.
- Much better cache locality which means less IO.
- Resulted in a 30% hardware reduction.
- Reduced replica lag to 0.
- Can now scale database almost arbitrarily.

## Data Center Strategy

1. Used **manage hosting** providers at first. Living off credit cards so it was the only way.
2. Managed hosting can't scale with you. You can't control hardware or make favorable networking agreements.
3. So they went to a colocation arrangement. Now they can customize everything and negotiate their own contracts.
4. Use 5 or 6 data centers plus the CDN.
5. Videos come out of any data center. Not closest match or anything. If a video is popular enough it will move into the CDN.
6. Video bandwidth dependent, not really latency dependent. Can come from any colo.
7. For images latency matters, especially when you have 60 images on a page.
8. Images are replicated to different data centers using BigTable. Code looks at different metrics to know who is closest.

## Lessons Learned

1. **Stall for time.** Creative and risky tricks can help you cope in the short term while you work out longer term solutions.
2. **Prioritize.** Know what's essential to your service and prioritize your resources and efforts around those priorities.
3. **Pick your battles.** Don't be afraid to outsource some essential services. YouTube uses a CDN to distribute their most popular content. Creating their own network would have taken too long and cost too much. You may have similar opportunities in your system. Take a look at **Software as a Service** for more ideas.
4. **Keep it simple!** Simplicity allows you to rearchitect more quickly so you can respond to problems. It's true that nobody really knows what simplicity is, but if you aren't afraid to make changes then that's a good sign simplicity is happening.
5. **Shard.** Sharding helps to isolate and constrain storage, CPU, memory, and IO. It's not just about getting more writes performance.
6. **Constant iteration on bottlenecks:**
  - Software: DB, caching
  - OS: disk I/O
  - Hardware: memory, RAID
7. **You succeed as a team.** Have a good cross discipline team that understands the whole system and what's underneath the system. People who can set up printers, machines, install networks, and so on. With a good team all things are possible.

[Todd Hoff](#) | [49 Comments](#) | [Permalink](#) | [Share Article](#) [Print Article](#) [Email Article](#)

in [Apache](#), [CDN](#), [Example](#), [Linux](#), [MySQL](#), [Python](#), [Shard](#), [lighttpd](#)

# 连接超时

[Tweet](#)

## Reader Comments (49)

Fantastic article.

Thanks very much for this.

Dimitry.

---

November 29, 1990 | [Dimitry](#)

The real meat of this is skipped over in a couple of lines: keep the popular content on a CDN. In other words, throw money at Akamai and let Akamai worry about it.

That is, of course, the right answer, but whether you're using Python or not hardly matters. Without Akamai's services, YouTube could never have kept up with the demand given the infrastructure described above.

---

November 29, 1990 | [Anonymous](#)

*- Living off credit cards so they leased hardware. When they needed more hardware to handle load it took a few days to order and get delivered.*

How did this work exactly? When we looked into this, we found that because we were a new startup, we had no credit (I guess not 'found', its pretty obvious), so hardware leasing companies would only lease to us if one of us personally backed the loans. Given that startup risk was high and the bill was large, we ended up buying hardware and putting it on various low-intro-APR CC's, etc. All the big h/w vendors were like "unless we can see your last N years of tax returns etc., we're not leasing to you." Made it seem like leasing wasn't a real option for 'living off credit cards' startups.

---

November 29, 1990 | [Anonymous](#)

wow, thats a great article, didnt think about the CDN :P

---

November 29, 1990 | [alex](#)

be sure to accept private venture capital financing from Sequoia, who was also the largest shareholder of Google and controlled its board. Sequoia used its influence to force Google to massively overpay for Youtube, and the sequoia partners made an instant 500 million dollars in profit.

---

November 29, 1990 | [Anonymous](#)

The article and video have been extremely hopeful for a few projects I am working on. Thank you!

---

November 29, 1990 | [Quinton](#)

totally amazing; 100% worth the reading.

---

November 29, 1990 | [gadgetoo](#)

wow that's pretty insane.

---

November 29, 1990 | [Live TV](#)

A very good article yet I have not learned anything new. The current YouTube architecture is already applied to one of our customer youtube-like, a romanian website called <http://www.trilulilu.ro/> . The only thing that it wasn't yet implemented by our customer is database sharding, no need for now as the total MySQL database is under 250MB and the MySQL server handles at east more than 650 qps.

---

November 29, 1990 | [George](#)

Thats a great article.

---

November 29, 1990 | [Aaron](#)

I think it's interesting to note that they used mod\_fastcgi. I mean, I use it because I'm forced to on my shared host, but I've always heard of tons of problems when trying to scale big with it (even though that's what it's designed for). I guess if done right, FastCGI can be a great asset to a server farm.

---

November 29, 1990 | [Joeophone](#)

This is a great article, very interesting!

---

November 29, 1990 | [Danny](#)

Would love to hear more stories like this, Flickr, Twitter, MySpace, Meebo... Clients are still brain washed by big enterprise players thinking they need BEA Portal Server or the likes to achieve a robust, scalable enterprise solutions. It's a battle to convince them not to invest their money on something that's way to expensive, takes forever to deploy and cost a fortune (and takes forever) to make it do what you want it to do from the user experience perspective. I keep saying, "MySpace is registering 350,000 users a day and they aren't using Aqualogic - lets save that extra cash for some killer AJAX UI, widgets and an API that's actually useful.

---

November 29, 1990 | [Dean Whitney](#)

You were right in that living off YOUR personal credit card... that's for sure the case to Youtube back in the early days as well...

---

November 29, 1990 | [fjordan](#)

Thanks for the good article

---

November 29, 1990 | [Kiran Vaka](#)

Organize <http://anton.shevchuk.name/php/php-cookbook-myphptubecom/>'>MyPHPTube.com (YouTube clone) on LAMP (Linux, Apache, MySQL, PHP)

---

November 29, 1990 | [Anton Shevchuk](#)

This is a wonderful read, it's good proof that Python is not the

slow coach it's made out to be. In any language, the biggest setback will always be the programmer's skillset.

---

November 29, 1990 | **Codervoid**

Great Article. !

Does anybody know the evolution in the number of servers they had to use in the course of their ascension. ?  
How many did they start with and what kind of config each server had.

Also any idea which host they were using. ?

Thanks

---

November 29, 1990 | **Jeffos**

Thank you!

---

November 29, 1990 | **Namik**

OK that was interesting information, but let's stop calling a large collection of unordered bullet points an 'article' shall we? An article has, you know, sentences. Probably arranged into paragraphs.

---

November 29, 1990 | **Anonymous**

This is a great example what makes the web 2.0. Remmber those evil old days of starting a startup with millions of spending on Oracle, SUN or other big dude just for getting the the basic program running. Now, who needs them.

---

November 29, 1990 | **Flex RIA**

Does anyone know how Youtube or tinyURL generate unique ID? Are they using some kind of hash function? tinyURL doesn't generate unique ID if the URL is the same. For example, for [www.google.com](http://www.google.com), it always generates <http://tinyurl.com/1c2>. How do they encode/decode the URL?

---

November 29, 1990 | **Crt**

I guess I always assumed they were preallocated so they were always minimal and unpredictable, but I couldn't find an authoritative answer. It's an interesting question though.

---

November 29, 1990 | **Todd Hoff**

TinyURL needs a mapping of code -> URL, so when you type [www.google.com](http://www.google.com), it searches for the URL on its DB, and gives you the previously-generated code.

---

November 29, 1990 | **Anonymous**

Does anybody know the evolution in the number of servers they had to use in the course of their ascension. ?  
How many did they start with and what kind of config each server had.

Also any idea which host they were using. ?

Thanks

November 29, 1990 | [JEffos](#)

Page [1](#) [2](#) [Next](#) [25 Comments](#) [Most Recent Comment](#) »



## Tumblr: 150亿月浏览量背后的架构挑战 (下)

2012-02-15 21:15 | 910次阅读 | 【已有3条评论】[发表评论](#)  
来源: High Scalability | 作者: Todd Hoff | [收藏到我的网摘](#)

导读: 和许多新兴的网站一样, 著名的轻博客服务Tumblr在急速发展中面临了系统架构的瓶颈。每天5亿次浏览量, 峰值每秒4万次请求, 每天3TB新的数据存储, 超过1000台服务器, 这样的情况下如何保证老系统平稳运行, 平稳过渡到新的系统, Tumblr正面临巨大的挑战。近日, HighScalability网站的Todd Hoff采访了该公司的分布式系统工程师Blake Matheny, 撰文系统介绍了网站的架构, 内容很有价值。我们也非常希望国内的公司和团队多做类似分享, 贡献于社区的同时, 更能提升自身的江湖地位, 对招聘、业务发展都好处多多。欢迎通过[@CSDN云计算的微博](#)向我们投稿。

以下为译文的第二部分, 第一部分在[这里](#) (括号内小号为CSDN编辑所注):

### 内部的firehose(通信管道)

- 内部的应用需要活跃的信息流通。这些信息包括用户创建/删除的信息, liking/unliking的提示, 等等。挑战在于这些数据要实时的分布式处理。我们希望能够检测内部运行状况, 应用的生态系统能够可靠的生长, 同时还需要建设分布式系统的控制中心。
- 以前, 这些信息是基于Scribe(Facebook开源的分布式日志系统。)/Hadoop的分布式系统。服务会先记录在Scribe中, 并持续的长尾形式写入, 然后将数据输送给应用。这种模式可以立即停止伸缩, 尤其在峰值时每秒要创建数以千计的信息。不要指望人们会细水长流的发布文件和grep。
- 内部的firehose就像装载着信息的大巴, 各种服务和应用通过Thrift与消防管线沟通。(一个可伸缩的跨语言的服务开发框架。)
- LinkedIn的Kafka用于存储信息。内部人员通过HTTP链接firehose。经常面对巨大的数据冲击, 采用MySQL显然不是一个好主意, 分区实施越来越普遍。
- firehose的模型是非常灵活的, 而不像Twitter的firehose那样数据被假定是丢失的。
- firehose的信息流可以及时的回放。他保留一周内的数据, 可以调出这期间任何时间点的数据。
- 支持多个客户端连接, 而且不会看到重复的数据。每个客户端有一个ID。Kafka支持客户群, 每个群中的客户都用同一个ID, 他们不会读取重复的数据。可以创建多个客户端使用同一个ID, 而且不会看到重复的数据。这将保证数据的独立性和并行处理。Kafka使用ZooKeeper(Apache推出的开源分布式应用程序协调服务。)定期检查用户阅读了多少。

### 为Dashboard收件箱设计的Cell架构

- 现在支持Dashboard的功能的分散-集中架构非常受限, 这种状况不会持续很久。
- 解决方法是采用基于Cell架构的收件箱模型, 与Facebook Messages非常相似。
- 收件箱与分散-集中架构是对立的。每一位用户的dashboard都是由其追随者的发言和行动组成的, 并按照时间顺序存储。
- 就因为收件箱就解决了分散-集中的问题。你可以会问到底在收件箱中放了些什么, 让其如此廉价。这种方式将运行很长时间。
- 重写Dashboard非常困难。数据已经分布, 但是用户局部升级产生的数据交换的质量还没有完

### 本周热点排行

[更多](#)

- 37signal创始人: SaaS开头容易收获难
- 黑客的方式: 一切皆可自动化
- Tumblr: 150亿月浏览量背后的架构挑战 (上)
- NASA的大型电脑时代告終了, 敬礼!
- 超级计算机模拟显示核弹可引爆碰撞地球小行星
- 免费云存储的秘密: 贡献多余的存储
- UNIX/Linux网络技术语
- 揭秘Dell新罕布什尔的新办公室
- 印度Microsoft Store被黑, 用户密码泄露
- 闪存时代来临: 传统机械硬盘难满足云计算需求

### 热门招聘职位

[更多](#)

- 【上海骏丰数码】诚聘软件开发人员、研发经理。
- 【FT中文网】诚聘JavaScript 开发人员、PHP Senior F
- 【杭州海康威视数字】诚聘英才C/C++开发工程师、
- 【CSDN】高薪急聘PHP开发/UI设计/网站编辑/社区
- 【广东南航天合信息】诚聘需求分析、网页前端开发
- 【杭州驰度】高薪诚聘C++搜索引擎工程师、window

### 精彩专题

[更多](#)

Metro UI完全解析



国内云计算第一刊

全搞定。

- 数据量是非常惊人的。平均每条消息转发给上百个不同的用户，这比Facebook面对的困难还要大。大数据+高分布率+多个数据中心。
- 每秒钟上百万次写入，5万次读取。没有重复和压缩的数据增长为2.7TB，每秒百万次写入操作来自24字节行键。
- 已经流行的应用按此方法运行。
- cell
- 每个cell是独立的，并保存着一定数量用户的全部数据。在用户的Dashboard中显示的所有数据也在这个cell中。
- 用户映射到cell。一个数据中心有很多cell。
- 每个cell都有一个HBase的集群，服务集群，Redis的缓存集群。
- 用户归属到cell，所有cell的共同为用户发言提供支持。
- 每个cell都基于Finagle（Twitter推出的异步的远程过程调用库），建设在HBase上，Thrift用于开发与firehose和各种请求与数据库的链接。（请纠错）
- 一个用户进入Dashboard，其追随者归属到特定的cell，这个服务节点通过HBase读取他们的dashboard并返回数据。
- 后台将追随者的dashboard归入当前用户的table，并处理请求。
- Redis的缓存层用于cell内部处理用户发言。
- 请求流：用户发布消息，消息将被写入firehose，所有的cell处理这条消息并把发言文本写入数据库，cell查找是否所有发布消息追随者都在本cell内，如果是的话，所有追随者的收件箱将更新用户的ID。（请纠错）
- cell设计的优点：
- 大规模的请求被并行处理，组件相互隔离不会产生干扰。cell是一个并行的单位，因此可以任意调整规格以适应用户群的增长。
- cell的故障是独立的。一个Cell的故障不会影响其他cell。
- cell的表现非常好，能够进行各种升级测试，实施滚动升级，并测试不同版本的软件。
- 关键的思想是容易遗漏的：所有的发言都是可以复制到所有的cell。
- 每个cell中存储的所有发言的单一副本。每个cell可以完全满足Dashboard呈现请求。应用不用请求所有发言者的ID，只需要请求那些用户的ID。（“那些用户”所指不清，请指正。）他可以在dashboard返回内容。每一个cell都可以满足Dashboard的所有需求，而不需要与其他cell进行通信。
- 用到两个HBase table：一个table用于存储每个发言的副本，这个table相对较小。在cell内，这些数据将与存储每一个发言者ID。第二个table告诉我们用户的dashboard不需要显示所有的追随者。当用户通过不同的终端访问一个发言，并不代表阅读了两次。收件箱模型可以保证你阅读到。
- 发言并不会直接进入收件箱，因为那实在太大了。所以，发言者的ID将被发送到收件箱，同时发言内容将进入cell。这个模式有效的减少了存储需求，只需要返回用户在收件箱中浏览发言的时间。而缺点是每一个cell保存所有的发言副本。令人惊奇的是，所有发言比收件箱中的镜像要小。（请纠错）每天每个cell的发言增长50GB，收件箱每天增长2.7TB。用户消耗的资源远远超过他们制造的。
- 用户的dashboard不包含发言的内容，只显示发言者的ID，主要的增长来自ID。（请Tumblr用户纠错）
- 当追随者改变时，这种设计方案也是安全的。因为所有的发言都保存在cell中了。如果只有追随者的发言保存在cell中，那么当追随者改变了，将需要一些回填工作。
- 另外一种设计方案是采用独立的发言存储集群。这种设计的缺点是，如果群集出现故障，它会影响整个网站。因此，使用cell的设计以及后复制到所有cell的方式，创建了一个非常强大的架构。
- 一个用户拥有上百万的追随者，这带来非常大的困难，有选择的处理用户的追随者以及他们的存取模式（见Feeding Frenzy）
- 不同的用户采用不同并且恰当的存取模式和分布模型，两个不同的分布模式包括：一个适合受欢迎的用户，一个使用大众。
- 依据用户的类型采用不同的数据处理方式，活跃用户的发言并不会被真正发布，发言将被有选择的体现。（果真如此？请Tumblr用户纠错）
- 追随了上百万用户的用户，将像拥有上百万追随者的用户那样对待。
- cell的大小非常难于决定。cell的大小直接影响网站的成败。每个cell归于的用户数量是影响力之



一。需要权衡接受怎样的用户体验，以及为之付出多少投资。

- 从firehose中读取数据将是对网络最大的考验。在cell内部网络流量是可管理的。
- 当更多cell被增添到网络中来，他们可以进入到cell组中，并从firehose中读取数据。一个分层的数据复制计划。这可以帮助迁移到多个数据中心。

### 在纽约启动运作

- 纽约具有独特的环境，资金和广告充足。招聘极具挑战性，因为缺乏创业经验。
- 在过去的几年里，纽约一直致力于推动创业。纽约大学和哥伦比亚大学有一些项目，鼓励学生到初创企业实习，而不仅仅去华尔街。市长建立了一所学院，侧重于技术。

### 团队架构

- 团队：基础架构，平台，SRE，产品，web ops，服务；
- 基础架构：5层以下，IP地址和DNS，硬件配置；
- 平台：核心应用开发，SQL分片，服务，Web运营；
- SRE：在平台和产品之间，侧重于解决可靠性和扩展性的燃眉之急；
- 服务团队：相对而言更具战略性，
- Web ops：负责问题检测、响应和优化。

### 软件部署

- 开发了一套rsync脚本，可以随处部署PHP应用程序。一旦机器的数量超过200台，系统便开始出现问题，部署花费了很长时间才完成，机器处于部署进程中的各种状态。
- 接下来，使用Capistrano（一个开源工具，可以在多台服务器上运行脚本）在服务堆栈中构建部署进程（开发、分期、生产）。在几十台机器上部署可以正常工作，但当通过SSH部署到数百台服务器时，再次失败。
- 现在，所有的机器上运行一个协调软件。基于Redhat Func（一个安全的、脚本化的远程控制框架和接口）功能，一个轻量级的API用于向主机发送命令，以构建扩展性。
- 建立部署是在Func的基础上向主机发送命令，避免了使用SSH。比如，想在组A上部署软件，控制主机就可以找出隶属于组A的节点，并运行部署命令。
- 部署命令通过Capistrano实施。

Func API可用于返回状态报告，报告哪些机器上有这些软件版本。

- 安全重启任何服务，因为它们会关闭连接，然后重启。
- 在激活前的黑暗模式下运行所有功能。

### 展望

- 从哲学上将，任何人都可以使用自己想要的任意工具。但随着团队的发展壮大，这些工具出现了问题。新员工想要更好地融入团队，快速地解决问题，必须以他们为中心，建立操作的标准化。
- 过程类似于Scrum（一种敏捷管理框架），非常敏捷。
- 每个开发人员都有一台预配置的开发机器，并按照控制更新。
- 开发机会出现变化，测试，分期，乃至用于生产。
- 开发者使用VIM和TextMate。
- 测试是对PHP程序进行代码审核。
- 在服务方面，他们已经实现了一个与提交相挂钩的测试基础架构，接下来将继承并内建通知机制。

### 招聘流程

- 面试通常避免数学、猜谜、脑筋急转弯等问题，而着重关注应聘者在工作中实际要做什么。
- 着重编程技能。
- 面试不是比较，只是要找对的人。
- 挑战在于找到具有可用性、扩展性经验的人才，以应对Tumblr面临的网络拥塞。
- 在Tumblr工程博客（Tumblr Engineering Blog），他们对已过世的Dennis Ritchie和John McCarthy予以纪念。

### 经验及教训

- 自动化无处不在
- MySQL（增加分片）规模，应用程序暂时还不行

- Redis总能带给人惊喜
- 基于Scala语言的应用执行效率是出色的
- 废弃项目——当你不确定将如何工作时
- 不顾用在他们发展经历中没经历过技术挑战的人，聘用有技术实力的人是因为他们能适合你的团队以及工作。
- 选择正确的软件集合将会帮助你找到你需要的人
- 建立团队的技能
- 阅读文档和博客文章。
- 多与同行交流，可以接触一些领域中经验丰富的人，例如与在Facebook、Twitter、LinkedIn的工程师多交流，从他们身上可以学到很多
- 对技术要循序渐进，在正式投入使用之前他们煞费苦心的学习HBase和Redis。同时在试点项目中使用 或将其控制在有限损害范围之内。

翻译：包研，张志平

英文原文出自[High Scalability](#)

【 发表评论 3条 】

分享到



5



CSDN  
全球最大中文IT社区



CSDN移动频道  
专注于移动应用开发者的创优和创富



CSDN云计算频道  
做领先的云计算技术传媒



程序员杂志  
面向软件开发 者及管理者的专业月刊



CSDN蒋涛  
CSDN和《程序员》创始人



刘江  
CSDN&《程序员》总编

一键关注

相关文章


- 云计算技术产业研讨会暨中国云计算技术产业联盟
- 专家质疑:云计算扼杀应用开发?

网友评论 (共3条评论) ..

- 

jianhuxiaoming 2012-02-17 11:25:30

海数据时代。。

回复 (0) 支持 (0) 反对 (0) 举报 (0) | 0条回复 ..
- 

lericzhang 2012-02-17 11:23:40

Cell的架构模式还是挺适合12306的

回复 (0) 支持 (0) 反对 (0) 举报 (0) | 0条回复 ..
- 

qingyunyunhua 2012-02-17 09:35:36

nice article!

回复 (0) 支持 (0) 反对 (0) 举报 (0) | 0条回复 ..

发表评论 / 共3条评论 ..

欢迎你, falcon05

发表评论

#### 请您注意

- 自觉遵守：爱国、守法、自律、真实、文明的原则
- 尊重网上道德，遵守《全国人大常委会关于维护互联网安全的决定》及中华人民共和国其他各项有关法律法规
- 严禁发表危害国家安全，破坏民族团结、国家宗教政策和社会稳定，含侮辱、诽谤、教唆、淫秽等内容的作品
- 承担一切因您的行为而直接或间接导致的民事或刑事法律责任
- 您在CSDN新闻评论发表的作品，CSDN有权在网站内保留、转载、引用或者删除
- 参与本评论即表明您已经阅读并接受上述条款

[公司简介](#) | [招贤纳士](#) | [广告服务](#) | [银行汇款帐号](#) | [联系方式](#) | [版权声明](#) | [法律顾问](#) | [问题报告](#)

北京创新乐知信息技术有限公司 版权所有, 京 ICP 证 070598 号

世纪乐知(北京)网络技术有限公司 提供技术支持

江苏乐知网络技术有限公司 提供商务支持

 Email:webmaster@csdn.net

Copyright © 1999-2011, CSDN.NET, All Rights Reserved





## Tumblr: 150亿月浏览量背后的架构挑战 (上)

2012-02-14 18:24 | 3812次阅读 | 【已有8条评论】[发表评论](#)  
来源: High Scalability | 作者: Todd Hoff | [收藏到我的网摘](#)

导读: 和许多新兴的网站一样, 著名的轻博客服务Tumblr在急速发展中面临了系统架构的瓶颈。每天5亿次浏览量, 峰值每秒4万次请求, 每天3TB新的数据存储, 超过1000台服务器, 这样的情况下如何保证老系统平稳运行, 平稳过渡到新的系统, Tumblr正面临巨大的挑战。近日, HighScalability网站的Todd Hoff采访了该公司的分布式系统工程师Blake Matheny, 撰文系统介绍了网站的架构, 内容很有价值。我们也非常希望国内的公司和团队多做类似分享, 贡献于社区的同时, 更能提升自身的江湖地位, 对招聘、业务发展都好处多多。欢迎通过[@CSDN云计算的微博](#)向我们投稿。

以下为译文的第一部分。第二部分点[这里](#)。(括号内小号为CSDN编辑所注):

Tumblr每月页面浏览量超过150亿次, 已经成为火爆的博客社区。用户也许喜欢它的简约、美丽, 对用户体验的强烈关注, 或是友好而忙碌的沟通方式, 总之, 它深得人们的喜爱。

每月超过30%的增长当然不可能没有挑战, 其中可靠性问题尤为艰巨。每天5亿次浏览量, 峰值每秒4万次请求, 每天3TB新的数据存储, 并运行于超过1000台服务器上, 所有这些帮助Tumblr实现巨大的经营规模。

创业公司迈向成功, 都要迈过危险的迅速发展期这道门槛。寻找人才, 不断改造基础架构, 维护旧的架构, 同

时去面对逐月大增的流量, 而且曾经只有4位工程师。这意味着必须艰难地选择应该做什么, 不该做什么。这就是Tumblr的状况。好在现在已经有20位工程师了, 可以有精力解决问题, 并开发一些有意思的解决方案。

Tumblr最开始是非常典型的LAMP应用。目前正在向分布式服务模型演进, 该模型基于Scala、HBase、Redis (著名开源K-V存储方案)、Kafka (Apache项目, 出自LinkedIn的分布式发布-订阅消息系统)、Finagle (由Twitter开源的容错、协议中立的RPC系统), 此外还有一个有趣的基于Cell的架构, 用来支持Dashboard (CSDN注: Tumblr富有特色的用户界面, 类似于微博的时间轴)。

Tumblr目前的最大问题是如何改造为一个大规模网站。系统架构正在从LAMP演进为最先进的技术组合, 同时团队也要从小的创业型发展为全副武装、随时待命的正规开发团队, 不断创造出新的功能和基础设施。下面就是Blake Matheny对Tumblr系统架构情况的介绍。

网站地址

<http://www.tumblr.com/>

主要数据

- 每天5亿次PV (页面访问量)
- 每月超过150亿PV
- 约20名工程师
- 峰值请求每秒近4万次
- 每天超过1TB数据进入Hadoop集群
- MySQL/HBase/Redis/memcache每天生成若干TB数据
- 每月增长30%

### 本周热点排行

[更多](#)

- 01 37signal创始人: SaaS开头容易收获难
- 02 黑客的方式: 一切皆可自动化
- 03 Tumblr: 150亿月浏览量背后的架构挑战 (上)
- 04 NASA的大型电脑时代告終了, 敬礼!
- 05 超级计算机模拟显示核弹可引爆碰撞地球小行星
- 06 免费云存储的秘密: 贡献多余的存储
- 07 UNIX/Linux网络技术
- 08 揭秘Dell新罕布什尔的新办公室
- 09 印度Microsoft Store被黑, 用户密码泄露
- 10 闪存时代来临: 传统机械硬盘难满足云计算需求

### 热门招聘职位

[更多](#)

- 【上海骏丰数码】诚聘软件开发人员、研发经理。
- 【FT中文网】诚聘JavaScript 开发人员、PHP Senior F
- 【杭州海康威视数字】诚聘英才C/C++开发工程师、
- 【CSDN】高薪急聘PHP开发/UI设计/网站编辑/社区
- 【广东南航天合信息】诚聘需求分析、网页前端开发
- 【杭州驰度】高薪诚聘C++搜索引擎工程师、window

### 精彩专题

[更多](#)



Metro UI 完全解析



国内云计算第一刊

- 近1000硬件节点用于生产环境
- 平均每位工程师每月负责数以亿计的页面访问
- 每天上传大约50GB的文章，每天跟帖更新数据大约2.7TB（CSDN注：这两个数据的比例看上去不太合理，据Tumblr数据科学家Adam Laiacano在Twitter上解释，前一个数据应该指的是文章的文本内容和元数据，不包括存储在S3上的多媒体内容）

#### 软件环境

- 开发使用OS X，生产环境使用Linux（CentOS/Scientific）
- Apache
- PHP, Scala, Ruby
- Redis, HBase, MySQL
- Varnish, HAProxy, nginx
- memcache, Gearman（支持多语言的任务分发应用框架），Kafka, Kestrel（Twitter开源的分布式消息队列系统），Finagle
- Thrift, HTTP
- Func——一个安全、支持脚本的远程控制框架和API
- Git, Capistrano（多服务器脚本部署工具），Puppet, Jenkins

#### 硬件环境

- 500台Web服务器
- 200台数据库服务器（47 pool，20 shard）
- 30台memcache服务器
- 22台Redis服务器
- 15台Varnish服务器
- 25台HAproxy节点
- 8台nginx服务器
- 14台工作队列服务器（Kestrel + Gearman）

#### 架构

##### 1. 相对其他社交网站而言，Tumblr有其独特的使用模式：

- 每天有超过5千万篇文章更新，平均每篇文章的跟帖又数以百计。用户一般只有数百个粉丝。这与其他社会化网站里少数用户有数百万粉丝非常不同，使得Tumblr的扩展性极具挑战性。
- 按用户使用时间衡量，Tumblr已经是排名第二的社会化网站。内容的吸引力很强，有很多图片和视频，文章往往不短，一般也不会太长，但允许写得很长。文章内容往往比较深入，用户会花费更长的时间来阅读。
- 用户与其他用户建立联系后，可能会在Dashboard上往回翻几百页逐篇阅读，这与其他网站基本上只是部分信息流不同。
- 用户的数量庞大，用户的平均到达范围更广，用户较频繁的发帖，这些都意味着有巨量的更新需要处理。

##### 2. Tumblr目前运行在一个托管数据中心中，已在考虑地域上的分布性。

##### 3. Tumblr作为一个平台，由两个组件构成：公共Tumblelogs和Dashboard

- 公共Tumblelogs与博客类似（此句请Tumblr用户校正），并非动态，易于缓存
- Dashboard是类似于Twitter的时间轴，用户由此可以看到自己关注的所有用户的实时更新。与博客的扩展性不同，缓存作用不大，因为每次请求都不同，尤其是活跃的关注者。而且需要实时而且一致，文章每天仅更新50GB，跟帖每天更新2.7TB，所有的多媒体数据都存储在S3上面。
- 大多数用户以Tumblr作为内容浏览工具，每天浏览超过5亿个页面，70%的浏览来自Dashboard。
- Dashboard的可用性已经不错，但Tumblelog一直不够好，因为基础设施是老的，而且很难迁移。由于人手不足，一时半会儿还顾不上。

#### 老的架构

Tumblr最开始是托管在Rackspace上的，每个自定义域名的博客都有一个A记录。当2007年Rackspace无法满足其发展速度不得不迁移时，大量的用户都需要同时迁移。所以他们不得不将自定义域名保留在Rackspace，然后再使用HAProxy和Varnish路由到新的数据中心。类似这样的遗留问题很多。

开始的架构演进是典型的LAMP路线：

- 最初用PHP开发，几乎所有程序员都用PHP
- 最初是三台服务器：一台Web，一台数据库，一台PHP
- 为了扩展，开始使用memcache，然后引入前端cache，然后在cache前再加HAProxy，然后是MySQL sharding（非常奏效）
- 采用“在单台服务器上榨出一切”的方式。过去一年已经用C开发了两个后端服务：[ID生成程序](#)和[Staircar](#)（用Redis支持Dashboard通知）

Dashboard采用了“扩散-收集”方式。当用户访问Dashboard时将显示事件，来自所关注的用户的事件是通过拉然后显示的。这样支撑了6个月。由于数据是按时间排序的，因此sharding模式不太管用。

## 新的架构

由于招人和开发速度等原因，改为以JVM为中心。目标是将一切从PHP应用改为服务，使应用变成请求鉴别、呈现等诸多服务之上的薄层。

这其中，非常重要的一项是选用了Scala和Finagle。

- 在团队内部有很多人具备Ruby和PHP经验，所以Scala很有吸引力。
- Finagle是选择Scala的重要因素之一。这个来自Twitter的库可以解决大多数分布式问题，比如分布式跟踪、服务发现、服务注册等。
- 转到JVM上之后，Finagle提供了团队所需的所有基本功能（Thrift, ZooKeeper等），无需再开发许多网络代码，另外，团队成员认识该项目的一些开发者。
- Foursquare和Twitter都在用Finagle，Meetup也在用Scala。
- 应用接口与Thrift类似，性能极佳。
- 团队本来很喜欢Netty（Java异步网络应用框架，2月4日刚刚发布3.3.1最终版），但不想用Java，Scala是不错的选择。
- 选择Finagle是因为它很酷，还认识几个开发者。

之所以没有选择Node.js，是因为以JVM为基础更容易扩展。Node的发展为时尚短，缺乏标准、最佳实践以及大量久经测试的代码。而用Scala的话，可以使用所有Java代码。虽然其中并没有多少可扩展的东西，也无法解决5毫秒响应时间、49秒HA、4万每秒请求甚至有时每秒40万次请求的问题。但是，Java的生态链要大得多，有很多资源可以利用。

内部服务从C/libevent为基础正在转向Scala/Finagle为基础。

开始采用新的NoSQL存储方案如HBase和Redis。但大量数据仍然存储在大量分区的MySQL架构中，并没有用HBase代替MySQL。HBase主要支持短地址生产程序（数以十亿计）还有历史数据和分析，非常结实。此外，HBase也用于高写入需求场景，比如Dashboard刷新时一秒上百万的写入。之所以还没有替换HBase，是因为不能冒业务上风险，目前还是依靠人来负责更保险，先在一些小的、不那么关键的项目中应用，以获得经验。MySQL和时间序列数据sharding（分片）的问题在于，总有一个分片太热。另外，由于要在slave上插入并发，也会遇到读复制延迟问题。

此外，还开发了一个公用服务框架：

- 花了很多时间解决分布式系统管理这个运维问题。
- 为服务开发了一种Rails scaffolding，内部用模板来启动服务。
- 所有服务从运维的角度来看都是一样的，所有服务检查统计数据、监控、启动和停止的方式都一样。
- 工具方面，构建过程围绕SBT（一个Scala构建工具），使用插件和辅助程序管理常见操作，包括在Git里打标签，发布到代码库等等。大多数程序员都不再操心构建系统的细节了。

200台数据库服务器中，很多是为了提高可用性而设，使用的是常规硬件，但MTBF（平均故障间隔时间）极低。故障时，备用充足。

为了支持PHP应用有6个后端服务，并有一个小组专门开发后端服务。新服务的发布需要两到三周，包括Dashboard通知、Dashboard二级索引、短地址生成、处理透明分片的memcache代理。其中在MySQL分片上耗时很多。虽然在纽约本地非常热，但并没有使用MongoDB，他们认为MySQL的可扩展性足够了。

Gearman用于会长期运行无需人工干预的工作。

可用性是以达到范围（reach）衡量的。用户能够访问自定义域或者Dashboard吗？也会用错误率。



历史上总是解决那些最高优先级的问题，而现在会对故障模式系统地分析和解决，目的是从用户和应用的角

来定成功指标。（后一句原文似乎不全）

最开始Finagle是用于Actor模型的，但是后来放弃了。对于运行后无需人工干预的工作，使用任务队列。而且Twitter的util工具库中有Future实现，服务都是用Future（Scala中的无参数函数，在与函数关联的并行操作没有完成时，会阻塞调用方）实现的。当需要线程池的时候，就将Future传入Future池。一切都提交到Future池进行异步执行。

Scala提倡无共享状态。由于已经在Twitter生产环境中经过测试，Finagle这方面应该是没有问题的。使用Scala和Finagle中的结构需要避免可变状态，不使用长期运行的状态机。状态从数据库中拉出、使用再写回数据库。这样做的好处是，开发人员不需要操心线程和锁。

22台Redis服务器，每台的都有8-32个实例，因此线上同时使用了100多个Redis实例。

- Redis主要用于Dashboard通知的后端存储。
- 所谓通知就是指某个用户like了某篇文章这样的事件。通知会在用户的Dashboard中显示，告诉他其他用户对其内容做了哪些操作。
- 高写入率使MySQL无法应对。
- 通知转瞬即逝，所以即使遗漏也不会有严重问题，因此Redis是这一场景的合适选择。
- 这也给了开发团队了解Redis的机会。
- 使用中完全没有发现Redis有任何问题，社区也非常棒。
- 开发了一个基于Scala Futures的Redis接口，该功能现在已经并入了Cell架构。
- 短地址生成程序使用Redis作为一级Cache，HBase作为永久存储。
- Dashboard的二级索引是以Redis为基础开发的。
- Redis还用作Gearman的持久存储层，使用Finagle开发的memcache代理。
- 正在缓慢地从memcache转向Redis。希望最终只用一个cache服务。性能上Redis与memcache相当。

（先到这里吧，敬请期待下篇，包括如何用Kafaka、Scribe、Thrift实现内部活动流，Dashboard的Cell架构，开发流程和经验教训等精彩内容。）

翻译：包研，张志平，刘江；审校：刘江

英文原文出自[High Scalability](#)

【发表评论8条】

分享到

10



CSDN  
全球最大中文IT社区



CSDN移动频道  
专注于移动应用开发者的创优和创富



CSDN云计算频道  
做领先的云计算技术传媒



程序员杂志  
面向软件开发 者及管理者的 专业月刊



CSDN蒋涛  
CSDN和《程序员》创始人



刘江  
CSDN&《程序员》总编

一键关注

相关文章

云计算技术产业研讨会暨中国云计算技术产业联盟 专家质疑:云计算扼杀应用开发?

网友评论 (共8条评论) . .



hao05010323 2012-02-17 11:25:24

完全不懂，做cs太久了

回复(0) 支持(0) 反对(0) 举报(0) | 0条回复 . .



• binglang8632 2012-02-17 11:25:06

确实才知道。先觉得点点网创意不错，现在才知道，还是令人恶心的抄袭。我们国人就只能干这个吗？

回复(0) 支持(0) 反对(0) 举报(0) | 0条回复..



• phpppk 2012-02-17 11:24:59

高仿

回复(0) 支持(0) 反对(0) 举报(0) | 0条回复..



• ideation\_shang 2012-02-17 10:41:16

点点网不就是高仿 tumblr 吗？哈哈

回复(0) 支持(0) 反对(0) 举报(0) | 0条回复..



• iamxi 2012-02-17 10:33:08

第一次听说这个网站，落伍啦。的确很不错的网站。

回复(0) 支持(0) 反对(0) 举报(0) | 0条回复..



• Aselan 2012-02-17 09:45:41

主页很简洁很漂亮

回复(0) 支持(0) 反对(0) 举报(0) | 0条回复..



• AntiPro 2012-02-17 09:25:04

我访问了这个网站

We're sorry  
Our servers are over capacity and certain pages may be temporarily unavailable.  
We're working quickly to resolve the issue.

回复(0) 支持(0) 反对(0) 举报(0) | 0条回复..



• codeallen 2012-02-17 09:17:46

学习学习，天天向上。

回复(0) 支持(0) 反对(0) 举报(0) | 0条回复..

第一页 上一页 1 下一页 最末页

发表评论/共8条评论..

欢迎你, falcon05

发表评论

请您注意

- 自觉遵守：爱国、守法、自律、真实、文明的原则
- 尊重网上道德，遵守《全国人大常委会关于维护互联网安全的决定》及中华人民共和国其他各项有关法律法规
- 严禁发表危害国家安全，破坏民族团结、国家宗教政策和社会稳定，含侮辱、诽谤、教唆、淫秽等内容的作品
- 承担一切因您的行为而直接或间接导致的民事或刑事责任
- 您在CSDN新闻评论发表的作品，CSDN有权在网站内保留、转载、引用或者删除
- 参与本评论即表明您已经阅读并接受上述条款



