# Varnish 应用技术指南 V2.1

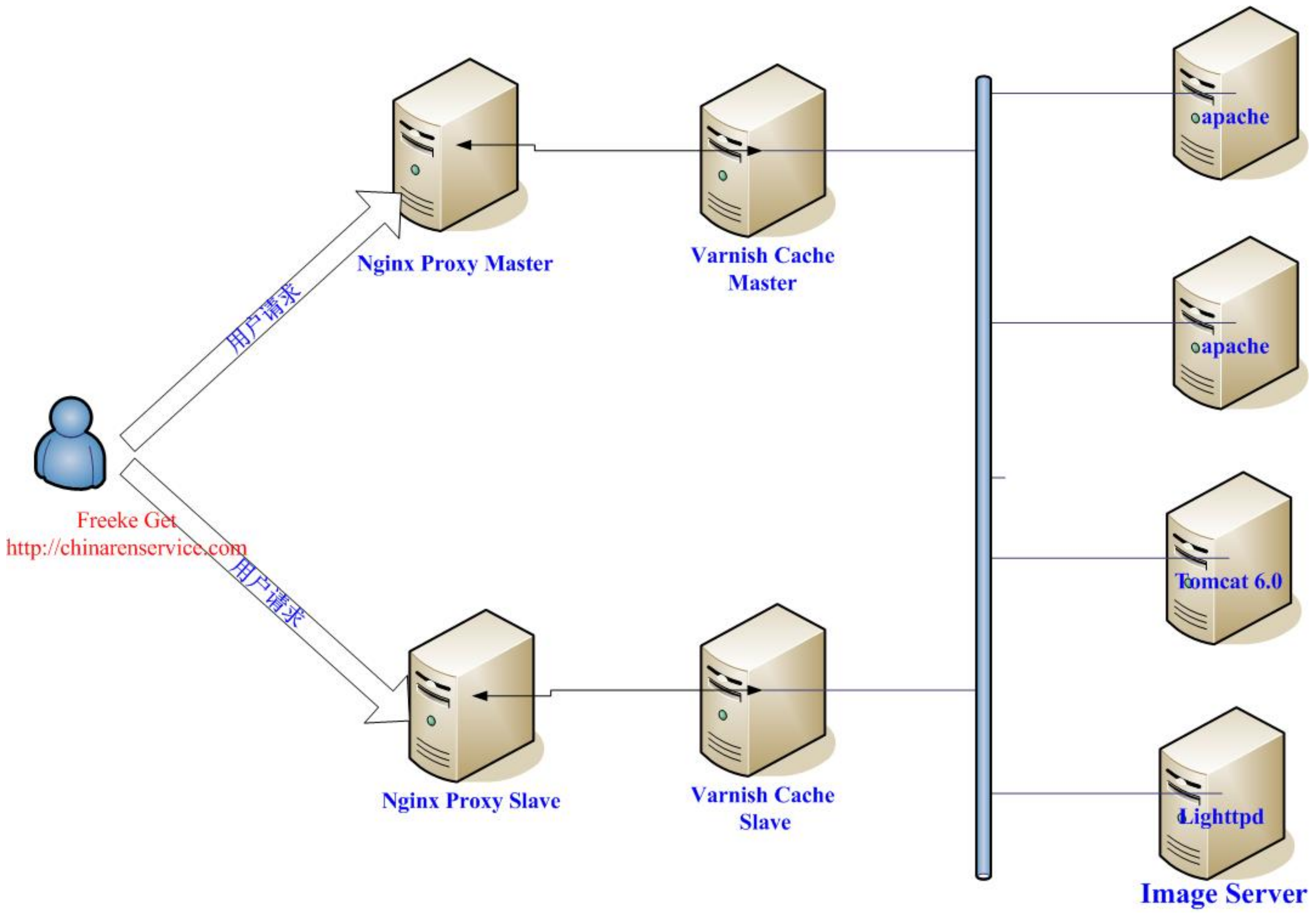
一、概述：

Varnish 是一款高性能的开源 HTTP 加速器，挪威最大的在线报纸 Verdens Gang (vg.no) 使用 3 台 Varnish 代替了原来的 12 台 squid，性能比以前更好。

Varnish 的作者 Poul-Henning Kamp 是 FreeBSD 的内核开发者之一，他认为现在的计算机比起 1975 年已经复杂许多。在 1975 年时，储存媒介只有两种：内存与硬盘。但现在计算机系统的内存除了主存外，还包括了 cpu 内的 L1、L2，甚至有 L3 快取。硬盘上也有自己的快取装置，因此 squid cache 自行处理物件替换的架构不可能得知这些情况而做到最佳化，但操作系统可以得知这些情况，所以这部份的工作应该交给操作系统处理，这就是 Varnish cache 设计架构。

Varnish 可以在 FreeBSD 6.0 和 Linux 2.6 内核上运行。

二、结构：



三、安装：

1、安装

```
#wget http://downloads.sourceforge.net/Varnish/Varnish-2.0.2.tar.gz?modtime=1224271223&big_mirror=0
```

```
# tar zxvf Varnish-2.0.2.tar.gz
# cd Varnish-2.0.2
# ./autogen.sh
# ./configure --prefix=/usr/local/Varnish/
#make
#make install
#mkdir /usr/local/Varnish/log/
#mkdir /usr/local/Varnish/cache/
#mkdir /usr/local/Varnish/etc/
#
```

2、启动

```
/usr/local/Varnish/sbin/Varnishd -n /usr/local/Varnish/cache -f /usr/local/Varnish/etc/freeke.vcl -a
0.0.0.0:83 -s file,/usr/local/Varnish/cache/Varnish_cache.data,1G -g www -u www -w 30000,51200,10 -T
127.0.0.1:3500 -p client_http11=on -P /usr/local/Varnish/var/Varnish.pid -p thread_pool_max=40000 -p
thread_pools=10 -p listen_depth=4096
/usr/local/Varnish/bin/Varnishncsa -n /usr/local/Varnish/cache -w /usr/local/Varnish/cache/Varnish.log&
```

## 四、　代码修改优化：

5、如何去除 HTTP header 部分的 Via 字段和 X-Varnish 字段

```
[root@chinarenservice ~]# curl --head http://www.chinarenservice.com/index.php
HTTP/1.1 200 OK
Server: nginx/0.5.33
Content-Type: text/html; charset=gb2312
X-Powered-By: PHP/5.2.5
Expires: Sun, 13 Jul 2008 05:43:16 GMT
Last-Modified: Fri, 13 Jun 2008 05:43:16GMT
Content-Encoding: gzip
Content-Length: 22403
Date: Wed, 18 Jun 2008 01:09:33 GMT
X-Varnish: 1426517043 1413292132
Age: 415577
Via: 1.1 Varnish
Connection: keep-alive
[root@chinarenservice Varnish-1.1.2]#  grep  -r Via *
bin/Varnishd/cache_response.c:  http_SetHeader(sp->wrk, sp->fd, sp->http, "Via: 1.1 Varnish");
bin/Varnishd/cache_response.c:  http_SetHeader(sp->wrk, sp->fd, sp->http, "Via: 1.1 Varnish");
include/http_headers.h:HTTPH("Via",                    H_Via,                  2, 0, 0, 0, 0)  /* RFC2616
14.45 */
[root@chinarenservice Varnish-1.1.2]#  grep -r X-Varnish *
bin/Varnishd/cache_synthetic.c: /* DO NOT generate X-Varnish header, RES_BuildHttp will */
bin/Varnishd/cache_response.c:  http_PrintfHeader(sp->wrk, sp->fd, sp->http, "X-Varnish: %u", sp->xid);
bin/Varnishd/cache_response.c:              "X-Varnish: %u %u", sp->xid, sp->obj->xid);
bin/Varnishd/cache_response.c:              "X-Varnish: %u", sp->xid);
bin/Varnishd/cache_http.c:      http_PrintfHeader(sp->wrk, sp->fd, hp, "X-Varnish: %u", sp->xid);
```

```
[root@chinarenservice  ~]# curl -I http://www.chinarenservice.com:81/
HTTP/1.1 200 OK
Server: Apache/2.2.6 (Unix) PHP/5.2.5
X-Powered-By: PHP/5.2.5
Content-Type: text/html; charset=gb2312
Content-Length: 27002
Date: Mon, 15 Dec 2008 00:59:27 GMT
Koncept: 1087701250 1087667318
Age: 323713
Via: 7.2 Koncept
Connection: keep-alive
```

## 5、对 Varnishstat 进行输出信息修改：

修改了一下 Varnishstat.c，方便自己查看信息，加了个 cache_hit /client_req 的统计。然后用 cron 每分钟运行一下 Varnishstat -1>info.txt，就可以直接在页面上访问到了。

```
hits/reqs                  57           .    HitRate(%)
client_conn             13368        12.55 Client connections accepted
client_req              13367        12.55 Client requests received
cache_hit                7625         7.16 Cache hits
cache_hitpass            3298         3.10 Cache hits for pass
```

在 static void do_once(struct Varnish_stats *VSL_stats)里加上

```
        intmax_t rr;
        rr=VSL_stats->cache_hit*100/VSL_stats->client_req;
        printf("%-16s %12ju %12s %s\n", "hits/reqs", rr, ".  ", "HitRate(%)");
```

当用-1 参数的时候在第一行显示缓存命中请求占所有请求的百分数。原来在持续输出中带的比率是 hit 和 miss 的比值。

## 5、对 Varnish 进行修改，避免 HTTP 头部信息丢失

之前碰到的Varnish的LostHeader的问题。HTTP应答里出现很多Vary: Accept-Encoding
觉得可能是 php 的问题，但由于出问题的那个文件是加密过的，问楼上做网站的同事，说也不知道程序是怎么回事情，估计改是不可能的了。一开始还怀疑是 nginx 的问题，但其他 PHP 并没有同样的问题。
刚才还是有些无聊的，所以随意看了一下代码，虽然没有看懂什么，但猜测还是有效的。
修改了 cache.h 里的 HTTP_HDR_MAX 值，原来是 32，改大一些，就 OK 了。猜测正确，程序里 HTTP 头部变量空间不够，原来丢掉的 gzip 信息正好是第 33 行。
索性改成 64，重新编译了一个，试了下，没有 LostHeader 了。

## 5、修改日志输出格式

修改 Varnish-1.1.2/bin/Varnishncsa/Varnishncsa.c

● 修改日志输出格式

```
[%d/%b/%Y:%T %z]
改为
%Y-%m-%d %T
```

将日志分隔符由空格改为"
" 改为\""

修改前后的日志输出对比

```
默认(修改前)
192.168.1.2 - - [15/May/2008:09:51:02 +0800] "GET http://dl.test.com:3128/pic/jpg/2007/11/03/045244/cw432_25299_0176.jpg HTTP/1.1" 200 30541 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; zh-CN; rv:1.8.1.14) Gecko/20080404 Firefox/2.0.0.14"
修改后
192.168.1.2"2008-05-15 11:34:29"GET http://dl.test.com:3128/pic/jpg/2007/12/04/125212/cw392_80737_0240.jpg HTTP/1.1"200"88057"-"curl/7.12.1 (x86_64-redhat-linux-gnu) libcurl/7.12.1 OpenSSL/0.9.7a zlib/1.2.1.2 libidn/0.5.6"
```

- 解决 127.0.0.1"2008-05-16 16:32:42"(null) (null) (null)"(null)"(null)"-"-"
  日志输出问题

```
    将 333 行的
    if (!lp->bogus)
    改为
    if (!lp->bogus && lp->df_h )
```

5、

五、 Varnish 配置：

1、2.0 配置文件内容参考：

```
#This is a basic VCL configuration file for varnish.  See the vcl(7)
#man page for details on VCL syntax and semantics.
#
#Default backend definition.  Set this to point to your content
#server.
#
backend shutter {
    .host = "212.66.20.167";
    .port = "80";
    }

backend ikuaimen {
    .host = "172.28.3.11";
    .port = "6060";
    }

backend sns {
    .host = "172.28.3.42";
    .port = "8080";
    }

####################   BBS Config
```

```
backend bbsas67 {
    .host = "172.28.3.69";
    .port = "80";
}
backend bbsas60 {
    .host = "172.28.3.62";
    .port = "80";
    }
director bbs random {
    { .backend = bbsas67; .weight = 1; }
    { .backend = bbsas60; .weight = 1; }
    }

#director bbs round-robin {
#   { .backend = bbsas60; }
#   { .backend = bbsas67; }
#}
#director bbs round-robin {
#    { .backend = { .host = "172.28.3.62"; .port = "http"; } }
#    { .backend = { .host = "172.28.3.69"; .port = "http"; } }
# }
##########################################################

acl purge {
       "localhost";
       "172.28.3.0"/24;
}

#
#Below is a commented-out copy of the default VCL logic.  If you
#redefine any of these subroutines, the built-in logic will be
#appended to your code.
#
sub vcl_recv {
# Add a unique header containing the client address
    remove req.http.X-Forwarded-For;
    set   req.http.X-Forwarded-For = client.ip;
    set req.grace = 30s;

 #### do not cache these files
    if (req.request != "GET" &&
        req.request != "HEAD" &&
        req.request != "PUT" &&
        req.request != "POST" &&
        req.request != "TRACE" &&
        req.request != "OPTIONS" &&
        req.request != "DELETE") {
       /* Non-RFC2616 or CONNECT which is weird. */
        pipe;
        }

    if (req.request != "GET" && req.request != "HEAD") {
       /* We only deal with GET and HEAD by default */
        pipe;
        }

    if (req.http.Expect) {
              pipe;
        }
    if (req.http.Authenticate || req.http.Authorization) {
              pass;
        }
##### always cache these items:
    if (req.http.host ~"^www.chinarenservice.com") {
        set req.backend = shutter;
        } elsif (req.http.host ~"^chinarenservice.com") {
            set req.backend = shutter;
        } elsif (req.http.host ~"^www.ichinarenservice.cn") {
            set req.backend = ikuaimen;
        } elsif (req.http.host ~"^www.ichinarenservice.cn.cn") {
            set req.backend = ikuaimen;
        } elsif (req.http.host ~"^www.ikuaimen.net") {
            set req.backend = ikuaimen;
        } elsif (req.http.host ~"^www.ikuaimen.org") {
            set req.backend = ikuaimen;
        } elsif (req.http.host ~"^www.1chinarenservice.cn") {
            set req.backend = ikuaimen;
        } elsif (req.http.host ~"^www.lchinarenservice.cn") {
            set req.backend = ikuaimen;

        } elsif (req.http.host ~"^static.ichinarenservice.cn") {
            set req.backend = sns;

        } elsif (req.http.host ~"^bbs.chinarenservice.com") {
            set req.backend = bbs;



    } else {
```

```
              error 404 "Unknown virtual host";
       }

       /* Always cache images and multimedia */
       if (req.url ~ "\.(jpg|jpeg|gif|png|tiff|tif|svg|swf|ico|mp3|mp4|m4a|ogg|mov|avi|wmv)$") {
              lookup;
       }

        /* Always cache CSS and javascript */
       if (req.url ~ "\.(css|js)$") {
              lookup;
       }

       /* Always cache static files */
       if (req.url ~
"\.(pdf|xls|vsd|doc|ppt|pps|vsd|doc|ppt|pps|xls|pdf|sxw|zip|gz|bz2|tgz|tar|rar|odc|odb|odf|odg|odi|odp|
ods|odt|sxc|sxd|sxi|sxw|dmg|torrent|deb|msi|iso|rpm)$") {
              lookup;
       }

#### if there is a purge make sure its coming from $localhost
       if (req.request == "PURGE") {
        if(!client.ip ~ purge) {
            error 405 "Not Allowed";
        }
            purge_url(req.http.X-Purge-Url);
            error 200 "Purged";
        }
 #### unknown function to "normalize the Accept-Encoding headers"
    if (req.http.Accept-Encoding) {
        if (req.http.Accept-Encoding ~ "gzip") {
            set req.http.Accept-Encoding = "gzip";
        } elsif (req.http.Accept-Encoding ~ "deflate") {
            set req.http.Accept-Encoding = "deflate";
        } else {
            # unkown algorithm
            unset req.http.Accept-Encoding;
              }
        }

 ####  don't cache authenticated sessions
    if (req.http.Cookie && req.http.Cookie ~ "is_logged_in=") {
        pipe;
        }
```

```
//  Varnish doesn't do INM requests so pass it through if no If-Modified-Since was sent
    if (req.http.If-None-Match && !req.http.If-Modified-Since) {
        pass;
        }

#### if it passes all these tests, do a lookup anyway;
    lookup;
}

sub vcl_pipe {
    pipe;
}

sub vcl_pass {
    pass;
}

sub vcl_hash {
    set req.hash += req.url;
    if (req.http.host) {
        set req.hash += req.http.host;
    } else {
        set req.hash += server.ip;
    }
    hash;
}

sub vcl_hit {
    if (req.request == "PURGE") {
        set obj.ttl = 0s;
        error 404 "Not in cache.";
       }

    if (!obj.cacheable) {
        pass;
    }
    deliver;
}

sub vcl_miss {
    fetch;
}

sub vcl_fetch {
    set obj.grace = 30s;

    if (!obj.cacheable) {
        pass;
     }

    if (obj.http.Set-Cookie ~ "is_logged_in=deleted(.*)") {
        #deliver;
     }

    if (obj.http.Set-Cookie) {
        pass;
     }

    if (req.request == "GET" && req.url ~
"\.(jpg|jpeg|gif|png|tiff|tif|svg|swf|ico|mp3|mp4|m4a|ogg|mov|avi|wmv)$") {
        set obj.ttl = 3600s;
        #deliver;
    } else {
        set obj.ttl = 30d;
        #deliver;
       }

    if (req.request == "GET" && req.url ~ "\.(css|js)$") {
        set obj.ttl = 3600s;
        #deliver;
    } else {
        set obj.ttl = 30d;
        #deliver;
       }

    if (req.request == "GET" && req.url ~
"\.(pdf|xls|vsd|doc|ppt|pps|vsd|doc|ppt|pps|xls|pdf|sxw|zip|gz|bz2|tgz|tar|rar|odc|odb|odf|odg|odi|odp|
ods|odt|sxc|sxd|sxi|sxw|dmg|torrent|deb|msi|iso|rpm)$") {
        set obj.ttl = 3600s;
        #deliver;
    } else {
        set obj.ttl = 30d;
        #deliver;
    }

#    set obj.prefetch =  -30s;
    #deliver;
}
```

```
sub vcl_deliver {
#    deliver;
}

sub vcl_discard {
   /* XXX: Do not redefine vcl_discard{}, it is not yet supported */
#    discard;
}

sub vcl_prefetch {
   /* XXX: Do not redefine vcl_prefetch{}, it is not yet supported */
#    fetch;
}

sub vcl_timeout {
   /* XXX: Do not redefine vcl_timeout{}, it is not yet supported */
#    discard;
}

#sub vcl_error {
#    set obj.http.Content-Type = "text/html; charset=utf-8";
#    synthetic {"
#<?xml version="1.0" encoding="utf-8"?>
#<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
# "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
#<html>
#  <head>
#    <title>"} obj.status " " obj.response {"</title>
#  </head>
#  <body>
#    <h1>Error "} obj.status " " obj.response {"</h1>
#    <p>"} obj.response {"</p>
#    <h3>Guru Meditation:</h3>
#    <p>XID: "} req.xid {"</p>
#    <address><a href="http://www.chinarenservice.com/">Shutter</a></address>
#  </body>
#</html>
#"};
###   deliver;
#}
```

### 2、1.2 配置文件内容参考：

```
backend server235 {
       set backend.host = "192.168.2.235";
       set backend.port = "80";
}

backend server237 {
       set backend.host = "192.168.2.237";
       set backend.port = "80";
}

backend server800 {
       set backend.host = "192.168.2.237";
       set backend.port = "800";
}

backend server98 {
       set backend.host = "192.168.2..98";
       set backend.port = "80";
}
sub vcl_recv {

        if (req.http.User-Agent ~ "(Baiduspider|google)") {
                        error 405 "Not allowed.";
        }

        if (req.http.Referer ~
"(baidu\.com|220\.181\.38\.82|7878758\.com|tom\.com|zj\.com|gougou\.com|soso\.com|7878758\.com|easyker\.co
m|wlyes\.com|sogou\.com|cn\.yahoo\.com|qianqian\.com|jq520\.cn|yupoo\.com|202\.108\.23\.172|192\.168\.1\.1
4)") {
                        error 405 "Not allowed.";
        }

             if (req.http.host ~ "^dl[1-6].idconline.cc") {
                 set req.backend = server237;
             } elsif (req.http.host ~ "^dl[8-9].idconline.cc") {
                 set req.backend = server235;
             } elsif (req.http.host ~ "^dl1[1-3].idconline.cc") {
                 set req.backend = server98;
             } elsif (req.http.host ~ "^dl10.idconline.cc") {
                 set req.backend = server235;
             } elsif (req.http.host ~ "^dl.idconline.cc") {
                 set req.backend = server237;
             } elsif (req.http.host ~ "^www.idconline.cc") {
                    set req.backend = server800;
                } else {
                    error 404 "Unknown virtual host";
                }
            }
                sub vcl_fetch {
                if (obj.ttl < 180s) {
                    set obj.ttl = 36000s;
                    insert;                             #插入缓存，关键。不然 Varnish 和源服务器交互会很多，缓
存命中率也会极低
                }
            }
         sub vcl_recv {
      if (req.request == "GET" && req.url ~
"\.(jpeg|png|amr|mid|mmf|mp3|3gp|avi|mp4|mpeg|rm|wmv|gif|jar|jad|sql|jpg|sis|nth|thm|utz|mtf|sdt|hme|tsk|z
ip|rar|sx|pxl|nes|cab|mpkg|mbm|app|exe)\?*.*")
       {
        lookup;
 }
           else {
         error 405 "Not allowed.";
                    }
}
```

3、1.1 配置文件内容参考：

```
backend www1 {
    set backend.host = " 192.168.0.1";
    set backend.port = "80";
}
backend www2 {
    set backend.host = " 192.168.0.3";
    set backend.port = "80";
}
backend www3 {
    set backend.host = " 192.168.0.3";
    set backend.port = "8080";
}
#可以做多个类似的后端
acl purge {
                    "localhost";
                    " 192.168.0.1";
                    " 192.168.0.2";
```

```
          }

sub vcl_recv {

    if (req.http.host ~ "www.chinarenservice.com") {
        set req.backend = www1;
    }
    else
    {
        if(req.http.host ~ "blog.chinarenservice.com"){
            set req.backend = www2;
        }
        else{
            error 200 "No cahce for this domain";
        }
    }

    if (req.request == "PURGE") {
        if (!client.ip ~ purge) {
            error 405 "Not allowed.";
        }
        else{
            lookup;
        }
    }

#####HTTP协议通常有三种方法，GET、HEAD和POST。而PURGE是一种由Squid作者定义的非HTTP官方方法，用来清除Squid缓存，我为了兼容Squid，也沿用Squid的PURGE方法来清除Varnish缓存。
##
#通过浏览器访问一个URL地址，发送的HTTP请求头是：
#GET http://www.abc.com/test.php
#当遇到GET或HEAD方法，Varnish会从缓存中返回网页：
#####当从浏览器提交表单时（请求头中的方法为 POST），需要透过 Varnish 将信息传递给后端 Web 服务器上的 PHP 程序处理，下面这几行表示如果请求头中的方法不是 GET 和 HEAD，则透过 Varnish 访问后端 Web 服务器.
    if (req.request != "GET" && req.request != "HEAD") {
        pipe;
    }

    if (req.http.Expect) {
        pipe;
    }

    if (req.http.Authenticate ){
        pass;
    }
   ##WEB 服务指明不能缓存的内容就不会被缓存了
    if (req.http.Cache-Control ~ "no-cache") {
        pass;
    }
    #对 ASP 或者 PHP 文件不缓存
    if(req.url ~ "\.asp" || req.url ~ "\.php" ){
        pass;
    }
    ### lookup 在缓存里查找被请求的对象，根据查找结果把控制权交给 vcl_hit 或 vcl_miss
    lookup;
}
####进入 pipe 模式时被调用。请求被直接发送到 backend，后端和客户端之间的后继数据不进行处理，只是简单传递，直到一方关闭连接。
sub vcl_pipe {
    pipe;
}
####进入 pass 模式时被调用。请求被送到后端，后端应答数据送给客户端，但不进入缓存。同一连接的后继请求正常处理。
sub vcl_pass {
    pass;
}
sub vcl_hash {
    set req.hash += req.url;
    set req.hash += req.http.host;
    #这里看情况，影响缓存命中率
    #set req.hash += req.http.cookie;
    hash;
}
####当遇到 PURGE 方法时，Varnishd 会 set obj.ttl = 0s;使某个 URL 的缓存失效，从而达到刷新 Varnish 缓存的目的。Varnish 配置了只接收并处理配置文件上面 acl 配置的 IP 发送 PURGE 请求.
sub vcl_hit {
    if (req.request == "PURGE") {
         set obj.ttl = 0s;
         error 200 "Purged.";
    }
##以前看 Varnish 配置，上面的这段只是从眼前经过，没有细想，今天看到一段话，才发现其中自有奥妙。
问题在于，既然已经 hit 了，说明这个内容已经是从缓存里 lookup 到的了，那为什么还有可能是“不能缓存”的呢？
##原来 Varnish 还真有“不能缓存”的对象在缓存里的情况：
##当客户端请求一个页面或其它什么东西，Varnish 在 lookup 了以后发现缓存里没有，于是就向后端请求。如果此时，在后端服务器有应答来之前，另一个客户端也向 Varnish 请求了这个东西，后来的请求将被挂起等待。当后端服务器的应答回来了，而这个内容是不可缓存的（貌似根据文档是说 HTTP 状态不是 200，203，300，301，302，404，410 中的任何一个，或者有 Expires 或者 Cache-Control 字段的时候，TTL 是 0 的），则会在缓存里插入一个标记为“不可缓存”的对象，被挂起的其他客户端将在 vcl_hit 里取得这个对象并被 pass 到后端。在处理对同一对象的并发请求时，能降低对后端服务器的压力。
```

```
    if (!obj.cacheable) {
        pass;
    }
    deliver;
}
#### lookup 后但没有找到缓存内容时调用，可以用于判断是否需要从后端服务器取内容。
sub vcl_miss {
    if (req.request == "PURGE") {
        error 404 "Not in cache.";
    }
    fetch;
}
sub vcl_fetch {
    if (!obj.valid) {
        error;
    }
    if (!obj.cacheable) {
        pass;
    }
    ##WEB 服务指明不能缓存的内容就不会被缓存了
    if (obj.http.Pragma ~ "no-cache" || obj.http.Cache-Control ~ "no-cache" || obj.http.Cache-Control ~
"private") {
        pass;
    }
    if (obj.ttl < 180s) {
        set obj.ttl = 180s;
    }
    ###将取到的内容插入缓存，然后发送给客户端，把控制权交给 vcl_deliver
    insert;
}
###缓存内容发动给客户端前调用。
sub vcl_deliver {
    deliver;
}
##在缓存内容到期前调用。
sub vcl_timeout {
    discard;
}
###由于到期或者空间不足而丢弃缓存内容时调用。
sub vcl_discard {
    discard;
}
```

## 六、　Varnish 启动：

1、 硬件文件启动方式：

```
/usr/local/varnish/sbin/varnishd -n /var/InfiNET/cache -f /usr/local/varnish/etc/shutter.vcl -a 0.0.0.0:81
-s file,/var/InfiNET/cache/varnish_cache.data,1G -g www -u www -w 30000,51200,10 -T 59.64.112.160:3500 -p
client_http11=on -P /usr/local/varnish/var/varnish.pid

/usr/local/varnish/bin/varnishncsa -n /var/InfiNET/cache -w /var/InfiNET/cache/varnish.log&
```

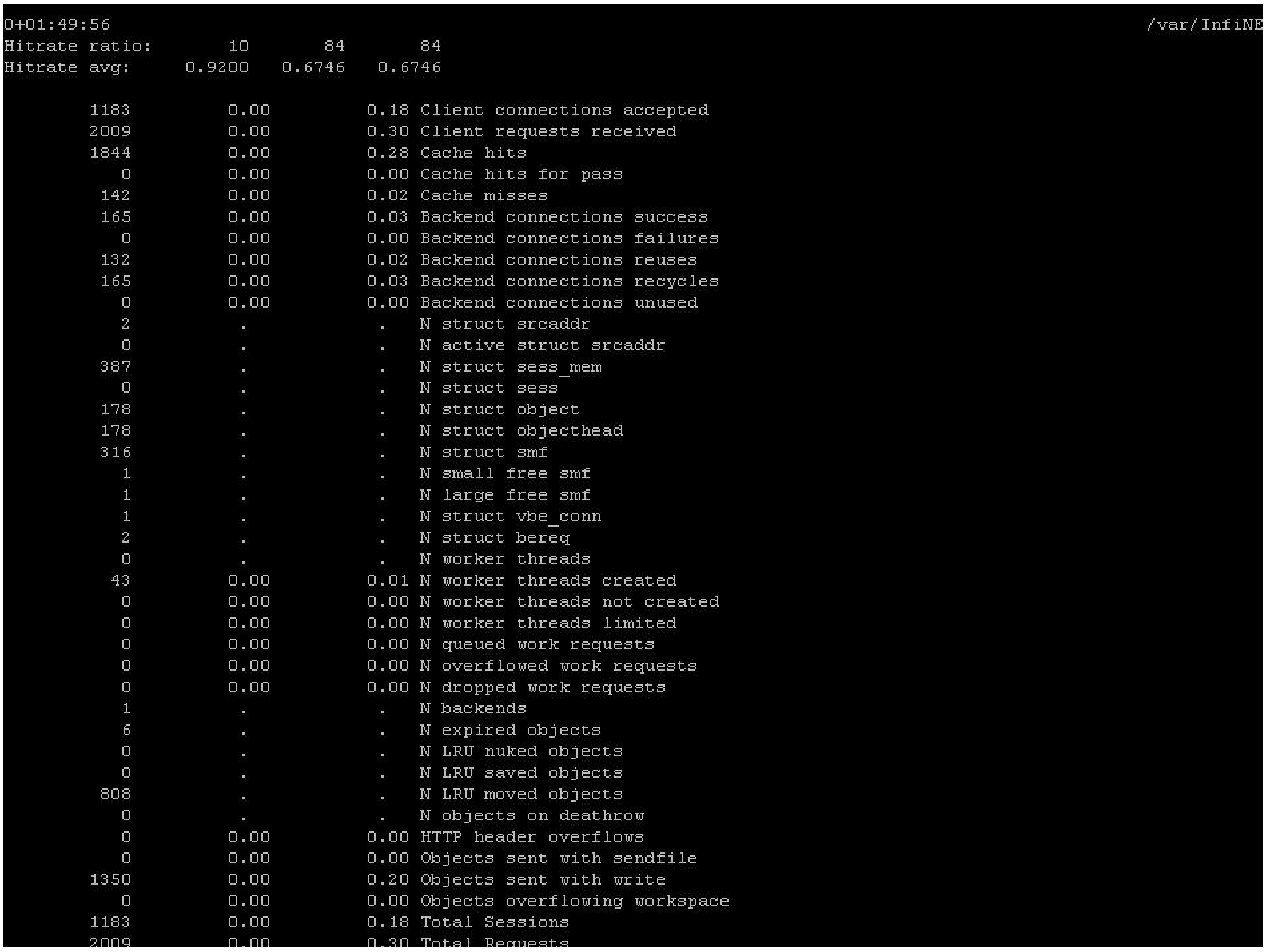
2、 SWAP 启动方式：

```
/usr/local/varnish/sbin/varnishd  -f /usr/local/varnish/etc/freeke.vcl -a 0.0.0.0:10086 -s malloc,20G -g www
-u www -w 30000,51200,10  -p client_http11=on -T 127.0.0.1:10087  -P /usr/local/varnish/var/varnishmalloc.pid
```

## 七、　　监控管理：

### 1、查看 Varnish 服务器连接数与命中率：

```
/usr/local/Varnish/bin/Varnishstat -n /usr/local/Varnisht/cache/
```



```
0+01:49:56                                                                                    /var/InfiNE
Hitrate ratio:        10        84        84
Hitrate avg:      0.9200    0.6746    0.6746

        1183          0.00         0.18 Client connections accepted
        2009          0.00         0.30 Client requests received
        1844          0.00         0.28 Cache hits
           0          0.00         0.00 Cache hits for pass
         142          0.00         0.02 Cache misses
         165          0.00         0.03 Backend connections success
           0          0.00         0.00 Backend connections failures
         132          0.00         0.02 Backend connections reuses
         165          0.00         0.03 Backend connections recycles
           0          0.00         0.00 Backend connections unused
           2            .            .  N struct srcaddr
           0            .            .  N active struct srcaddr
         387            .            .  N struct sess_mem
           0            .            .  N struct sess
         178            .            .  N struct object
         178            .            .  N struct objecthead
         316            .            .  N struct smf
           1            .            .  N small free smf
           1            .            .  N large free smf
           1            .            .  N struct vbe_conn
           2            .            .  N struct bereq
           0            .            .  N worker threads
          43          0.00         0.01 N worker threads created
           0          0.00         0.00 N worker threads not created
           0          0.00         0.00 N worker threads limited
           0          0.00         0.00 N queued work requests
           0          0.00         0.00 N overflowed work requests
           0          0.00         0.00 N dropped work requests
           1            .            .  N backends
           6            .            .  N expired objects
           0            .            .  N LRU nuked objects
           0            .            .  N LRU saved objects
         808            .            .  N LRU moved objects
           0            .            .  N objects on deathrow
           0          0.00         0.00 HTTP header overflows
           0          0.00         0.00 Objects sent with sendfile
        1350          0.00         0.20 Objects sent with write
           0          0.00         0.00 Objects overflowing workspace
        1183          0.00         0.18 Total Sessions
        2009          0.00         0.30 Total Requests
```

Cache hits 代表缓存命中情况

Cache misses 代表缓存非命中情况

2、通过 Varnish 管理端口进行管理：

用 help 看看可以使用哪些 Varnish 命令：

```
/usr/local/Varnish/bin/Varnishadm -T 127.0.0.1:5000 help
# /usr/local/Varnish/bin/Varnishadm -T 127.0.0.1:5000 help
help [command]
ping [timestamp]
status
start
stop
stats
vcl.load <configname> <filename>
vcl.inline <configname> <quoted_VCLstring>
vcl.use <configname>
vcl.discard <configname>
vcl.list
vcl.show <configname>
param.show [-l] [<param>]
param.set <param> <value>
quit
purge.url <regexp>
purge.hash <regexp>
purge.list
```

3、Varnish 中最常用的 Url 及 Referer

```
#/usr/local/Varnish/bin/Varnishtop -i rxurl
# /usr/local/Varnish/bin/Varnishtop -i rxheader -I Referer
```

4、通过 Varnish 管理端口，使用正则表达式批量清除缓存：

(1)、例：清除类似http://chinarenservice.com/a/index.html的URL地址）：

```
/usr/local/Varnish/bin/Varnishadm -T 127.0.0.1:5000 url.purge /a/
```

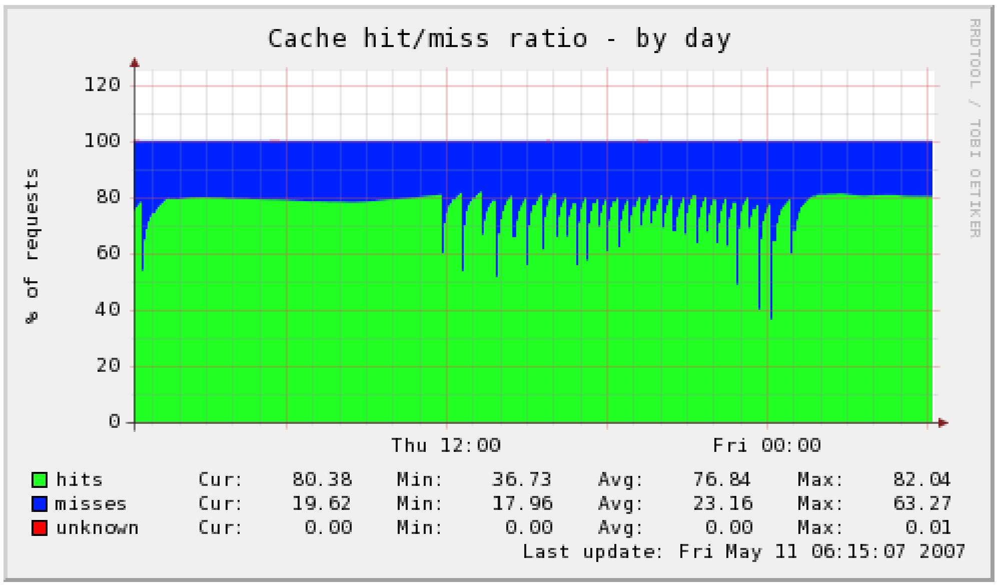
(2)、例：清除类似http://chinarenservice.com/tech的URL地址：

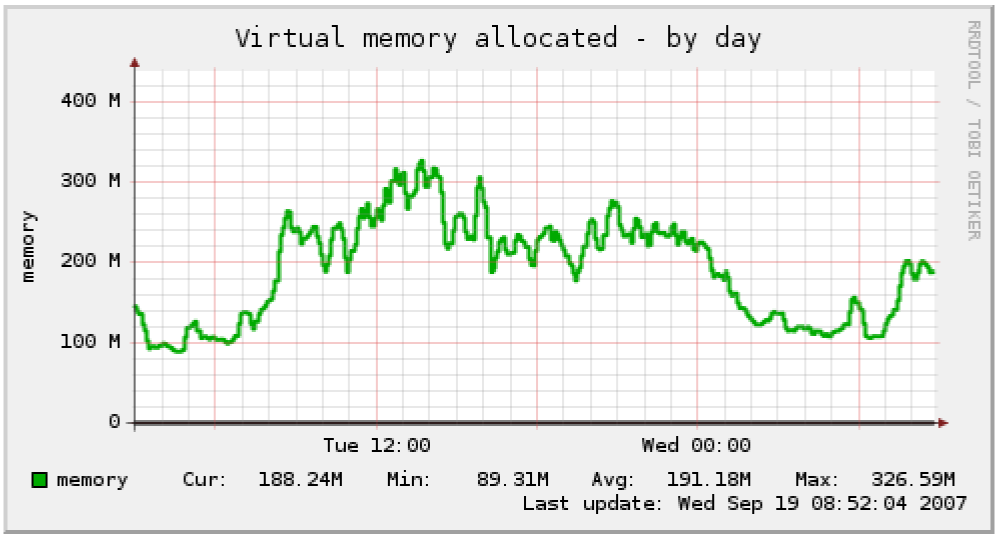
```
/usr/local/Varnish/bin/Varnishadm -T 127.0.0.1:5000 url.purge w*$
```

(3)、例：清除所有缓存：

```
/usr/local/Varnish/bin/Varnishadm -T 127.0.0.1:5000 url.purge *$
```

5、How To Use RRDtools Monitor For Varnish





## 八、　原理分析：

### 1、VCL man page 意译 （一）

VCL 语法比较简单，和 C 类似，if(){}的形式，=和==的区别，!、&&和||等等。但\符号没有特别的意思。VCL 里除了用==、!、&&、||做逻辑判断意外，还可以用~来表示与正则表达式或 ACL 的匹配。VCL 其实只是配置，并不是真正的编程语言，没有循环，没有自定义变量。

声明 Backend

```
backend 名称 {
set backend.host = "域名";
set backend.port = "端口";
}
```

比如

```
backend www {
set backend.host = "www.example.com";
set backend.port = "http";
}
```
声明的 Backend 可以用在判断请求针对哪个后端服务器
```
if (req.http.host ~ "^(www.)?example.com$") {
{
set req.backend = www;
}
```

声明 ACL
```
acl 名称 {
"IP";
"IP 子网"/反掩码位数;
! "IP 或 IP 子网"/反掩码位数;
}
```
比如
```
acl local {
"locahost"; /* myself */
"10.0.0.1"/8; /* and everyone on the local network */
! "10.0.0.23"; /* except for the dialin router */
}
```
判断 ACL 也很简单
```
if (client.ip ~ local) {
pipe;
}
```

还可以定义子程序
```
sub pipe_if_local {
if (client.ip ~ local) {
pipe;
}
}
```
用 call 来调用
```
call pipe_if_local;
```

内置的例程

vcl_recv

有请求到达后成功接收并分析时被调用，一般以以下几个关键字结束。

error code [reason] 返回 code 给客户端，并放弃处理该请求

pass 进入 pass 模式，把控制权交给 vcl_pass

pipe 进入 pipe 模式，把控制权交给 vcl_pipe

lookup 在缓存里查找被请求的对象，根据查找结果把控制权交给 vcl_hit 或 vcl_miss

vcl_pipe

进入 pipe 模式时被调用。请求被直接发送到 backend，后端和客户端之间的后继数据不进行处理，只是简单传递，直到一方关闭连接。一般以以下几个关键字结束。

error code [reason]
pipe

vcl_pass
进入 pass 模式时被调用。请求被送到后端，后端应答数据送给客户端，但不进入缓存。同一连接的后继请求正常处理。一般以以下几个关键字结束。
error code [reason]
pass

vcl_hash
目前不使用

vcl_hit
在 lookup 以后如果在 cache 中找到请求的内容事调用。一般以以下几个关键字结束。
error code [reason]
pass
deliver 将找到的内容发送给客户端，把控制权交给 vcl_deliver.

vcl_miss
lookup 后但没有找到缓存内容时调用，可以用于判断是否需要从后端服务器取内容。一般以以下几个关键字结束。
error code [reason]
pass
fetch 从后端取得请求的内容，把控制权交给 vcl_fetch.

vcl_fetch
从后端取得内容后调用。一般以以下几个关键字结束。
error code [reason]
pass
insert 将取到的内容插入缓存，然后发送给客户端，把控制权交给 vcl_deliver

vcl_deliver
缓存内容发动给客户端前调用。一般以以下几个关键字结束。
error code [reason]
deliver 内容发送给客户端

vcl_timeout
在缓存内容到期前调用。一般以以下几个关键字结束。
fetch 从后端取得该内容
discard 丢弃该内容

vcl_discard
由于到期或者空间不足而丢弃缓存内容时调用。一般以以下几个关键字结束。
discard 丢弃
keep 继续保留在缓存里

如果这些内置例程没有被定义，则执行缺省动作

一些内置的变量
now 当前时间，标准时间点（1970？）到现在的秒数

backend.host 后端的 IP 或主机名
backend.port 后端的服务名或端口

请求到达后有效的变量
client.ip 客户端 IP
server.ip 服务端 IP
req.request 请求类型，比如 GET 或者 HEAD 或者 POST
req.url 请求的 URL
req.proto 请求的 HTTP 版本号
req.backend 请求对应的后端
req.http.header 对应的 HTTP 头

往后段的请求时有效的变量
bereq.request 比如 GET 或 HEAD
bereq.url URL
bereq.proto 协议版本
bereq.http.header HTTP 头

从 cache 或后端取到内容后有效的变量
obj.proto HTTP 协议版本
obj.status HTTP 状态代码
obj.response HTTP 状态信息
obj.valid 是否有效的 HTTP 应答
obj.cacheable 是否可以缓存的内容，也就是说如果 HTTP 返回是 200、203、300、301、302、404、410 并且有非 0 的生存期，则为可缓存
obj.ttl 生存期，秒
obj.lastuse 上一次请求到现在间隔秒数

对客户端应答时有效的变量
resp.proto response 的 HTTP 版本
resp.status 回给客户端的 HTTP 状态代码
resp.response 回给客户端的 HTTP 状态信息
resp.http.header HTTP 头

变量可以通过 set 来赋值或通过 remove 来删除（清空）

```
sub vcl_recv {
if (req.http.host ~ "^(www.)?example.com$") {
set req.http.host = "www.example.com";
}
}

sub vcl_fetch {
remove obj.http.Set-Cookie;
}
```

2、VCL man page 意译 （二）

**名称**: VCL - Varnish Configuration Language Varnish 配置语言

**描述**:这里没啥意思，如果大家感兴趣的话安装完 Varnish 以后 man 7 vcl

**语法**:

VCL 的语法非常简单,语法跟 C 和 perl 有点相似.每个指令是以分号结束,同时可以按照你自己的参数来选择跟 C,C++,perl 相同的注释语法.VCL 不但有跟 C 语言类似的指定运算符(=),比较运算符(==),逻辑运算符(!,&&,!!)以外，vcl 还支持正则表达样和用~进行的 ACL 匹配运算.

不像 C 和 Perl，反斜杠(\)字符在 VCL 中没有什么特别的含义,VCL 使用%xx 这个代表 URLS 的排除结构.

可以使用 set 这样的关键字引进分配,他们并非用户定义的变量,这个值可以仅分配给附加给 backend（后端）请求或者文本类的变量.大多数的变量需要指定,这些值必须有同样单元后缀.

VCL 有 if,但是没有循环语句。

用 include 语句可以指定其他的 VCL 文件并且包含再当前 VCL 配置中.

**后端声明：**

后端声明创建和初始化一个以 backend 命名的类：

```
backend www {
      .host = "www.example.com";
      .port = "http";
   }
```

后端类也能再请求的时候选择一个后端:

```
if (req.http.host ~ "^(www.)?example.com$") {
      set req.backend = www;
   }
```

Directors

指导不同的后端是基于有一个在线的健康状态的.他们现在存在全的随机的指引方式.

定义:

```
director b2 random {
      .retries = 5;
      {
          /* We can refer to named backends */
```

```
                .backend        = b1;
                .weight         = 7;
            }
             {
                /* Or define them inline */
                .backend        = {
                    .host = "fs2";
                }
                .weight         = 3;
             }
         }
```

**随机指导**

随机指导有一个.retries 的参数,这个是指定有多少个个尝试找到一个工作的后端.默认和前面定义的后端数目相同。

另外一个选择是.weight 这个参数指定了一部分的流量发送到指定的后端上.

**后端探针**

后端可以被检测他们是否是健康的状态,它使用 req.backend.healthy 来检测返回的状态..windows 检测最后有多少 polls.

```
         backend www {
             .host = "www.example.com";
             .port = "http";
             .probe = {
                 .url = "/test.jpg";
                 .timeout = 0.3 s;
                 .window = 8;
                 .threshold = 3;
             }
         }
```

同时也能指定一个http的请求

```
   backend www {
             .host = "www.example.com";
             .port = "http";
             .probe = {
                 # NB: \r\n automatically inserted after each string!
                 .request =
                       "GET / HTTP/1.1"
                     "Host: www.foo.bar"
                     "Connection: close";
             }
         }
```

ACLS

3、Varnish 源代码分析的一些总结

yaoweibin2008@163.com
2008-11-21 http://yaoweibin2008.blog.163.com/blog/

看了两个星期 Varnish2.0.1 的源代码，作一些总结，给那些将要分析其代码的朋友一些方便：

**1)、Varnish 的总体结构**

Varnish 主要有两个进程：管理进程和 cache 子进程。cache 子进程又包含命令行接受处理线程（CLI_Run），放牧线程（wrk_herder_thread），放牧超时线程（wrk_herdtimer_thread），请求接受线程（vca_acct），数据接受线程（vca_main），很多工作线程（wrk_thread），HTTP 对象超时线程（exp_timer），后台服务器连接探测线程（vbp_wrk_poll_backend）。

**2)、Varnish 各进程线程的作用**

管理进程的行为主要包括：读入命令参数，并做出相应配置；编译 VCL 配置文件，生成 C 语言代码以后编译成动态连接库，由子进程载入并使用；产生子进程，并能处理各种子进程的信号；处理通过 CLI 接口传过来的命令，做出相应决定。

Cache 子进程处理所有具体工作，各个线程的任务包括：

命令行接受处理线程（CLI_Run）接受从管理进程通过管道传过来的命令，做出相应决定。其中初始时由管理进程默认产生三个命令(vcl.load、vcl.use、start）来启动后台服务器连接探测线程 和两个接受线程。

放牧线程（wrk_herder_thread）用于产生工作线程池。线程不足时会增加线程池。

放牧超时检查线程（wrk_herdtimer_thread）清理一些工作超时的工作线程。

请求接受线程（vca_acct）接受 HTTP 初次请求，并叫醒某个工作线程，处理请求。

数据接受线程（vca_main）在发送数据以后，继续可能的再次请求，并把请求交给工作线程。

工作线程（wrk_thread）不断处理请求，进入状态机。如果缓存没有命中，还需要从后台服务取过数据，存入缓存并回复。然后把该连接通过管道转给数据接受线程并睡去。

HTTP 对象超时检查线程（exp_timer）检查二叉堆中 HTTP 超时对象，删除之。

后台服务器连接探测线程（vbp_wrk_poll_backend）针对不同的后台服务器组进行轮询，检查存活与否。

**3)、几个碰到难点**

a、后台服务器连接探测线程是在“vcl->conf->init_func(cli);”被调用的，但是该函数的定义在源代码中并未出现，都是在 VCL 的编译文档中出现的。

b、CLI 接口是程序启动入口，比如后台服务器连接探测线程是由命令 vcl.load 产生的，
两个接收线程是 start 命令产生的。

c、 FreeBSD 的尾队列是 Varnish 用到的基本数据结构。特别要注意的是，尾队列成员入口的 prev 是双重指针，指向上一个成员的 next 指针。这样做主要为了通用，即使 element 成员是不同类型也可以组成链表。这与我们一般教科书上的 prev 操作不一样。

所以其 last 和 prev 的宏定义也颇为让人费解：

```
#define VTAILQ_LAST(head, headname) \
(*(((struct headname *)((head)->vtqh_last))->vtqh_last))
#define VTAILQ_PREV(elm, headname, field) \
(*(((struct headname *)((elm)->field.vtqe_prev))->vtqh_last))
```

注意尾队列的 head 和 entry 结构体的定义是一样的，所以在 element 类型是一样的时候，VTAILQ_LAST(head, headname) 也等于*(struct type *)head->vtqh_last->vtqe_prev， 相应的 VTAILQ_PREV(elm, headname, field)也等于*(struct type *)elm->field.vtqe_prev->vtqe_prev。FreeBSD 写成这样诡异的形式据说是为了可以方便得删除不同类型的 element。

**4)、Varnish 源代码中一些特定的缩写字母含义：**

BH:Binary Heap
CLI:Command-Line Interface,part of the published Varnish-API,see "cli.h"
CNT:CeNTer
EVB:Event Variable Base
EXP:EXPire

HCL:Hash CLassic
HTC:HTtp Connection
MCF:Main ConFigure
mgt:managment
PAN:PANic
PFD:PoolFD, Poll File Description
SES:SESsion
SMF:Storage Mmaped File
SMS:Storage Mutex Synth
STV:STeVedore
TMO:TiMe Out
VBE:Varnish BackEnd
VBM:Varnish Bit Map
VBP:Varnish Backend Polling
VCA:Varnish Cache Acceptor
VCC:Varnish Configure Compile?
VCL:Varnish Configure Language
VCT:Varnish Character Type
VDI:Varnish DIrector
VEV:Varnish Event Variable
VLU:Varnish Line Up
VPF:Varnish PID File
VSB:Varnish Storage Buffer
VSS:Varnish addreSS?
VSL:Varnish Share-memory Log
VTAILQ:Varnish Tail Queue,
WQ:Work Queue
WRK:WoRKer
WSL:Worker Share-emory Log

## 4、新浪开发人员代码分析资料：

- Varnish 总体架构：

Varnish 介绍
1 Varnish is HTTP accelerator.
2 Varnish stores data in virtual memory and leaves the task of deciding what is stored in memory and what gets paged out to disk to the operating system
3 The Varnish web site claims that Varnish is ten to twenty times faster than the popular Squid cache on the same hardware.
4 Varnish is heavily threaded
Varnish 总体架构
1.1 总体流程
主进程 fork 子进程，主进程等待子进程的信号，子进程退出后，主进程重新启动子进程
子进程生成若干线程。
　Accept 线程：接受请求，将请求挂在 overflow 对列上

Work 线程： 多个，从对列上摘除请求，对请求进行处理，直到完成，然后处理下一个请求

Epoll 线程： 一个请求处理称作一个 sesion，在 sesion 周期内，处理完请求后，会交给 Epoll 处理，监听是否还有事件发生。

Expire 线程：对于缓存的对象，根据过期时间，组织成二叉堆，该线程周期检查该堆的根，处理过期的文件。

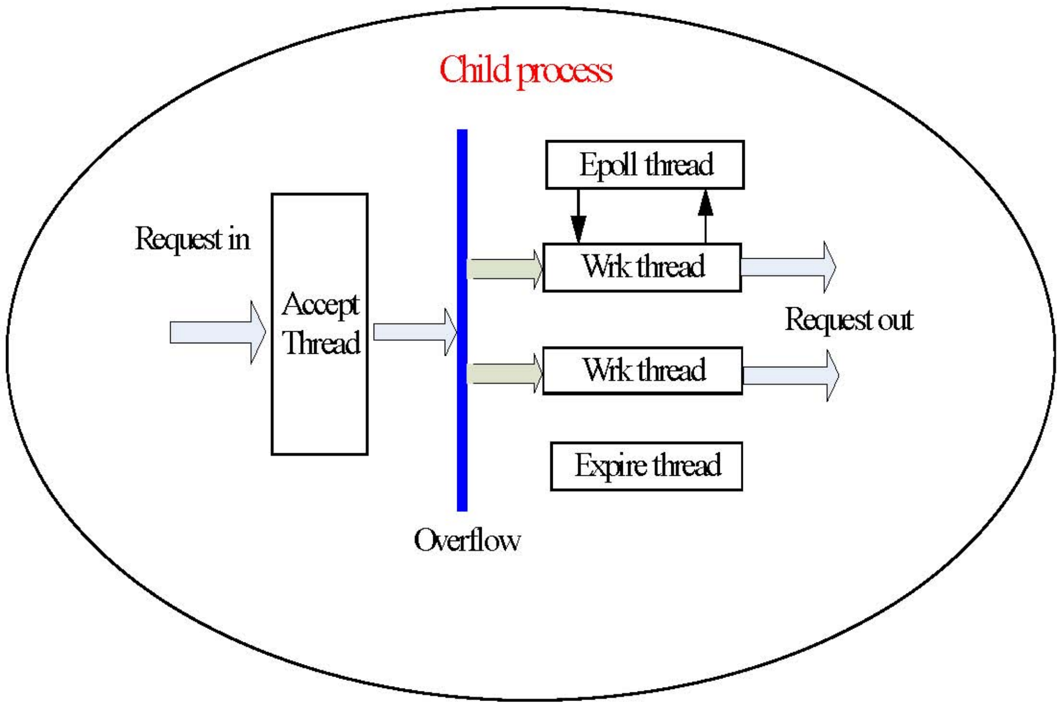


图 1-1 Varnish 总体架构简化图

线程之间的关系：

1.1.1 accept 线程

监听端口，接受连接。

接受后组织成 struct ses（session 结构），看是否有空闲的工作线程，如果有，将请求给它，pthread_cond_signal 信号通知它没有空闲线程，如果 overflow 过大，则放弃该请求。否则，将其挂在 overflow 上（需要更多工作线程，发通知）。继续监听

1.1.2 work 线程

从 overflow 队列上摘取请求（struct ses），进入状态机处理，处理结束后，通过 pipe 通信，将 struct ses 发送给 epoll 线程。

1.1.3 Epoll 线程,得到传过来的 struct ses，若还没有过期，将 socket 放入 epoll 的事件中，事件发生时，也会将其放入到 overflow 中进行。

关于 Expire thread，比较独立，下面专门介绍。

**1.2 work 线程的处理过程**

1.2.1 请求的处理过程称为 session，主要是由 work 线程处理的。

请求的是通过进入状态转换机进行分步处理,通过 Varnish Configuration Language（VCL）进行定制。

request 进入状态机后的状态变化

对于每种状态，都可以通过 VCL 进行配置，丰富功能。

状态的基本转换如下图所示：

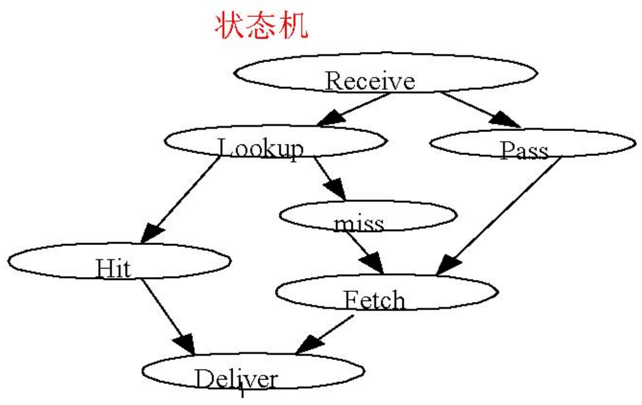
Child process 状态机－通过链表解决冲突 大文件

图 2-2 http 请求处理状态图

Work 线程处理请求的过程是根据 VCL 的配置而定制的状态机，典型的处理流程如下

1.
 Receive,请求处理的入口状态（之前还有 first 等状态），根据 VCL 判断该请求是 Pass（跳过）还是进行 Lookup(本地查询)
2.
 Lookup，在 hash 表中查找数据，若找到则进入 hit 状态，否则进入 fetch 状态。
3.
 Pass， 选择后台，进入 fetch 状态
4.
 Fetch，对请求进行后端的获取，发送请求，获得数据，并进行本地的存储
5.
 Deliver,，将数据发送给客户端，然后进入 done
6.
 Done，处理结束事宜，对于一些请求需要做重新处理则可能重新进行状态转换或交给　epoll

1.2.2 Work 线程总体工作如下：
接受到请求，按状态机处理，请求结束后，关闭连接或交给 Epoll
重新取请求，若没有请求，挂入空闲队列，等待信号唤醒（pthread_cond）
唤醒它有两个途径，除了前面说的 accept 线程外，还有就是 herdtimer 线程
如果是 accept 唤醒的，则继续按照状态机的方式处理请求，如果是 herdtimer 唤醒的，则自杀
**1.3 工作线程的管理**
1.3.1 Herd 线程
－根据配置生成指定数目的线程（min）
－动态检查线程数目，生成需要的线程
1.3.2 Herdtimer 线程
定期检查空闲的线程，对于空闲超过指定时间的线程，通知它可以自杀
工作线程管理的目的是根据请求的数量动态的调整工作线程的数目
**1.4 expire 线程**
对缓存的数据采用二叉堆的方式进行组织，线程检测堆的 root，判断是否过期，对过期的数据进行删除或重取，由 VCL 设置。
对于过期的数据，如果需要重新取，则会调用状态机中的 fetch 去后台获取，然后更新



● Cache 详解：
2.1 Hash 方式

2.1.1 简单 hash 方式
－单一链表，按 key 大小排序，通过 memcmp 比较查找和添加
－缺点：查询效率低
2.1.2 Hash classic
－第一层 hash backet（较大的素数）包含锁
－采用 CRC32 方法，key 可配，一般是 url + host
－优点：查询较快
－值得参考的地方：采用查找和添加分两遍进行

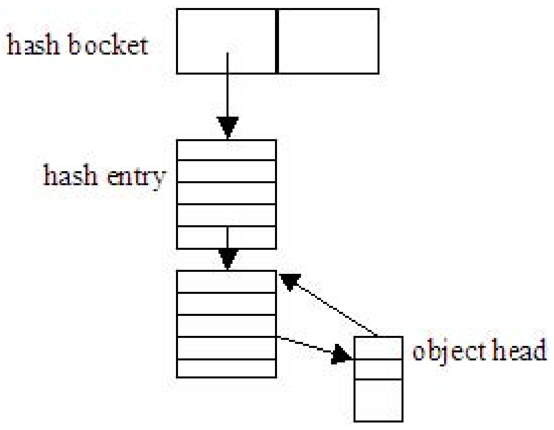


图 2-1 hash 结构图
2.2 Storage 方式
2.2.1 Malloc
－通过 malloc 获取内存
－通过 free 释放
－－特点：简单
－有什么不好呢？
2.2.2 Mmap file
创建大文件，通过二分法分段映射成 1G 以内的大块

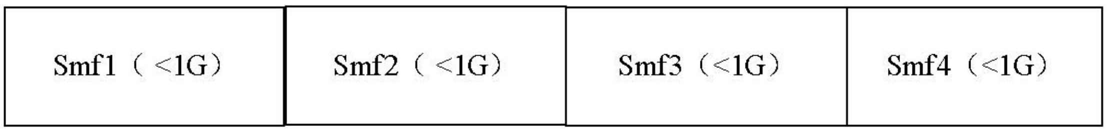


图 2-2 hash classic 存储信息图
数据的初始化
A 初始化时，将大文件分段进行 mmap，每段大小在 1G 以内，映射好的段分配到 free block 数组链表中，数组下标便是页的倍数向下取整，如果块大于数组倒数第二个元素与页的乘积，则将该块连接到数组的最后一个元素的链表中。
B 分配，遍历数组，找到满足要求的空闲块，若是前 B-2 个没有，则从最后一个中满足要求的大块中切出一块。如果找出的块大于需要的容量，则就对其进行拆分，然后将剩下的插入到空闲块中。
C 回收，对于释放的块，看能否和相邻的块进行合并，如果可以，则合并后再重新插入到合适位置。
2.3 数据输出
Object 结构表示一个请求对象（文件），通过其 store 链表指出数据块信息
2.3.1 采用 writev

－将 store 链表上的数据组成 iov，通过 writev 输出
2.3.2 采用 sendfile
－通过循环使用 sendfile，将 store 链表中的数据输出

● VCL 配置：

通过 vcl 脚本对程序进行定制，主要是对请求的定制处理，如过滤某些请求等，脚本配置生成的函数是嵌套在状态机中的。默认的配置如下
http://Varnish.projects.linpro.no/browser/trunk/Varnish-cache/bin/Varnishd/default.vcl
purge 删除配置如下
http://Varnish.projects.linpro.no/wiki/VCLExamplePurging

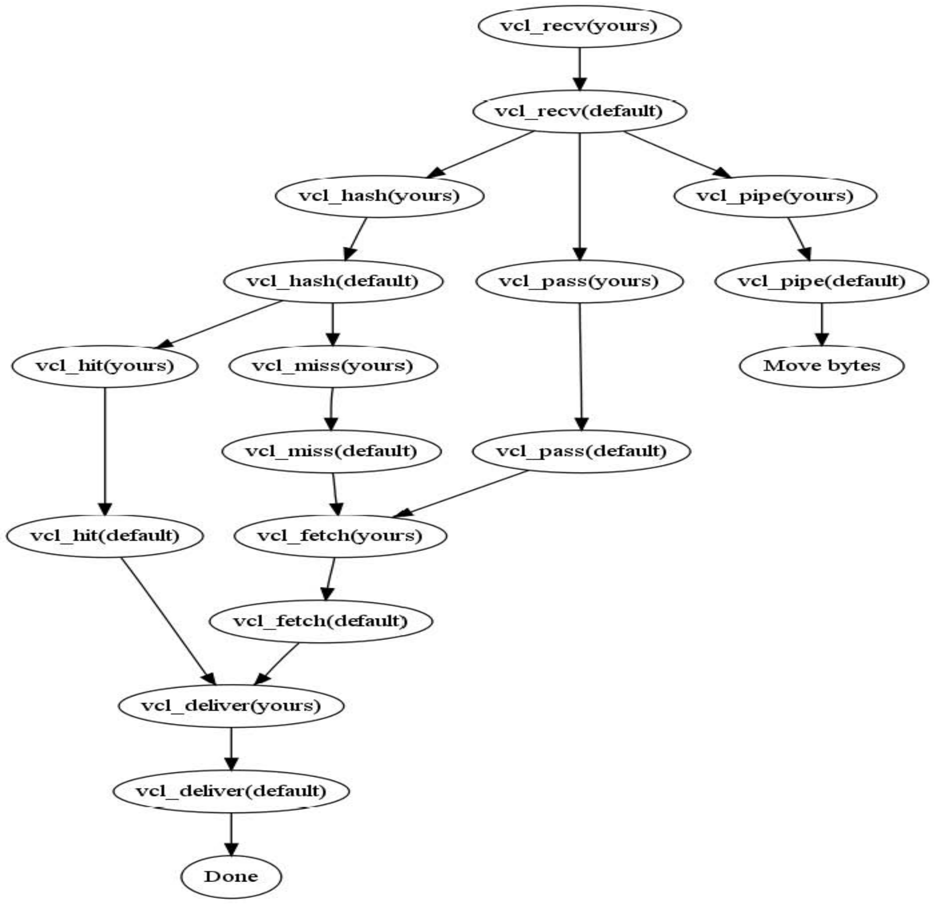


图 3-1 vcl 与状态机的关系
分析与总结
Varnish 比较轻便，总共的代码量不大，功能上有待丰富和加强。
1.
•利用虚拟内存方式，io 性能好•状态机设计巧妙，结构清晰•利用二叉堆管理缓存文件，达到积极删除目的
4.
•VCL 比较灵活
5.
•强大的管理功能，top，stat，admin，list 等
6.
•是内存缓存，重启数据消失

7.
•32 位机器上文件大小为 2G

讨论
1.
二叉堆方式的插入和删除对于缓存文件较多时，性能是不是影响较大
2.
这么多的线程，分工清晰，如 epoll，expire，herd，herdtimer 等对性能的影响？
3.
Hash 中的 key 保存完整的 url 和 host，信息量是不是太大？优点是：信息全，可重新组成请求用于过期的重取

参考
1 http://Varnish.projects.linpro.no/
2 http://en.wikipedia.org/wiki/Varnish_cache
3http://en.wikipedia.org/wiki/Virtual_memory
4http://en.wikipedia.org/wiki/Squid_cache

## 九、 FAQ:

### 1、Multiple backends with Varnish

Varnish has been able to provide caching for more than one backend for quite some time. The achilles heel with this has up until now been that it hasn't been able to determine whether a backend is healthy or not. This is now a problem of the past! The backend health polling code is available in 2.0 beta1 Sadly it had a bug, so when using the 'random' director, it was unable to use the remaining healthy backend if all but one went MIA. I reported this bug and it was fixed in changeset r3174.

So as of now, you can **safely** use one Varnish instance for several front-ends, thus eliminate double-caching (memory waste, unnecessary load on back-ends), reduce network traffic, do rudimentary load balancing, ease management etc.

With the obscene amount of traffic Varnish can push without putting a fairly basic system under any load worth mentioning, you can use a single front-end to serve several nodes in most setups.

Here's an elementary sample VCL for how to do this:

```
backend node0 {
  .host = "127.0.0.1";
  .port = "80";
  .probe = {
          .url = "/";
          .timeout = 50 ms;
          .interval = 1s;
          .window = 10;
          .threshold = 8;

  }
}

backend node1 {
  .host = "10.0.0.2";
  .port = "80";
  .probe = {
#          .url = "/";
          .timeout = 100 ms;
          .interval = 1s;
          .window = 10;
          .threshold = 8;
       .request =
          "GET /healthcheck.php HTTP/1.1"
          "Host: 10.0.0.2"
          "Connection: close"
          "Accept-Encoding: foo/bar" ;
  }
}
director cl1 random {
    { .backend = node0; .weight = 1; }
    { .backend = node1; .weight = 1; }
}

#director cl1 round-robin {
#   { .backend = node1; }
#   { .backend = node0; }
#}

sub vcl_recv {
        set req.backend = cl1;
}
```

As you can see I’m defining the backends slightly differently. You need to define one of .url or .request, but not both for obvious reasons. If you go for the slighly simpler .url the default request looks like this:

```
GET / HTTP/1.1
Host: something
Connection: close
```

If this does not suit your need, comment out .url and use .request to roll your own. This aspect of Varnish is actually quite well documented, so I won’t repeat what’s on the trac page.

There is clearly a lot more you can and, more often than not, should do in the VCL than the above. This is a stripped down version which only pertains to the backend polling functionality.

## 2、configuring Varnish to use custom http headers

By default caches uses the field “Cache-Control” to get the TTL for objects they care about. In case of a server side cache the parameter “s-maxage” is responsible for the life time of an object. (see rfc2616). This is fine for almost all cases. The bad thing about this is a bad configured cache proxy also uses this parameter to get it’s caching informations. This causes the backend to have no control to invalidate such an object because it will remain in memory of such a bad configured cache.

So we decided to use a custom field in the http header to tell Varnish how long an object should stay in cache. In the backend it was very easy to implement a few lines of code to set the custom field. I called it “Varnish-Control”. The field will contain an integer value which indicates the TTL in seconds. The main part of the work was to configure Varnish to use this value instead of s-maxage form the field “Cache-Control”.

In words i did the following:

*Once an item was fetched from the backend (so it was not found in the cache) I will set the TTL of the item and stuff it into the cache. Since the field “Varnish-Control” is useless for all request handling nodes on the way to the client i removed the field once the TTL was set.*

The first approach was pretty simple:

```
sub vcl_fetch {
if (obj.http.Varnish-Control) {
set obj.ttl = 7d;
remove obj.http.Varnish-Control;
}
insert;
}
```

Unfortunately this solution did not care about the value I’ve set into “Varnish-Control” but uses a static value of seven days to cache items. This was not satisfying. So i tried to make it more dynamic. Since it is not possible to assign the value from obj.http.Varnish-Control to obj.ttl i had to use inline C code. This is done by using C{ … }C somewhere in the VCL file.

```
sub vcl_fetch {
if (obj.http.Varnish-Control) {
C{
char *ttl;
ttl = VRT_GetHdr(sp, HDR_OBJ, "\020Varnish-Control:");
VRT_l_obj_ttl(sp, atoi(ttl));
}C
remove obj.http.Varnish-Control;
}
insert;
}
```

Finding the right commands to access the header informations and write them into the TTL of the object was PITA.

Anyway, I try to explain this two commands:

`char *VRT_GetHdr(const struct sess *sp, enum gethdr_e where, const char *n)`

- The first parameter is the session pointer…nothing more to say about this.

- The second parameter is a constant indicating from which object will be used to get the header informations. In this case it was “HDR_OBJ” which is equivalent to “obj” in plain vcl configuration scenarios. possible values are: HDR_REQ, HDR_RESP, HDR_OBJ and HDR_BEREQ.
- The third parameter is the name of the field i like to access. NOTE: The field name has to be suffixed with a colon and it has to be prefixed with the length of the field name (including the colon). This value has to be noted octal.

void VRT_l_obj_ttl(const struct sess *, double)

- The first parameter is again the session pointer.
- The second parameter is the TTL as double.

Thats it!

All VRT_x commands will be available in this context. This commands can be found in Varnish-cache/include/vrt.h and Varnish-cache/include/vrt_obj.h

## 3、Varnish 的一个图片缓存应用

发现 Varnish 2.0 beta1，在同一 URL 的文件有多份缓存时，发送 PURGE 请求总是返回 404

所以必须杜绝这样的情况发生，做法如下

1. 防止不同浏览器的 Accept-Encoding 请求头不同，造成多份缓存：

```
if (req.http.Accept-Encoding) {
        remove req.http.Accept-Encoding;
}
```

因为是缓存图片，所以可以写得像上面这样简单，直接把 Accept-Encoding 头去掉

2. 去掉 URL 中带”?”的参数：

```
set req.url = regsub(req.url, “\?.*”, “”);
```

顺便，加上简单的防盗链：

```
if (!req.http.referer ~ “^(.*)?example.com(.*)$”) {
    error 403 “Forbidden”;
}
```

http://www.sigmablog.cn/?p=271

## 4、Varnish linux 内核

Edit /etc/sysctl.conf

These are numbers from a highly loaded Varnishe serving about 4000-8000 req/s

```
net.ipv4.ip_local_port_range = 1024 65536

net.core.rmem_max=16777216

net.core.wmem_max=16777216

net.ipv4.tcp_rmem=4096 87380 16777216

net.ipv4.tcp_wmem=4096 65536 16777216

net.ipv4.tcp_fin_timeout = 3

net.ipv4.tcp_tw_recycle = 1

net.core.netdev_max_backlog = 30000
net.ipv4.tcp_no_metrics_save=1

net.core.somaxconn = 262144

net.ipv4.tcp_syncookies = 0

net.ipv4.tcp_max_orphans = 262144

net.ipv4.tcp_max_syn_backlog = 262144

net.ipv4.tcp_synack_retries = 2

net.ipv4.tcp_syn_retries = 2
```

## 5、 Varnish 和 x-forwarded-for

BBS 提出来说显示发帖 IP 不正常，检查原来 Varnish 在某些往后 pass 或 pipe 的情况下会加上一个 x-forwarded-for 头，而这个头的内容竟然是 127.0.0.1，怀疑是 Varnish 内部通信或者什么其他原因造成的。
本来没什么，可是 Varnish 却很霸道，不管原来有没有 x-forwarded-for 头，都给加上一个。
没办法了，只好修改 VCL，加个判断并修改 x-forwarded-for，在 pass 和 pipe 的时候处理一下 HTTP 头。

```
        if (bereq.http.x-forwarded-for) {
                set bereq.http.X-forwarded-for = bereq.http.X-forwarded-for ", " regsub(client.ip, ":.*", "");
        } else {
                set bereq.http.X-forwarded-for = regsub(client.ip, ":.*", "");
        }
```

这样一来，多次转发的 IP 信息以 ip1, ip2, ip3 的形式记录在 x-forwarded-for 里。
另外一个办法就是设置一个 127.0.0.1 的 ACL，然后匹配到的话就使用原头部信息。

```
       if (client.ip ~ locall)
       {
               set bereq.http.X-forwarded-for = bereq.http.X-forwarded-for;
       } else {
               set bereq.http.X-forwarded-for = regsub(client.ip, ":.*", "");
       }
```

## 6、 Varnish 2.0 load balancing

Varnish 是一个高效的 HTTP accelerator，目前正在开发的 2.0 版本有一些非常有意思的改进，最重要的就是对 load balancing 的支持。 从目前的 trunk 里面可以看到，其功能已经基本实现，目前支持 round robin 和 weighted random 两种算法对后端进行负载均衡，同时对后端服务器也有简单的健康检查机制。负载均衡的一组后端在 vcl 中也是一个 backend, 不过有较特殊的定义语法。

```
backend_round_robin rr {
set backend.set = {
{ "bbs1.chinarenservice.com", "http" }
{ "bbs2.chinarenservice.com", "http" }
{ "bbs3.chinarenservice.com", "http" }
};
}
```

这就定义了一个名字为 rr 的 backend, 它对应的是一组使用 round robin 算法负载均衡的后端

```
backend_random rrr {
set backend.set = {
{ ″bbs1.chinarenservice.com″, ″http″, 0.3 }
{ ″bbs2.chinarenservice.com″, ″http″, 0.6 }
{ ″bbs3.chinarenservice.com″, ″http″, 0.1 }
};
}
```

这则是定义了一个名字为 rrr 的 backend，它对应的是一组使用 weighted random 算法负载均衡的后端，花括号中的小数是每个后端的权重，可选，默认是每个后端分配相同的权重。
具体参见：#1931
找时间测了一下，功能已经基本可以用，不过测试的过程中 Varnish 经常当掉，XD
http://blog.jianqing.net/2007/09/Varnish-2-load-balancing

```
backend bbsas67 {
  .host = "172.28.3.69";
  .port = "80";
  .probe = {
          .url = "/";
          .timeout = 50 ms;
          .interval = 1s;
          .window = 10;
          .threshold = 8;
  }
}

backend bbsas60 {
  .host = "172.28.3.62";
  .port = "80";
  .probe = {
          .url = "/";
          .timeout = 100 ms;
          .interval = 1s;
          .window = 10;
          .threshold = 8;
       .request =
           "GET /healthcheck.php HTTP/1.1"
           "Host: 10.0.0.2"
           "Connection: close"
           "Accept-Encoding: foo/bar" ;
  }
}
director bbs random {
    { .backend = bbsas67; .weight = 1; }
    { .backend = bbsas60; .weight = 1; }
}

#director bbs round-robin {
#   { .backend = bbsas60; }
#   { .backend = bbsas67; }
#}
#director bbs round-robin {
#     { .backend = { .host = "172.28.3.62"; .port = "http"; } }
#     { .backend = { .host = "172.28.3.69"; .port = "http"; } }
# }
#bbs_director
```

```
    if (req.http.host ~"^www.chinarenservice.com") {
        set req.backend = shutter;
        } elsif (req.http.host ~"^chinarenservice.com") {
            set req.backend = shutter;
        } elsif (req.http.host ~"^www.ichinarenservice.cn") {
            set req.backend = ikuaimen;
        } elsif (req.http.host ~"^www.ichinarenservice.cn") {
            set req.backend = ikuaimen;
        } elsif (req.http.host ~"^www.ichinarenservice.net") {
            set req.backend = ikuaimen;
        } elsif (req.http.host ~"^www.ichinarenservice.org") {
            set req.backend = ikuaimen;
        } elsif (req.http.host ~"^www.1chinarenservice.cn") {
            set req.backend = ikuaimen;
        } elsif (req.http.host ~"^www.lchinarenservice.cn") {
            set req.backend = ikuaimen;

        } elsif (req.http.host ~"^static.ichinarenservice.cn") {
            set req.backend = sns;

        } elsif (req.http.host ~"^bbs.chinarenservice.com") {
            set req.backend = bbs;


       } else {
              error 404 "Unknown virtual host";
       }
```

7、 日志处理：

- Varnish 日志的 rotate

#touch /etc/logrotate.d/Varnish

```
/usr/local/Varnish/logs/Varnish.log {
   Daily
   rotate 60
   copytruncate
    notifempty
    missingok
     Prerotate
     killall Varnishncsa
     endscript
      postrotate
     /usr/local/Varnish/bin/Varnishncsa -a -w /usr/local/Varnish/logs/Varnish.log &
       Endscript
     }
```

● cron 日志自动切割

```
cat Varnishlog.sh
DATE=$(date +%Y%m%d%H --date='1 hours ago')
cd  /usr/local/Varnish/
kill -9 $(ps aux | grep Varnishncsa | awk '{print $2}')
mv access.log backup.log
/usr/local/Varnish/bin/Varnishncsa -a -n /data/Varnish -w /usr/local/Varnish/logs/access.log &
cat backup.log | grep -v "^server" >access$DATE.log
rm -f backup.log
tar zcvf access$DATE.log.tar.gz access$DATE.log
rsync -vza --progress access$DATE.log.tar.gz log@192.168.2.222::227/squid --password-file=/etc/rsync.pass
find /log/Varnish -daystart -type f -mtime 7 -exec rm -f '{}' ';'
```

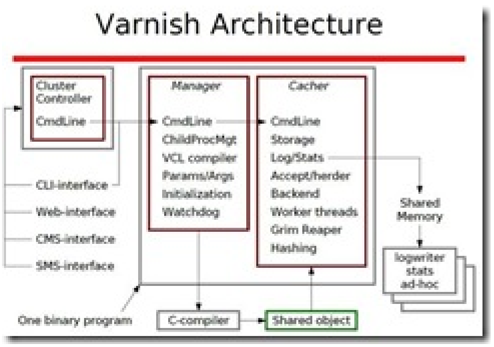
## 8、Scaling Apps with Varnish

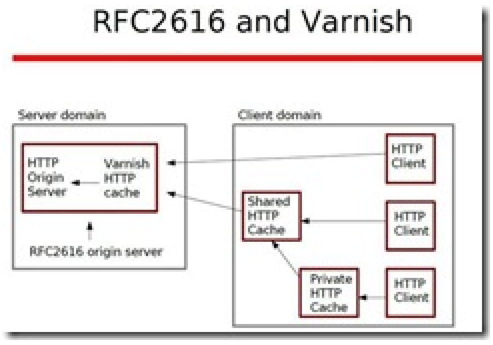
Varnish是一款高性能的反向代理和HTTP加速器，并非传说中Http Cache Server。
这是Varnish作者的 Varnish http accelerator文档，少不了PK Squid。
不过本人认为其在一定程度上还无法达到完全替换 Squid 的功效。保留意见于此：
1、没有 cache 检测机制，当内存满了无有效处理机制；
2、一但进程 Hang、Crash 或者重启，内存完全释放出 cache 内容，所有 requests 都回 origin servers；
3、高并发状态下 CPU、IO、内存等开销均高于 Squid
针对
1、 确实 Squid 在内存控制着实不尽如人意。cache_mem 设定理想的内存总量用于：In-Transit OBJECTs、Hot OBJECTs、Negative-Cached OBJECTs。 这些对象所对应的数据都存储在 4KB 的块里面。此参数指定理想中分配 4KB 块总数的最大限制，其中 In-Transit OBJECTs 具有最高的优先级。当输入数据需要额外空间 来存储的时候，negative-cached 和 hot OBJECTs 将被释放。配合上 disk 和 ram 之间的交互 cache_swap。在 Cache 达设置的峰值时会有按优先动作处理 cache object。这点姑且看作略胜于 Varnish

2、Varnish 基于 Kernel 和 Ram 的处理机制上比 Squid 成熟很多。进程挂了或 reboot 后，所有 Cache 内容将释放，Requests 将全部回到 Original servers。这点上 Squid 还有 Disk Cache 即通俗的 filesystem cache，尽管如 Varnish 作者表述的”It’s very slowly”，但总比直接去源端访问来的高效。估计 Varnish 设计中的 2 进程 – 是想表达 Automatic restarts on crash 的意思吧。
3、 不用多说了，在高并发情况下 Varnish 处理的请求远大于 Squid，系统 Load 会高于 Squid。这也是 Varnish 相比 Squid 高效便捷的一点。当然有利肯定需要承担相应的系统开销
总之 Squid 固然经典，但不妨碍 Varnish 作为一款性能卓越的 HTTP Accelerator
还是交给应用来选择吧。本人管理的 10 台 Squid 中应用相对稳定，暂不打算用 Varnish 来替换
其中一台 Cache info 如下

```
Cache information for squid:
Request Hit Ratios: 5min: 88.8%, 60min: 88.1%
Byte Hit Ratios: 5min: 66.7%, 60min: 65.5%
Request Memory Hit Ratios: 5min: 25.6%, 60min: 25.8%
Request Disk Hit Ratios: 5min: 1.8%, 60min: 1.8%
Storage Swap size: 75470884 KB
Storage Mem size: 524016 KB
Mean Object Size: 31.30 KB
Requests given to unlinkd: 6197478
```

Varnish成功应用在挪威最大的在线报纸http://www.vg.no/：
使用 3 台Varnish代替了原来的 12 台squid。参见Here
有关Varnish架构手记请参见Here





```
backend default {
    .host = ″x.com″;
    .port = ″80″;
    }
backend B1 {
    .host = "x.com";
    .port = "80";
    }
backend B2 {
    .host = "y.com";
    .port = "80";
    }
backend B3 {
    .host = "z.com";
    .port = "80";
    }
acl purge {
    "localhost";
    "127.0.0.1";
    "192.168.1.0"/24;
    "124.42.78.0"/32;
    }
sub vcl_recv {
    if (req.request == "PURGE") {
    if (!client.ip ~ purge) {
    error 405 "Not allowed.";
    }
    lookup;
    }
    if (req.http.host ~ "^(www.)?x.com$") {
               set req.http.host = "www.x.com";
              set req.backend = B1;
    } elsif (req.http.host ~ "^y.com$") {
             set req.http.host = "y.com";
             set req.backend = B2;
    } elsif (req.http.host ~ "^z.com$") {
             set req.http.host = "z.com";
           set req.backend = B3;
    } else {
              error 404 "NOT be cached";
    }
    if (req.request != "GET" &&
          req.request != "HEAD" &&
          req.request != "PUT" &&
          req.request != "POST" &&
          req.request != "TRACE" &&
          req.request != "OPTIONS" &&
          req.request != "DELETE") {
/* Non-RFC2616 or CONNECT which is weird. */
         pipe;
    }
    if (req.request != "GET" && req.request != "HEAD") {
/* We only deal with GET and HEAD by default */
    pass;
    }
    if (req.http.Authorization || req.http.Cookie) {
/* Not cacheable by default */
    pass;
    }
    if (req.request == "GET" && req.url ~ "\.(php)($|\?)") {
    pass;
    }
    lookup;
    }
sub vcl_pipe {
    pipe;
    }
sub vcl_pass {
    pass;
    }
#sub vcl_hash {
# set req.hash += req.url;
# if (req.http.host) {
# set req.hash += req.http.host;
# } else {
# set req.hash += server.ip;
# }
# hash;
#}
sub vcl_hit {
    if (req.request == "PURGE") {
    set obj.ttl = 0s;
    error 200 "Purged.";
    }
}
sub vcl_miss {
    if (req.request == "PURGE") {
    error 404 "Not in cache.";
    }
    }
sub vcl_fetch {
```

```
# if (!obj.cacheable) {
# pass;
# }
# if (obj.http.Set-Cookie) {
#remove obj.http.Set-Cookie;
# pass;
# }
# if (obj.http.Cache-Control ~ "no-cache" || obj.http.Cache-Control ~ "private") {
# pass;
# }
    if (req.request == "GET" && req.url ~ "\.(txt|js)$") {
    set obj.ttl = 3600s;
    }
    else {
    set obj.ttl = 30d;
    }
    }
```

Note
>1. 如果不提供命令行选项 (-b hostname:port )，则 backend default 部分将指定要连接的服务器。
>2.当守护进程收到一个客户机请求时，将调用 vcl_recv() 函数。反过来，当从实际的 Web 服务器检索到请求对象后或者对 Web 服务器的请求失败后，将调用 vcl_fetch()。如前述，如果 Cache-Control 或 Pragma 报头被设为 no-cache，则 vcl_fetch() 也拒绝缓存。
>3. 在代码中，pass 操作暗示 “通过”，或者对于这次单独的请求/响应交换不执行任何操作。pipe 还把数据不加改变地从客户机传递到服务器，但是对客户机与服务器之间的所有后续事务都这样做（pipe 是连续不间断的 pass，直至任何一端关闭连接）。lookup 将尝试在缓存中查找响应，将把响应添加到缓存中。
这段配置文件解释一下：
A、Varnish 通过反向代理请求后端 B1/B2/B3，端口为 80 的 web 服务器；
B、Varnish 允许 localhost、127.0.0.1、1124.42.78.0/32 三个来源 IP 通过 PURGE 方法清除缓存；
C、Varnish 对域名为 x.com/y.com/z.com 的请求进行处理，非 x/y/z 域名的请求则返回“NOT be cached”；
D、Varnish 对 HTTP 协议中的 GET、HEAD 请求进行缓存，对 POST 请求透过，让其直接访问后端 Web 服务器。之所以这样配置，是因为 POST 请求一般是发送数据给服务器的，需要服务器接收、处理，所以不缓存；
E、Varnish 对以.txt 和.js 结尾的 URL 缓存时间设置 1 小时，对其他的 URL 缓存时间设置为 30 天。

```
kill -HUP `cat /usr/local/Varnish/sbin/Varnish.pid`
```

**启动 Varnish**
#ulimit -SHn 65536
#./start.sh
**B>.管理 Varnish:**
**配置开机自动启动 Varnish**
vi /etc/rc.local
增加以下内容：
ulimit -SHn 65536
**优化 Linux 内核参数**
vi /etc/sysctl.conf
net.ipv4.ip_local_port_range = 1024 65536

```
net.core.rmem_max=16777216
net.core.wmem_max=16777216
net.ipv4.tcp_rmem=4096 87380 16777216
net.ipv4.tcp_wmem=4096 65536 16777216
net.ipv4.tcp_fin_timeout = 30
net.ipv4.tcp_keepalive_time = 300
net.ipv4.tcp_tw_recycle = 1
net.core.netdev_max_backlog = 30000
net.ipv4.tcp_no_metrics_save=1
net.core.somaxconn = 262144
net.ipv4.tcp_syncookies = 0
net.ipv4.tcp_max_orphans = 262144
net.ipv4.tcp_max_syn_backlog = 262144
net.ipv4.tcp_synack_retries = 2
net.ipv4.tcp_syn_retries = 2
#sysctl -p
```

附

1、Varnish官方网站： http://Varnish.projects.linpro.no/

2、常见FAQ参考http://Varnish.projects.linpro.no/wiki/FAQ

## 9、 Varnish 缓存的后端健康监测：

```
backend web1 {
    .host = "127.0.0.1";
    .port = "8080";
    .probe = {
        .url = "/test.html";
        .timeout = 3000 ms;
        .interval = 5 s;
        .window = 3;
        .threshold = 2; #3 次检查里 2 次失败，就算不健康
    }
}
backend web2 {
    .host = "127.0.0.1";
    .port = "8081";
    .probe = {
        .url = "/test.html";
        .timeout = 3000 ms;
        .interval = 5 s;
        .window = 3;
        .threshold = 2;
    }
}
```

http://www.2tutu.com/post/572.html

## 10、 rewrite URLS

```
    if (req.http.host ~"^www.chinarenservice.com") {
        set req.backend = shutter;
        set                    req.url                    =                    regsub(req.url,                    "^",
"/VirtualHostBase/http/chinarenservice.com:80/Sites/chinarenservice.com/VirtualHostRoot");
      ………………………
```

## 11、 压缩配置

```
if (req.http.Accept-Encoding) {
      if (req.url ~ "\.(jpg|png|gif|gz|tgz|bz2|tbz|mp3|ogg)$") {
         # No point in compressing these
         remove req.http.Accept-Encoding;
      } elsif (req.http.Accept-Encoding ~ "gzip") {
         set req.http.Accept-Encoding = "gzip";
      } elsif (req.http.Accept-Encoding ~ "deflate") {
         set req.http.Accept-Encoding = "deflate";
      } else {
         # unkown algorithm
         remove req.http.Accept-Encoding;
      }
   }
```

## 12、 Expires Check

```
     sub vcl_fetch {
            if (obj.cacheable) {
                  /* Remove Expires from backend, it's not long enough */
                  unset obj.http.expires;
                  /* Set the clients TTL on this object */
                  set obj.http.cache-control = "max-age = 900";
                  /* Set how long Varnish will keep it */
                  set obj.ttl = 1w;
                  /* marker for vcl_deliver to reset Age: */
                  set obj.http.magicmarker = 1;
            }
      }
      sub vcl_deliver {
            if (resp.http.magicmarker) {
                  /* Remove the magic marker */
                  unset resp.http.magicmarker;
                  /* By definition we have a fresh object */
                  set resp.http.age = "0";
            }
```

```
        }
```

## 13、 Varnish 实现图片防盗链

防盗链最普遍的就是通过 Referer 来判断，如果不是从指定的域名来访问的就咔咔。知道了这个思路，剩下的就是配置文件的事儿了，这里只给出相关的配置文件段（当然，是加在 vcl_recv 里面）：

```
                    if (req.http.referer ~ "http://.*") {
                     if (  !(req.http.referer ~ "http://.*a\.com"
                       || req.http.referer ~ "http://.*b\.com\.cn"
                       || req.http.referer ~ "http://.*c\.cn"
                       || req.http.referer ~ "http://.*google\.com"
                       || req.http.referer ~ "http://.*yahoo\.cn"
                       || req.http.referer ~ "http://.*google\.cn"
                       )) {
                           set req.http.host = "img.test.com";
                           set req.url = "/images/default.jpg";
                       }
                           lookup;
                       }
                   }
                          else {
                                  pass;
                          }
```

解释一下，此段配置文件的意思：将来 Varnish 服务器的请求进行判断，如果 referer 不是以 http://开头（其实这一段是针对如果来的访问没有 referer，想了很久最后终于在高人指点下想到了这个办法:P），且 referer 匹配下面的域名列表中的任意一个，就按正常请求处理，否则将请求重写为 img.test.com/images/default.jpg。

文章来源：http://iyubo.blogbus.com/logs/33581942.html

## 14、

## 十、 Varnish 命中率监控：

```
<?php
//PHP script to poll varnish management port for performance statistics, written by Jason "Foxdie" Gaunt
//http://ja.pastebin.ca/1268381
// This is just a code snippet written by Jason "Foxdie" Gaunt, its not meant to be executed, it may work
as-is, it may not.
// I freely acknowledge this code is unoptimised but it has worked in practice for 6 months :)

// Lets define our connection details
$adminHost = "59.64.112.165"; // IP address to connect to
```

```
$adminPort = "3500"; // Port to connect to

// pollServer(str) - this function connects to the management port, sends the command and returns the results,
or an error on failure
function pollServer($command) {
	global $adminHost, $adminPort;

	$socket = socket_create(AF_INET, SOCK_STREAM, getprotobyname("tcp"));
	if ((!socket_set_option($socket, SOL_SOCKET, SO_RCVTIMEO, Array("sec" => "5", "usec" => "0"))) OR
(!socket_set_option($socket, SOL_SOCKET, SO_SNDTIMEO, Array("sec" => "5", "usec" => "0"))))  {
		die("Unable to set socket timeout");
	}

	if (@socket_connect($socket, $adminHost, $adminPort)) {
		$data = "";

		if (!$socket) {
			die("Unable to open socket to " . $server . ":" . $port . "\n");
		}

		socket_write($socket, $command . "\n");
		socket_recv($socket, $buffer, 65536, 0);
		$data .= $buffer;

		socket_close($socket);

		return $data;
	}
	else {
		return "Unable to connect: " . socket_strerror(socket_last_error()) . "\n";
	}
}

// byteReduce(str) - this function converts a numeric value of bytes to a human readable format and returns
the result
function byteReduce($bytes) {
	// Terabytes
	if ($bytes > 1099511627776) {
		return round($bytes / 1099511627776) . "TB";
	}
	else if ($bytes > 1073741824) {
		return round($bytes / 1073741824) . "GB";
	}
	else if ($bytes > 1048576) {
			return round($bytes / 1048576) . "MB";
		}
		else if ($bytes > 1024) {
			return round($bytes / 1024) . "KB";
		}
		else {
			return $bytes . "B";
		}
}

// This is where our main code starts
echo "<div class=\"inner\"><br />Statistics since last reset:<ul>";
$stats = pollServer("stats");
if (substr($stats, 0, 3) == "200") { // If request was legitimate
		// Clear all excessive white space and split by lines
		$stats = preg_replace("/ {2,}/", "|", $stats);
		$stats = preg_replace("/\n\|/", "\n", $stats);
		$statsArray = explode("\n", $stats);

		// Removes the first call return value and splits by pipe
		array_shift($statsArray);
		$statistics = array();
		foreach ($statsArray as $stat) {
			@$statVal = explode("|", $stat);
			@$statistics[$statVal[1]] = $statVal[0];
		}
		unset($stats, $statsArray, $stat, $statVal);

		// Start outputting statistics
		echo "<li>" . $statistics["Client connections accepted"] . " clients served over " . $statistics["Client
requests received"] . " requests";
		echo "<li>" . round(($statistics["Cache hits"] / $statistics["Client requests received"]) * 100) .
"% of requests served from cache";
		echo "<li>" . byteReduce($statistics["Total header bytes"] + $statistics["Total body bytes"]) . "
served (" . byteReduce($statistics["Total header bytes"]) . " headers, " . byteReduce($statistics["Total
body bytes"]) . " content)";

		// The following line is commented out because it only works when using file storage, I switched to
malloc and this broke
		// echo "<li>" . byteReduce($statistics["bytes allocated"]) . " out of " .
byteReduce($statistics["bytes allocated"] + $statistics["bytes free"]) . " used (" .
round(($statistics["bytes allocated"] / ($statistics["bytes allocated"] + $statistics["bytes free"])) *
100) . "% usage)";
}
```

```
else {
	echo "Unable to get stats, error was: \"" . $stats;
}

echo "</ul></div>";

?>
```

## 十一、 参考资料

http://bart.motd.be/using-Varnish-http-accelerator-experiences-so-far
http://longrujun.name/blogs/longrujun/archive/2008/11/08/Share-Reading_3A00_.aspx
http://www.ibm.com/developerworks/cn/opensource/os-php-Varnish/
http://www.mumonitor.com/blog/?p=514
http://www.2tutu.com/guestbook.asp?page=2
http://blog.s135.com/read.php/313.htm
http://www.sunnyu.com/?p=14 Nignx 配合 Memcached 提升 400%性能(阅读笔记)
http://user.qzone.qq.com/56802890/blog/1229330100
http://Varnish.projects.linpro.no/wiki/VarnishInternals
http://www.chinaunix.net/jh/38/987084.html
http://www.sysbus.com/?p=117
http://www.idconline.cc/?q=node/292