

Varnish 浅析

付根希 genxi@staff.sina.com.cn

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

2.1 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 session，在 session 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。

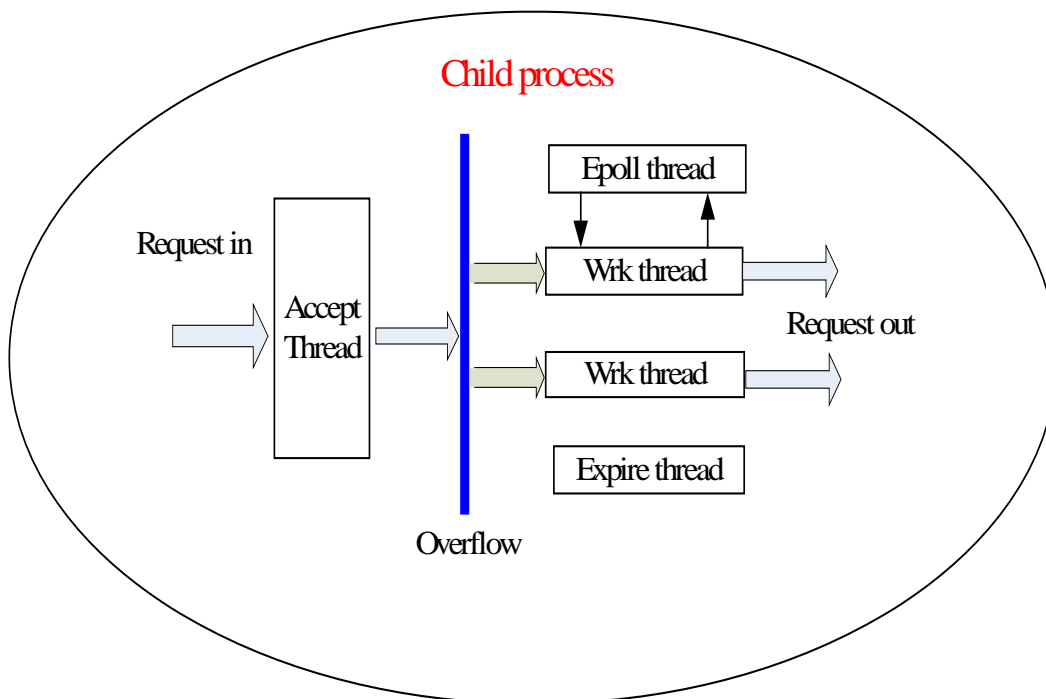


图 2-1 varnish 总体架构简化图

线程之间的关系：

2.1.1 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 2.1.2 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。

2.1.3 **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。

2.2 work 线程的处理过程

2.2.1 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：

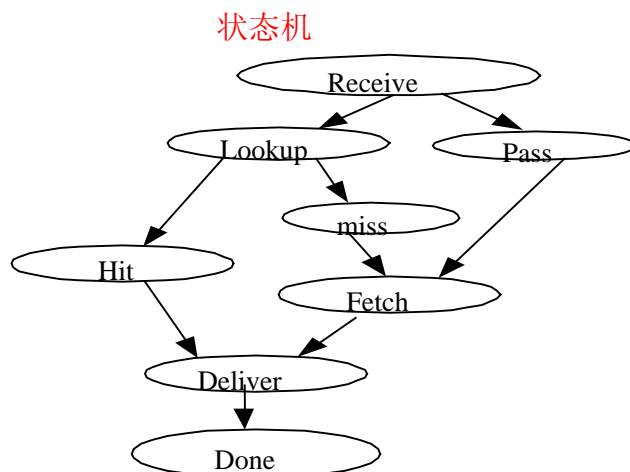


图 2-2 http 请求处理状态图

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

2.2.2 Work 线程总体工作如下：

接收到请求，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

2.3 工作线程的管理

2.3.1 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

2.3.2 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

2.4 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

3.1 Hash 方式

3.1.1 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

3.1.2 Hash classic

-第一层 hash bucket（较大的素数）包含锁

-采用 CRC32 方法，key 可配，一般是 url + host

-通过链表解决冲突

-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

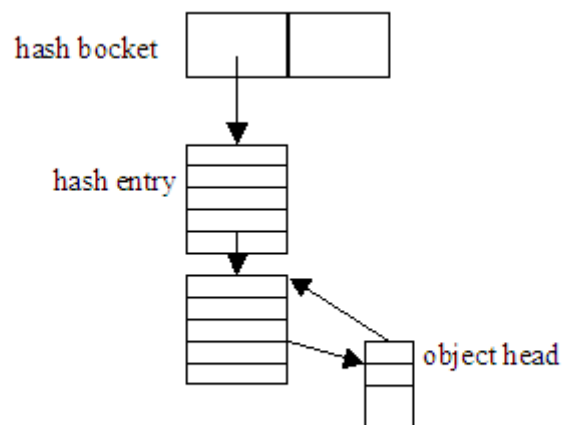


图 3-1 hash 结构图

3.2 Storage 方式

3.2.1 Malloc

-通过 malloc 获取内存

-通过 free 释放

—特点：简单

-有什么不好呢？

3.2.2 Mmap file

创建大文件，通过二分法分段映射成 1G 以内的大块

大文件

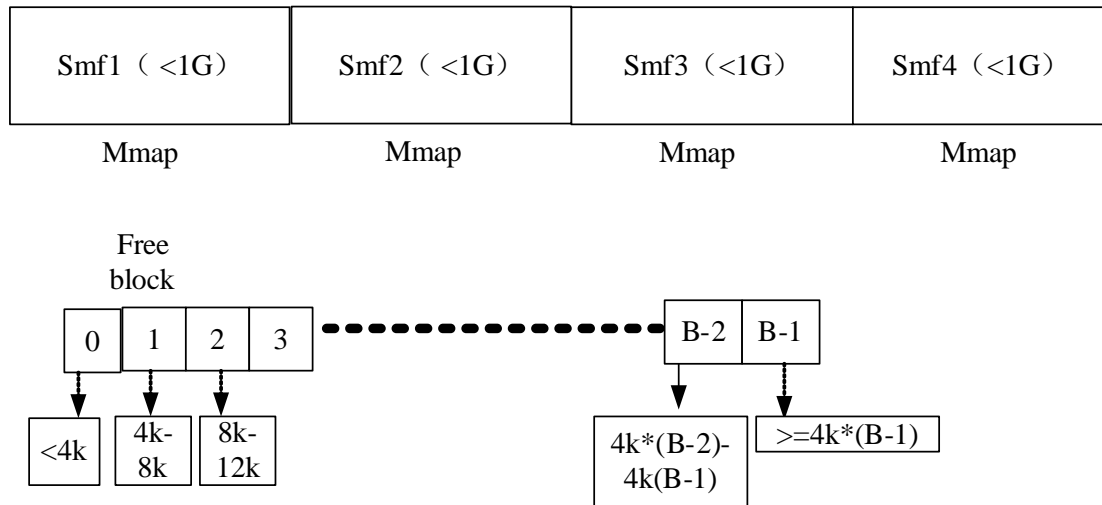


图 3-2 hash classic 存储信息图

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。

3.3 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

3.3.1 采用 writev

-将 store 链表上的数据组成 iov，通过 writev 输出

3.3.2 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>

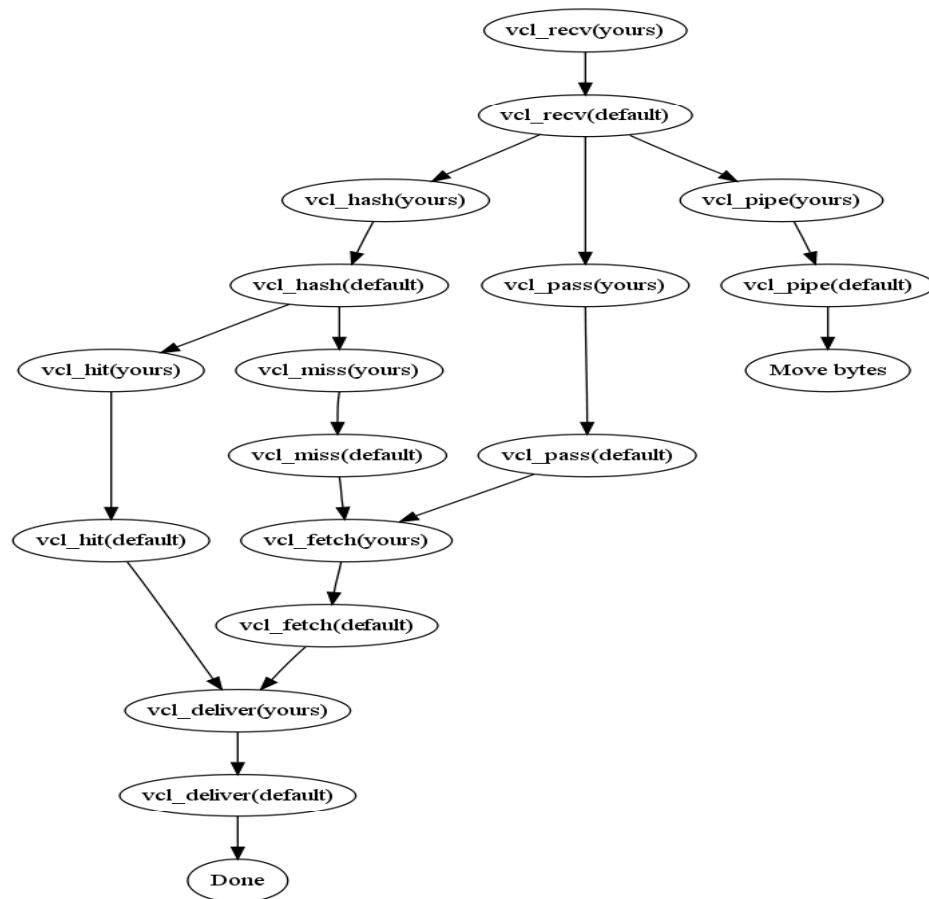


图 4-1 vcl 与状态机的关系

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