# 让程序运行更快

## 从varnish比squid快说起

12年10月19日星期五

很高兴有这样的一个机会跟大家一起交流，我今天要讲的主题是“让程序运行更快”
先作个小调查，在工作中考虑过程序的速度的问题的人请举手

# 为什么还考虑速度?

- 服务端程序一直在提升速度
- 操作系统不断提升io和网络处理速度
- 特定任务，如数据分析必须在受限时间内完成
- 系统关键环节提升速度的收益大

12年10月19日星期五

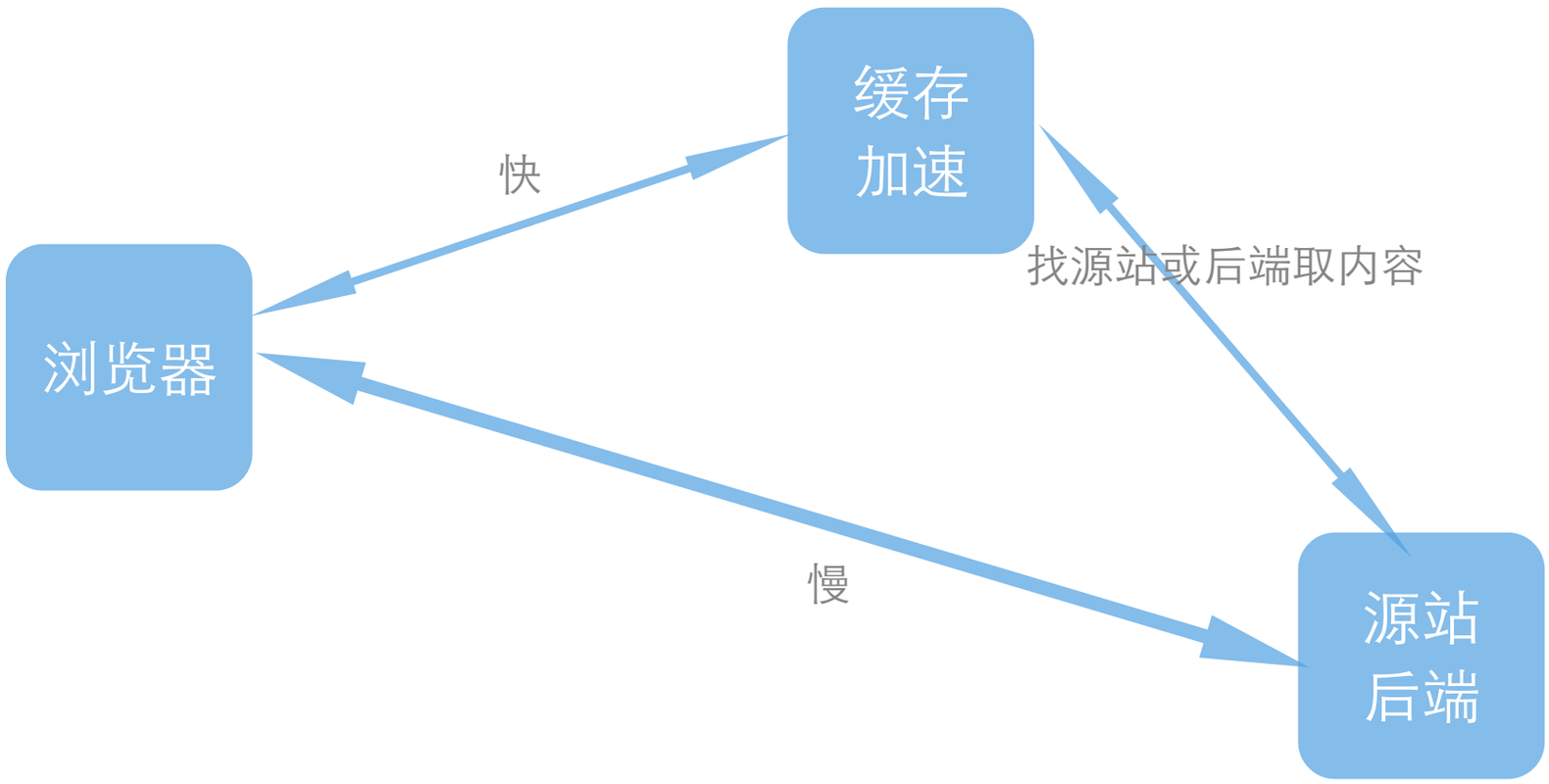
1、对于服务器端程序来说，如果速度跟不上潮流，就会被新的程序超越，比如lightd、nginx这样的程序就超越了apache
2、操作系统向下与硬件有接口，向上跟应用程序以系统调用为接口，操作系统不断在提升老接口的处理速度，同时不断增加新接口来取代旧的慢速接口，在后面还会比较多的说这一块的内容
3、前面两条或许跟日常工作没有啥关系，你或许会说我不写服务器端的dameon程序，也不写操作系统，后面两条在工作碰到的可能性比较大
4、关于最后一条中的收益，我遇到的是写好代码能为公司一次性省下几百万元，也有每月省上百万元的情况
5、既便是你没有优化速度的需求，这种思路也能用在优化成本等别的目标上，了解一下也没有啥坏处。呵呵

# 为何varnish比squid快?



我再问一下，知道squid或varnish是干什么的请举手。

# 什么是缓存加速软件?



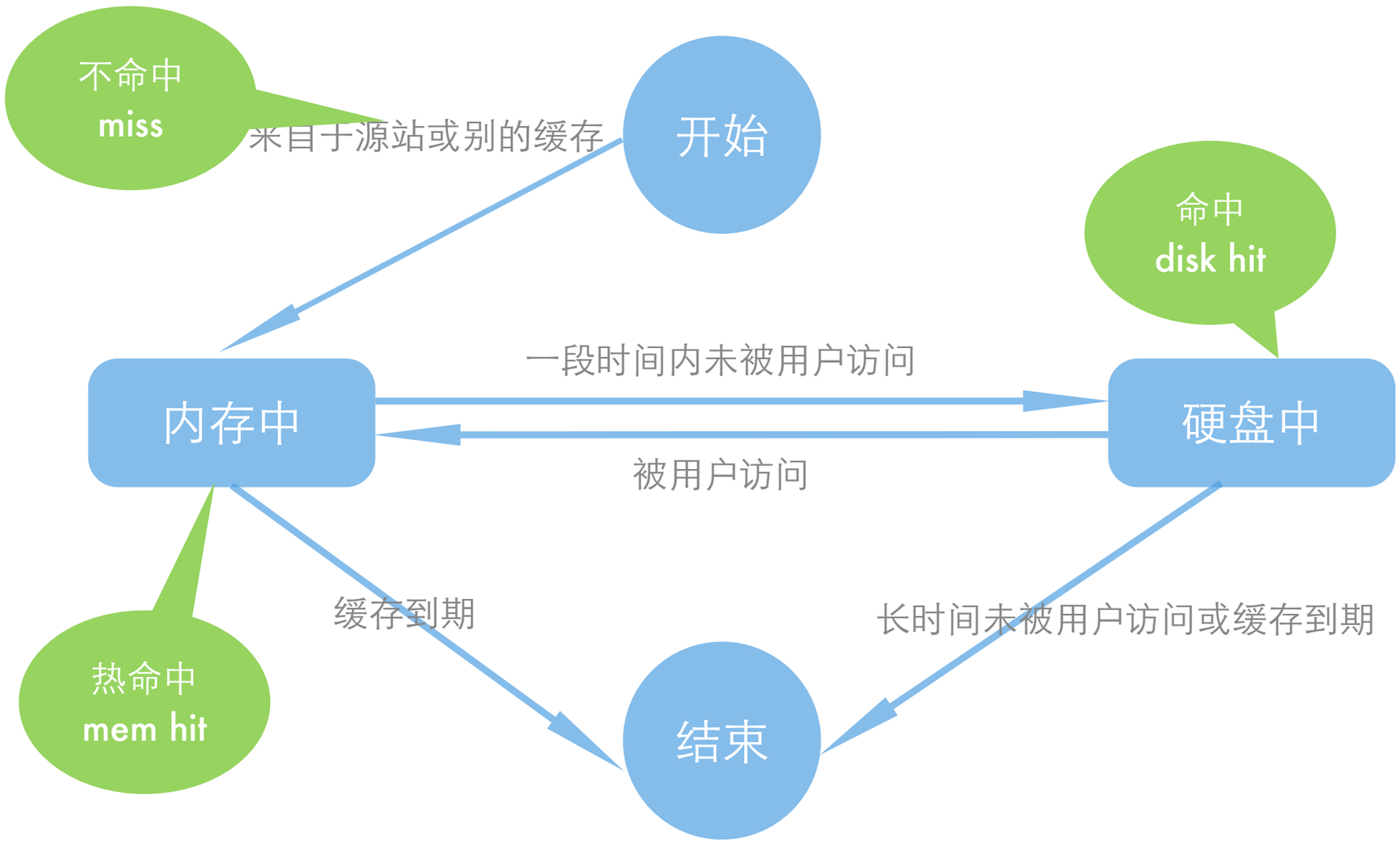

12年10月19日星期五

1、源站或后端可以直接对客户端提供服务，但是这个速度较慢，或者服务提供商因各种原因会切断这个服务路径

2、缓存加速软件目标是给客户端提供更快的服务，但它本身不产生内容，还得找源站或后端去取内容，拿到内容后会缓存下来，后续如果有客户访问同样的内容时，直接返回缓存的内容，用这种方法起到加速的效果

3、缓存加速软件的理想是它缓存下来的内容在未来的一段时间内还会被客户访问，相反在未来一段时间内不会被访问的内容将会从缓存中清除，毕竟用来缓存的内存或硬盘都是有限的；可惜的是，缓存加速软件同样没有预测未来的能力，所以事实上它一般会缓存最近被访问的内容，除此之外，在内容的取舍上有时也考虑过去一段时间访问次数这个因素。

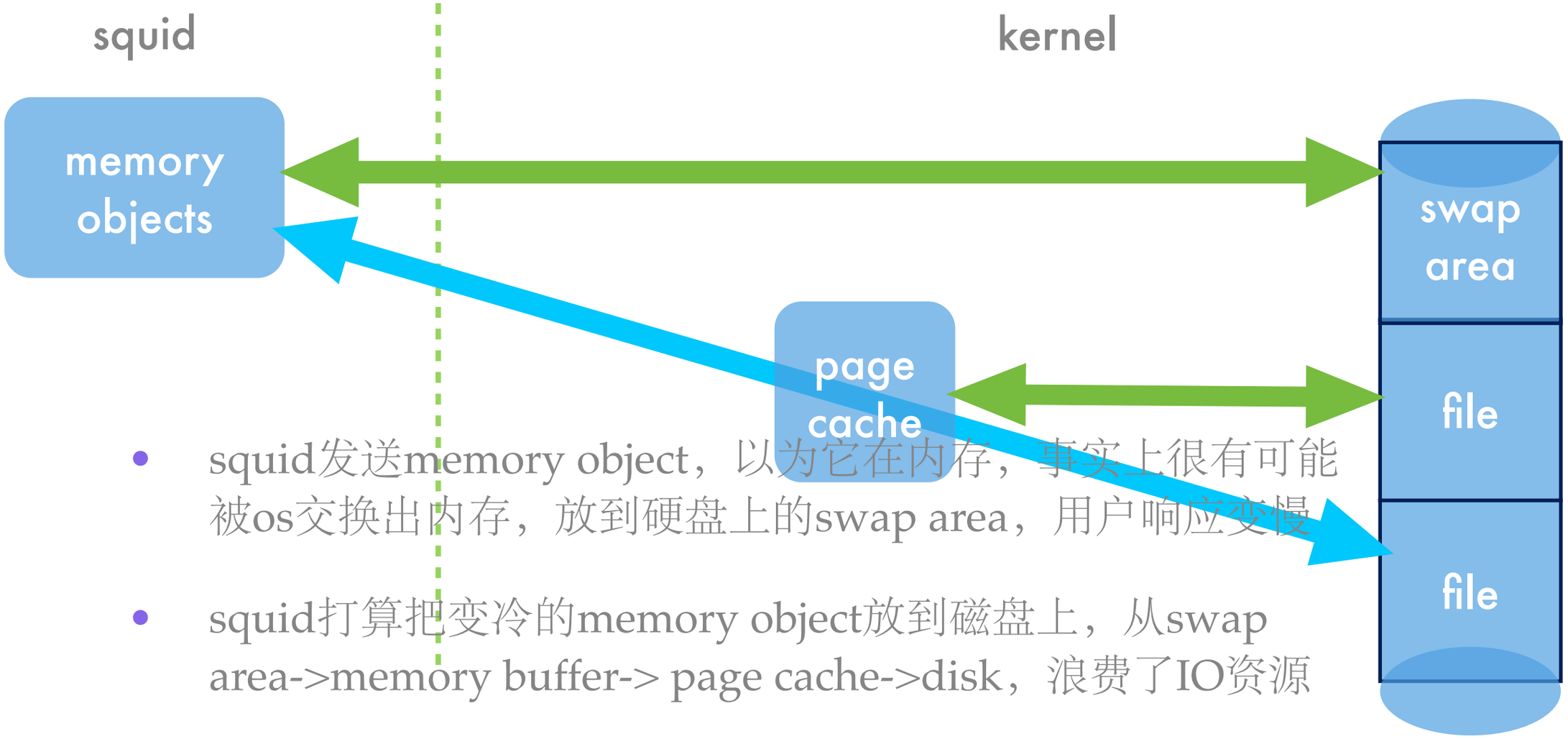
# 缓存对象的生命周期




12年10月19日星期五

问个小问题，从源站取内容返回给用户，一定比从缓存中特别是硬盘上中取内容快么?

# squid内存管理跟vm冲突

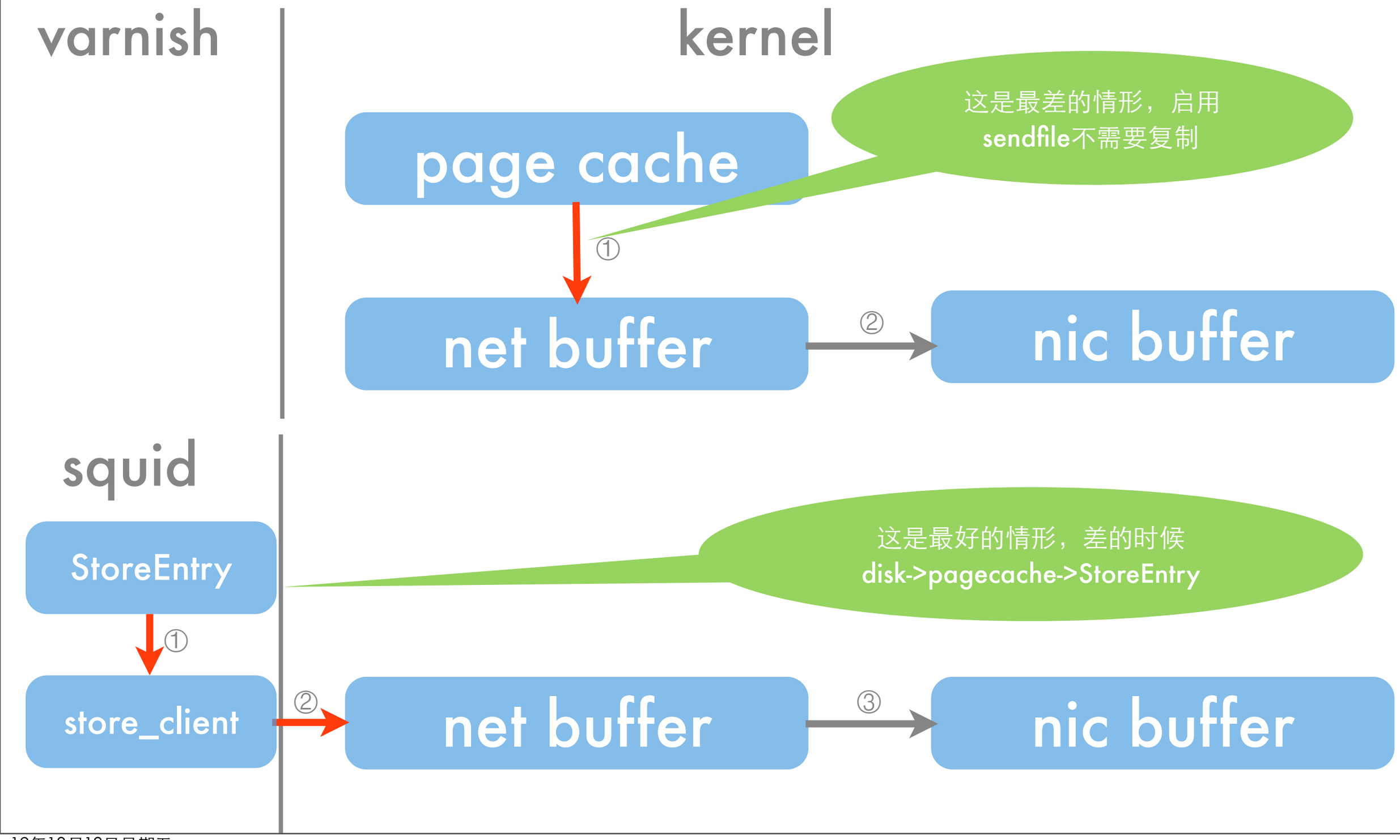



- squid发送memory object，以为它在内存，事实上很有可能被os交换出内存，放到硬盘上的swap area，用户响应变慢
- squid打算把变冷的memory object放到磁盘上，从swap area->memory buffer-> page cache->disk，浪费了IO资源
- squid打算把硬盘中的object放在内存中时，pagecache中另有一份，浪费内存资源
- 增大squid管理的内存大小，验证实际效果反而变差

12年10月19日星期五

1、vm指os的虚拟内存管理系统
2、在squid刚开发的年代，操作系统还没有虚拟内存系统，squid把最近访问的对象放在内存中，一段时间不被访问的内容放在硬盘中，也就是自己在作对应的管理
3、由于硬盘访问比内存访问慢很多，而现代操作系统会被最近访问的硬盘内容放在内存中，即page cache中，在内存紧张的时候，会被一段时间未被访问的内容写回到文件或交换区(swap)中
4、这样在内存和文件之间有一个操作管理的内存缓冲(pagecache)存在，squid以为从硬盘中把缓存对象复制到自己的内存中，实际上会是从page cache中复制过来的
5、同样，squid认为它管理的内存对象在内存中，有可能会在交换区(swap area)中
6、这样squid内存管理，跟操作系统的虚拟内存管理，目标是一致的，但同时使用会存在冲突

# 热命中时内存复制次数对比



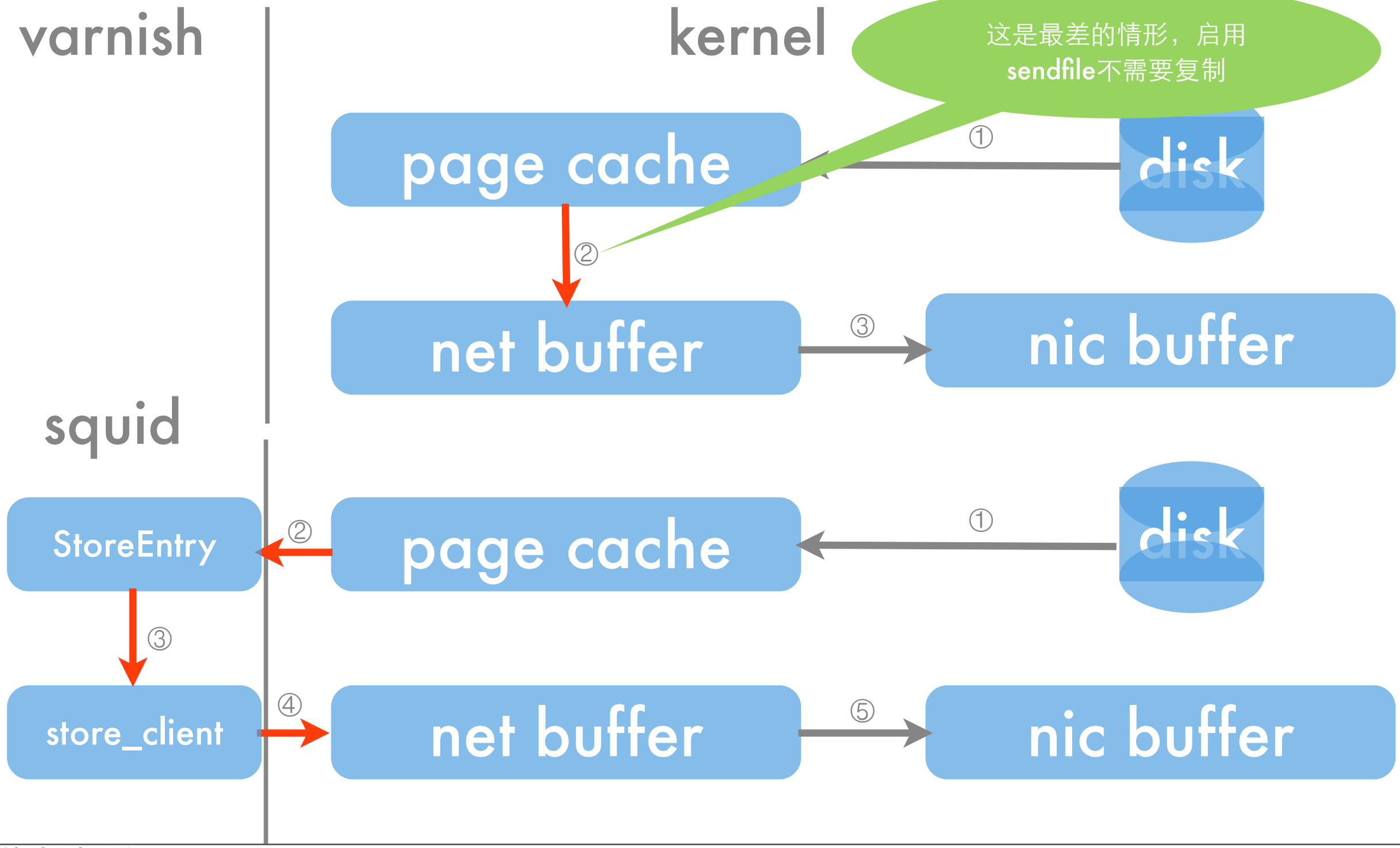

12年10月19日星期五

1、接下来比较squid和varnish在各种情形下处理用户请求时的内存复制次数，为什么这么关注内存复制次数？我请大家先想想这个问题。

2、这是用户请求的对象在内存中时，两者的复制次数对比图

3、红色的线条对应的复制由cpu进行，黑色的线条对应的复制是dma复制

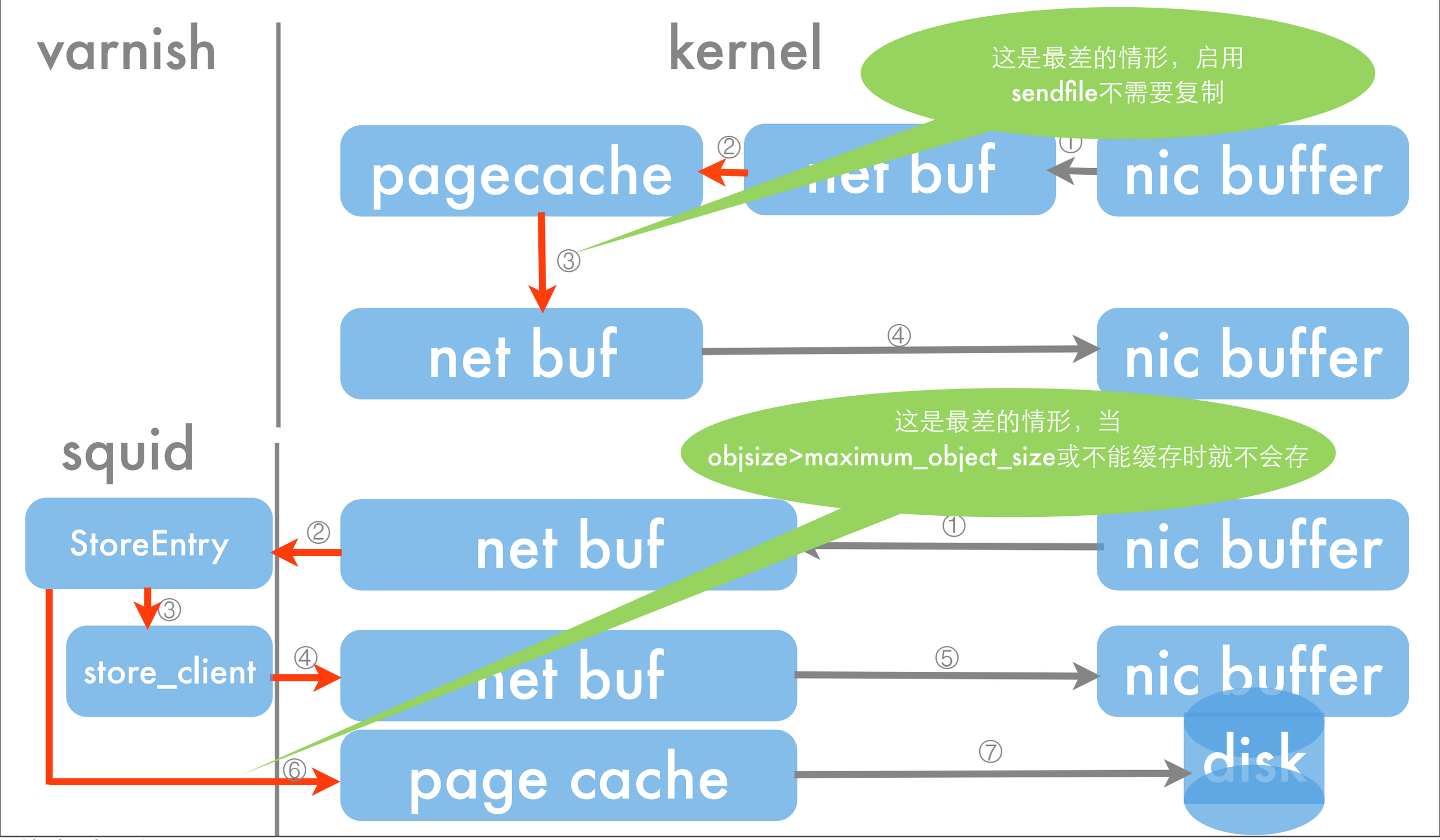
# 硬盘命中时内存复制次数对比




12年10月19日星期五

1、这是用户请求的对象在硬盘中时两者复制次数对比图
2、同样，红色的线条对应的复制由cpu进行，黑色的线条对应的复制是dma复制

# 不命中时内存复制次数对比

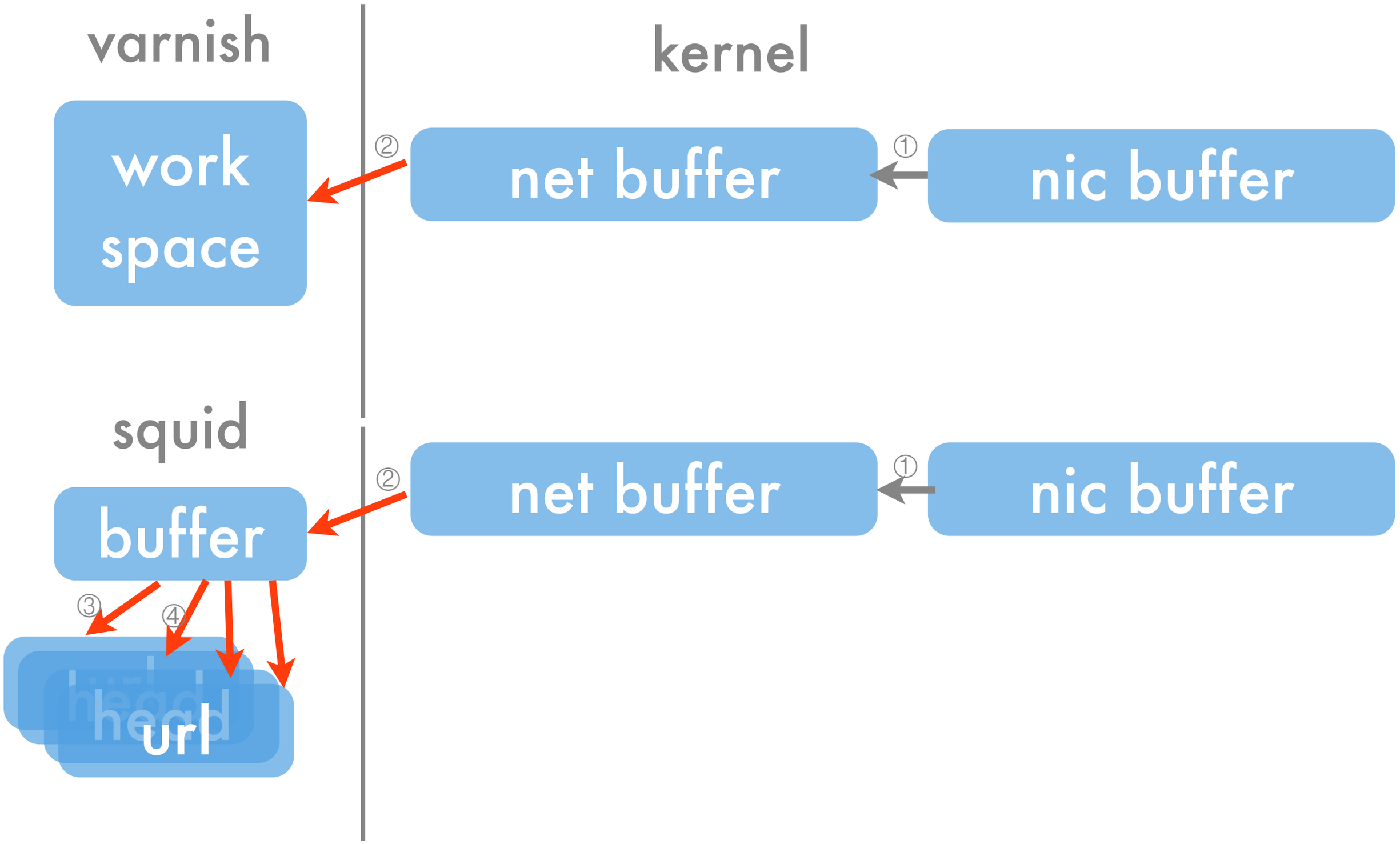





1、这是当用户请求的对象不在缓存中，即在内存和硬盘中都没有，这时候需要从源站去取内容，两者内存复制次数对比图

2、同样，红色的线条对应的复制由cpu进行，黑色的线条对应的复制是dma复制

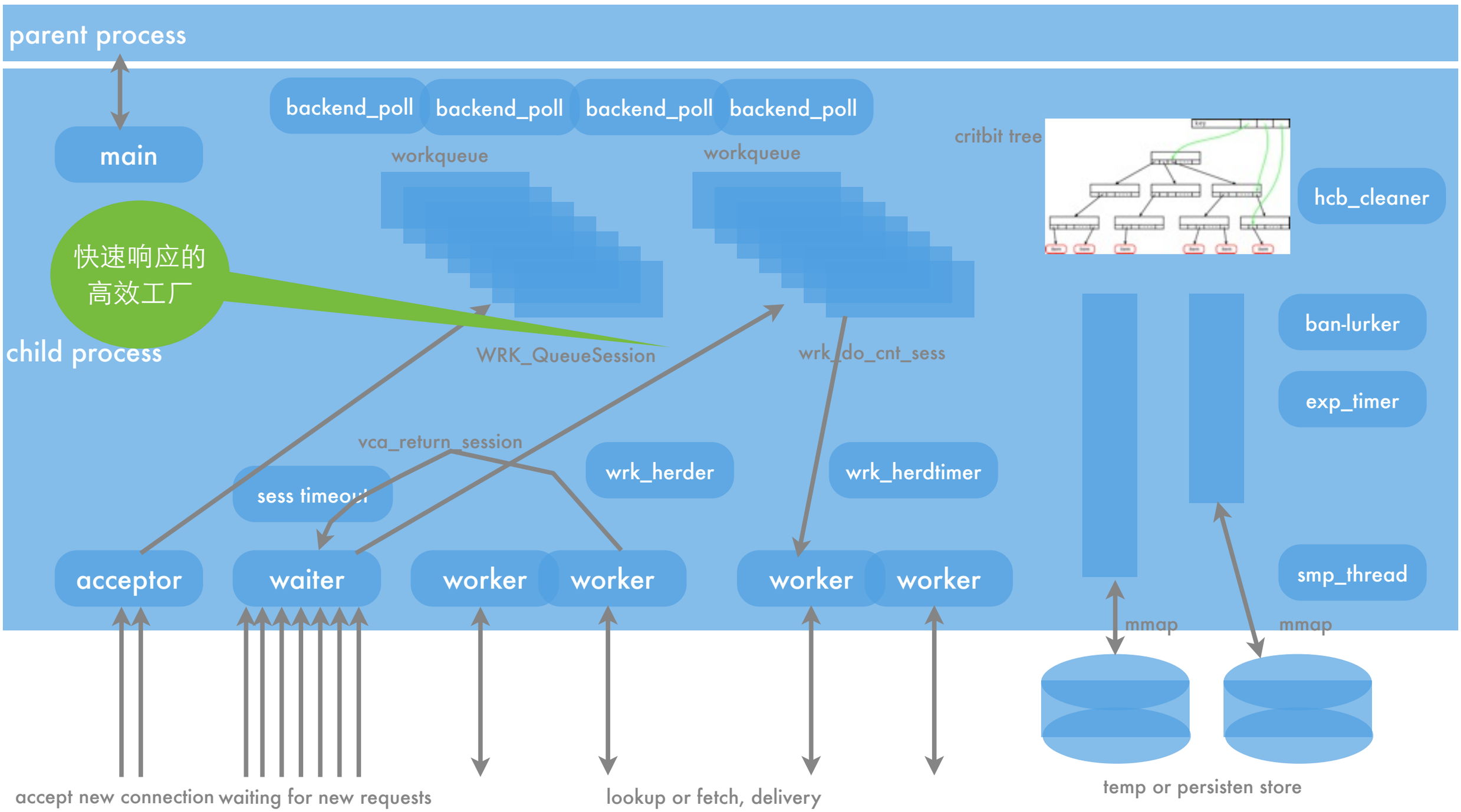
# 分析请求时内存复制次数对比




12年10月19日星期五

1、这是两者刚收到一个用户请求，进行分析时两者的内存复制次数对比图
2、同样，红色的线条对应的复制由cpu进行，黑色的线条对应的复制是dma复制
3、从处理一个用户请求时对内存的复制次数来看，squid多出不少，内存复制次数越多，处理一个请求费的时间就越大，也就是说处理请求的速度越慢

# varnish是高效多线程程序




12年10月19日星期五

1、图中每一个圆角长方形是对应一个线程。varnish象一个快速响应的高效工厂，由于时间的关系，不在这里细讲线程之间的关系了

2、varnish每一个工作者线程，收到一个请求后会负责到底，直到完成才会接下一个处理任务，这样只有任务能执行，就执行没有等待时间，而squid是单线程事件驱动，当请求可执行时，有可能它在处理别的请求，这样在任务对列中进行等待，从而单个请求速度会因处理的请求数量多少而变慢，同时不可控

# varnish更快的原因

- 多线程，有效利用多核、CPU资源
- 内存访问和复制次数少
- 处理请求系统调用次数少
  - 命中请求产生11个系统调用，获取7次锁
- 有效地利用操作系统虚拟内存

# 现代硬件的变化

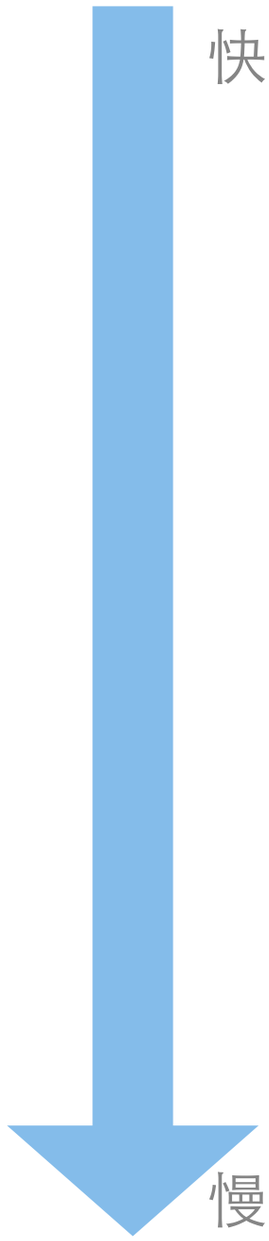
- CPU
  - 流水线，一个时钟周期运行多条指令
  - 打乱顺序执行指令
- CPU和内存之间有多级高速缓存
- CPU之间需要通讯以确保从内存中读取数据的一致性

12年10月19日星期五

1、为什么前面提到的原因会产生大的速度影响呢，这就需要从现代硬件的变化说起
2、第三条解释一下，当一个cpu从内存的某个位置读取数据时，有可能该数据存在别的cpu的高速缓存中，而且已经被修改，所以cpu不能简单地从内存读取数据，复视别的cpu的高速缓存的存在，不然会读到过期的数据，产生数据的不一致问题
3、而cpu之间通讯保存高速缓存间的数据一致性是有代价的，当不同的cpu频繁地读写同一位置，更准确来讲，是同一条cache line中的数据时，数据明显变慢

# 不同操作速度

- 访问cpu寄存器
- 访问cpu高速缓存
- 访问内存
- 原子操作
- 锁操作
- 系统调用
- 上下文切换
- 访问硬盘




12年10月19日星期五

1、上面从上放下，操作的速度不断变慢。访问cpu寄存器小于一个时钟周期，访问cpu高速缓存需要几十个时钟周期，访问内存需要上百个时钟周期，原子操作需要几百个时钟周期，锁操作和系统调用需要上千个时钟周期，而上下文切换需要几千个时钟周期。访问硬盘需要10ms左右

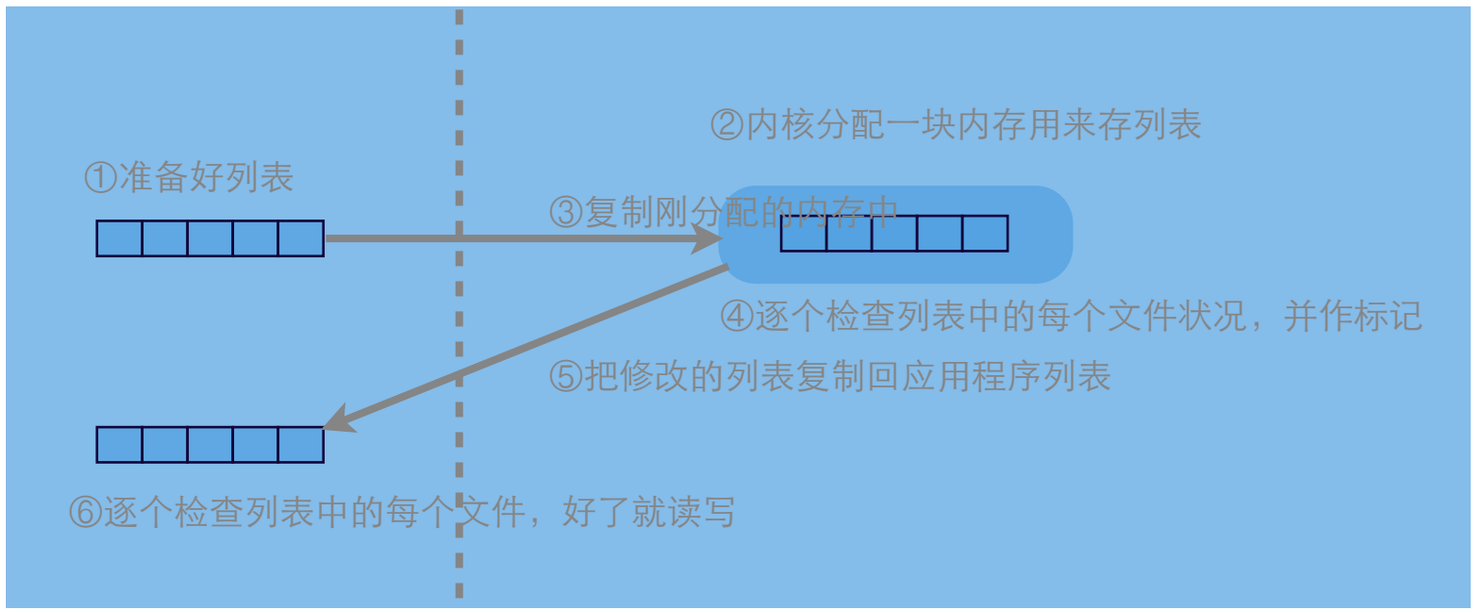
2、重要是上面的操作完成任务，就不用下面的操作完成任务。比如，能用原子操作实现一个无锁的算法，就比用一个加锁的版本快。

# 操作系统速度

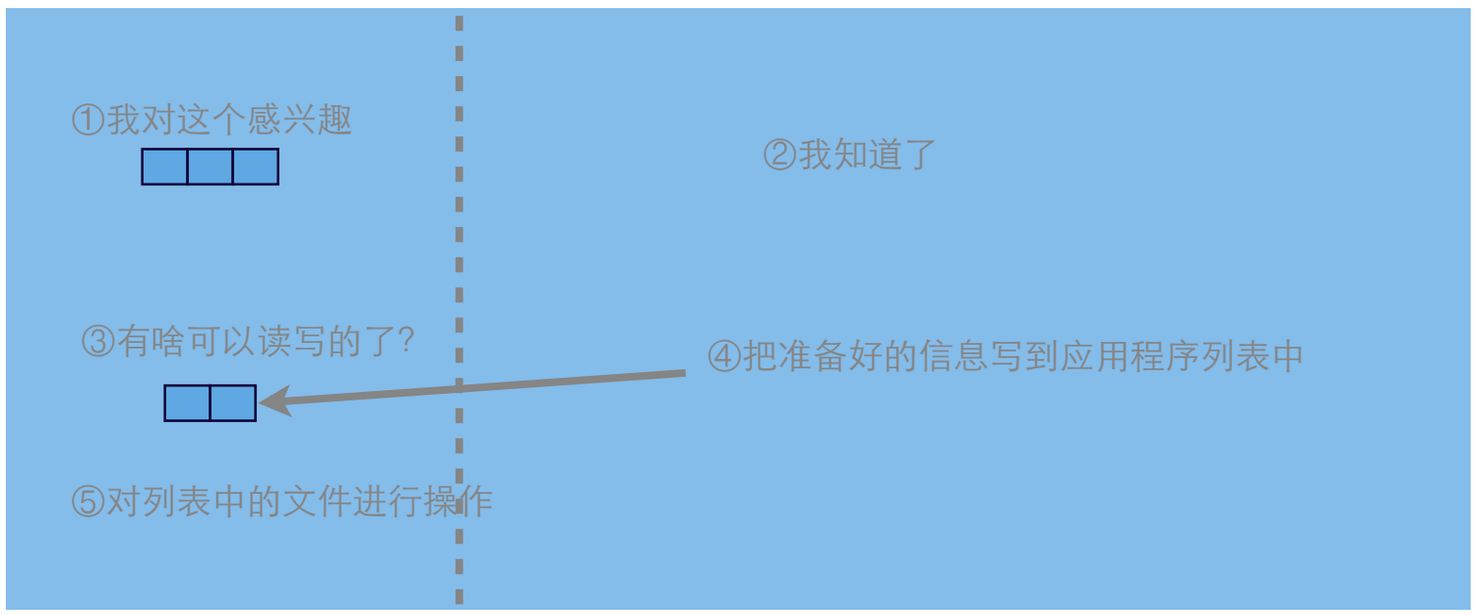
12年10月19日星期五

接下来分析一下操作系统在提高速度方面的一些措施

# poll到epoll




- 两次内存复制，列表越多，复制时间也越长
- 应用程序和内核分别扫描整个列表，至少应用程序的扫描是多余的操作
- 上面都是O(n)复杂度
- 内核需要分配单独的内存来存放列表




- 内存核存有应用程序关心的句柄状态信息，有变动用单独的系统调用通知内核
- epoll调用返回时，内核返回符合条件的句柄信息列表

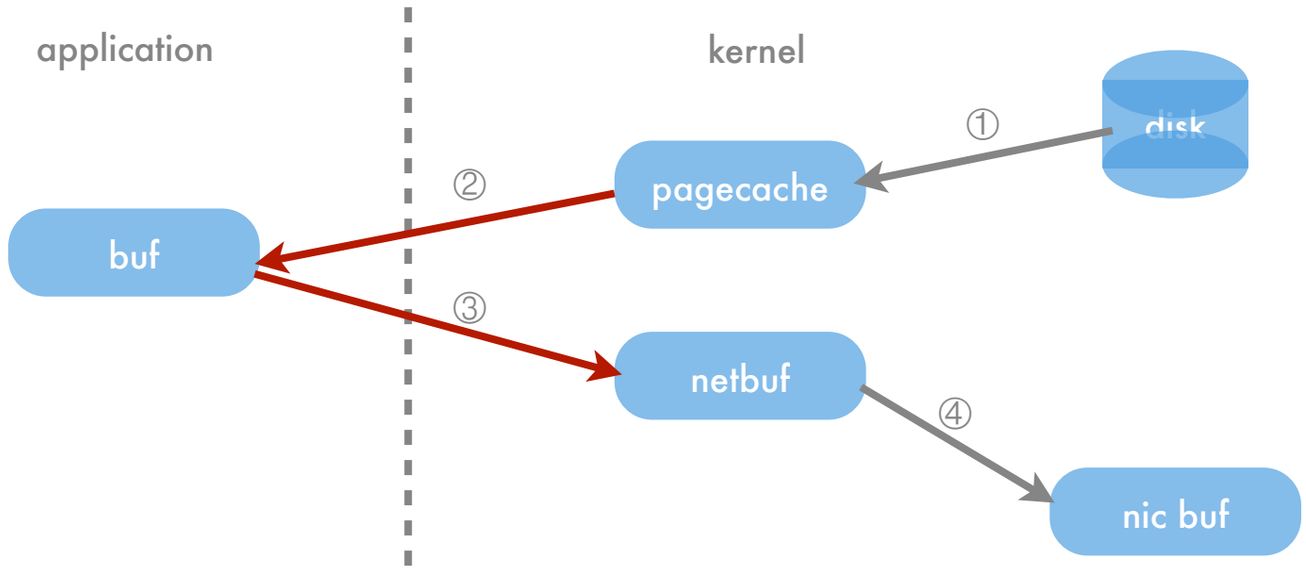
12年10月19日星期五

1、单线程事件驱动方式处理多个任务，需有一个机制在一些任务可处理时及时处理，这就是poll接口，应用程序把一个文件句柄列表及关注的条件(可读，可写等)给操作系统，操作系统返回符合条件的句柄，应用程序则进行读写操作

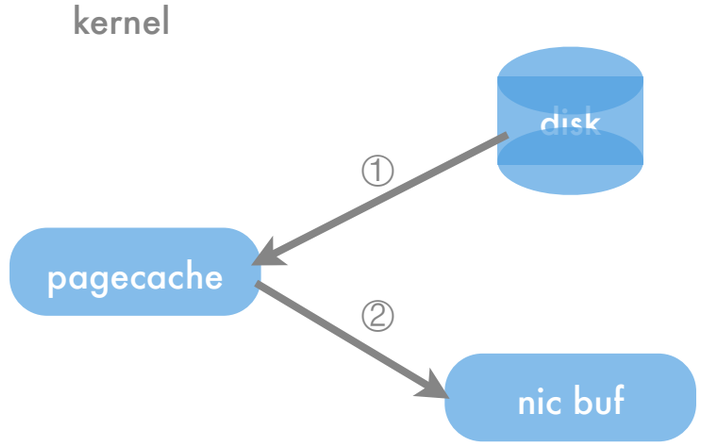
2、随着服务器并发处理的请求数量不断增大，原有的接口的低效使得应用程序处理单个请求的速度越来越慢，而这个低效不是操作系统单方面进行优化就能解决的，所以产生了新的接口，在这一页就是对老接口的问题和新接口的解决方法进行说明

# 零内存复制sendfile接口

- read(file, buf, len)
- write(socket, buf, len)




- sendfile(file,socket, offset, len)




12年10月19日星期五

1、服务器端程序常见的一个操作是从文件中读取数据，通过socket发给客户端，老的方式需要应用程序进行两次系统调用，cpu对数据进行两次复制

2、为了提高速度，操作系统增加了sendfile接口，同样的操作是一次系统调用，cpu不需要进行内存复制，即零复制

3、sendfile的问题，是不同操作系统接口不统一，有的能一次发送文件的不同部分，有的还能实现文件到文件的发送，不仅是文件到网络的发送

# gather io减少系统调用

- read,read,read -> readv
- write,write,write -> writev

12年10月19日星期五

1、用readv能实现一次性从文件的读取内容放到应用程序指定的不同的缓冲区中，这样不用多次调用read了

2、同样，writev可以将应用程序不同位置的多个缓冲区数据，一次性写到文件中，也不用多次调用write了

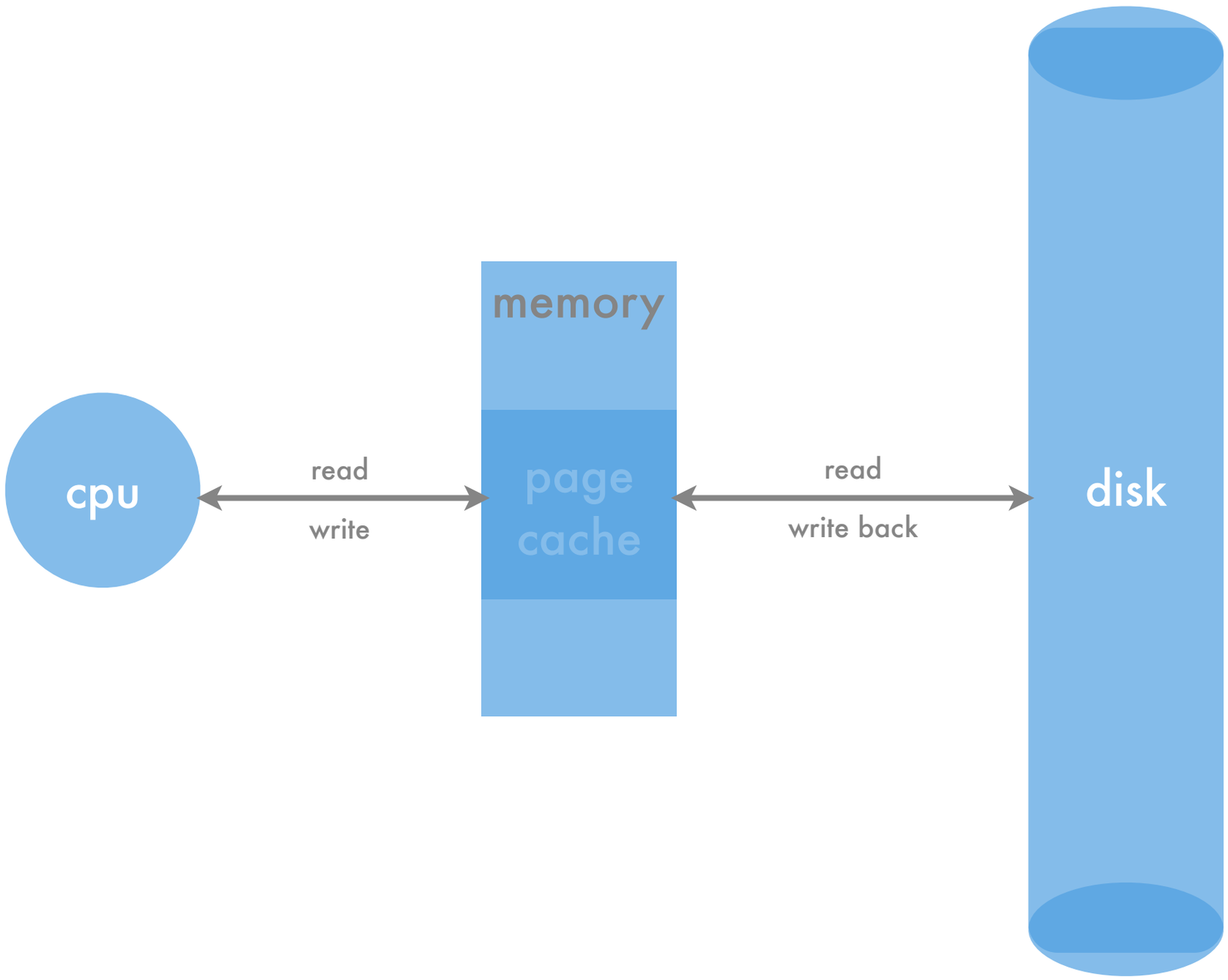
# 中断bottom half处理演进

- 2.2静态编译(32个bhs), 动态task queue
  - 同一时刻只能有一个运行，高时延
- softirqs, tasklets, work queues
  - 低时延, 多cpu的扩展性好

12年10月19日星期五

1、这个主要是针对linux操作系统。中间处理在程序运行时，一般会屏蔽相同类似的中断，所以需要快速运行，不然系统的响应变差，所以设备产生中断时的任务的上半部分在中断处理程序执行，下半部分用一个bottom half机制运行，下半部分运行时cpu能响应中断
2、拿网络部分来说，网卡收到一个数据后的处理工作，中断处理程序只会跟网卡硬件部分确认收到，放到设备驱动程序管理的缓存中、启动新数据接受工作就返回了，真正的处理工作都在bottom half部分执行
3、对数据包的处理速度的要求直接推动了linux的bottom half机制的优化

# 操作系统page cache



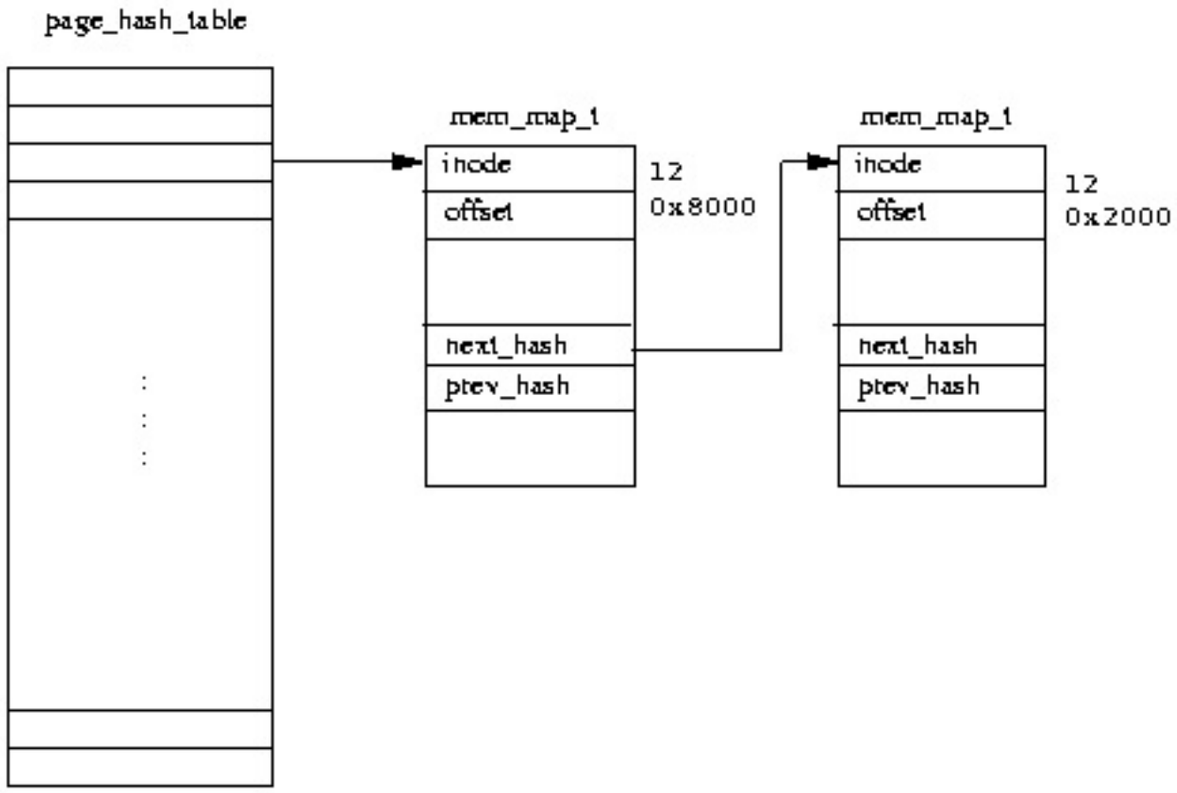

12年10月19日星期五

1、跟内存的访问速度比，硬盘的访问实在是太慢了，为了提高数据读取速度，操作系统会把硬盘上的数据放在内存中，期待后续的数据读写可以直接从内存中读取，而不是去访问硬盘。同样最理想的情况时，放在内存中的缓冲内容在未来的一段时间会被读取，而不被读取的内容将从内存中清理出去，跟squid这样的缓存软件一样，操作系统也无法预测未来，只能根据过去对数据的访问行为进行决策。结果现实的状况是缓存最近访问过的硬盘数据

2、当需要写入数据到文件中时，操作系统会写到内存缓冲区中然后返回。在合适的时间点或内存不足时才会真正写到硬盘上，这样应用程序不必等待漫长的硬盘写操作完成就可以返回处理别的事情

3、这个操作系统管理的缓存以page(页)为单位，所以也称为page cache。page cache实现方法对应用程序的处理速度影响很大，接下来会介绍page cache在提升速度方面的一些大的变化

# 最老page cache实现方法




- 所有文件的page cache信息都放在一个hash表里，锁争用严重
- 查询一个特定文件的page效率低，特别是内容不在page cache中更是如此

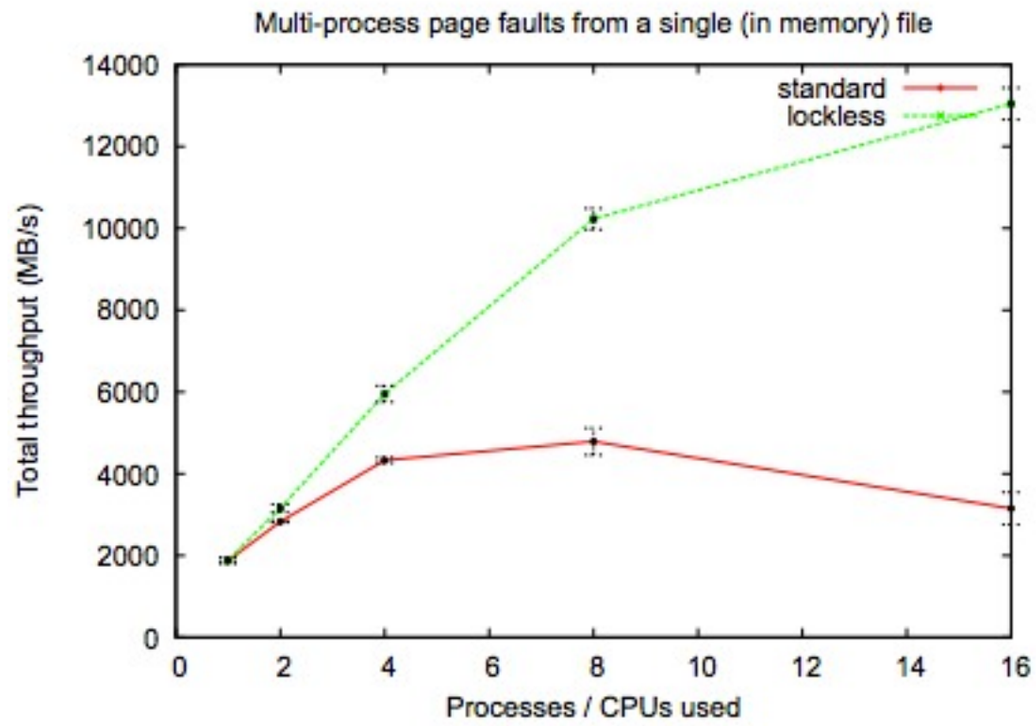
| CPUs | throughput |
|---|---|
| 1 | 1.00 |
| 2 | 1.51 |
| 4 | 2.15 |
| 8 | 2.27 |

# radix-tree page cache

- page cache信息记录在radix-tree上
- 每个文件有自己单独的radix-tree
- 访问不同的文件不存在锁争用情况，同时查询速度更快
- 访问同样的文件仍然需要获取锁(spinlock)

# 无锁的page cache版本

- 多个线程或进程同时读取同一个文件时速度更快
- 修改时仍需要获取锁



# os网络新接口想法

- 每次发送或接收需要动态分配和释放内存(skbuf) -> 预先分配
- 一次系统调用只能针对一个socket -> 一次系统调用，批量发送或接送，减少系统调用开销
- 应用程序和内核进行数据复制 -> zero copy

12年10月19日星期五

1、现在服务器使用万兆网卡已经不是什么稀奇的事情了，象土豆实现了最高一台机器能吐出15G的带宽来，未来会出现40G的网卡，这要求应用程序和操作系统一起协作，更快地处理数据
2、我认为优化老的socket接口已经不能满足这个发展的需求，需要产生一个新的接口

# python速度

12年10月19日星期五

1、咱们这是关于python的大会，不讲点python好象说不过去，下边简单说一下。我只写过hello world版的python程序，所以没有能力说太细。呵呵

# python dict实现特点

- 碰撞解决方法: open addressing
- 结构体中自带8个桶小哈希表，整个结构体占两个cache line
- 保持最多80个的不同的dict，加快dict的生成和释放
- 删除无素不会缩减桶数量，减少内存访问次数

12年10月19日星期五

1、我特意看了一下python的dict实现源代码
2、open addressing比链接方式更能有效地利用cpu与内存的高速缓存，缓存的命中率要高不少。别的语言，如ruby、perl、erlang的dict相关的实现都是采用链表方式。为此让我对python的好感增加不少
3、

# python案例：从几小时到不到一分钟

- 涉及到数值计算和海量循环，Python表现极其糟糕
- Cython + NumPy 可解决部分计算问题和内存问题，但GIL无法避免
- multiprocessing能解决SMP/GIL问题，但内存问题解决不了，也许共享内存+ctypes是个办法，没尝试过
- ctypes + C的动态链接库创建系统线程能解决GIL和内存共享问题，但无法在C中操作Python对象
- ctypes + Python C扩展意义不大，因为C扩展无法直接操作ctypes的数据指针
- 改写后的ctypes + Python C扩展解决了性能和内存消耗问题
- 具体参考：http://blog.sina.com.cn/s/blog_6d2cab390100vmok.html



这是一个python的案例，详细的介绍请参看我同事的blog

# python解释器问题

- 大锁，多线程效果差
- 可以向java vm/erlang beam学习
- 也可以象linux kernel学习

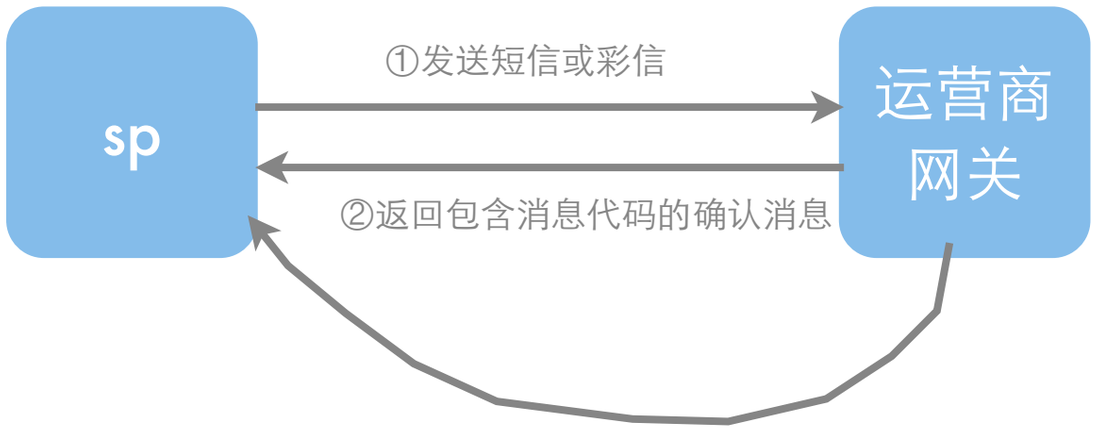
# 数据分析速度

12年10月19日星期五

1、前面也提过，数据分析的任务一般要求在指定的时间内运行完，土豆这边会根椐数据分析的结果，快速响应调整调度系统的配置，以实现好的用户播放体验

2、我们的分析程序一般是shell/awk/python脚本实现，简单高效

# 短信彩信勾对问题

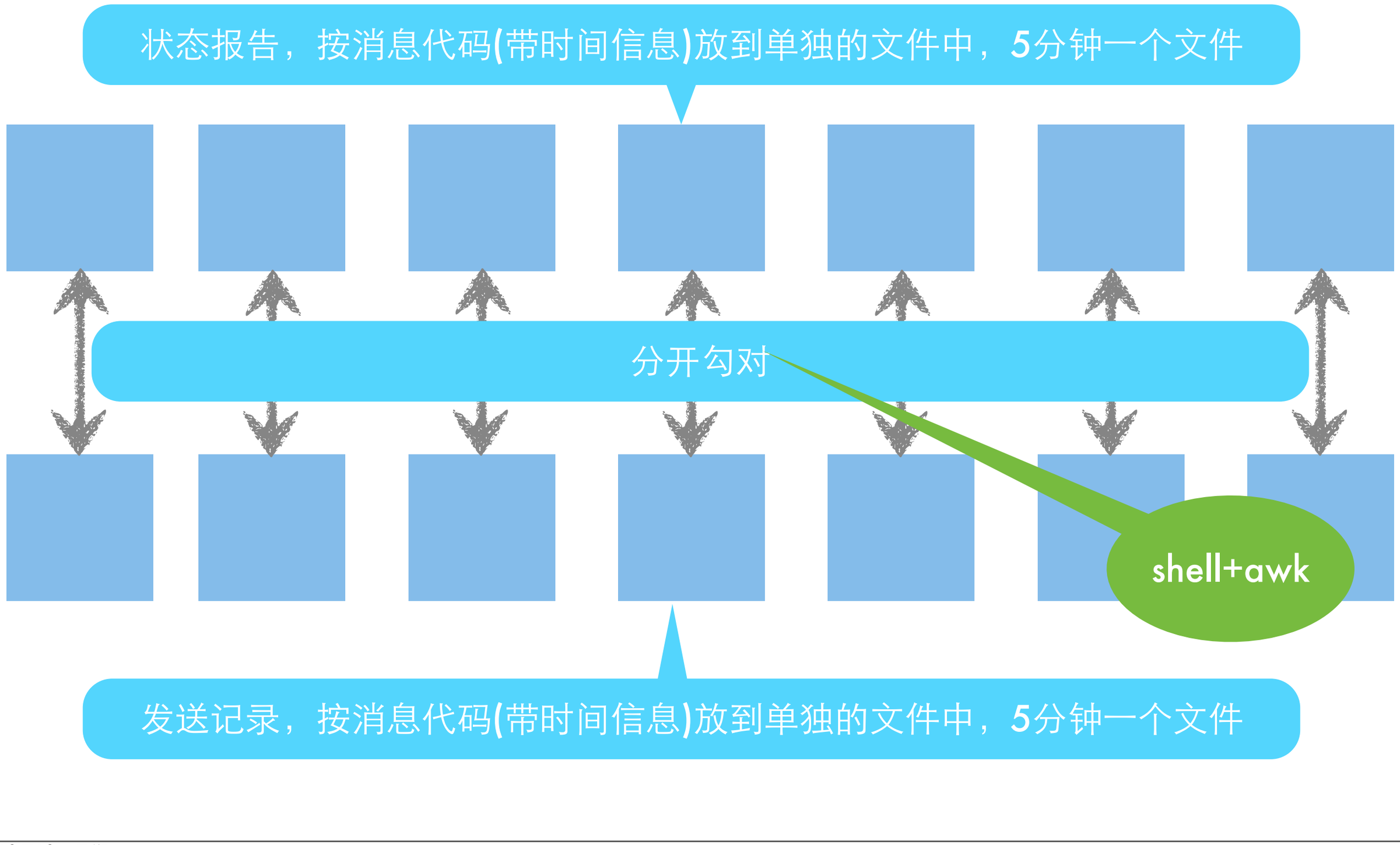



- 消息发送报告与状态报告之间需要勾对
- 第n天的短彩接受状况要拿第n天、第n+1天、第n+2天的状态报告进行勾对，才能出最终的结果
- 2千万的短信和2千万的彩信，你需要多长时间勾对完?

12年10月19日星期五

1、这是以前解决过的一个问题，值得说一下
2、短彩的发送，和状态报告的接受是异步的，中间最长时延是48小时，这是问题的关键

# 勾对从5小时到20分钟以内



# 土豆用户体验分析

- 百G左右的日志文件，客户端发送日志和服务器下载日志
- shell/python/awk/c
- 每次针对半个小时的日志计算，出半个小时的结果
- 半个小时的日志最长需要1.5小时，通过优化减少到14分钟

# 土豆用户体验分析

- 临时文件 -> 管道传输
  - 临时文件不压缩占硬盘io，压缩占cpu
- 分机房并发计算，再合并

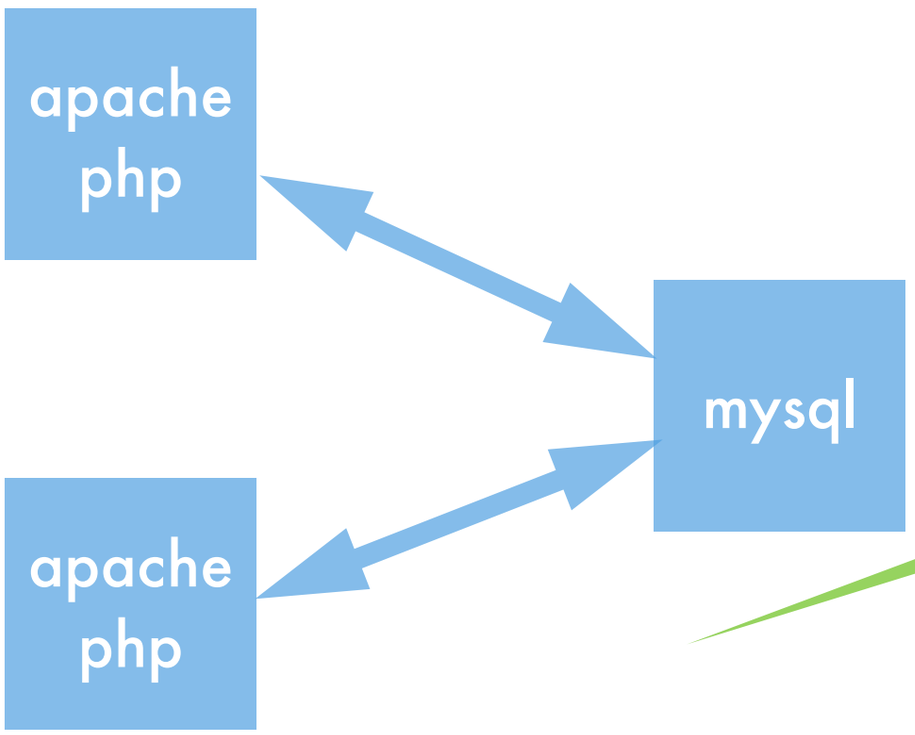
- 清洗环节用c写

# 案例

12年10月19日星期五

下面再说一个提高一个网站的整体响应速度的案例

# 网站用户响应速度问题




- image/css/php都由同样的2台httpd机器提供服务
- 前台和后台程序都在同样2台机器上
- 没有cms内容管理，活动发布靠修改php程序
- 网站帮助也是由php生成，且会建立数据库连接
- 2千万pv

如何快速提高响应速度

12年10月19日星期五

1、网站帮助为什么会建立数据库连接？数据库的配置和连接处理都在一个公用的文件中，被别的文件包含，而网站帮助程序，在开头会包含大的公用文件，这样后面虽不用使用数据库连接，但是也会建立

# 提高响应速度的方法

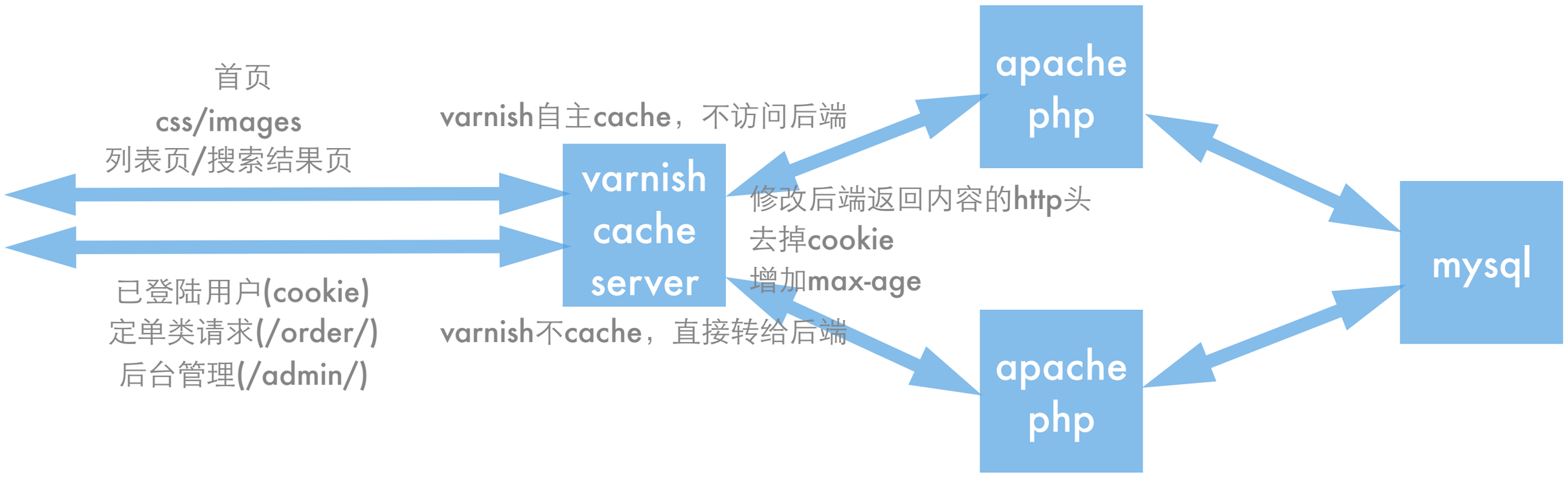
- 增加辅库，读写分离
- 增加expire/cache-control头
- 动静分离
- 生成静态html
- php cache
- 压缩内容
- 页面优化(如延迟加载)

有没有更快的方法?

12年10月19日星期五

1、按上面的方法整个优化下来，你认为需要多长时间？同时还需要响应别的需求，如发布新的活动，不断修改现有的php代码
2、有没有更快的方法?

# 快速提升响应速度方法




- 不改动原有系统，快速实施
- 方便测试和回滚

# 理念

# 提升速度的想法

- 多度量、多思考
- 少改动
- 找到改动的支点
- 不到20%的努力提升超过80%的速度

# 谢谢

- 关于我
  - 土豆网技术运营高级总监
- email：lixiaohong@gmail.com
- sina weibo：@槛内小红
- blog：http://blog.sina.com.cn/lixiaohong