第41卷 第2期 西 安 交 通 大 学 学 报 Vol.41 №2
2007年2月 JOURNAL OF XI′AN JIAOTONG UNIVERSITY Feb. 2007

应用控制的Web服务器磁盘缓冲方法

李卫，郑卫斌，管晓宏
（西安交通大学电子与信息工程学院，710049，西安）

摘要：针对大数据负载时磁盘I/O阻塞造成的Web服务器性能下降的问题，提出了应用程序控制缓冲(ACC)方法.其核心是，缓冲跟踪模块根据应用程序的文件访问过程来跟踪内核中的文件缓冲状态，缓冲控制模块进行缓冲替换和预取，保持文件缓冲有足够的空闲空间.这样，服务器可在用户空间控制文件缓冲，从而准确判断文件是否在缓冲之中，并依此来调度请求，以提高处理器和磁盘的I/O并行度.同时，服务器可采用适应自身特点的缓冲和预读策略，以提高缓冲的命中率.作为示例，将ACC在Flash服务器中实现，实现中选用了“金字塔选择”缓冲算法.实验表明，在大数据负载下使用ACC的Flash服务器性能有很大的提高，即便在数据负载稍大于物理内存空间的情况下，服务器的吞吐率仍可提高约24.4%，而当数据负载超出物理内存2～3倍时，吞吐率可提高3～4倍.

关键词：服务器；应用程序控制缓冲；缓冲替换
中图分类号：TP393 **文献标识码**：A **文章编号**：0253-987X(2007)02-0153-05

Application-Controlled Caching for Web Server

Li Wei, Zheng Weibin, Guan Xiaohong
(School of Electronics and Information Engineering, Xi′an Jiaotong University, Xi′an 710049, China)

Abstract: Focusing on the problem that under workloads of large dataset, the performance of a Web server is dropped due to the disk I/O blockings, an application-controlled caching (ACC) approach is presented. In ACC, the cache tracking module traces the status of file caches in kernel according to the files access process of applications, and cache replacement and prefetch are performed in cache control module to keep enough free space for file caches. With ACC, servers can control the file caches in user space, and determine whether a file is in caching, then schedule requests based on this to improve the parallelism of the processor and disk I/O. Meanwhile, servers can choose a suitable policy of caching and prefetching to increase cache hit ratio. ACC is implemented in Flash server as a demonstration, in which pyramidal selection scheme is adopted. Experiments show that the performance of Flash with ACC can be largely enhanced under workloads of a large dataset. Even when the dataset load is slightly larger than the physical memory space, its throughput is improved by about 24.4%; when the dataset load is 2-3 times larger, the throughput improved by 3-4 times.

Keywords: server; application-controlled caching; cache replacement

随着Web的发展和大量多媒体内容的引入，Web服务器的页面容量(又称数据负载)越来越大，其性能在很大程度上受到磁盘I/O的影响.早期的多进程服务器(MP)模型，如Apache，试图通过多个进程的并行工作来隐藏磁盘访问延迟，但进程创建、通信和上下文切换等开销都很大，严重影响了服务

收稿日期：2006-06-29. 作者简介：李卫(1965～)，男，副教授. 基金项目：国家高技术研究发展计划资助项目(2001AA140213).

器的性能.单进程事件驱动(SPED)模型可避免这些开销,但在大数据负载情况下会出现频繁阻塞磁盘的I/O,而非对称多进程事件驱动(AMPED)模型则结合SPED和MP二者的优点,能有效隐藏磁盘延迟和保证CPU与磁盘系统的并行度[1].

如何判断一个文件的访问是否需要磁盘I/O,早期Flash服务器采用mmap()/mincore()机制,虽判断准确但效率并不高.后来,Flash改用sendfile()来提高性能,通过记录文件的访问情况,试图在应用程序空间跟踪内核中文件缓冲的状态[2],但由于各个操作系统的文件缓冲替换算法差别比较大,所以跟踪可能会最终偏离内核中的真实缓冲状态,造成Flash主进程错误地阻塞在磁盘I/O之中,特别是在Linux环境下,阻塞问题比较突出,严重影响了性能.

受"应用控制的缓冲和预读"方法[3]的启发,本文提出了应用控制的缓冲管理方法(ACC),由服务器主动控制文件缓冲,从而避免出现服务器主进程阻塞在磁盘I/O之中的现象.特别是通过ACC,服务器可以采用适应自身特点的缓冲和预读策略,从而提高缓冲的命中率.

1 应用控制磁盘缓冲方法

在应用控制缓冲方法中有2个重要部件模块——缓冲跟踪模块和缓冲控制模块,它们都可在用户空间实现,其结构如图1所示.前者根据应用程序的文件访问过程来跟踪内核中的文件缓冲状态,后者运行特定的缓冲替换算法保持文件缓冲有足够的空闲空间.ACC的基本工作原理就是,跟踪并保持内核文件缓冲有足够多的空闲空间,从而减少乃至消灭内核的缓冲管理线程运行的机会,达到由应用程序控制系统缓冲的目的.实现ACC也比较简单,文件缓冲仍然在内核中,缓冲跟踪模块只需要根据文件名及其描述符为文件建立索引,而缓冲控制模块通过系统调用就可以释放某文件的缓冲.

ACC的一个重要工作前提是,系统的大部分文件访问都来自应用程序,这在Web服务器系统中是可以满足的,而其他的FTP、NFS和文件服务器等网络应用也都可满足这样的条件.从实现角度来看,在多进程并发服务器(如Apache)中实现缓冲跟踪与管理较困难,因此要求应用程序尽可能是单进程或单进程为主(如Flash)的架构.本文仅以Flash为例,讨论ACC的实现和Web服务器的缓冲替换算法的选择,并以实验来验证ACC在提高应用程序性能上的作用.

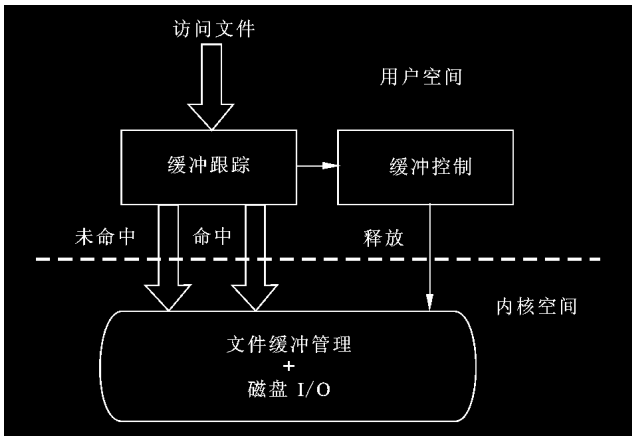

图1 应用控制缓冲方法

2 ACC在Flash中的实现

2.1 文件缓冲管理与磁盘预读

在服务器中,有些Web对象的数据量很大,如果将其全部读入缓冲,则给系统的内存造成很大的压力.因此,在具体实现中应将文件分成块来管理,一个文件由一个或多个块组成,块再根据需要读入缓冲和从缓冲中释放,同时缓冲替换算法针对整个文件来考虑是否替换出缓冲.另外,Web服务器的磁盘I/O具有很大的顺序性,可采用磁盘预读来加快磁盘I/O速度.

在实现中,可采用符合POSIX标准的编程接口(posix_fadvise())实现文件的磁盘预读和缓冲释放,所以程序有很好的平台可移植性.

2.2 缓冲替换算法的选择

适合Web的磁盘缓冲替换算法很多,这些算法通常考虑时间久远度(如LRU)、访问频率(Web对象流行度,如LFU)和Web对象大小等因素[4].但是,以往的研究多针对Web缓冲器,很少针对Web服务器.具体到缓冲容量较小的Web服务器,久远度往往较之访问频率更为重要[4],而访问频率反映的是Web对象在较长时间尺度上的访问可能性,因此不适用于Web服务器.另外,Web对象大小不一(有时差别非常大),如果在替换策略中倾向于保留小对象,即其他条件相近时优先替换大对象,则可以留出空间以容纳更多的小对象.在服务器中,优先替换大对象的理由是,大对象的磁盘读速远比相同大小的多个小对象的读速快,这是因为在磁盘的访问中,寻道操作引入的延迟相当大,磁盘读速则相对比较快.例如,一个普通硬盘的平均寻道时间大约为8 ms,而一个普通磁盘的峰值读速就能够达到

20 MB/s,这样读 16 个 4 kB 的文件需消耗时间大约 131 ms,而读一个 64 kB 的文件时间则仅需 11 ms.

GreedyDual-Size (GDS)[5] 和 PSS[6] 是 2 个比较好的综合了时间久远度和 Web 对象大小的缓冲替换算法,其中 PSS 的计算开销更小,比较适用于运算负担较重的 Web 服务器.

PSS 将对象放置在 N 个 LRU 队列中,队列按照对象大小呈现"金字塔"排列,队列 i 的对象大小在 2^{i-1} 和 2^i-1 之间. 在此,本文定义对象 o 的大小为 S_o,则 o 在队列 i 中,当且仅当 $2^{i-1}\leqslant S_o<2^i-1$. 替换时,选择每个队列中的最长久的对象进行比较,选择其中 $S_o\Delta T_o$ 值最大的对象,而 ΔT_o 是对象 o 上一次被访问以来的累积访问次数,它反映了对象 o 的久远度. 在实现中,S_o 的取值是对象 o 的块数,这比取对象 o 的大小更易于优化计算,也不影响算法的有效性.

3　实　验

在本文实验中,Web 服务器运行在一个普通 PC 机上,它有一个 2.4 GHz 的 Intel P4 处理器,2 块 Intel E1000 千兆网卡(双绞线接口),配有 512 MB 的物理内存和一个 10 GB 的 Seagate 5400 r/min 的硬盘. 另外,还有 2 个 PC 机作为测试客户机,在硬件配置上与服务器相同. 测试服务器的 2 个网卡通过双绞线分别与 2 台测试客户机相连. 3 台机器都运行 Linux 2.6,其中服务器的运行内核为 2.6.15.3 版本,而客户机运行 Redhat 的 Linux 2.6.9 版本. 实验还对比了 ACC 方法优化前、后 2 个 Flash(以 Ruan 修改过的版本[2] 为基础)的性能,为了保证实验的客观准确性,测量时关闭了无用的进程,内核只有一个可加载模块 e1000.ko,运行中 Flash 关闭了日志功能. 由于 Linux 系统自身存在内存开销,特别是网络 I/O 的内存开销,所以实验中又设定 Flash 程序的缓冲空间阈值为 420 MB,超过该阈值 ACC 将主动释放部分缓冲.

可以看出,实验的硬件配置比较差,特别是硬盘,其最大读速为 12 MB/s,这样做是为了使硬件配置与 Ruan 的实验硬件[2,7] 相当,便于进行结果对比和参照.

实验采用测试程序 flexiclient 进行性能测试[7],该工具可产生与 SPECWeb99[8] 一致的遵从 Zipf 分布规律的负载. 所谓的 Zipf 分布规律,即 Web 对象的访问概率分布在第 n 位流行的对象具有 $1/n^{\alpha}$ 的权重,在此 $\alpha=1$. 在测试平均吞吐率时,先测试 200 s 以"预热"缓冲,然后测试 200 s 取其平均值.

3.1　Flash 性能和磁盘 I/O 阻塞

通过实验观察到,Flash 性能严重地受到了磁盘 I/O 阻塞的影响. 在 720 MB 数据负载下,Flash 修改前、后的吞吐率随时间的变化(ACC 表示采用应用控制缓冲方法修改的 Flash)如图 2 所示. 在运行开始的约 100 s 之内,由于 Linux 对磁盘文件的缓冲是逐步的,所以 Flash 性能经历了一个明显的"爬坡"期,吞吐率逐步上升,但当吞吐率达到峰值后,性能急剧下降,造成这种性能下降的原因应当是主进程阻塞. 为了确认这一点,实验记录了 Flash 运行期间的系统 CPU 时间分布情况(见图 3),其中 CPU 磁盘 I/O 等待时间为 CPU 等待 I/O 完成时间占 CPU 运行总时间的百分比. 显然,100 s 以后的性能下降是 Flash 阻塞在磁盘 I/O 中所致,之所以如此,是因为 Flash 在"跟踪"内核的缓冲状态时严重偏离了实际情况.

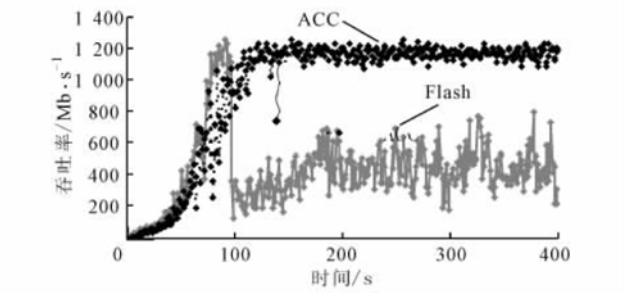

图 2　720 MB 数据负载下 Flasch 吞吐率随时间的变化

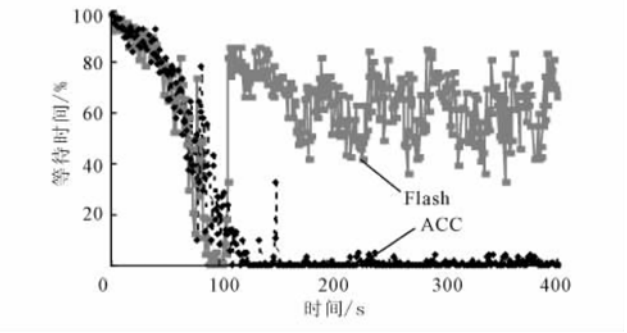

图 3　Flash 运行中的 CPU 磁盘 I/O 等待时间

同时,观察图 3 的 ACC 曲线可以看出,应用了 ACC 方法的 Flash 在缓冲"满"后,其 CPU 空闲等待时间一直维持在一个非常低的水平,这说明 ACC 方法有效解决了 Flash 主进程的阻塞问题. 从图 2 也可以看出,ACC 方法带来了吞吐率的大幅度提升,但缓冲的作用如此明显,这应与 Web 访问的 Zipf 分布特点(这里 $\alpha=1$)密不可分.

Ruan 在对比了 Flash 运行于 FreeBSD 和 Linux 2 个系统的吞吐率之后，推测 Linux 在文件系统方面逊色于 FreeBSD[7]. 根据本文实验，可以认为这很可能是由于 Linux 的磁盘缓冲替换算法与 LRU 差别较大造成的.

3.2 多种优化方式的比较

在多种优化方法下进行了 Flash 吞吐率的实验测量，结果如图 4 所示. 其中，LRU 指的是 ACC 方法中的缓冲替换策略采用 Flash 原有的 LRU 算法，RA 指的是在辅助进程的“读磁盘”程序中应用磁盘预读等优化方法，PSS 指的是 ACC 方法中的缓冲替换策略采用 PSS 算法，数据负载大小是实验测量中请求到的网页文件大小的总和.

从图 4 中可以看出，应用 ACC 方法后 Flash 吞吐率有了极大的提高，当数据负载稍大于物理内存空间时（480 MB 数据负载），也大约提高了 24.4 %；当数据负载超出物理内存空间 2～3 倍时，吞吐率提高了 3～4 倍. 磁盘预读也能提高 Flash 的吞吐率，在大数据负载下，LRU 采用预读提高的吞吐率达到 20%～30%之多，其原因是预读提高了磁盘 I/O 性能. 对比 2 种缓冲替换策略，可以明显看出 PSS 比之 LRU 在处理 Web 对象的缓冲方面更优，特别在大数据负载下，吞吐率提高达 10%～47%. 采用 ACC 方法后，Flash 在大数据负载下的性能也比较平稳，例如在 2 倍于可用内存的数据负载（960 MB）下的吞吐率比之最大吞吐率仅下降约 16%，在 3 倍的数据负载下则下降不到 50%.

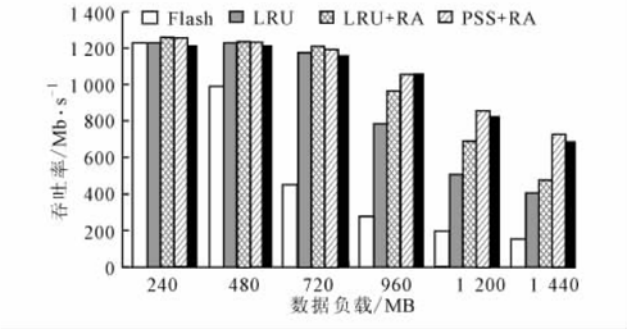

图 4 不同优化方式下 Flash 吞吐率

Web 服务器性能的另外一个重要指标是延迟，图 5 显示了不同优化下的 Flash 延迟累积概率分布 F，其对应的数据负载大小为 960 MB. 采用 ACC 之后，延迟也明显降低，超过 50%的访问延迟在3 ms 之内，超过 95%的在8 ms之内，超过 99%的在15 ms 之内，而原 Flash 的延迟分别约为 4 ms、50 ms和 100 ms，可见在解决了磁盘 I/O 阻塞问题的同时，也改善了服务器的延迟. 另外，值得注意的是磁盘预读可提高服务器吞吐率，但要为延迟付出一点代价. 如图 5，LRU 的 50%访问的延迟不到 1 ms，而 LRU＋RA 则超过 2 ms；LRU 的 90%访问的延迟不到 2 ms，而 LRU＋RA 的约为6 ms. 这说明，磁盘预读加大了调度与合并磁盘 I/O 的请求量，内核在某些 I/O 请求上有意延迟服务，以获得更大的吞吐率.

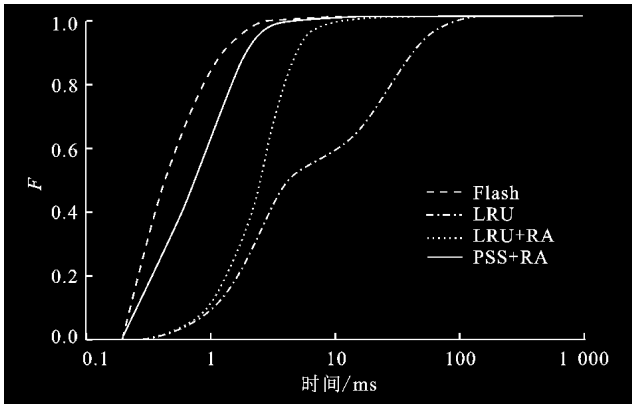

图 5 960 MB数据负载下不同优化的 Flash 延迟累积分布

3.3 Web 服务器的性能对比

将 Flash 修改前、后的性能与 Apache、Zeus 等 Web 服务器的性能进行对比. Apache 是目前 Internet 上使用最广泛的 Web 服务器，新的 Apache 2.0 由于采用了多进程、多线程混合的服务器模型，其性能有了很大提高，而 Zeus 则一直以高性能 Web 服务器著称. 实验测量中的 Apache 采用 Worker 多处理（这是 Apache 在 Linux 平台下性能最好的多处理方式）模块，Apache 和 Zeus 都参照有关说明进行了编译优化和配置优化，如禁止无用模块、关闭日志等，实验结果如图 6 所示.

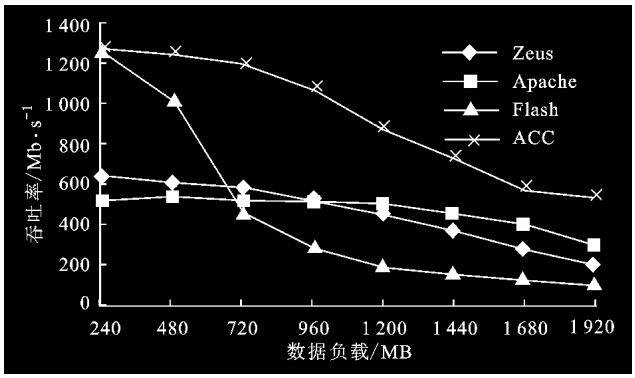

图 6 应用 ACC 方法前、后 Flash 与其他 Web 服务器的性能对比

从图 6 可以看出，由于 Apache 服务器基于多进程、多线程的混合模型，存在进程与线种切换等开销，所以其主要性能瓶颈是 CPU. 在小数据负载下，Apache服务器的性能是几种服务器中最低的，但在

大数据负载下,多进程、多线程混合模型的隐藏磁盘访问延迟的能力发挥了作用,此时它的性能保持稳定,并优于 Zeus 和 Flash. Zeus 虽然在小数据负载下的性能优于 Apache,但比之 Flash 的性能则相差甚远,而在大数据负载下,其性能下降也不如 Flash 那样明显. 应用 ACC 方法后,Flash 充分发挥了 AMPED 的优势,无论数据负载是大是小,其性能远远超出其他服务器.

4　结　论

本文提出了应用控制缓冲方法,并以 Flash 为例实现并验证了 ACC 的有效性. 实验表明,在大数据负载下的 ACC 能够大大提高 Web 服务器的性能.

虽然本文的研究集中于静态页面,但有两种途径可以解决 Web 服务器的动态页面问题:一是可以将一个 Web 站点的动态页面和静态页面分流,采用专门的服务器来处理、计算要求很高的动态页面,再利用 ACC 方法来加速静态页面的处理;二是通过磁盘来缓冲动态页面,将之转化为静态页面的处理.

ACC 方法不单适用于 AMPED 模型的服务器,也适用于其他模型,即便服务器通过采用异步方式工作来消除磁盘 I/O 的阻塞问题,但它还可以通过更优化的磁盘缓冲替换算法来提高缓冲命中率,以获得更好的性能.

参考文献:

[1] Pai V S, Druschel P, Zwaenepoel W. Flash: an efficient and portable web server [C]// 1999 Annual USENIX Technical Conference. Monterey, USA: USENIX, 1999: 199-212.

[2] Ruan Yaoping, Pai V S. The origins of network server latency and the myth of connection scheduling [C]// Joint International Conference on Measurement and Modeling of Computer Systems. New York: Association for Computing Machinery, 2004: 424-425.

[3] Cao Pei, Felten E W, Karlin A R, et al. Implementation and performance of integrated application-controlled file caching, prefetching, and disk scheduling [J]. ACM Transactions on Computer Systems, 1996, 14(4): 311-343.

[4] Podlipnig S, Boszormenyi L. A survey of Web cache replacement strategies [J]. ACM Computing Surveys, 2003, 35(4): 374-398.

[5] Cao Pei, Irani S. Cost-aware WWW proxy caching algorithms [C]// Proceedings of the USENIX Symposium on Internet Technologies and Systems. Monterey, USA: USENIX, 1997: 193-206.

[6] Aggarwal C, Wolf J L, Yu P S. Caching on the world wide web [J]. IEEE Transactions on Knowledge and Data Engineering, 1999, 11(1): 94-107.

[7] Ruan Yaoping, Cohen M S, Pai V. Finding speed bumps: web server performance analysis and anomaly detection via wide spectrum microbenchmarking [EB/OL]. [2005-12-15]. http://www.cs.princeton.edu/~yruan/.

[8] Standard Performance Evaluation Corporation. SPEC Web99 benchmark [EB/OL]. [2005-12-15]. http://www.spec.org/osg/web99/.

(编辑　苗凌)