第39卷第11期 中国科学技术大学学报 Vol. 39, No. 11
2009年11月 JOURNAL OF UNIVERSITY OF SCIENCE AND TECHNOLOGY OF CHINA Nov. 2009
文章编号:0253-2778(2009)11-1207-05

一种自适应的流媒体服务器缓存调度算法

李 靖,杨 坚,奚宏生

(中国科学技术大学网络传播系统与控制联合实验室 网络传播系统与控制安徽省重点实验室,安徽合肥 230027)

摘要:在分析现有的流媒体缓存技术优缺点的基础上,提出一种采用间隔缓存的自适应混合型间隔缓存算法(adaptive hybrid interval cache,AHIC).该策略充分考虑用户行为模式和影片冷热度对缓存策略性能的影响,采取分段缓存和间隔缓存相结合的方式,通过周期性地统计影片的流行度和用户访问行为,实现了缓存的接纳和替换.仿真实验表明,与传统的间隔缓存策略相比,AHIC 策略能够有效的提高缓存的字节命中率.

关键词:流媒体服务器;间隔缓存;自适应
中图分类号:TP37 **文献标识码**:A

An adaptive caching scheme for multimedia streaming servers

LI Jing, YANG Jian, XI Hong-sheng

(*Joint Lab of NCSC, Key Lab of Anhui NCSC, USTC, Hefei* 230027, *China*)

Abstract: A new cache management algorithm, adaptive hybrid interval cache (AHIC) was proposed based on an analysis of the advantages and disadvantages of the existing multimedia streaming caching schemes. The new scheme considered the influence of user activities as well as the popularity of multimedia objects and proposed a hybrid scheme which blended some speciality of segment cache and interval cache. Caching admission and replacement algorithms based on the segment popularity estimated periodically were implemented. Simulation results show that the AHIC scheme can effectively improve the byte hit ratio of multimedia streaming systems compared to the traditional interval cache policy.

Key words: streaming server; interval cache; adaptive

0 引言

近几年来,随着网络带宽的快速发展,在 Internet 上支持视频点播服务(video-on-demand, VoD)变得越来越流行.在一个 VoD 系统中,为了保证影片播放的连续性,必须预留足够的资源(网络带宽、磁盘带宽、CPU 运算时间等).然而随着 CPU 速度和网络带宽的飞速增长,磁盘 I/O 逐渐变成 VoD 系统服务能力的瓶颈.为了减小对磁盘 I/O 的压力,流媒体服务器中的缓存机制就变得格外重要.一种有效的缓存机制不但能够降低系统的服务延迟,而且还能减小对磁盘 I/O 的负载,增强系统的并发服务能力.

现有 VoD 系统中的缓存策略一般分为两类:基于分块的和基于流的.LRU[1],LFU[2],LRU-k[3]等是传统的分块缓存算法,这些算法操作简洁且易于实现.LFU 总是替换出使用频率最小的对象,认为使用频率越高,未来使用价值越大,但 LFU 算法存

收稿日期:2008-06-13;**修回日期**:2008-10-08
基金项目:中国高技术研究发展(863)计划(2006AA01Z114),中国科学院院长奖获得者科研启动专项基金和 41 批博士后基金资助.
作者简介:李靖,男,1982 年生,博士生. 研究方向:流媒体系统的接入控制以及调度. E-mail: jingl3@mail.ustc.edu.cn
通讯作者:奚宏生,教授. E-mail: xihs@ustc.edu.cn

在缓存污染问题. LRU 算法总是替换最长时间没有被使用的对象,但是很容易出现对象刚被替换出缓存又被访问的问题. LRU-k 算法将访问频率和最近访问时刻综合到性能函数中,虽然获得了较好的性能,但是同样具有上述两种算法的缺点;而且由于流媒体文件体积很大且大多是顺序地被访问,这些传统的基于页面请求的替换算法工作的效果很差,并不适用于流媒体系统. 文献[4]提出了一种基于流的间隔缓存策略(interval cache,IC),其原理是选择访问同一节目的两个相继的视频流的间隔进行缓存,使后继视频流请求从缓存中服务. 它仅缓存两个连续流间的间隔,前一个流所访问的内容被缓存下来,下一个对同样文件点播流直接从缓存中读取数据,这样能够保证流的连续性和实时性,同时减少了磁盘 I/O 的占用率,增大了缓存的利用率. 然而,当流媒体文件的时间长度很短或者对相同文件连续两次点播的间隔时间太长时,IC 算法的效果较差. 文献[5]针对该问题对算法进行了改进,提出的一种 generalized interval caching(GIC)算法,该算法将小文件完全缓存下来,并通过缓存替换来保证只缓存最小的间隔,从而提高缓存的利用率.

文献[6]在文献[4,5]的基础上提出了一个"虚拟间隔"的概念,通过对某部影片过去访问时间来预测对该影片下次访问的到达时刻并提前进行缓存,从而提高缓存的字节命中率. 这些基于流的间隔缓存算法一般都选择最小的间隔来缓存,但是这并不能保证系统达到最优. 文献[7]以缓存容量和磁盘带宽的均衡为原则来决定哪些间隔被缓存,哪些间隔不被缓存. 这些缓存策略都侧重于考虑如何缓存间隔使得内存的利用效率最大,很少考虑用户行为对算法性能的影响. 文献[8]提出了一种基于流行程度的间隔缓存策略,即将内存分为两部分,一部分采用经典的 interval 算法,另一部分保存最流行的几部影片的全部数据. 该策略需要系统有很大的内存空间,且没有考虑相同影片不同数据段的流行程度是不同的,也没有给出影片流行度发生变化时相应的缓存替换策略. 文献[9]提出了 interval 与 prefix 相结合的方式,但是没有给出需缓存的文件以及其分段长度,也没有给出相应的替换策略. 文献[10]通过对 HP 公司的 VoD 服务系统日志的分析表明:用户对影片的访问不是平均的,在一段时间内大量用户请求集中在少数比较流行的影片上,同时影片的流行程度是随时间不断变化的,通常随着时间的推移,影片的被访问程度会越来越低. 用户对同一部影片在不同时间段的访问量也是不同的,只有少部分用户会将影片看完,大部分用户只观看影片的头几分钟后就退出(这种情况下,基于流的缓存策略效果很差).

本文提出了一种自适应的混合型间隔缓存策略(adaptive hybrid interval caching,AHIC),将缓存分为两部分,一部分用于缓存部分热门影片的片头,另一部分用作间隔缓存. 通过定期对用户行为进行统计分析来调整缓存的分配与替换,增大缓存的字节命中率,从而提高系统的服务能力.

1 系统描述

1.1 流媒体服务器的架构

如图 1 所示,流媒体服务器是一台高性能的计算机,通过高速的 I/O 接口和磁盘阵列相连. 阵列上存储有各种码率和时间长度的影片,并能提供一定的输出带宽. 我们的流媒体服务器主要由三个模块组成:I/O 管理模块、缓存管理模块以及流化模块. I/O 管理模块负责从磁盘读取文件并送往缓存管理模块,缓存管理模块负责管理系统中的内存,流化模块负责从缓存管理模块中读取数据,并流化给客户端. 当一个客户请求到达时,流媒体服务器会从缓存中或者从硬盘中读取数据,为其进行流化服务.

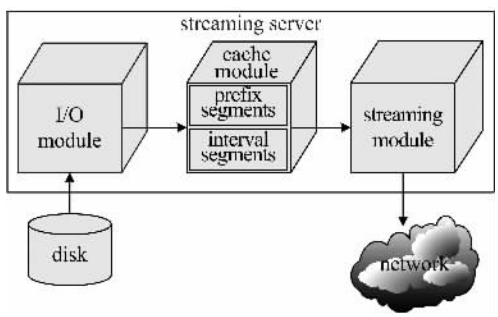

图 1 流媒体服务器结构

Fig. 1 Architecture of streaming server

1.2 间隔缓存

间隔缓存策略的基本原理是缓存访问同一节目的连续视频流请求的间隔,使后继视频流请求从缓存中服务.

如图 2 所示,当请求 S_{11} 在 t 时刻到达时,由于它是该节目的第一个点播,所以将从磁盘中读取数据进行服务. 当请求 S_{12} 在时刻 $t+b_{12}$ 到达时,由于 S_{11} 存在,因此(S_{11},S_{12})形成一个配对,流 S_{11} 称为前向流,流 S_{12} 称为后继流. S_{11} 在 $t+b_{12}$ 时刻开始缓存时间长度为 b_{12} 的数据,而流 S_{12} 先从磁盘中读取 b_{12}

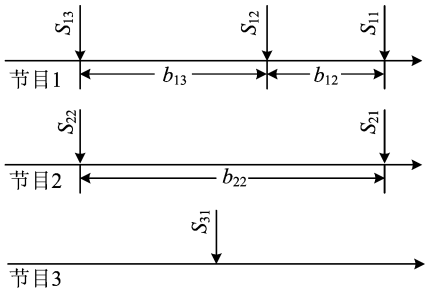

图 2　间隔缓存的基本原理

Fig. 2　Theory of interval cache

时间长度的数据后，再从流 S_{11} 的缓存中读取数据. 当流 S_{13} 在时刻 $t+b_{12}+b_{13}$ 到达后，(S_{12},S_{13}) 就形成一个配对，S_{13} 从硬盘中读取 b_{13} 时间长度的数据后，再从 S_{12} 的缓存中读取数据. S_{12} 相对 S_{11} 来说是后继流，相对 S_{13} 来说是前向流. 由于内存空间有限，所以不是所有的配对都被缓存下来，IC 算法只缓存间隔最小的那些配对. 当一个请求到达后，首先与其前向流形成一个配对，如果空闲的内存比这个配对大，那么就分配内存，缓存下这个配对，否则就从已经分配内存的配对中，找出间隔最大配对与当前配对进行比较. 若当前配对比已有配对要小则进行替换，这时被替换出来的配对将释放内存，供新形成的配对使用. 系统经过一段时间达到稳态时，缓存中的都是间隔较小的配对，间隔较大的配对都会被替换出去. 这样就保证了内存利用率最高.

1.3　Adaptive hybrid interval cache(AHIC)

由间隔缓存的算法可以看出，在间隔缓存发挥作用之前，后向流需要从磁盘中读取间隔时间长度的数据，因此当请求的间隔变大或者用户过早结束时，算法的效率会降低. 因此本文提出一种混合型的缓存策略 AHIC，将缓存分为两部分，一部分用来做常规的间隔缓存 C_{IC}，另外一部分用来缓存部分热门影片一定长度的文件头 C_{PR}，并动态的调整两部分缓存的大小.

假设系统中共有 N 部影片，用户对第 i 部影片的访问请求符合参数为 λ_i 的 Poisson 过程. 设第 i 部影片的码率为 $b(i)$，影片的长度为 $L(i)$. 定义内存的效率 e 为单位内存单位时间内输出的字节数.

若第 i 部影片的前 $L_p(i)$ 时间长度的数据被缓存在内存中，则 t 时间段内从 $L_p(i)b(i)$ 内存中输出的字节数为 $\lambda_i t L_p(i)b(i)$，内存的利用效率为

$$e=\frac{\lambda_i t L_p(i)b(i)}{tL_p(i)b(i)}=\lambda_i \tag{1}$$

假设对第 i 部影片的第 j 次点播请求到达且与其前向流的时间间隔为 $t_w^j(i)$，若采用间隔缓存算法，则在该请求整个访问期间 d_i 内($d_i\leqslant L(i)$)，服务器需要从 $t_w^j(i)b(i)$ 大小的内存中读取 $b(i)(d_i-t_w^j(i))$ 字节的数据，故内存的利用率为

$$e=\frac{b(i)(d_i-t_w^j(i))}{t_w^j(i)b(i)d_i}=\frac{1}{t_w^j(i)}-\frac{1}{d_i} \tag{2}$$

由式(2)可以看出影片点播长度 d_i 与时间间隔对间隔缓存算法的影响. 在 d_i 相对 $t_w^j(i)$ 比较大的时候，间隔缓存算法对内存的利用效率的高低由间隔大小 $t_w^j(i)$ 决定，$t_w^j(i)$ 越小，缓存算法的效率越高. 当 d_i 变小的时候，间隔缓存算法的效率会变差. 由式(1)，(2)可以看出，若 $t_w^j(i)\leqslant\frac{1}{\lambda_i}$，即与前向流的时间间隔小于平均间隔的请求，采用间隔缓存算法能够获得更高的字节命中率. 若 $t_w^j(i)>\frac{1}{\lambda_i}$，采用间隔缓存的方式就不如缓存部分热门影片的头部对内存的利用效率高. 因此对于点播到达率最高的 k 部影片，求出它们请求到达之间的平均间隔 $T_x=\left(\sum_k\frac{1}{\lambda_k}\right)\Big/K$. 当对影片 i 的第 j 次请求到达时，只有当 $t_w^j(i)\leqslant T_x$ 时才进行 IC，其他的内存空间用来缓存请求到达率最高的几部影片的片头.

由于对第 i 部影片的点播服从到达率为 λ_i 的 Poisson 过程，因此两次点播的时间间隔服从参数为 λ_i 的负指数分布，即 $P(t_w(i)\leqslant t)=F_i(t)=1-\mathrm{e}^{-\lambda_i t}$(其无条件概率密度函数为 $f_i(t_w(i)=t)=\lambda_i\mathrm{e}^{-\lambda_1 t}$). 因为系统只缓存间隔小于 T_x 的配对，所以其条件概率密度函数为

$$f_i(t_w=t\mid t_w\leqslant T_x)=(f_i(t_w=t,t_w\leqslant T_x))/F_i(T_x) \tag{3}$$

因此缓存下来的平均间隔为

$$t_w(i)=\int_0^{T_x}tf_i(t_w=t\mid t_w\leqslant T_x)\mathrm{d}t=\frac{1}{F_i(T_x)}\int_0^{T_x}t\lambda_i\mathrm{e}^{-\lambda_i t}\mathrm{d}t \tag{4}$$

故第 i 部影片所占用的缓存为 $\lambda_i L(i)t_w(i)$，因此用于间隔缓存所需要的间隔缓存大小为

$$C_{\mathrm{IC}}=\sum_{i=1}^{N}\lambda_i L(i)t_w(t) \tag{5}$$

2　缓存策略和替换算法

由上面的分析可以看出，在可用内存较小，请求

到达率较高的情况下,IC 的字节命中率是很高的.当可用内存变大时,对于点播率比较高的影片,缓存一段时间长度的片头是有效的.因此在本文的算法中,将定期统计影片请求到达率的高低,并缓存到达率最高的影片的片头.直接计算式(5)的复杂度比较高,因此算法采用一些近似的方法来代替.

缓存算法保存的一些全局变量,同时对每一部影片都保持着一个数据结构,如表 1 所示.

表 1 算法保存的全局变量

Tab. 1 Global parameters

参数	意义
T_x	表示允许采用间隔缓存的最大时间间隔
C	可用缓存的总大小
C_{IC}	当前周期可用来做间隔缓存的总大小
C_{free}	当前系统中空闲内存大小
$L_p(i)$	第 i 部影片需要缓存的片头的长度
$T_l(i)$	第 i 部影片最近一次被点播的时间
$L_c(i)$	第 i 部影片实际缓存的片头的长度
$T_{\mathrm{cast}}(i)$	第 i 部影片在本周期内被点播的次数

2.1 参数更新

由于用户访问行为是随时间变化的,因此需要定期更新相关参数.用 T 表示更新周期,T_c 表示当前时刻.当更新周期 T 到达时:

① 将所有影片的 T_{cast} 从大到小进行排序,并取最大的 k 个,并通过式 $T_x=\left(\sum_{i=1}^{k}\frac{T}{T_{\mathrm{cast}}(i)}\right)\Big/k$ 计算出允许采用间隔缓存的最大时间间隔;

②计算将当前时刻系统中所有间隔小于 T_x 的间隔之和,即为 C_{IC},以粗略地表示当前周期所需最小的间隔缓存的大小.同时采用指数平滑的方式估计下一个周期所需要的间隔缓存的大小,即

$$\hat{C}_{\mathrm{IC}}^{(N+1)}=\alpha\hat{C}_{\mathrm{IC}}^{(N)}+(1-\alpha)C_{\mathrm{IC}}^{(N)} \tag{6}$$

式中,$\hat{C}_{\mathrm{IC}}^{(N)}$ 当前周期的估计值,$C_{\mathrm{IC}}^{(N)}$ 为当前周期的实际值;

③计算可以用来缓存文件头部的内存大小 $C-C_{\mathrm{IC}}$.将其平均分配给最热门的 k 部影片(即 T_{cast} 最大的 k 部影片),更新这 k 部影片的 L_p 值.

2.2 点播请求到达

当对一部影片的点播请求到达时,若存在前向流:①若 $b(i)(T_c-T_l)\leqslant C_{\mathrm{free}}$,则将这两路流形成一个间隔缓存,其前向流从 $\max(T_c,T_l+L_c)$ 时刻开始缓存数据,同时更新 C_{free}.②若 $b(i)(T_c-T_l)>C_{\mathrm{free}}$ 且 $T_c-T_l<T_x$,则对缓存中的配对从大到小进行排序.若缓存中存在比 T_x 大的配对就进行替换,若不存在,则从 k 部影片所存的片头中,取出足够的缓存使之形成一个间隔缓存,其前向流从 $\max(T_c,T_l+L_c)$ 时刻开始缓存数据,同时更新片头缓存被替换的影片的 L_c 值.③若 $b(i)(T_c-T_l)>C_{\mathrm{free}}$ 且 $T_c-T_l>T_x$,则不进行间隔缓存.④更新 T_l 与 T_{cast}.

2.3 点播结束

当对一部影片的点播请求结束时:①若不存在前向流,也不存在后向流,直接结束;②若仅存在前向流,则删除与其前向流之间的间隔,释放缓存;③若仅存在后向流,则删除与后向流之间的间隔,释放缓存;④如果同时存在前向流与后向流,则将其前向流和后向流合成一个配对;⑤判断缓存片头的前 k 部影片的 L_p 与 L_c 值.若 $L_p>L_c$,则分配相应的内存.

3 仿真与分析

为了使仿真更加接近实际情况,测试数据全部从中国科学技术大学的 VoD 服务器日志中取得,而不是采用常用的 Poisson 过程和 ZIPf 分布的假设.本文用字节命中率作为衡量算法性能的指标.字节命中率是指从缓存输出的字节数与总的流量的比值.字节命中率越高,意味着从缓存中输出的字节数越多,对磁盘 I/O 的压力越小,缓存算法的效率越高.通过对日志数据的分析,测试比较 AHIC 算法以及间隔缓存算法在给定缓存下的性能以及缓存大小对算法性能的影响.

仿真实验随机选择了 2007-05-21 的数据.当天共接收点播请求46 076次,有3 581部不同影片被点播.测试中,我们每 10 min 记录一次当前并发的用户数量,同时也记录这 10 min 内缓存的字节命中率.图 3 给出了在缓存大小为 1 GB 时,AHIC 算法

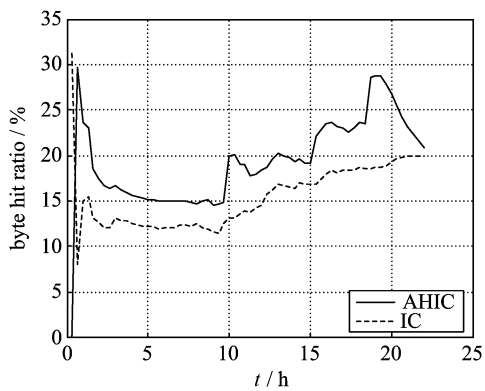

图 3 字节命中率

Fig. 3 Byte hit ratio

与间隔缓存算法的字节命中率随时间的变化情况. 由实验结果可以看出,AHIC 算法比 IC 算法的缓存字节命中率要高出 5%左右,两种算法的字节命中率在当天的凌晨和晚上这两个时间段相对较高. 在开始阶段较高是因为是从 0 点开始进行仿真,缓存都是空闲的,因此命中率较高,但是随着点播次数的增多而下降. 在晚上较高是因为这个时间段并发的用户数比白天要大,在对日志的分析中我们发现晚上 8～10 点,平均的并发数达到近6 000个,因此缓存算法的命中率也相应较高.

本文进一步比较了缓存大小对算法性能的影响. 图 4 给出了 AHIC 算法与间隔缓存算法在缓存大小从 512 MB～4 GB 时,其相应的字节命中率的变化情况. 由仿真结果可以看出,间隔缓存算法在内存空间大于 2 GB 后,其性能的提升很小. 其原因可能是系统中比较小的间隔全部被缓存了,只能缓存许多大的间隔,使得缓存的利用效率降低. 我们进一步分析了间隔缓存算法对 23∶00～23∶59 这段时间约3 000多个点播请求时缓存的状态,发现其最多只会用到 600 MB 左右的内存. 而 AHIC 算法在缓存空间比较小的时候,会缓存系统中小的间隔,当缓存空间变大时,AHIC 并不会去继续缓存那些大的间隔,而是把热门影片文件头部缓存下来,这样可以有效地提高缓存的字节命中率. 因此,AHIC 算法比经典的间隔缓存算法对用户行为有更好的适应性.

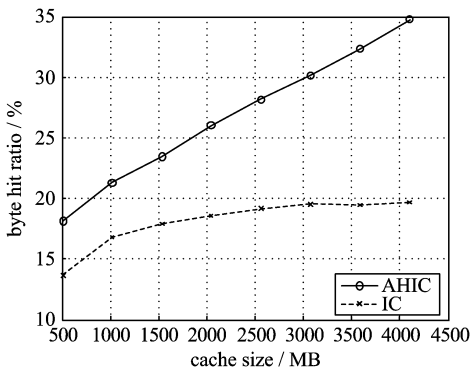

图 4　缓存大小的影响

Fig. 4　Impact of cache size

4　结论

本文在考虑实际系统中用户点播行为的基础上,结合经典的间隔缓存算法,提出了一种基于用户行为的缓存调度算法. 该算法采用缓存热门影片片头和间隔缓存相结合的方式,并通过周期性地更新参数来动态调整缓存间隔和缓存片头内存的大小,使得系统在用户请求到达率较高、影片点播间隔较小时,采用间隔缓存提高内存的利用效率;而在用户请求到达率较低、点播的间隔较大时,增大缓存片头的空间,使得算法对内存的利用效率增大. 仿真结果表明,该算法可以达到比经典的间隔缓存更高的字节命中率.

参考文献(References)

[1] Robinson J T, Devarakonda M V. Data cache management using requency-based replacement[C]// Proceedings of SIGMETRIC on Measuring and Modeling of Computer Systems. Colorado, 1990:134-142.

[2] Alghazo J, Akaaboune A, Botros N. SF-LRU cache replacement algorithm [C]// Proceedings of the Records of the 2004 International Workshop on Memory Technology, Design and Testing. Washington: IEEE Computer Society, 2004: 19-24.

[3] Shin S W, Kim K Y, Jang J S. LRU based small latency first re placement (SLFR) algorithm for the proxy cache [C]//Proceedings of IEEE/WIC International Conference on Web Intelligence. Halifax, Canada: IEEE Press, 2003: 499-502.

[4] Dan A, Dias D M, Mukherjee R, et al. Buffering and caching in large-scale video servers[C]// Proceedings of the 40st IEEE Computer Society International Conference. San Francisco: IEEE Computer Society, 1995: 217-224.

[5] Dan A, Sitaram D. A generalized interval caching policy for mixed interactive and long video workloads [C]// Proceedings of Multimedia Computing and Networking. San Jose: IEEE Press, 1996, 2 667: 344-351.

[6] Cho K, Ryu Y, Won Y, et al. Virtual interval caching scheme for interactive multimedia streaming workload [J]. Computer and Information Sciences, 2003, 2 869: 276-283.

[7] Lin W, Yong L S, Leong Y K. A client-assisted interval caching strategy for video-on-demand systems [J]. Computer Communications, 2006, 29(18): 3 780-3 788.

[8] 余宏亮,陈婧,李毅, 等. 一种 IA-64 架构下的大规模流媒体服务器缓存调度算法[J]. 计算机研究与发展, 2006, 43(4): 729-737.

[9] 佘堃,杨四铭,周明天. 一种多媒体服务器混合缓存策略[J]. 小型微型计算机系统, 2005, 26(1): 143-146.

[10] Cherkasova L, Gupta M. Analysis of enterprise media server workloads: access patterns, locality, content evolution, and rates of change [J]. IEEE/ACM Transactions on Networking, 2004, 12(5): 781-794.