文章编号:1001 - 9081(2008)03 - 0823 - 03

移动网络中流媒体统计缓存算法设计

崔欣辰[1],曲 宁[1],吴晓男[2]
(1. 海军航空工程学院 兵器科学与技术系,山东 烟台 264001; 2. 海军航空工程学院 控制工程系,山东 烟台 264001)
(cuixinchen@tom.com)

摘 要:为了改进移动环境下流媒体的数据访问技术,提出了一种可以根据对先前数据的统计所得结果预测用户在移动网络系统单元中的移动趋向的新算法,包括 Cache 块排定算法和 SAA* 搜索算法。并通过模拟实验验证了该算法的有效性。

关键词:缓存算法; 流媒体; 移动网络
中图分类号: TP393 **文献标志码**:A

Statistical buffering for streaming media in mobile networks

CUI Xin-chen[1], QU Ning[1], WU Xiao-nan[2]
(1. *Department of Enginery Weapon Science and Technology, Naval Aeronautical Engineering Institute, Yantai Shandong* 264001, *China*;
2. *Department of Control Engineering, Naval Aeronautical Engineering Institute, Yantai Shandong* 264001, *China*)

Abstract: To investigate streaming media access in mobile networks, a new mechanism that made use of prior knowledge to predict the trend of user movement among cells was proposed, which included cache block scheduling and SAA* search algorithm. Experimental studies show that, this algorithm can obtain good performance on buffering streaming media data.

Key words: buffering algorithm; streaming media; mobile networks

由于移动网络设备受其有限的存储空间的限制使其很难有一个较大的高速缓存,使流媒体数据访问这一问题变得越来越重要。在蜂窝式移动网络系统中一个比较可行的解决方法是在基站中对流媒体数据进行缓存,作为移动设备的“高速缓存”使用。然而当一个移动网络设备(如笔记本电脑,PDA)原来在一个基站覆盖区内,然后移动到另一个基站覆盖区内时,为使移动网络用户能够连续、流畅的欣赏流媒体形式的电影,被高速缓存的数据也应当随之被移动至相应的基站,这就需要一种切换机制[1]在基站之间进行的数据缓存和数据转移。有效的切换算法可以提高蜂窝移动网络的容量和 QoS[2]。如果移动网络设备拥有一个小的本地 Cache,基站的缓冲器将作为二级 Cache,如果移动设备没有本地 Cache,基站的 Cache 将直接作为移动网络设备的 Cache。然而,如何有力的在较多数量的基站间维持缓冲器(从移动用户角度看是“高速缓存块”)是一个具有挑战性的问题。

1 系统建模

如图 1 所示,是一个三级的移动网络分级结构模型。流媒体服务器(用 S_i 表示),提供 Internet 上的多媒体服务,它将同时处理数以万计的请求。移动网络设备通过 N 个基站连接到 Internet 上,分别记作 $B_1, B_2, \cdots, B_N$。移动网络设备通过与其蜂窝单元所属的 B_i 与整个网络连接,于是 B_i 就承担起了将用户的请求传送给指定流媒体服务器 S_i[1]。S_i 处理请求并经由 B_i 传送被请求的内容。如果 B_i 已经缓存了用户所请求的数据,那么,该请求不必被传送给 S_i。当一个移动设备从一个单元移动至另一单元时,新的基站决定是否移动移动设备(用户)在原来基站 Cache 块中缓存的数据。基站之间通过有线网络进行通信。

每个基站拥有两种类型的数据:“缓存数据表”和“未响应请求数据表”。缓存数据表由基站 B_i 上的一系列 Cache 块 $C_1, C_2, \cdots, C_{Ki}$ 组成。其中每个 $C_p(1 \leqslant p \leqslant Ki)$ 由四部分组成:Req(用户的请求),N(B_i 基站中请求与 $B_i.Req$ 相同的用户数),Len(流媒体缓冲区长度)和 $Data$(流媒体缓冲数据)。基站 B_i 单元之内,请求未被缓存媒体数据的其他移动用户记录在 B_i 基站的未响应请求数据表中,以 $U_1, U_2, \cdots, U_{Li}$ 表示。这里 $U_q(1 \leqslant q \leqslant Li)$ 由两部分组成:Req 和 N。通过使用两个哈希表实现对缓存数据表和未响应请求数据表的快速访问。

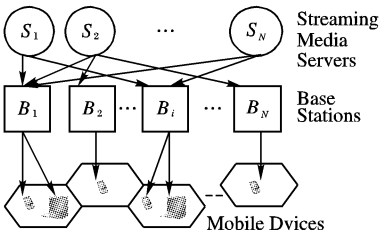

图 1 三级分级结构模型

如果基站 B_i 认为其有必要缓存一个数据块,它将计算相应的 Req 值,并将其广播至所有其他的基站。所有收到广播信息的基站将比较所收到的信息与本地记录的 Req:如果基站 B_j 中存在 $B_j.C.Req(1 \leqslant k \leqslant K)$ 等于所收到的 Req 值,则判定基站 B_j 拥有 B_i 所请求的 Cache 块,于是基站 B_j 回复一个确认信息给基站 B_i 允许其复制所请求的 Cache 块。如果 B_i 没有收到任何的回应信息,B_i 将需要与相应的媒体服务器联系以获得所需的数据块。

我们采用两种广泛使用的标准:字节命中率(Byte Hit Ratio, BHR)[3] 和平均服务延迟(Average Service Delay, ASD)[4]来评价我们的算法。BHR 定义为从缓存中获得的比特数与所有用户所请求比特总数之比。假设有 R 个移动网络设备,记为 $A_1, A_2, \cdots, A_R$,式(1)给出了在一个时间区间[$t1$,

收稿日期:2007 - 08 - 13;**修回日期**:2007 - 12 - 05。

作者简介:崔欣辰(1978 -),女,山东招远人,讲师,硕士,主要研究方向:网络、多媒体数据库; 曲宁(1972 -),男,山东龙口人,副教授,主要研究方向:网络、信息安全; 吴晓男(1973 -),女,吉林吉林人,讲师,硕士,主要研究方向:自动控制。

$t2]$ 中,BHR 的计算方法:

$$BHR = \frac{\sum_{i=1}^{N}\sum_{p=1}^{K}(B_i.C_p.Len \cdot B_i.C_p.N)}{\sum_{x=1}^{R}Q(A_x,T)} \quad (1)$$

其中,$Q(A_x,T)$ 表示在时间区间$[t1,t2]$ 中,移动设备 A_x 所请求的字节总数。

式(2)给出了移动设备平均服务延迟时间 ASD 的计算方法:

$$ASD(A_x) = \frac{1}{R}\sum_{x=1}^{R}\left[\frac{\overline{B}(A_x,T)}{b(A_x,T)} - T\right]^{+} \quad (2)$$

其中,$b(A_x,T)$ 表示被请求的服务器为 A_x 提供服务时的带宽。在时间 T 内,当 A_x 所请求的数据块没有被任何基站缓存的情况下,$\overline{B}(A_x,T)$ 为所请求的字节数;否则,$\overline{B}(A_x,T)$ 为 0。所以函数 $u = [v]^{+}$ 表示如果 $v < 0$,则 $u = 0$;否则 $u = v$。

统计缓存机制的目的在于当在基站之间移动数据块时,力争取得最大的 BHR 值,同时确保一个较小的 ASD 值。

2 统计缓存算法设计

2.1 Cache 块的排定算法

如前面所提到的获得最大的 BHR 指标是通过在所有的基站间将 Cache 块进行有效的移动。Cache 块寻址排定算法如下程序所示。移动设备 A_x 访问基站 B_m 中的 Cache 块,在使用完该块之前移动至基站 B_n 的单元中。而且,程序中的 *Algorithm_enter*(A_x,B_n) 和 *Algorithm_leave*(A_x,B_m) 可以在基站 B_n 和 B_m 上各自独立执行。

```
Algorithm_enter( A_x,B_n )
if ∃p ∈ [1,K_n],A_x. req = B_n. C_p. req
  increase B_n. C_p. N by 1;
else if
  Q_n - ∑(B_n. C_p. Len · B_n. C_p. N) > A_x. Len
create cache block for A_x;
else
make a decision based on SAA* (of section 2.2)
Algorithm_leave( A_x,B_m )
if ∃p ∈ [1,K_m],A_x. req = B_m. C_p. req;
  decrease B_m. C_p. N by 1;
if B_m. C_p. N = 0
  destroy B_m. C_p and free its memory;
else
  find ∃p ∈ [1,L_m] , So that A_x. req = B_m. U_p. req
  decrease B_m. U_p. N by 1;
if B_m. U_p. N =0
destroy B_m. U_p and free its memory
```

Algorithm_leave(A_x,B_m) 在系统中相对简单。不管基站 B_m 是否拥有所需的数据块只是影响到在 Cache 数据列表和未响应请求列表中的一些域,而且在必要时可以随时释放内存。

对于 *Algorithm_enter*(A_x,B_n),如果基站 B_n 中已经存在与 $B_m.C_p$ 相等的 Cache 块,移动设备 A_x 可以直接对其进行访问。如果基站 B_n 中没有所请求的 Cache 块,但是却有足够的空间来缓存一个新的数据块,那么基站 B_n 将运用前面提到的广播的方法将 Cache 块 $B_m.C_p$ 复制到基站 B_n。如果基站 B_n 中没有足够的空间来缓存一个新的数据块,那么基站 B_n 将会有两种选择:一是将移动设备 A_x 的请求加入未响应请求列表;二是删除一个或多个已有的 Cache 块,空闲出来的空间用来缓存移动设备 A_x 请求的数据块。为了最大化全局的 BHR,基站 B_n 将所有属于同一时间区间的消息集中到一起并且使用 SAA* 来作出选择。

寻找恰当的 Cache 块进行删除比较困难。最好的解决方法是从$\{B_m.C_p \mid p = 1,2,\cdots,K_n\}$ 中按条件 $\sum_{k=1}^{Y}B_n.C_k.Len > A_x.Len$ 来找到一个集合 $C_g = \{C_k \mid k = 1,2,\cdots,Y\}$,使得 $\sum_{k=1}^{Y}(B_n.C_k.N \cdot B_n.C_k.Len)$ 最小。

2.2 SAA* 搜索算法

每次当一个基站发现它没有缓存被请求的数据块时将面临两种选择。所以如果在时间 T 内出现 k 次这样的情况时,会有 2^k 种选择。这个决策过程可以转化为在完全二叉树 G 中具有指数复杂度的搜索问题。在此完全二叉树中从一个节点到其直接子节点的代价函数可以定义为变化的 BHR 值。由上下文,统计缓存算法的核心问题归结为寻找从根节点 *root*(s) 到完全二叉树 G 叶子节点的最短路径问题[7]。SAA 搜索是基于 SA 搜索的[5],是有望解决 n 叉树搜索问题的技术之一。SAA 搜索技术将所有从根节点到叶子节点的路径长度视为随机变量,并假设时间间隔为渐进的,有效的,连续的,固定宽度的置信区间[6](Asymptotic Efficient Sequential Fixed - width Vonfidence Intervals, ASM)。SAA 算法的优势在于不必要知道在我们模型中很难获得的评价函数 $a(n)$。这里把 ASM(本质上是一个连续的测试过程)当作统计推理的方法。假定$\{x_i\}$ 是一组独立同分布的变量。并且具有共同的分布函数 F。给定 $\delta > 0$ 且 $0 < \gamma < 1$,被 ASM 采用的停止变量 $R(\delta)$ 定义为满足式(3)的最小值。

$$R \geqslant \frac{a^2}{\delta^2}\left\{\frac{1}{R}\left[1 + \sum_{i=1}^{R}(x_i - \bar{x}_R)^2\right]\right\} \quad (3)$$

其中:

$$\bar{x}_R = \frac{1}{R}\sum_{i=1}^{R}x_i$$

$$\Phi(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty}e^{-t^2/2}\mathrm{d}t \quad (4)$$

$$a = \Phi^{-1}\left(\frac{1+r}{2}\right)$$

设 u 为$\{x_i\}$ 的平均数,式(3)具有这样的性质:$\forall F, u \in (\bar{x}_R - \delta, \bar{x}_R + \delta)$ 的概率大于 γ。这里$(\bar{x}_R - \delta, \bar{x}_R + \delta)$ 是一个固定宽度的置信区间,记为:$I(\bar{x}_R(\delta),\delta)$。从而,$\delta$ 越小,u 值越精确。

SAA 搜索过程从根节点开始,首先给定 $R = 1$,代入式(3)中验证,如果满足式(3),计算间隔 $I(\Gamma,\delta)$,否则重新设置 R 值进行验证。但是 SAA 可能会出现拒绝问题。特别地,处于同一层次的子树可能全部被 SAA 拒绝。于是这里对 SAA 进行了改进,基本算法如下:给定任意常数 δ_1,置信区间 $I(p_i,\delta_i)$,这里 p_i 是第 i 个节点当前的层次。如果 $I(p_1,\delta_1)$ 与 $I(p_2,\delta_2)$ 交叉,则取新的常量 δ_2($< \delta_1$)取代 δ_1 的位置进行重试。如此重复直到找到一个使 $I(p_1,\delta_n)$ 与 $I(p_2,\delta_n)$ 互不交叉的 δ_n。这个调整后的算法称其为 SAA* 搜索算法。SAA* 算法由两部分组成。

第一部分是统计获悉过程 Proc1,算法如下所示:

```
Algorithm_stat( )
let i = 0, left = right = 0;
find goal g by Branch-and-Bound search;
while g ≠ s
m = g's parent node
if g is the left subnode of m left = left + Subroutine_sum(g);
else
  right = right + Subroutine_sum(g);
g = m;
increase i by 1;
return c = (left - right)/i.
Subroutine_sum(N)
sum = 0;
```

```
for i = 1 to N
  sum = sum + el(i) + er(i);
return sum.
```

第二部分是 SAA* 搜索过程 Proc2,算法如下所示:

```
Algorithm_search()
given α0;
for i = 0 to D
let γ = 1 - α0 and δ = c/4;
let x1 = el(i) and x2 = er(i);
for j = 1 to 2
calculate x_R(δ) using formula (3);
let I(pj, δ) = I(x_R(δ), δ);
if I(p1, δ) does not intersect I(p2, δ)
  increase i by 1;
if p1 < p2
  prune the right branch;
else
  prune the left branch;
break;
else
  δ = δ/2;
decrease i by 1.
```

在算法的第一部分中,$e_l(i)$ 是节点 i 到其左子节点的代价函数,$e_r(i)$ 是节点 i 到其右节点的代价函数。

在算法的第二部分中,D 为搜索树 G 的深度(或称高度)。在第 1 行定义的变量 α_0,在统计推论中被称为重要性级别。事实上,α_0 越小,所得的结果越精确。然而如果 α_0 太小时,$I(p_1, \delta)$ 与 $I(p_2, \delta)$ 互不相交的可能性将变得很大,这将使第 8 行中 if 条件为假并且增加循环的次数。通过实验得出 $\alpha_0 = 0.34$ 时取得最佳精确度和效率。

运用过程 Proc1 和 Proc2,SAA* 算法执行过程如下:一个基站首先被选作为“协调者”,它的任务是收集用户移动的统计数据,做调用 Proc1 之前的初始化工作,使其获得 c 的值,然后此基站将 c 值广播至所有其他的基站。Proc2 在每个时间间隔 T 内执行一次以获得每个基站近似最佳的解决办法。在 Proc1 中,为了获得最佳的 g 值,第三行将占用多一些的时间间隔执行,这使得 Proc1 的计算代价升高。所以与 Proc2 相比,Proc1 被调用的次数少一些,这是因为在实际情况中,随着用户缓慢平滑的移动,“协调者”自然要花费相对多的时间间隔来执行 Proc1,以便更新 G 树中的分布结构估计。

由于我们的模型中没有中心服务器,所有的基站将轮流充当“协调者”。总的说来,Proc1 收集前面的移动信息并获得统计参数 c, 而 Proc2 运用此参数在决策树中执行统计搜索。理论上讲 SAA* 的复杂度略高于 $O(D \ln^2 D)$,出错的概率[7] 为 $2\alpha_0$。

3　实验结果

取某个地区的地图平面数据信息建立用户分布的模拟环境。在此模拟环境中包括 60 000 多个点,从中随机选取 8 000 个点代表用户在地图上的初始位置,每个位置记为 (x,y),其中 $x,y \in [0,10\,000]$。六边形单元覆盖了整个地图,每个单元以其中心和半径来定位。

建立好模拟环境的分布和单元之后,每个用户开始随机选择一个新的点作为目的地并向其移动。速度服从 $[10/T, 20/T]$ 的 Zipf 分布[8]。到达目的地后,用户随机选择另外一个点继续移动。每个媒体目标被标记一个值来标识其流行程度,移动用户通过这个值来选择相应媒体目标。这个值越大,移动用户访问这个媒体目标的可能性越大。我们假设访问每个媒体目标的时间服从 $[60T, 600T]$ 的 Zipf 分布[9],这样用户请求媒体目标的分布近似为 Zipf 分布(α 约为 0.5)。

为了测试缓存算法的性能,我们比较了三种缓存算法:最近最少使用算法(Least Recently Used, LRU)、最近最不常使用算法(Least Frequently Used, LFU)和 SAA* 搜索算法在相同缓存空间情况下的 BHR 值。如图 2 所示,SAA* 算法在不同大小缓存空间情况下性能均优于其他两种算法。尽管在测试中时间间隔 T = 1 s,Cache 块的大小为 40 kB,六边形单元的半径为 500,在相同缓存大小的情况下,三种算法下的 BHR 值均存在相同的关系,即:SAA* 算法高于 LFU 算法,LFU 算法高于 LRU 算法。

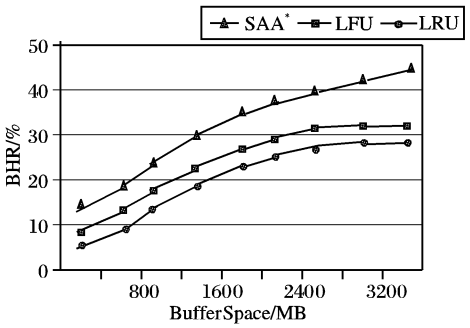

图 2　BHR 相对缓存空间比较

图 3 中展示了我们的缓存算法在缓存大小为 600 MB,随时间变化 BHR 值的变化情况。参数 c 每 80 个时间间隔被刷新一次,这里我们从另一个角度比较三种算法。如图 3 所示,随着时间增加,LRU 和 LFU 算法下的 BHR 曲线近似直线几乎没有什么变化。而 SAA* 算法的 BHR 曲线呈现“Z”字型。这种现象可以解释为:每一次当 Cache 调度程序根据历史数据统计并计算 c 值时,BHR 值会有一个增大的跳变,这是因为 SAA* 算法同时也在预测用户的下一次请求,然而随着时间过去,用户的移动,这种预测就变得不那么精确了。并且原来的 c 值已经不再能够产生出近似最佳的解决方案了,这正是 BHR 每次跳变之后又逐渐降低的原因。

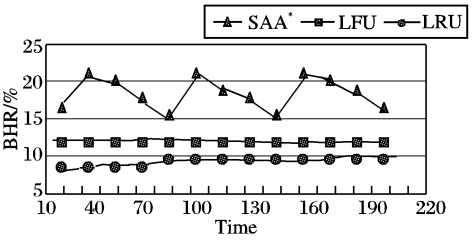

图 3　BHR 相对时间比较

同样的,我们可以从 Average Service Delay(ASD)角度将 SAA* 算法与 LRU 和 LFU 算法作比较。如图 4、图 5 所示。

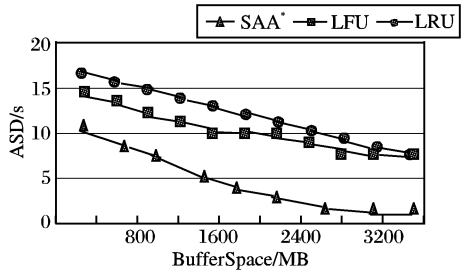

图 4　ASD 相对缓存变化比较

ASD 值受缓存空间大小和时间影响变化各不相同。可以看出 SAA* 从这个角度仍然优于 LRU 和 LFU 算法。如图 5 所示,SAA* 的 ASD 曲线受时间影响仍然呈现“Z”字状,几乎是 BHR 曲线的反转。

(下转第 828 页)

SDP,获得 SDP 后注册本机的这个服务,获得的分片由终端任务为播放器提供。

4)本站内的另一个主机上的播放器向本地数据互传客户端 S1-C2 请求组播数据 app://channel1,通过调度接口请求服务节点,获得了 S1-S 和 S1-C1,则同时向这两个节点请求数据,对方会先发送自己的分片情况,然后通过查看分片情况决定向它们请求那些数据,请求的原则是尽快满足用户的播放请求。

5)再有新的节点加入,同样会获得本站内有这个服务的所有节点,调度接口已经根据节点的负载情况作了排序,再加上在伙伴获取模块对伙伴节点表的优化,可以保证当前服务的节点是近似最优的。

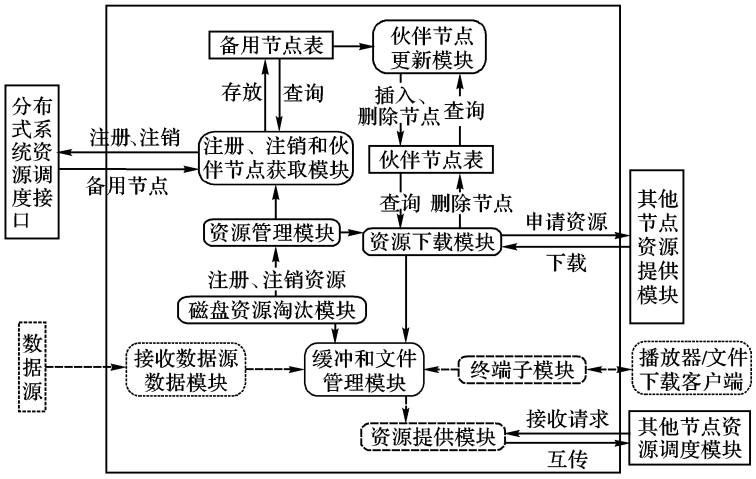

图 3 数据互传模块逻辑结构

6)当某个终端模块发现数据已经不够多时调用伙伴节点更新模块,并加入若干新的伙伴,删除速度较慢的节点。

7)服务器节点针对某一个资源允许有一定数目的常连接节点,超过此数目则会尽量让那些节点从本站内的客户端节点请求数据,本站内的客户端现在完全有能力提供这个服务。

8)某个播放器停止接收组播时,本地的数据源模块会停止接收这个资源的数据,并且注销本节点的服务,此时也不再为别的节点提供服务。

5 系统测试

测试环境:1)点播时限速为发送者速率(每秒允许发送给一个请求者的分片数)和接收者速率(每秒从一个邻接节点请求的分片数)都是 3;2)服务器分每站三台服务器,一个站为 IBM PC710;3)客户机在同一站内,共 20 台,CPU Intel 赛扬 1.7 GHz;4)片源码率为 400 kbps ~ 1 Mbps。

测试结果:本地没有所需资源,调度成功率为 95% 左右;从请求到播放点播需要 3 ~ 6 s 延时、组播需要 4 ~ 7 s 延时;点播时从每个节点下载的速率为每片 300 ~ 630 ms;组播时开始由于跟播放点同步缓冲采取加速下载,每片 20 ~ 45 ms,在同步完缓冲后保持与服务器同步的速率下载数据每秒一片;点播拖放时如果拖放点处数据已经下载则会马上开始播放,平均总的下载速率是播放速率的 4 倍以上;对于组播各个客户的播放点的开始值都是参考服务器的播放点,基本上同步,差值在 0 ~ 5 s 范围内,在播放过程中可能会由于解码速率的差异使播放点拉近或者缩小。

6 结语

本文提出的透明下载系统是一种兼有 P2P 技术和 C/S 技术优点的分布式系统数据分发技术,实现了点播、组播和文件下载功能。经过三个月的测试,本系统点播和组播功能已经稳定,文件下载功能正在测试当中,基本实现了设计目标。

参考文献:

[1] 王典荫, 刘心松. 下一代计算机系统——数字有机体[J]. 西部广播电视, 2005(1):4 - 6.

[2] 李勇, 彭宇行. 大规模视频点播磁盘 Cache 替换算法[J]. 计算机研究与发展, 2000, 37(2):207 - 212.

[3] 唐辉, 张国杰, 黄建华, 等. 一种混合 P2P 网络模型的研究与设计[J]. 计算机应用, 2005, 25(3):521 - 524.

[4] 高伟, 韩华, 代亚非. 一种 P2P 环境下分布式文件存储系统的缓存策略[J]. 计算机工程与应用, 2004, 40(30):45 - 48.

[5] 胡放明, 李俊兵, 贺贵明, 等. 对 P2P 网中发现机制的研究[J]. 计算机应用, 2004, 24(6):46 - 47.

[6] PRESLAN K, TEIGLAND D, O'KEEFE M, *et al*. An overview of the global file system[J]. IEEE Transactions on Parallel and Distributed Systems, 2000: 1252 - 1273.

[7] RFC 3016, RTP payload format for MPEG-4 audio/visual streams[S/OL]. [2007 - 09 - 10]. http://www.faqs.org/rfcs/rfc3016.html.

(上接第 825 页)

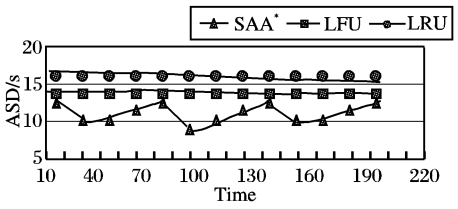

图 5 ASD 相对时间变化对照

参考文献:

[1] TANENBANM A S. 计算机网络[M]. 潘爱民, 译. 4 版, 北京: 清华大学出版社, 2004: 18 - 19, 57 - 58.

[2] 张丹丹. 无线多媒体网络中呼叫允许控制策略研究[D]. 成都: 西南交通大学, 2006: 6 - 9.

[3] PAKNIKAR S, KANKANHALLI M, RAMAKRISHNAN K R, *et al*. A caching and streaming framework for multimedia[C]// Proceedings of the eighth ACM international conference on Multimedia. New York: ACM Press, 2000: 13 - 20.

[4] JIN S D, BESTAVROS A B, LYENGAR A. Accelerating Internet streaming media delivery using network-aware partial caching[C]// Proceedings of the 22nd International Conference on Distributed Computing Systems. Washington, DC: IEEE Computer Society, 2002: 153.

[5] ZHANG B, ZHANG L. Theory and applications of problem solving[M]. [S.l.]: North-Holland, 1992.

[6] 置信区间[M/OL]. [2007 - 07 - 8 - 01]. http://www.core.org.cn/CN_NR/rdonlyres/Civil-and-Environmental-Engineering/1-017Computing-and-Data-Analysis-for-Environmental-ApplicationsFall2003/A4479E6E-4330-4656-9F7D-3099EBD7B6E7/0/class 03_15.pdf.

[7] 刘璟. 计算机算法引论——设计与分析技术[M]. 北京: 科学出版社, 2005: 7 - 9, 72 - 85.

[8] 文献信息词频分布规律——齐普夫定律[EB/OL]. [2007 - 08 - 01]. http://221.232.129.83/jpkc2007/wxjlx/Course/down/05.ppt.

[9] CHESIRE M, WOLMAN A, VOELKER G M, *et al*. Measurement and analysis of a streaming-media workload[C]// Proceedings of the 3rd conference on USENIX Symposium on Internet Technologies and Systems. Berkeley: USENIX Association, 2001: 1.