文章编号:1671-4598(2021)02-0166-05 DOI:10.16526/j.cnki.11—4762/tp.2021.02.033 中图分类号:TP393 文献标识码:A

基于高斯混合模型的 Web 代理服务器缓存替换策略

唐 榜[1],吴 珏[1],杨福军[2],杨 雷[1]
(1. 西南科技大学 计算机科学与技术学院,四川 绵阳 621000;
2. 中国空气动力研究与发展中心 计算空气动力研究所,四川 绵阳 621000)

摘要:Web 代理服务器缓存能够在一定程度上解决用户访问延迟和网络拥塞问题,Web 代理缓存的缓存替换策略直接影响缓存的命中率,从而影响网络请求响应的效果;为此,使用一种通过固定大小的循环滑动窗口提取 Web 日志数据的多项特征,并使用高斯混合模型对 Web 日志数据进行聚类分析,预测在窗口时间内可能再次访问到 Web 对象,结合最近最少使用(LRU)算法,提出一种新的基于高斯混合模型的 Web 代理服务器缓存替换策略;实验结果表明,与传统的缓存替换策略 LRU、LFU、FIFO、GDSF 相比,该策略有效提高了 Web 代理缓存的请求命中率和字节命中率。

关键词:Web 缓存;替换策略;循环滑动窗口;高斯混合模型;访问预测机制

Web Proxy Server Cache Replacement Strategy Based on Gaussian Mixed Model

Tang Bang[1], Wu Jue[1], Yang Fujun[2], Yang Lei[1]
(1. School of Computer Science and Technology, Southwest University of Science and Technology, Mianyang 621000, China;
2. Institute of Computational Aerodynamics, China Aerodynamics Research and Development Center, Mianyang 621000, China)

Abstract: Web proxy cache can solve the problems of user access delay and network congestion to a certain extent. The cache replacement strategy of web proxy cache directly affects the hit rate of cache, thereby affecting the effect of network request response. To solve this problem, using a fixed-size sliding window to extract multiple features of Web log data, and using a Gaussian mixture model for cluster analysis of Web log data, predicting that the Web object may be accessed again within the sliding window time, combining the least using (LRU) algorithm, a new cache replacement strategy of web proxy server based on Gaussian mixture model is proposed. The results show that compared with the traditional cache replacement strategies such as LRU, LFU, FIFO, GDSF, the proposed strategy effectively improves the request hit rate and byte hit rate of web proxy cache.

Keywords: Web cache; replacement strategy; circular sliding window; Gaussian mixture model; access prediction mechanism

0 引言

Web 缓存技术通过将 Web 资源保存在缓存中,减少了网络拥塞和服务器资源的负载,有效地提高了网站的响应能力[1]。Web 缓存技术充分利用了时间局部性原理,通过代理服务器缓存用户经常访问的 Web 资源,降低了用户访问时的页面响应延迟,属于主动缓存技术的一种。而缓存替换技术通过设定缓存阈值,当缓存的大小达到阈值时就会触发缓存替换策略,对缓存中的内容进行替换。因此,能够预测用户未来可能访问的资源,并提前将其放入缓存中的缓存预取技术得到了广泛的应用[2]。在仅使用缓存替换机制的服务器中,缓存的容量是制约 Web 响应速度的关键因素,面对大量 Web 访问时的缓存命中率一般都比较低。面对有限的缓存空间和大量的 Web 访问,基于空间局部性原理的缓存预取技术能够很好地弥补缓存替换算法的局限性,显著提高了缓存命中率,极大地降低了访问延迟。

缓存替换算法的准确性是决定缓存替换性能的关键。学者们对 Web 访问数据的很多特征进行了广泛研究,并利用这些特征和特性对 Web 缓存替换方法进行改进。文献[3]调研发现基于频率(过去对象的引用次数)、新近度(距最后一次引用所经过的时间)、大小(对象的大小)和成本(从服务器获取对象的延迟和带宽成本)的缓存替换方案效果更为理想。

目前,缓存替换算法一般分为两大类:一类是基于特

收稿日期:2020-06-21; **修回日期**:2020-07-09。
基金项目:国家数值风洞工程支持项目;国家重点基础研究发展计划基金项目(2014CB744100);西南科技大学博士基金(13zx7102)。
作者简介:唐 榜(1996-),女,四川广元人,硕士研究生,主要从事数据挖掘、大数据技术等方向的研究。
吴 珏(1978-),女,四川绵阳人,博士,副教授,主要从事大数据技术、数据挖掘、人工智能等方向的研究。
引用格式:唐 榜,吴 珏,杨福军,等.基于高斯混合模型的 Web 代理服务器缓存替换策略[J].计算机测量与控制,2021,29(2):166-170,175.

征统计的方法，典型的有最近最少使用算法（LRU）、最少使用频次算法（LFU）、贪婪对偶大小算法[4]（GDS，greedy dual size）和贪婪对偶大小频率算法[5]（GDSF，GDS－frequency）等。其中，LRU 算法会删除最近最少使用的对象，并将新的对象填入空出的缓存中。LFU 算法会使用一个计数器来计算对象的使用频率，并删除最近最不频繁使用的对象。以上两种方法只考虑了对象的其中一个数据特征来进行缓存替换，适用场景单一，因此准确度较低。而 GDS 算法考虑了对象的局域性、大小、延迟、替换代价等因素，综合替换权值最小的对象。GDSF 算法在 GDS 算法的基础上加入了频率因素，改进了 GDS 算法的性能。另一类是基于预测的方法，通过机器学习模型来预测会被再次访问的对象，并提前将其放入缓存中。文献［6］使用树朴素贝叶斯分类器对 Web 数据进行分类，预测可能被再次访问的 Web 对象，结合 LRU 算法提高了缓存替换的效率。文献［7］根据用户的访问日志提取多种特征作为训练数据集，通过训练 SVM 分类器将预测为不会再次访问的缓存对象删除以留出缓存空间。文献［8］提出了一种结合 GDSF 算法和支持向量机（SVM）重新访问概率预测的 Web 缓存替换策略。文献［9］提出了一种用来评估 Web 对象访问的空间局部性算法，提高了缓存替换策略的性能。文献［10］使用随机索引方法和权重分配策略机制的 Web 对象聚类的方法增强 Web 代理缓存的性能。

而高斯混合模型（GMM，gaussian mixed model）由多个高斯分布函数线性组合而成，理论上高斯混合模型可以拟合出任意类型的分布，因此在诸多领域都有应用。文献［11］使用神经网络与 GMM 相结合，学习点云的生成空间来生成新颖的点云形状。文献［12］使用 GMM 对医疗文本数据进行聚类分析，实现帕金森病的早期预测。

尽管一些工作使用机器学习模型通过固定窗口内的访问频率等特征对缓存替换模型进行训练，但是由于 Web 访问存在较高的时间相关性，将长时间内的访问频率及访问新近性作为特征进行缓存替换的方法，无法较好地捕获时间序列数据所拥有的时间相关性。而循环滑动窗口机制在处理时间序列数据上能够起到能够降低运算量、划分时间周期的作用[13-14]，提高对时间序列数据的处理效果。

基于以上分析，本文采用基于循环滑动窗口的 GMM 模型对 Web 日志数据进行聚类分析，结合 LRU 缓存替换算法对缓存对象进行替换。利用循环滑动窗口策略学习一段时间内的 Web 访问数据特征，可以更好地捕获 Web 访问数据的时间相关性，从而使模型获得更好的预测结果。在进行缓存替换时，将 GMM 聚类分析预测的结果与 LRU 算法进行结合，在保证了较好的计算性能的同时，缓存替换的命中率也比传统算法更佳。实验结果证明了本文方法的有效性。

1　模型概述

高斯混合模型可以看作是由 K 个单高斯模型组合而成的模型。由于高斯分布具有良好的数学性质和优秀的计算性能，能够很好地刻画参数空间中数据的分布及其特性，在拟合数据分布时有很强的建模能力，因此在数据科学界被广泛使用。高斯混合模型结合了参数估计法和非参数估计法的优点，并使用了期望最大（EM，expectation maximization）算法进行训练。由于高斯混合模型使用多个高斯分布的组合来刻画数据的分布，在模型中的样本点足够丰富的情况下，任意精度上的连续分布都能被高斯混合模型所拟合。

当样本数据是一维数据时，高斯混合模型为每个类别下的特征分布都假设了一个服从高斯分布的概率密度函数如公式（1）所示：

$$P(x \mid \theta) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp(-\frac{(x-\mu)^2}{2\sigma^2}) \tag{1}$$

其中：μ 为数据均值（期望），σ 为数据标准差（Standard deviation）。

当样本数据是多维数据（Multivariate）时，高斯分布遵从的概率密度函数如公式（2）所示：

$$P(x \mid \theta) = \frac{1}{(2\pi)^{\frac{n}{2}} \mid \sum \mid^{\frac{1}{2}}}\exp(-\frac{(x-\mu)^T\sum^{-1}(x-\mu)}{2}) \tag{2}$$

其中：μ 为数据均值（期望），$\sum$ 为协方差（Covariance），D 为数据维度。

高斯混合模型包含的 K 个子模型就是混合模型的隐变量（hidden variable，$\alpha_1,\alpha_2,\cdots,\alpha_k$）。一般来说，一个混合模型可以使用任何概率分布，这里使用高斯混合模型是因为高斯分布具备很好的数学性质以及良好的计算性能。因此高斯混合模型的概率分布表示如公式（3）所示：

$$P(x \mid \theta) = \sum_{k=1}^{K}\alpha_k\varphi(x \mid \theta_k) = \sum_{k=1}^{K}\alpha_k N(x \mid \mu_k,\sigma_k) \tag{3}$$

其中：x_i 表示第 i 个观测数据，$i = 1,2,\ldots,N$；K 是混合模型中高斯模型的数量，$k = 1,2,\ldots,K$；α_k 是观测数据属于第 k 个子模型的概率，且 $\alpha_k \geqslant 0$，$\sum_{k=1}^{K}\alpha_k = 1$；$\varphi(x \mid \theta_k)$ 是第 k 个子模型的高斯分布密度函数，$\theta_k \sim N(\mu_k,\sigma_k)$。

对于多维数据的每个观测点来说，由于事先不知道它所属的分布，因此无法使用最大似然法来求导获得使最大似然函数最大的参数。最大期望算法（expectation maximization algorithm，EM 算法）是一种常用的高斯混合模型参数估计方法，由 Dempster 等人于 1977 年提出，用于求解含有隐变量（Hidden variable）的概率模型参数的最大似然估计。在初始时随机生成 K 个高斯分布，然后不断地迭代 EM 算法，直至似然函数变化不再明显或者达到了最大迭代次数为止。因此，本文选择 EM 算法作为迭代算法对高斯混合模型的参数进行求解。

EM 算法的迭代更新过程分为如下几步：

1）初始化参数：定义分量数目 K，对每个分量 k 设置 α_k,μ_k 和 $\sum_k$ 的初始值。

2）E－step：在给定的多维高斯分布下，根据参数初始值或上一轮的迭代值来计算对数似然函数的期望及后验概率，计算每个数据 j 来自子模型 k 的可能性，如式（4）所示：

$$\gamma_{jk}=\frac{\alpha_k\varphi(x_j\mid\mu_j,\sum_k)}{\sum_{i=1}^{K}\alpha_i\varphi(x_j\mid\mu_j,\sum_k)},j=1,2,\ldots,N;\quad k=1,2,\ldots,K \tag{4}$$

3）M－step：根据 E－step 中得到的后验概率计算新一轮迭代的模型参数，将似然函数最大化以获得新的参数值。计算过程如下：

$$\mu_k=\frac{\sum_{j=1}^{N}(\gamma_{jk}x_j)}{\sum_{j=1}^{N}\gamma_{jk}},k=1,2,\ldots,K \tag{5}$$

$$\sum\nolimits_k=\sum_{j=1}^{N}\gamma_{jk}(x_j-\mu_k)(x_j-\mu_k)^T,k=1,2,\ldots,K \tag{6}$$

$$\alpha_k=\frac{\sum_{j=1}^{N}\gamma_{jk}}{N},k=1,2,\ldots,K \tag{7}$$

4）计算对数似然函数：

$$\ln P(x\mid\pi,\mu,\sum)=\sum_{j=1}^{N}\ln\{\sum_{k=1}^{K}\pi_k\varphi(x_k\mid\mu_k,\sum\nolimits_k)\} \tag{8}$$

5）检查参数或者对数似然函数是否收敛，若不收敛则返回第 2）步。

2 基于 GMM 的缓存替换策略

2.1 Web 缓存框架

图 1 展示了基于高斯混合模型的 Web 缓存替换算法框架。当代理服务器收到用户的访问请求后，用户与代理服务器进行通信，并将请求记录保存到日志文件中。代理服务器将收集的日志构建成数据集发送到预测模块进行数据预处理和预测[15]。文献［16］展示了高斯混合模型结合时间序列方法在处理具有时间特征的数据上有较好的效果。

预测模块的作用是将数据集中的数据进行预处理，经过数据过滤和特征提取等方式将得到的数据放入 GMM 预测模型进行学习，预测数据是否应该被放入缓存中。而替换模块的作用是统一管理缓存中的对象，根据缓存的请求命中情况，对缓存的内容进行替换。当代理服务器接收到用户的访问请求后，首先在缓存中检索是否存在用户请求的对象。若该对象存在，则直接返回给用户。若该对象不存在，则将请求转发至源服务器，获取到请求对象后由源服务器将该对象返回给用户。通过获取预测模块中 GMM 模型的预测结果，结合 LRU 缓存替换策略判断是否将用户请求的对象拷贝到代理服务器的缓存中，以提高缓存中 Web 对象的访问效率。

本文使用的 GMM 模型在获得日志数据后通过聚类分析计算每个 Web 对象被重新访问的概率，将可能被再次访问的数据替换到代理缓存中。当 Web 对象被标记为可访问时，结合其文件大小等特征将其分别放置在替换队列的不同位置，直到缓存未命中时通过 LRU 算法进行缓存替换。通过 GMM 对 Web 对象的预测结果来决定将其放在缓存队列的位置，以实现更为高效的缓存替换。算法 1 显示了基于高斯混合模型的缓存替换算法的具体流程。

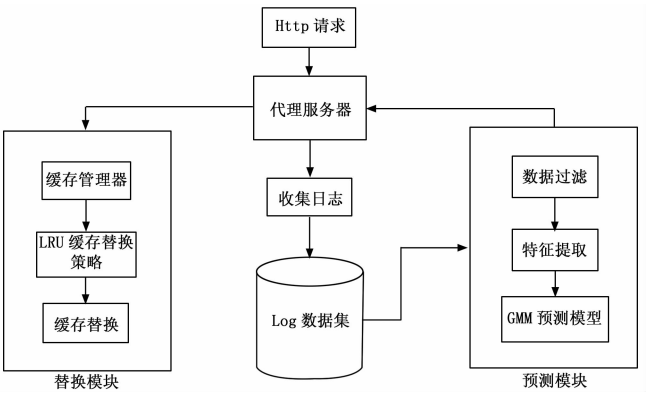


图 1 基于 GMM 的缓存替换模块架构图

算法 1：缓存替换算法

参数：Size 为目标大小，Object 为目标地址，CachedSeq 为缓存队列，Capacity 为缓存大小，Used _ capacity 为已使用的缓存大小，Will _ visit 为是否会再次访问。

```
1. If (Size > Capacity)
2. 缓存未命中
3. Get Object // 从代理服务器获取 Object
4. If (Object 存在于缓存中)
5. 缓存命中
6. 将 Object 移动至 CachedSeq 的头部位置
7. End if
8. If (Object 不存在于缓存中)
9. 缓存未命中
10. End if
11. While (Size + Used_capacity > Capacity)
12. 获取 CachedSeq 尾部节点
13. 从缓存中删除尾节点存储的 Web 对象
14. 更新 CachedSeq
15. End While
16. 通过 GMM 预测得到 Object 的 Will_visit
17. If (Will_visit=1 && Size < Capacity * 0.3)
18. 将 Object 移动至 CachedSeq 的头部位置
19. Else if (Will_visit = 1 && Size >=Capacity * 0.3)
20. 将 Object 移动至 CachedSeq 的中间位置
21. End if
22. If (Will_visit=0)
23. 将 Object 移动至 CachedSeq 的尾部位置
24. End if
```

2.2 特征提取

在代理服务器中，用户访问信息会记录在代理日志中。Web 日志文件中包含了多种访问信息，如用户 IP 地址、访问的 URL 及端口、请求方式、请求时间，访问对象字节大小等。但是数据中包含一定数量的无效数据（访问失败、

地址失效等），因此需要对数据集进行预处理。一方面，将 Web 日志文件数据集进行了过滤，去除不相关的访问及错误的 Web 请求，抽取有用的数据来进行特征提取。另一方面，Web 数据集的构建是从日志代理文件中提取所需的信息，考虑到访问具有时序性，使用了循环滑动窗口机制对数据集进行了分段处理，从中提取并计算出可以用作聚类分析的特征。经过预处理后的具体参数如表 1 所示。

表 1　预处理后的参数列表

参数名称	参数意义
x_1	Web 对象访问时间戳
x_2	Web 对象访问地址
x_3	Web 对象的大小
x_4	Web 对象访问的频次
x_5	访问当前 Web 对象的时间间隔
x_6	滑动窗口内访问当前 Web 对象的时间间隔
x_7	滑动窗口内 Web 对象访问的频次

其中：x_1 表示访问 Web 对象的时间，x_2 表示访问 Web 对象的地址，x_3 表示访问 Web 对象的大小，x_4 表示 Web 对象被访问的频次，x_5 表示 Web 对象上一次访问距离现在的时间差，若 Web 对象首次被访问，x_5 会被初始化为 -1。以上参数均从原始数据集计算获得。而 x_6 表示循环滑动窗口内 Web 对象最近一次访问的时间间隔（Recency），x_7 表示循环滑动窗口内 Web 对象的访问频次。若 Web 对象在滑动窗口内首次出现，则 x_6 初始化为滑动窗口的长度，x_7 初始化为 1。x_6 和 x_7 的计算方式如式（9）和式（10）所示：

$$x_6 = \begin{cases} \min(SWL, \Delta T), Web \text{ 对象在滑动窗口内被访问过} \\ SWL, Web \text{ 对象在滑动窗口内首次出现} \end{cases} \tag{9}$$

$$x_7 = \max[x_7 + 1, 1], \Delta T \leqslant SWL \tag{10}$$

其中：循环滑动窗口的长度设置为 SWL，距离上次请求 Web 对象的时间间隔设置为 ΔT。

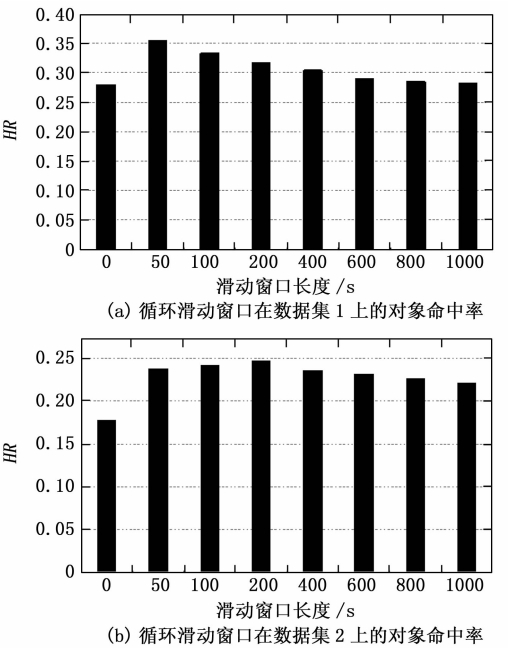
3　实验结果与分析

在本文的实验中，采用了由实验室代理服务器收集的不同时段的用户访问日志数据。数据集 1 包含了该站点从 17：00 到 24：00 收集的约 300 000 条访问数据，数据集 2 包含了该站点从 6：00 到 13：00 收集的约 130 000 条访问数据。在这两个真实数据集上对本文提出的算法与 4 种传统缓存替换算法在对象命中率和字节命中率上进行了比较，验证了该算法的性能。在聚类算法的选择上，比较了几种聚类算法在对本文数据集进行聚类时所耗费的时间，证明了 GMM 模型在进行聚类分析时具有较好的计算性能。

3.1　实验设置

用户的访问请求被记录在代理服务器的日志文件中。当到达指定时间后，通过统计用户访问在缓存空间中的命中次数来获取访问成功的缓存副本。代理服务器的日志文件中包含了每条访问记录的条目，包括以下 8 个字段：日志标记、客户端端口号、请求时间戳、HTTP 状代码、请求和响应报文的大小、URL 地址、主机名和内容类型。本文通过循环滑动窗口机制对日志数据集进行了预处理，再去除掉非法访问、无效访问后的数据集。

为了探究循环滑动窗口的长度对不同时间段获取的数据特征的影响，使用本文的替换算法在 1 M 的缓存空间下对两个数据集进行了验证实验。实验将滑动窗口长度设置为 0～1 000 s，0 s 表示不设置滑动窗口，即使用全局时间长度来提取数据特征。实验结果如图 2 所示，可以发现，循环滑动窗口的设置在一定程度上提高了缓存替换的对象命中率。而由于时间段不同，用户访问频率也不同，在两个数据集上的滑动窗口长度分别设置为 50 s 和 200 s 时获得最高的对象命中率。本文在综合两个数据集的实验结果之后，决定统一选择 100 s 作为循环滑动窗口长度来进行后续实验。



(a) 循环滑动窗口在数据集 1 上的对象命中率

(b) 循环滑动窗口在数据集 2 上的对象命中率

图 2　循环滑动窗口大小对对象命中率的影响

3.2　性能指标

对象请求命中率（HR，hit ratio）和字节命中率（BHR，byte hit ratio）是两个最常用的用来评估不同算法的缓存性能的测试指标。对象请求命中率是指缓存命中的请求次数占总请求次数的百分比，提高对象请求命中率的目的在于减少用户的响应时间。字节命中率是指缓存命中的请求对象字节数占请求对象总字节数的百分比。提高字节命中率的目的则侧重于降低网络通讯量，减少网络带宽开销。对象请求命中率和字节命中率的计算公式如下：

$$HR = \frac{\sum_{i=1}^{N} Q_i}{N} \times 100\% \tag{11}$$

$$BHR = \frac{\sum_{i=1}^{N} Q_i \times S_i}{\sum_{i=1}^{N} S_i} \times 100\% \tag{12}$$

其中：S_i 为对象 i 的大小，Q_i 为命中对象 i 的请求数量，N 为被访问的对象集合。

3.3 性能评估

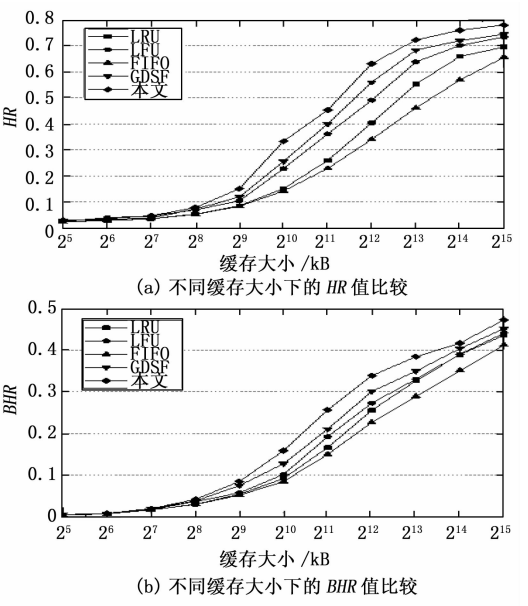
在聚类算法的选择上，本文综合比较了 K－Means 聚类算法、Mini Batch K－Means 算法、DBSCAN 算法、GMM 和 Birch[17]算法的计算性能，实验结果如表 2 所示。可以发现 GMM 聚类算法在面对较大的数据集时，计算性能仅次于 K－Means 算法和 Mini Batch K－Means 算法。K－Means 算法选择的初始聚类中心是随机的，在不同实验中可能产生不同的结果，不具备可重复性，因此并不适用于大型 Web 日志数据。Mini Batch K－means 算法是 K－Means 算法的优化变种，训练时从数据集中随机抽取数据子集来减少计算时间，但是聚类效果也比 K－Means 算法稍差。DBSCAN 算法是一种基于密度的聚类算法，其优点是对噪声鲁棒，能很好地拟合不同形状的数据。但是 DBSCAN 算法的聚类速度较慢，无法满足 Web 缓存替换的高效性需求。Birch 算法只需一遍扫描数据集就能建立 CF Tree，并且对噪声鲁棒，聚类速度也比较快。但是对数据集的分布要求较高，不适合具有高维特征的数据集。而 GMM 使用均值和标准差进行计算，使得簇的形状更加灵活。而且 GMM 给出的是数据集中的项分布在不同簇的概率，因此可以对从 Web 日志数据中得到的概率进行进一步的处理，得到更好的预测效果。因此，综合考虑了计算速度和 Web 日志数据的特点，本文决定采用 GMM 来对已访问的 Web 日志数据进行聚类划分。

表 2 不同聚类算法的训练时间对比 s

次数	K－Means	MiniBatch K－Means	DBSCAN	GMM	Birch
1	1.88	0.90	25.66	1.72	5.34
2	1.29	0.91	25.96	1.32	5.38
3	1.20	0.93	25.35	1.28	5.32
4	1.23	0.88	25.10	3.66	5.13
5	1.18	0.88	25.28	1.50	5.15
平均	1.36	0.90	25.47	1.90	5.27

本文在两个数据集上使用了 LRU、LFU、FIFO 和 GDSF 作为对比算法，与本文提出的缓存替换算法在不同的缓存大小上进行了对比试验。实验结果如图 3、图 4 所示。其中，图 3 显示了在数据集 1 上不同缓存大小下，各种替换策略的 HR 值和 BHR 值。图 4 显示了数据集 2 上不同缓存大小下，各种替换策略的 HR 值和 BHR 值。

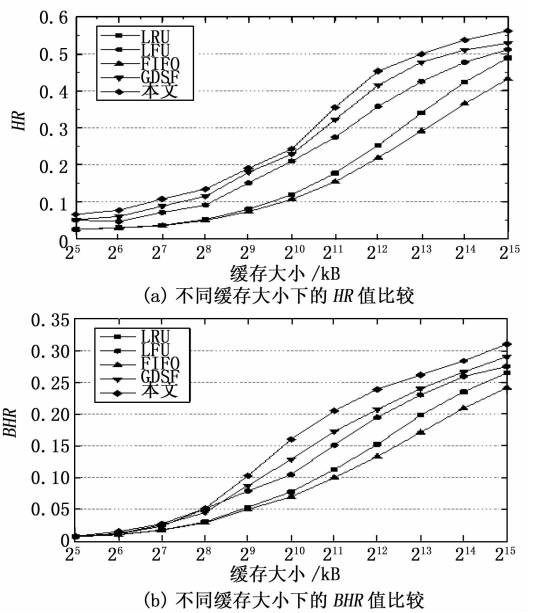
观察模型在两个数据集上的实验结果，不难发现随着缓存大小的增加，缓存对象命中率和字节命中率也呈上升态势。当缓存较小时，各种替换策略的表现相差不大，缓存命中率都比较低。这是因为在缓存较小时，所能承载的缓存对象较少，在面对大量的用户访问时经常需要进行缓存替换的操作，因此缓存的命中率较低。当缓存大小增大时，由于 FIFO、LRU 和 LFU 在进行缓存替换时，使用的数据特征较为单一，因此即使使用了更多的缓存，其缓存命中率与使用多种特征进行缓存替换的 GDSF 相比始终处于劣势。与传统方法相比，本文提出的基于 GMM 的缓存替换算法始终有着较高的 HR 值和 BHR 值，这说明使用了基于 GMM 的聚类分析后得到的预测结果对缓存内容进行替换，有效地提高了缓存的命中率，显著地改善了 Web 代理服务器的缓存效果。



(a) 不同缓存大小下的 HR 值比较

(b) 不同缓存大小下的 BHR 值比较

图 3 数据集 1 上的 HR 值和 BHR 值



(a) 不同缓存大小下的 HR 值比较

(b) 不同缓存大小下的 BHR 值比较

图 4 数据集 2 上的 HR 值和 BHR 值比较

4 结束语

本文提出了一种基于 GMM 访问预测机制的 Web 缓存替换策略。在原有数据的基础上，使用循环滑动窗口机制提取时序特征，根据用户之前的访问日志构建包含多项特征

(下转第 175 页)