计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2022,31(11):380−386 [doi: 10.15888/j.cnki.csa.008797]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于谱聚类的 Web 多级缓存替换策略①

刘　露[1], 吴　珏[1], 杨　雷[1], 杨福军[2]

[1](西南科技大学 计算机科学与技术学院, 绵阳 621010)
[2](中国空气动力研究与发展中心 计算空气动力研究所, 绵阳 621000)
通信作者: 杨福军, E-mail: fintan_yang@163.com

摘　要: 服务器缓存性能的核心是缓存替换策略, 缓存替换策略直接影响缓存的命中率, Web 缓存可以解决网络拥塞和用户访问延迟问题, 提高服务器的性能. 传统缓存替换算法的命中率往往不高, 为此文中提出了一种基于谱聚类的多级缓存替换策略. 该策略利用循环滑动窗口机制提取日志文件的多项时序特征和访问属性, 通过谱聚类对过滤后的数据集进行聚类分析从而得到访问预测结果. 多级缓存替换策略综合考虑了缓存对象的局部频率、全局频率以及资源大小能更好地对低价值资源进行剔除, 同时对高价值资源进行保留. 通过与传统替换算法 LRU、LFU、RC、FIFO 进行实验对比, 实验结果表明本文将谱聚类和多级缓存替换策略进行结合有效地提高了缓存请求命中率和字节命中率.

关键词: Web 缓存; 缓存替换策略; 谱聚类; 多级缓存; 循环滑动窗口

引用格式: 刘露,吴珏,杨雷,杨福军.基于谱聚类的 Web 多级缓存替换策略.计算机系统应用,2022,31(11):380–386. http://www.c-s-a.org.cn/1003-3254/8797.html

Replacement Strategy of Web Multi-cache Based on Spectral Clustering

LIU Lu[1], WU Jue[1], YANG Lei[1], YANG Fu-Jun[2]

[1](School of Computer Science and Technology, Southwest University of Science and Technology, Mianyang 621010, China)
[2](Institute of Computational Aerodynamics, China Aerodynamics Research and Development Center, Mianyang 621000, China)

Abstract: The core of server cache performance is the cache replacement strategy which directly affects the cache hit ratio. Web cache can solve the problems of network congestion and user access delay and improve server performance. A multi-cache replacement strategy based on spectral clustering is proposed because of the low cache hit ratio of traditional cache replacement algorithms. The strategy uses the circular sliding window mechanism to extract multiple temporal features and access attributes of log files and conducts cluster analysis on the filtered data set through spectral clustering to obtain access prediction results. Multi-cache replacement strategy takes into account the local frequency, global frequency, and resource size of the cache object to eliminate the low-value resources and retain the high-value resources. In comparison with traditional replacement algorithms such as LRU, LFU, RC, and FIFO, the experimental results show that the combination of spectral clustering and multi-cache replacement strategy in this study can effectively improve the cache request hit ratio and byte hit ratio.

Key words: Web cache; cache replacement strategy; spectral clustering; multi-cache; circular sliding window

随着计算机的发展, 对 Web 服务器缓存性能提出了更高的要求. Web 缓存是提高 Web 服务质量的一种重要技术, 用于减少客户端延迟、网络流量和服务器负载[1]. Web 缓存的性能取决于其框架、缓存策略、替换策略

① 基金项目: 国家数值风洞工程
收稿时间: 2022-02-28; 修改时间: 2022-03-28; 采用时间: 2022-04-13; csa 在线出版时间: 2022-08-26

和一致性处理[2]. 预取和替换的准确性决定了缓存性能, 对 Web 缓存性能至关重要的是缓存对象的置换算法[3].

Web 访问存在较高的时间相关性, 将长时间内的访问频率和访问时间间隔作为特征进行缓存替换的参考依据, 无法较好地捕获时间序列数据所拥有的时间相关性. 循环滑动窗口机制在处理时间序列数据上有划分时间周期的作用, 能提高对时间序列数据的处理效果[4].

目前缓存替换算法主要分为两类: 一类是基于特征统计的方法, 另一类是基于智能预测算法的方法[5].

基于特征统计的方法有最少使用频次 (least frequently used, LFU)、最近最少使用 (least recently used, LRU) 和贪婪对偶大小频率 (greedy dual size frequency, GDSF) 等算法. LRU 和 LFU 都只考虑用其中一个特征来替换缓存, 准确性不高. 而 GDSF[6] 考虑了对象的局域性、大小、延迟、替换代价、频率等因素, 综合考虑后选择权值最小的对象进行替换. 这些传统方法都侧重用户的访问历史, 缺乏对用户访问请求的预测功能[7]. 将谱聚类用在分析处理日志文件上, 能有效地对日志文件中每条记录进行类型归类, 从而预测某个资源被再次访问到的概率, 这样能提前对该资源进行处理, 提高资源的命中率.

基于智能预测算法的方法是指学者通过机器学习模型来预测会被再次访问的对象, 依此设定替换策略. 文献 [8] 使用了人工神经网络 (artificial neural network, ANN) 提出了一种自适应方法来预测网页的未来访问, 但如 ANN 这类的智能算法在本质上是复杂的, 并且在做出缓存替换决定时的计算量很大, 花费较长时间. 用谱聚类进行小批次聚类分析时, 它所需要的计算量远小于 ANN 这类智能算法, 从而能适应 Web响应所需的低延迟需求. 又由于谱聚类的聚类分析部分在实时变化, 所以能很好地反映当前环境下用户的访问倾向, 比用整个日志文件所生成的数据模型更具针对性.

基于以上分析, 本文采用基于循环滑动窗口的谱聚类对 Web 日志数据进行聚类, 通过构建这种预测模型来预测请求资源是否有可能再次被访问到, 且循环滑动窗口的性质保留了数据特征提取的时序特征. 在缓存替换模块的多级缓存中该算法综合了 LFU 和 MQ 的各自优点, 提出了综合频率和资源大小的多级缓存替换策略, 使得该缓存替换在保证计算性能的前提下, 其命中率也较传统算法更优. 实验结果也证明了本文方法的有效性.

1 谱聚类算法

1.1 谱聚类概述

聚类[9] 是对探索性数据分析最广泛使用的技术, 在现在各个科学领域中处理没有类标的数据时, 人们总是想通过确定数据中不同样本的归类, 来获取对数据的直观印象.

K-均值 (K-means) 算法[10] 是使用最广泛的聚类算法之一, 它易于理解和实现并且收敛速度快, 但是对于初始点选取敏感, 不同的随机初始点得到的聚类结果可能完全不同, 对样本形状包容性差. 为能在任意形状的样本空间上达到良好聚类性能, 研究者们提出了谱聚类 (spectral clustering, SC).

谱聚类[11] 是一种基于图论的聚类方法, 通过对样本数据的拉普拉斯矩阵的特征向量进行聚类, 从而达到对样本数据聚类的目的. SC 可以理解为将高维空间的数据映射到低维, 然后在低维空间用其他聚类算法 (如 K-means) 进行聚类. SC 拥有对样本形状敏感度低、收敛于全局最优解、对高维数据支持较好等特点[12]. SC 如今广泛地应用于数据挖掘、图像分割[13]、计算机视觉以及生物等领域.

1.2 谱聚类算法基本流程

(1) 构建输入样本的相似矩阵 S, 再由相似矩阵构建邻接矩阵 W 和度矩阵 D.

(2) 计算拉普拉斯矩阵 $L = D - W$, 再标准化处理 $L = D^{-1/2} \cdot L \cdot D^{-1/2}$.

(3) 计算 L 矩阵的 k 个最小特征值及对应的特征向量, 构建特征向量矩阵 Q.

(4) 用 K-means 对 Q 矩阵的每一行聚类, 得到聚类结果.

在日志文件中每一行有效记录都包含着数据信息, 一行记录进行特征提取后得到特征属性集, 将作为谱聚类输入矩阵的行向量, 多行记录进而能形成谱聚类输入矩阵. 在谱聚类的最后步骤中用迭代求解的聚类分析算法对特征向量矩阵进行聚类时, 其本质也就是对日志文件中用户的访问记录进行归类处理. 高访问频率和低访问频率的资源文件在聚类结果上便有了差异.

2 基于 SC 的多级缓存替换策略

2.1 缓存策略框架结构

本文的缓存框架分为两个部分, 分别为预测模块

和替换模块, 如图 1 所示. 其中预测模块包含特征提取和谱聚类应用; 替换模块包含多级缓存队列的管理.

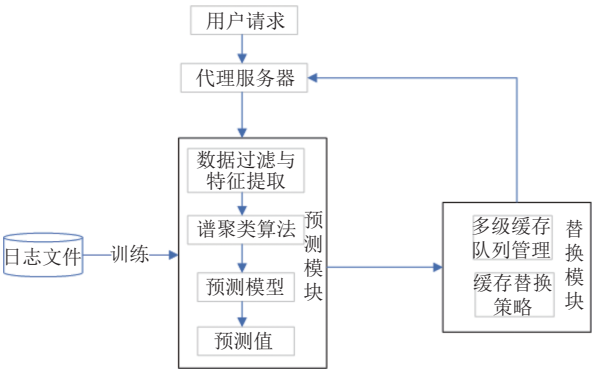


图 1 基于谱聚类的缓存替换策略架构图

当用户的资源请求从客户端发送到代理服务器时, 首先将请求添加到服务器的日志文件中, 同时代理服务器会检查自身是否有该条请求的缓存资源. 如果存在缓存资源则直接将该缓存资源返回给客户端. 若不存在则由预测模块对该条请求结合本地的日志文件进行特征提取, 得到请求资源大小、在日志文件中出现的频率等特征属性; 然后由谱聚类训练出的预测模型进行缓存价值预测, 得到该条资源的预测值. 预测值代表未来是否有再次请求的概率, 在本文中为二分类结果. 最终这个从源服务器请求到的资源文件会随着预测值一起送往替换模块进行缓存, 若多级缓存空间已满则会由多级缓存管理调用本文的多级缓存替换策略进行缓存资源更新.

2.2 数据预处理和特征提取

服务器的日志文件收集了在一段时间内用户的请求记录, 其中包含了主机 ip 地址、访问时间、URL、状态码、访问资源大小等. 除了大部分记录是正常外, 也不免存在一些异常记录, 如: 信息缺失或异常状态码等无效记录, 除此以外在极小时间间隔内来自同一主机的异常高频率访问也被认定为无效. 原始日志文件中存在几十个特征属性, 但很多属性对于聚类分析是没用的, 如: 主机 ip 地址、访问设备等, 所以需要进行特征提取, 特征提取后的值将按照一定排列规则形成特征属性集, 该属性集作为聚类算法的输入. 故在将日志数据进行聚类前, 我们需要对原始数据进行过滤清洗和特征提取.

在考虑到日志文件的数据具有一定的时间相关性和时间序列, 故引入滑动窗口机制对请求记录进行进一步的特征提取. 在对每个请求记录进行过滤清洗和特征提取后会形成统一格式的特征属性集 $< x_1, x_2, x_3, x_4, x_5, x_6, x_7 >$. 其中 x_1 表示为请求的资源地址; x_2 表示为客户端请求到达代理服务器的时间; x_3 表示为请求资源大小; x_4 表示为 Web 对象距上一次访问的时间间隔; x_5 表示为 Web 对象的访问频率; x_6 表示为滑动窗口内距上一次访问的时间间隔; x_7 表示为滑动窗口内的访问频率.

其中, x_4 的初始值为−1, x_5 的初始值为 0, x_6 和 x_7 都与滑动窗口有关且在滑动窗口内计算得出, 其计算公式如式 (1) 和式 (2) 所示:

$$x_6 = \begin{cases} \min(SWL, \Delta T), & \text{Web对象在滑动窗口内被访问过} \\ SWL, & \text{Web对象在滑动窗口内首次出现} \end{cases} \tag{1}$$

$$x_7 = \max[x_7 + 1, 1],\ \Delta T \leqslant SWL \tag{2}$$

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.

2.3 基于谱聚类的预测模块

在预测模块中有一个特征集数组和两个谱聚类数组. 特征集数组是包含了所有请求过的资源对象的特征, 两个谱聚类数组分别是频率特征的谱聚类数组和时间间隔的谱聚类数组, 谱聚类数组记为 s. 每个谱聚类数组都是由冻结部分和活跃部分这两部分构成, 分别记为 $s1$ 和 $s2$, 每个部分都包含了若干条请求资源特征, 其中 $s = s1 + s2$.

(1) 冻结部分是代理服务器刚启动时收集的请求资源的特征集, 这部分的数据是不会发生变化的, 其作用是用来识别用户最新资源请求的预测值类别. 因为最初记录的资源特征变化不明显, 且将预测值设定为未来不会访问很合理, 毕竟在初始时刻每个请求资源都无法在日志文件中找到访问记录. 当最新的资源请求聚类结果和冻结部分的聚类结果相反, 则说明依据频率或者时间间隔该条资源请求是未来会访问的, 应该放入一级缓存, 反之亦然.

(2) 活跃部分是从特征集数组取的最新的若干条请求资源特征集, 这个部分的数据在每次预测时都不相同, 其中最后一个请求资源特征就是当前用户最新的资源请求特征. 活跃部分的作用是使聚类数据包含变化多样的和最新的资源特征, 从而使聚类结果更加贴近真实值, 使聚类结果更为准确.

由频率特征的谱聚类操作我们可以得到关于频率特征的预测值, 由时间间隔的谱聚类操作可以得到关于时间间隔的预测值, 然后将两个预测值进行与操作得到最终的预测值. 最终预测值为 1 则表明该请求资源应放入替换模块的多级缓存的一级缓存, 最终预测值为 0 则应该放入二级缓存. 当日志文件中的资源请求记录不够多, 也就是特征集数组的长度没达到谱聚类数量的开始阈值时, 预测模型对于每个资源请求的最终预测值都直接返回 0, 直到请求数量积累到可以开始谱聚类预测. 这样带来的结果是替换模块中的二级缓存空间会迅速地被请求资源填满, 随后的资源在二级缓存时会采用缓存替换策略进行旧资源剔除, 同时一级缓存却没有任何缓存资源, 会在初期极大影响请求命中率和字节命中率. 这个问题能由第 2.4 节的替换模块的占位标志方法解决. 本文基于谱聚类的预测模块具体算法流程如算法 1 所示.

算法 1. 对请求资源对象进行预测

输入: 请求资源对象所对应的特征属性集 *obj*
输出: 预测值 *will*

1. 把 *obj* 放入特征集数组 *s*1, 判定当前 *s*1 的长度是否达到可以聚类的长度.若为否直接 *will*=0 结束流程. 若 *s*1 的长度刚好等于聚类长度, 则生成聚类冻结部分 *fp*.
2. 从 *s*1 的尾部选择预设长度的元素组成聚类活跃部分 *ap*.
3. 把 *fp* 的频率特征与 *ap* 的频率特征拼接得到频率聚类数组 *s*2, 同理把 *fp* 和 *ap* 的时间间隔部分拼接得到时间间隔聚类数组 *s*3.
4. 分别对 *s*2 和 *s*3 进行谱聚类, 得到聚类结果数组 *r*2 和 *r*3, 其中 *r*2 和 *r*3 中的最后一位值即代表对请求资源对象频率特征和时间间隔特征的聚类结果.
5. 判定 *r*2 的第一位 (冻结部分首位) 和最后一位 (活跃部分末位) 是否相等, 再判定 *r*3 的第一位和最后一位是否相等, 将前两者结果进行与操作.

2.4 基于多级缓存的替换模块

替换模块管理着一个多级缓存空间, 这里是请求资源存放的地方, 多级缓存空间由两个数组组成, 分别是一级缓存数组和二级缓存数组. 当请求资源没有存储于多级缓存中时, 则会根据预测模块提供的预测值 *will* 进行选择性放入; 若存储于多级缓存时, 则会将命中的缓存资源放置在数组的首位, 由于每次一级缓存的缓存资源降级或者二级缓存的缓存资源移除都是从数组末尾开始的, 这样做能使刚才命中的活跃缓存资源能尽可能地保持在缓存空间中.

当请求资源放入的是二级缓存且二级缓存已满, 而此时一级缓存有多余的容量时, 本文会把该请求资源的占位标志改为临时占用, 然后将其存储在一级缓存中. 这样做可以在不移除二级缓存资源的同时, 将请求资源存入缓存空间, 增加了缓存命中的面积, 有利地增大了初期的请求命中率和字节命中率, 这也是第 2.3 节所提到的占位标志方法. 但是该方法会把本不属于一级缓存的资源放入一级缓存, 挤压了本属于一级缓存请求资源的空间, 所以替换模块会在每次调用缓存替换策略的最后根据占位标志对一级缓存的所有缓存资源进行排序, 保证临时占用的缓存资源是保持在一级缓存的末尾, 这样可以优先将这类临时占用的缓存资源替换出去.

替换模块会根据请求资源的大小将其划分为大资源和小资源, 这种资源大小划分的不同会影响请求资源放入缓存数组的位置, 若请求资源被判定为小资源则会被放入缓存数组的首位, 被判定为大资源则会被放入缓存数组的中间位置. 这样做在一定程度上能快速地替换出大资源, 从而使更多的请求资源被缓存, 提高命中率的同时降低代理服务器的缓存成本. 当一级缓存的缓存资源需要被替换出时, 该缓存资源不是直接被清除而是从一级缓存降级到二级缓存中存储; 当二级缓存的缓存资源需要被替换出时, 该缓存资源则是直接从二级缓存数组中移除, 释放出部分二级缓存空间. 这也说明了一级缓存的缓存资源会被更长时间的缓存在多级缓存中, 相反二级缓存的缓存资源则是较快频率地进行更新迭代, 这也是为什么要根据预测模块将不同缓存价值的请求资源放在不同的缓存空间中的原因. 本文基于多级缓存的替换模块具体算法流程如算法 2 所示.

算法 2. 多级缓存替换算法

输入: 请求资源对象 *obj*, *obj* 的预测值 *will*
1. 判定 *obj* 是否存在于一级缓存空间 *Lv*1 或者二级缓存空间 *Lv*2 中, 若 *obj* 已被缓存, 则在缓存空间中将 *obj* 置于缓存空间头部. 否则进入步骤 2.
2. 判定 *will* 值是否为 1, 若为 1 则说明 *obj* 由算法 1 判定为可能被再次访问对象, 进入步骤 3, 若为 0 则进入步骤 4.
3. 判定 *Lv*1 是否有足够空间放入 *obj*, 若否, 则循环剔除 *Lv*1 的末尾元素直至有足够空间. 随后判定 *obj* 是否为小资源, 若是, 则将 *obj* 放入 *Lv*1 的中部, 若否, 则将 *obj* 放入 *Lv*1 的尾部. 程序结束.
4. 判定 *Lv*2 是否有足够空间放入 *obj*, 本算法采用了临时占用的方式, 若 *Lv*2 没有足够空间, 会优先考虑 *Lv*1 是否有足够空间可以临时占用.
5. 若 *Lv*1 有足够空间则把 *obj* 标记为临时占用放入 *Lv*1 的尾部, 若没有则循环剔除 *Lv*2 末尾元素直至有足够空间, 然后把 *obj* 放入 *Lv*2.

在传统的缓存替换策略如 LFU、LRU 和 FIFO 等中, 都未曾涉及到多级缓存的概念, 算法 2 的实现采用了多级缓存概念. 采用多级缓存的好处在于能更加灵活地对系统缓存资源进行分配, 一级缓存的稳定存储能让具有高缓存价值和用户高频率访问的资源对象保持高命中率. 二级缓存的缓存资源频繁更改能更好适应不同时间段用户访问偏好.

3 实验结果与分析

3.1 数据和实验环境

(1) 数据描述

本文实验数据来自于 CSDN 上的一个网站日志文件, 压缩文件大小为 21.59 MB (数据的网站地址为: https://download.csdn.net/download/s1e2l3/10966831). 两份日志文件上记录了不同时段用户访问的历史记录. 数据集 1 是从 17:38 到 24:00 的日志文件数据, 一共有 548 160 条访问记录.数据集 2 是从当日 24:00 到第二天 24:00 的日志文件数据, 一共有 1 400 629 条访问记录. 此次实验通过谱聚类分析请求记录特征, 给出预测值辅助本文的多级缓存替换策略机制实现缓存存储. 在与常见的缓存替换策略 LRU、LFU、RC 和 FIFO 进行性能对比后, 可以看出本文提出的谱聚类与多级缓存的缓存策略具有良好的性能优势.

(2) 实验环境

实验环境采用百度平台下的飞桨 AI Studio, 其云算力配置为: GPU: Tesla V100 × 4; CPU: Intel Xeon × 32; RAM: DDR4 128 GB.

3.2 实验设置

在代理服务器上获取到一份日志文件的副本, 其中的每条访问记录包含用户 ip、访问时间、请求资源路径、资源大小、状态码等等. 除了对其中的缺省或异常数据进行剔除, 还利用滑动窗口机制得到了滑动窗口时间内的频率和时间戳, 最终得到了过滤后的 7 项属性: ip、时间戳、资源大小、时间间隔、频率、滑动窗口内时间间隔和滑动窗口内频率. 同时设置了多级缓存替换策略中一级缓存和二级缓存中的空间分配占比.

为了探究在多级缓存中空间如何分配从而达到最高性能, 在缓存空间设定为 8 MB 的前提下, 将一级缓存和二级缓存设置为 90% 和 10% 进行性能测试. 测试完成后将一级缓存降低且二级缓存升高 (其总和为 100%), 测试直至一级缓存 10% 且二级缓存 90%. 测试结果如图 2 和图 3 所示, 从图中可以看出, 不同的多级缓存空间设置对多级缓存替换策略的性能表现具有一定影响, 其中当一级缓存空间的容量占总缓存空间大小的 80% 时, 缓存替换策略的请求命中率和字节命中率都为最佳. 故多级缓存空间的分配一致采用 [0.8, 0.2], 即一级缓存占总空间 80% 且二级空间占 20% 来进行后续的综合实验.

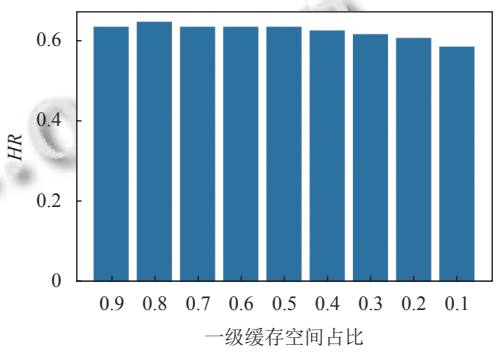


图 2 不同缓存分配比例下的 *HR*

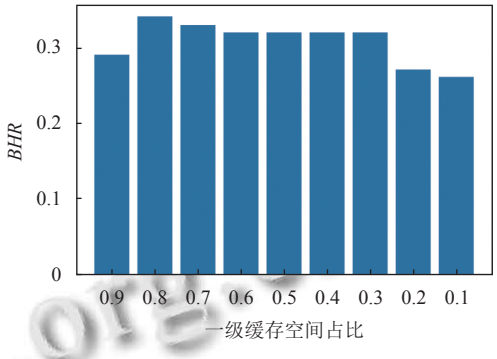


图 3 不同缓存分配比例下的 *BHR*

3.3 性能指标

在对缓存替换算法的研究过程中, 研究员需要有合适的缓存性能判断标准, 所有的性能判断方法几乎都是基于几个重要的参数实施的, 本文对缓存替换算法的评判是选取在该领域内常用的两个参数: 对象请求命中率 (hit ratio, *HR*) 和字节命中率 (byte hit ratio, *BHR*). 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[14], *HR* 的计算公式如下:

$$HR = \frac{\sum_{i=1}^{N}\sigma_i}{N} \times 100\% \tag{3}$$

其中, σ_i为命中对象 i 的请求数 ($\sigma_i = 1$时表示被命中, $\sigma_i = 0$时表示没有被命中) , N 为被访问的对象集合.

对象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[14], *BHR* 的计算公式如下:

$$BHR=\frac{\sum_{i=1}^{N}\sigma_i\times S_i}{\sum_{i=1}^{N}S_i}\times 100\% \quad (4)$$

其中, S_i为对象 i 的大小, σ_i为命中对象 i 的请求数.

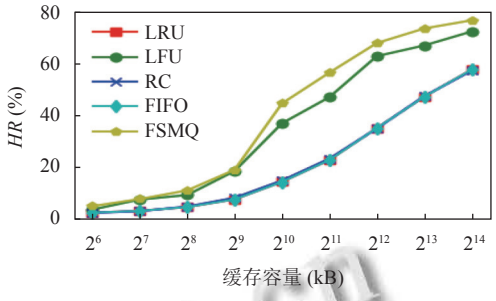
3.4 性能比较与分析

在不同的聚类算法的选择中, 本文比较了 K-means 算法、Agglomerative Clustering 算法、BIRCH 算法和谱聚类算法的表现性能和契合度. K-means 算法中需要选择一个聚类初始中心, 这个初始点的选择对聚类结果有很大的影响, 不具有唯一性故不适合 Web 日志数据聚类. Agglomerative Clustering 算法和 BIRCH 算法[15] 是层次聚类算法, 该类算法的特点是它将所有聚类的数据都存储在磁盘上, 且只需要扫描一遍聚类数据, 同时聚类的速度也不慢, 但它对于高维特征的数据聚类效果不理想, 在 Web 日志文件中属性特征是较多的. 谱聚类算法使用了降维技术, 它对于代理服务器日志中的多个特征属性具有良好的聚类效果, 且由于大部分服务器资源不会被经常访问故在特征属性集中容易形成稀疏数据. 由于谱聚类只需要数据之间的相似性矩阵, 因此对于处理稀疏数据的聚类十分有效, 这点在如 K-means 这种聚类算法中很难做到. 如表 1 所示, 实验采用了上述提到的不同聚类算法对固定大小的聚类分组进行了准确率测试. 从中可以看出谱聚类在处理日志文件这类具有高维特征的数据时具有一定优势.

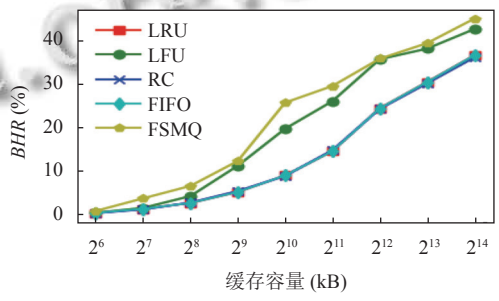
表 1 不同聚类算法的准确率比较 (%)

次数	K-means	Agglomerative clustering	BIRCH	SC
1	79.75	80	80	78.5
2	76	76.67	76.67	79
3	75.67	76	76	79.67
4	71.75	75.5	75.5	78
平均	75.79	77.04	77.04	78.79

本文在多级队列 MQ 的基础上考虑了部分频率、全局频率以及资源大小, 设计了多级缓存替换策略. 以传统缓存替换策略 LRU、LFU、RC 和 FIFO 作为对照算法, 在不同的缓存空间大小上进行了实验. 实验结果如图 4、图 5 所示. 图 4 显示了 5 种缓存替换策略在数据集 1 上不同缓存空间大小上的 *HR* 和 *BHR*, 图 5 显示了 5 种缓存替换策略在数据集 2 上不同缓存空间大小上的 *HR* 和 *BHR*.

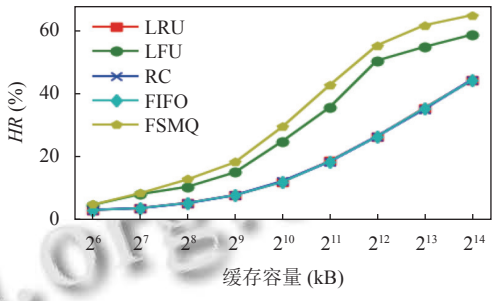


(a) 不同缓存大小下的 *HR* 值比较

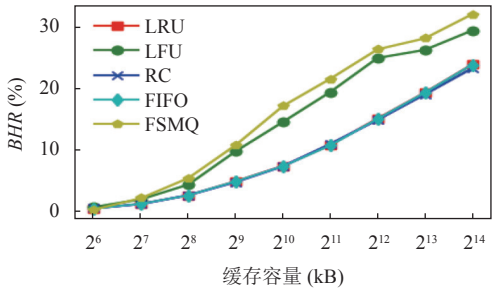


(b) 不同缓存大小下的 *BHR* 值比较

图 4 数据集 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 5 数据集 2 上的 *HR* 值和 *BHR* 值

从图 4、图 5 可以看到 *BHR* 一开始在所有的缓存替换策略下命中率都很小. 这是因为在缓存容量上限很小的时候, 缓存空间所能容纳的资源有限, 且这时命中的资源对象的大小也很小, 导致其 *BHR* 一直上不去.

但随着缓存容量上限不断提高, 能承载的对象数目和大小也在不断上升, 所以命中率也在不断上升. 类似 LRU、LFU、RC 和 FIFO 这类传统缓存替换策略对数据特征考虑的较为单一, 而本文的多级缓存替换策略与传统的替换算法相比考虑了更多的数据特征属性, 同时通过谱聚类的聚类结果进行预测能做出更好的缓存替换抉择, 从而得到 *HR* 和 *BHR* 一直处于较高的位置, 显著地提高了缓存的命中率.

除此以外不难发现本文提出的缓存替换策略在缓存容量较小的情况下 *HR* 值很高, 这是因为在谱聚类的聚类效果基础上能将高频访问、小资源的高缓存价值资源进行缓存, 从而达到在相同缓存空间容量的情况下能缓存更多资源对象的效果. 这也解释了在缓存空间不大时每个缓存替换算法 *BHR* 都差不多, 但本文的 *HR* 却高出很多的原因.

4 结束与展望

本文提出了一种基于谱聚类访问预测机制的 Web 多级缓存替换策略. 将用户访问的日志文件作为数据源, 利用循环滑动窗口机制从中提取出多项属性特征. 基于谱聚类的聚类结果进行访问预测, 将缓存资源连带预测值放入本文的多级缓存替换策略进行数据缓存.

从结果可以看出与传统替换算法 LRU、LFU、RC 和 FIFO 相比, 本缓存替换策略在命中率上有不错的提升, 在改善用户访问体验和减轻服务器网络拥塞的问题上有不错的效果, 有效地提高了 Web 性能. 下一步的研究方向会考虑搭建用户访问模型, 根据用户的访问兴趣动态地影响预测值, 从而得到高效且全面的预测算法.

现有的缓存框架中普遍使用的是传统缓存替换策略, 如 Redis 中使用 LFU, Ehcache 中支持 LRU、LFU 和 FIFO. 传统缓存替换策略自身具有局限性, 导致最终对于服务器性能提升不明显. 本文基于谱聚类的 Web 多级缓存替换策略能在高并发的情况下, 依然保持比传统算法更高的命中率.

参考文献

1 段洁, 邢媛, 赵国锋. 信息中心网络中缓存技术研究综述. 计算机工程与应用, 2018, 54(2): 1–10. [doi: 10.3778/j.issn.1002-8331.1704-0426]

2 Ma TH, Hao Y, Shen WH, *et al*. An improved Web cache replacement algorithm based on weighting and cost. IEEE Access, 2018, 6: 27010–27017.

3 赵中全, 刘丹. 基于树扩展朴素贝叶斯分类器的 Web 代理服务器缓存优化. 计算机工程, 2017, 43(1): 115–119.

4 叶阿勇, 孟玲玉, 赵子文, 等. 基于预测和滑动窗口的轨迹差分隐私保护机制. 通信学报, 2020, 41(4): 123–133. [doi: 10.11959/j.issn.1000-436x.2020049]

5 杨瑞君, 祝可, 程燕. 基于 SVM 访问预测机制的 Web 缓存数据库级替换策略. 计算机科学, 2019, 46(6): 201–205. [doi: 10.11896/j.issn.1002-137X.2019.06.030]

6 Ma TH, Qu JJ, Shen WH, *et al*. Weighted greedy dual size frequency based caching replacement algorithm. IEEE Access, 2018, 6: 7214–7223. [doi: 10.1109/ACCESS.2018.2790381]

7 戴敏. 基于 NB 分类器重访概率预测的 Web 缓存替换策略. 计算机工程与应用, 2019, 55(19): 134–140. [doi: 10.3778/j.issn.1002-8331.1808-0061]

8 王准, 何元烈. 基于混合价值计算的云存储缓存替换方案. 计算机工程与设计, 2017, 38(6): 1651–1656. [doi: 10.16208/j.issn1000-7024.2017.06.046]

9 Golalipour K, Akbari E, Hamidi SS, *et al*. From clustering to clustering ensemble selection: A review. Engineering Applications of Artificial Intelligence, 2021, 104: 104388. [doi: 10.1016/j.engappai.2021.104388]

10 Govender P, Sivakumar V. Application of K-means and hierarchical clustering techniques for analysis of air pollution: A review (1980–2019). Atmospheric Pollution Research, 2020, 11(1): 40–56. [doi: 10.1016/j.apr.2019.09.009]

11 Xia KJ, Gu XQ, Zhang YD. Oriented grouping-constrained spectral clustering for medical imaging segmentation. Multimedia Systems, 2020, 26(1): 27–36. [doi: 10.1007/s00530-019-00626-8]

12 白璐, 赵鑫, 孔钰婷, 等. 谱聚类算法研究综述. 计算机工程与应用, 2021, 57(14): 15–26. [doi: 10.3778/j.issn.1002-8331.2103-0547]

13 Ma X, Zhang SG, Pena-Pena K, *et al*. Fast spectral clustering method based on graph similarity matrix completion. Signal Processing, 2021, 189: 108301. [doi: 10.1016/j.sigpro.2021.108301]

14 Hou R. Performance analysis of cache replacement algorithm in information center network and construction of electronic music composition system. Alexandria Engineering Journal, 2022, 61(1): 863–872. [doi: 10.1016/j.aej.2021.04.082]

15 张谣谣. 基于 BIRCH 算法和深度神经网络在学科分析中的应用研究 [硕士学位论文]. 重庆: 重庆交通大学, 2020.

(校对责编: 牛欣悦)