基于伸展树的文件数据缓存管理策略研究

姚智海 徐宏喆 李 文 吴 夏
(西安交通大学陕西省计算机网络重点实验室 西安 710049)

摘 要 针对企业内部网络存储,研究并提出了一种基于伸展树的缓存管理策略,以对网络缓存空间进行组织和管理。在内部网络存储缓存链的基础上,引入并改进了伸展树结构和操作,将改进后的伸展树作为缓存节点数据组织和管理的索引结构,分析并设计了基于伸展树的文件数据缓存管理策略。实验结果表明,基于伸展树算法的缓存管理策略提高了缓存空间利用率和用户访问数据的效率,有较好的实时性。

关键词 伸展树,缓存链,替换价值度,缓存管理
中图法分类号 TP393 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2016.9.025

Research of Cache Management Strategy Based on Splay Tree

YAO Zhi-hai XU Hong-zhe LI Wen WU Xia
(Shaanxi Key Lab of Computer Network, Xi'an Jiaotong University, Xi'an 710049, China)

Abstract Taking the enterprise's internal network storage application as background, we put forward a cache management strategy based on splay tree to organize buffer memory space. We improved the structure and operation of splay tree and took it as the index structure for organizing and managing cached data. Finally, according to the actual application requirements, we analyzed and designed the model of cache management and made it come true. The experimental results show that, through combining with the improved splay tree algorithm, the buffer memory space and cached data can be effectively organized and managed, therefore, the cache management strategy proposed in this paper has good real time property and the efficiency of accessing data can also be improved.

Keywords Splay tree, Buffer chain, Replacement value, Cache management

1 引言

随着网络数据流量的迅猛增加,企业对网络存储系统的依赖性越来越强[1]。但是有限的网络带宽以及硬件条件的限制,导致了客户端请求响应缓慢、远程服务器负载严重及处理速度受限等问题[2],从而无法满足用户对服务质量和效率的要求。缓存技术的使用可以有效减轻远程服务器的负载,减低网络阻塞[3],从而成为网络应用研究的热点。目前,国内外已有很多成熟的缓存技术,这些缓存技术在网络存储领域得到了广泛的应用[1],其目的均是减少网络阻塞和降低客户端请求响应延时,提升系统的整体性能,只是侧重点不同[4-7]。而对缓存数据管理算法及数据结构相结合的研究相对较少,主要集中于树结构上[8-10]。

伸展树(SplayTree)是一种与平衡二叉树(AVL树)类似、具有自调整特征的改进二叉查找树[11],已被应用于FreeBSD系统的虚存管理中。在几种经典的缓存替换算法中,LRU算法因其额外资源开销小且适合于具有高局部性的数据访问,而被普遍应用于现有系统中,然而当需要循环访问一个数据集时,其执行效果却不理想[12],而且没有区分访问频率不同的数据[13]。

在深入研究在企业内部网络存储系统之后,基于内部网络存储缓存链,结合虚存对象管理的伸展树算法和缓存数据替换算法,提出一种满足用户访问需求的基于伸展树算法的文件数据缓存管理策略,以实现在企业网络存储应用中缓存数据的组织和管理。所提策略对网络应用中数据缓存问题的研究具有一定的实际意义和价值。

2 内部网络存储缓存链

企业内部网络存储系统一方面为企业用户提供数据访问服务,并满足企业内部各部门的资料共享等需求;另一方面通过存储网络将重要资料保存并集中管理,在可靠性、可用性和实时性相结合等方面,较其他存储系统有更高的要求[14]。本文研究的内部网络存储系统的体系结构包括数据中心、缓存服务器、客户端等,如图1所示。客户端与数据中心节点之间存在一条经过若干个服务器节点的路径,由路径上节点的缓存构成的多级缓存,被称为内部网络存储缓存链。在客户端和数据中心之间建立一条缓存链,即是寻找到一条客户端到数据中心的单源最短路径。在内部网络存储缓存链的基础上,通过改进的伸展树结构对缓存数据进行高效组织和管理,以满足客户端快速响应的需求,这是本文研究的主要内容。

到稿日期:2015-08-08 返修日期:2015-11-09

姚智海(1988-),男,硕士生,主要研究方向为计算机网络、数据挖掘,E-mail:yao.zhihai@foxmail.com;徐宏喆(1961-),女,博士,教授,CCF会员,主要研究方向为人工智能、数据挖掘;李 文(1966-),女,硕士,讲师,主要研究方向为计算机智能软件;吴 夏(1990-),男,硕士生,主要研究方向为云计算。

从缓存链中获取数据的基本过程如下：用户节点依据本地保存的缓存链缓存节点表向各个缓存节点广播数据消息；在收到用户节点发送的数据请求后，各缓存节点根据数据的基本信息组装成树的二叉查找关键字，之后依据伸展树的特性搜索树中节点，将结果反馈给用户；用户节点收到缓存链上各个节点反馈的消息后，若某一节点存在所需要的数据，则直接从该缓存节点获取数据并将其缓存在本地；若除去数据中心，所有缓存节点均不存在所需要的数据，则从数据中心获取数据并将其缓存在本地。

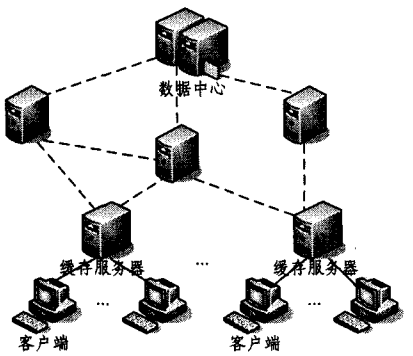

图 1　内部网络缓存系统

经典的 LRU 算法仅使用单一的时间因素作为替换的依据，而在内部网络存储缓存链中，访问次数往往也是某一数据块是否应被替换的重要依据。针对此种情况，将访问次数与时间共同作为缓存数据替换的依据——替换价值度。

定义 1（替换价值度，Replacement Value）　用于描述当前缓存节点中数据块未被访问的程度，其与最近一次访问时间成正比，与当前缓存链用户访问该数据块的次数成反比。设第 i 个数据块替换价值度为 $ReValue_i$，最近一次访问时间为 $Time_i$，被访问次数为 Num_i，则替换价值度可表示为式(1)所示形式：

$$ReValue_i = Time_i / Num_i \tag{1}$$

结合经典 LRU 算法，替换价值度越大的数据对象，越应当从当前缓存节点中替换并迁移至下一级缓存节点，本文称该算法为 ILRU 算法（Improved LRU Algorithm）。ILRU 替换缓存数据块的流程如下：

Step 1　初始化缓冲区、$ReValue_i$、$Time_i$、Num_i。

Step 2　记录每个数据块最近一次被访问的时间间隔 $Time_i$ 及访问次数 Num_i。

Step 3　按照式(1)计算每个数据块的替换价值度 $ReValue_i$。

Step 4　将 $ReValue_i$ 值中最大的数据块 j 替换出缓冲区。

Step 5　将数据块 j 的 $ReValue_j$，$Time_j$，Num_j 值置 0。

3　伸展树算法

伸展树在满足操作效率的前提下，具有实现简单、可以将本地经常被访问的数据节点置于靠近树根的位置、不需要额外附加空间等优点，故选取伸展树作为缓存数据管理的核心数据结构。为了满足整个系统的实际应用需求，还需要对伸展树的结构以及操作进行一些改进。

3.1　伸展树算法结构改进

3.1.1　增加关键属性

为了实现依据文件数据信息来组织并管理缓存数据和缓存空间，需要对伸展树的管理数据结构进行改进：增加 adj_free 和 replace_value 节点关键属性，其中 adj_free 记录数据存储右相邻的空闲空间大小，replace_value 记录树节点对应数据的最近一次访问时间与访问次数的比值即替换价值度。

3.1.2　缓存空间管理

为了能够组织并管理数据空间和缓存空间，采用两套数据结构：伸展树和双向链表，如图 2 所示。伸展树管理数据的各项操作，如数据查找、插入和删除等，用于索引缓存空间，达到快速查找目标节点的目的。双向链表直接管理缓存空间，并修改缓存数据的访问时间，如空间的合并和分裂等。缓存空间管理的数据结构（双向链表）以组织和管理数据的数据结构（伸展树）为中心进行各种操作。

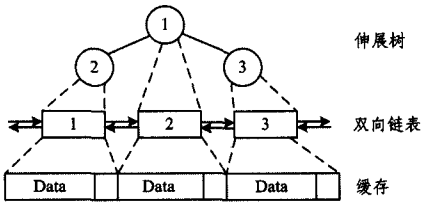

图 2　双重数据结构管理

为了能够有效管理缓存空间，双向链表中节点组织的依据除了文件路径的 MD5 值以及数据的偏移外，还包括数据存储空间的首地址。即同一个节点拥有两套不同的关键字，因此中序遍历伸展树产生的有序结果与链表并非一一对应。

3.2　伸展树算法操作优化

3.2.1　伸展树重构

伸展树的重构包括自底向上和自顶向下两种方式。为了能够在重构伸展树的过程中将自底向上得到的属性方便地计算出来，在生成左右树时，将“链倒置”操作引入到自顶向下查找目标节点的过程中，即将节点指针的链接在逻辑上倒置；之后加入自底向上的过程，从左右树的树根位置开始，在矫正左右树节点链接的同时，根据位置关系循环计算出节点的属性，并矫正“链”连接。

在生成左右树的过程中，发生位置变动的是经过旋转操作并被更改链接的节点，仅这些节点的属性值需要更新。因此仅需要自底向上逐个扫描左右树中被更改链接属性的节点，计算出调整树结构中被修改节点的属性值。之后，矫正链的链接，最终将矫正链接后的左右树与中树合并，构成调整结构后的伸展树。

3.2.2　节点合并

节点合并发生的前提条件是新到来的数据需要较大的空间，而任何空闲空间以及任何一个待替换出缓存的数据块所占用的空间与其相邻的空闲空间之和均不能满足空间需求，这样就必须从缓存中替换多个在物理空间相邻的缓存数据块，从而将新到数据存储至缓存空间。“合并”包含两部分操作：树结构和缓存空间。合并的方法为：令新到数据的大小为 $Length$，根据节点的 $replace\_value$（替换价值度）属性，进行二叉树查找操作，搜索到替换价值度最大的节点 $Node$。若该节点对应数据块所占用的缓存空间以及它与相邻空闲缓存空间 adj_free 属性之和均小于 $Length$，则需要根据链表上相邻节点的 $replace\_value$ 合并缓存空间。此时有如下两种情况需要处理。

(1)仅有左/右相邻节点。如图 3 所示，伸展树中每一个节点对应缓存空间中一个缓存数据块的存储，树的节点包含

了数据结构所需的属性、标示数据块、记录数据块参数和相邻空闲空间的属性等。查询到的节点仅有左/右相邻节点，说明搜索到的节点的标示关键字为最小值或最大值，其位置处于树结构的最左边或最右边，而对应的缓存数据块存储于缓存空间的起始或结束位置，此情况只能合并叶子节点的父节点。

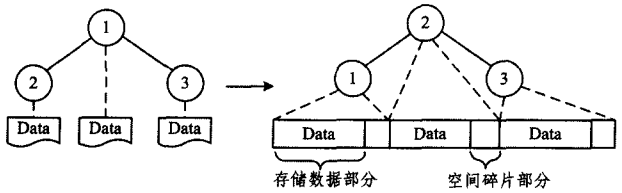

图 3　伸展树节点与缓存空间之间的对应关系

(2)否则，比较相邻节点的 *replace_value* 值，若一个相邻节点的 *replace_value* 属性值大于另一个相邻节点的值，说明相对于后者，前者在更长的时间内未被用户访问，则替换合并 *replace_value* 大的相邻节点；若两者的 *replace_value* 属性值相等，则替换合并占用缓存空间较多的相邻节点。循环以上处理过程，直至出现满足空间需求的存储空间。

通过数学分析可知，改进算法基本操作的平均时间复杂度为 O(log*n*)，整个算法的时间复杂度为 O(*n*log*n*)；虽然加入了各种改进操作，但是算法的整体执行效率不会被降低。

4　基于伸展树的文件数据缓存管理策略

基于对缓存链、缓存替换算法和伸展树算法的分析与改进，本文提出了一种基于伸展树的文件数据缓存管理策略。该策略在用户节点与数据中心之间建立的缓存链的基础上，以改进的伸展树算法为核心，对缓存链上节点的缓存空间及数据进行组织和管理，以达到企业内部网络快速存储和及时访问数据的目的。该策略应用于内部网络缓存链结构中的数据中心和各个缓存服务器上，其架构如图 4 所示，缓存空间用于存放缓存数据，缓存管理用于组织和管理数据，用户接口则为缓存节点用户提供了访问本地缓存数据的方式。其中缓存管理以缓存管理策略为核心，包括 splaying、查找、插入、删除、分裂及合并等基本的伸展树操作，并在这些操作的基础上完成了数据的缓存、替换等操作。

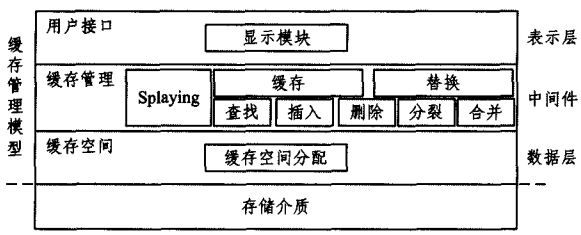

图 4　设计架构图

基于伸展树的文件数据缓存管理策略的详细描述如下所示。

输入：用户节点请求数据 ask

输出：查找结果

Step 1　依据伸展树特性和内部网络拓扑结构，构建内部网络缓存链。

Step 2　收到请求数据 ask，依据伸展树的二叉查找树特性，在各个缓存节点中根据 ask 关键字快速查找目标数据。

Step 3　若缓存服务器节点中存在请求数据，比较离用户最近的缓存节点 I 是否存在足够空间存放数据 ask。

Step 4　若节点 I 空余空间 space＞数据 ask 大小，则采用 first_fit 方法分裂足够空间，将新到数据 ask 依据伸展树操作 splaying 至该节点。

Step 5　若节点 I 空余空间 space＜数据 ask 大小，且节点 I 中替换价值度 replace_value 最大的数据块空间＞ask 大小，则根据伸展树特性，将该数据块迁移至缓存链的下一级缓存节点，回收该空间用于存放 ask。

Step 6　若节点 I 空余空间 space＜数据 ask 大小，且节点 I 中替换价值度 replace_value 最大的数据块空间＜ask 大小，则根据替换价值度和合并地址相邻的空间，并迁移相应数据块，直到获得足够存放 ask 的缓存空间为止，迁移 ask 到该空间。

Step 7　修改 ask 和被替换(迁移)数据块的 adj_free 值及替换价值度 replace_value。

Step 8　若缓存节点不存在 ask，则从数据中心获取 ask 并缓存在本地，修改 ask 的替换价值度。

5　仿真测试及性能分析

为证明本文所提策略的可行性及其对缓存数据访问性能的提高，本研究依托基于 iSCSI 技术的内部网络数据存储系统，在硬件条件为 AMD Athlon64 X2 双核 4000＋ 2.10GHz CPU，2.00GB 内存的实验室环境下，模拟实际的应用环境，对策略进行仿真测试，并对结果进行研究和分析。

5.1　空间利用率测试与分析

依据数据块大小范围，通过设定缓存空间大小，对本文提出的策略的空间利用率进行测试，结果如图 5 所示。

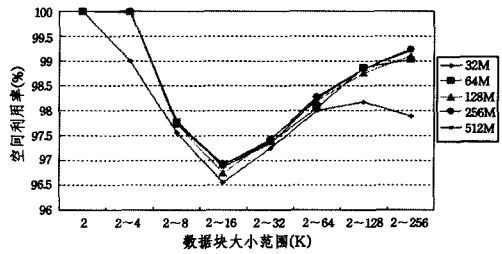

图 5　空间利用率

从图 5 看出，在数据块大小范围一定的前提下，空间利用率随着缓存空间的增大会有一定的提升；在缓存空间大小一定的前提下，空间利用率在前期会随着数据块可选择范围的增大而降低，并在 2kB～16kB 之后随着范围的增大而提高。由此可知，影响空间利用率的主要因素是数据块的大小范围，而缓存空间大小对其影响较小；如果迁移或缓存到缓存空间的数据块粒度越小，缓存利用率就越高。但是存储网络中用户与数据中心使用的文件系统不一定相同，因而数据单元大小分配也不一定相同；数据块的大小范围越大，缓存利用率也越大。

图 6 抽样显示了缓存空间为 256M、数据块范围在 2kB～16kB 时，缓存利用率在一段时间内的变化。可以看出，空间利用率在一段时间后能达到平衡，并在 96.8%上下浮动。

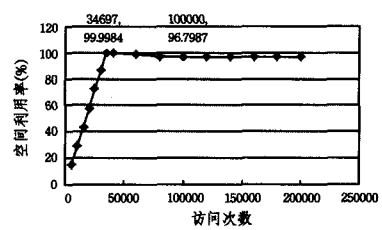

图 6　一段时间的空间利用率变化曲线

5.2　访问操作开销测试与分析

测试参数：缓存为 256MB，数据块范围为 2kB～4kB，文

件个数为 2，文件大小为 4GB，读取数据大小为 8GB。

(1)本文算法改进前后在删除与插入操作上消耗的时间对比情况如图 7、图 8 所示。

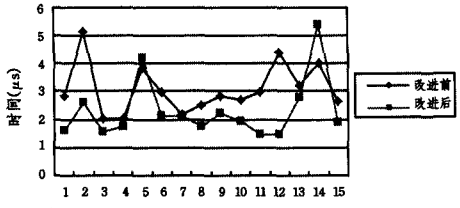

图 7　删除操作时间对比

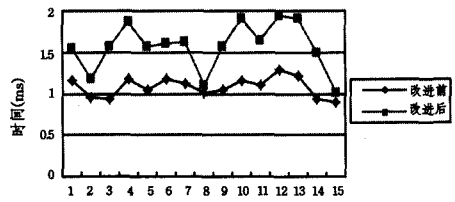

图 8　插入操作时间对比

本文改进后策略在删除操作上消耗时间较少，而在插入操作上消耗时间较多。原因在于：1)改进后策略的插入操作是缓存管理策略将优化的伸展树算法与 ILRU 替换算法相结合的结果，是缓存管理策略的核心部分；2)优化后伸展树结构的变化使得每产生新的节点，算法均需要将该节点链接到树和双向链表两种结构中，从而增加了时间的消耗。

(2)改进前后算法平均一次操作的时间消耗如图 9 所示。

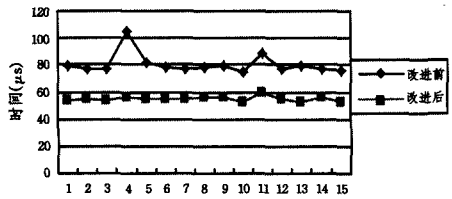

图 9　改进前后算法平均一次操作时间对比

由图 9 可知，以优化的伸展树算法和替换算法为核心的管理策略平均一次操作的时间消耗较少，能够满足网络存储系统缓存管理的实时性需求。

5.3　缺页次数测试与分析

图 10 显示了 LRU 优化前后缓存中数据替换的缺页请求发生次数。在相同的测试条件下，ILRU 算法虽然引入访问时间与访问次数的比值作为数据替换的依据，增加了一些额外存储空间，但其缺页次数少于 LRU 算法，可以满足策略的缓存替换需求。

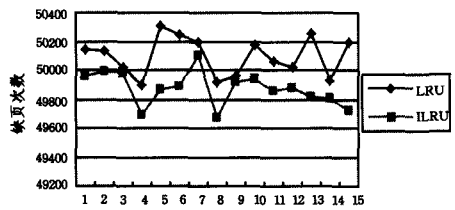

图 10　改进前后算法缺页次数对比

结束语　本文提出了以缓存链为基础、将改进的伸展树与优化的 LRU 替换算法(ILRU)结合的数据缓存管理策略。选择伸展树作为缓存组织的核心算法，根据实际需求以及伸展树的特性，对伸展树算法在操作与结构上进行了改进，并在 LRU 替换算法的基础上，引入访问次数共同作为缓存替换的依据，最终提出了基于伸展树算法的文件数据缓存替换策略；并且依据性能标准，对测试结果进行了研究和分析，进一步验证了本文给出的管理策略的有效性和可行性。

参 考 文 献

[1] 韩德志，傅丰著. 高可用存储网络——关键技术研究[M]. 北京：科学出版社，2009：1-2，146

[2] Tewari R，Dahlin M，Vin H M，et al. Design Considerations for Distributed Caching on the Internet[C]//Proc. of the 19th IEEE Distributed Computing Systems. Austin，TX，USA，1999

[3] Raunak M S. A survey of cooperative caching：Technical Report [R]. 1999

[4] Devi U C，Chetlur M，Kalyanaraman S. Object placement for cooperative caches with bandwidth constraints[C]//Jeannot E，Namyst R，Roman J，Eds. 17th International Conference on Parallel Processing. Euro-Par 2011，Part I，2011：579-593

[5] Wang K，Chen Z K，et al. Research on Storage Service-oriented Distributed Cache System[J]. Computer Engineering，2010，36(15)：80-82(in Chinese)
王侃，陈志奎，等. 面向存储服务的分布式缓存系统研究[J]. 计算机工程，2010，36(15)：80-82

[6] Morales K，Lee B K. Fixed Segmented LRU cache replacement scheme with selective caching[C]//2012 IEEE 31st International Performance Computing and Communications Conference (IPCCC). IEEE，2012：199-200

[7] Khulbe P，Pant S. Hybrid (LRU) Page-Replacement Algorithm [J]. International Journal of Computer Applications，2014，91(16)：25-28

[8] Zhu R，Xie J C，Xiao N，et al. Research on Autonomic Collaborative Caching in RAM Grids[J]. Computer Engineering & Science，2008，30(8)：111-115
褚瑞，谢健聪，肖侬，等. 内存网格中的自主协同缓存技术研究[J]. 计算机工程与科学，2008，30(8)：111-115

[9] Pan P，Lu Y S. R-tree Updating Caching and Batch Processing Mechanism Based on Gird[J]. Computer Engineering，2008，34(15)：28-30(in Chinees)
潘鹏，卢炎生. 基于栅格的 R 树更新缓存与批处理机制[J]. 计算机工程，2008，34(15)：28-30

[10] Yang C H，Wang L S. Prefetching T-Tree：A Cache-optimized Main Memory Database Index Structure[J]. Computer Sience，2011，38(10)：161-165(in Chinese)
杨朝辉，王立松. pT-树：高速缓存优化的主存数据库索引结构[J]. 计算机科学，2011，38(10)：161-165

[11] Sleator D D，Tarjan R E. Self-Adjusting Binary Search Trees [J]. Journal of the Association for Computing Machinery，1985，32(3)：652-686

[12] Huang M，Cai Z G. Survey of Cache Replacement Algorithm [J]. Computer Sience，2006，23(12)：191-193(in Chinese)
黄敏，蔡志刚. 缓存替换算法研究综述[J]. 计算机科学，2006，23(12)：191-193

[13] Jiang S，Zhang X. LIRS：An efficient low inter-reference recency set replacement policy to improve buffer cache performance[C]//ACM SIGMETRICS Conf (SIGMETRICS' 2002). Marina Del Rey，California，2002

[14] 姜宁康，时成阁. 网络存储导论[M]. 北京：清华大学出版社，2007：5-6