集群协作缓存机制研究*)

魏文国[1] 陈潮填[2] 闫俊虎[1]

(广东技术师范学院电子信息工程系 广州 510665)[1] (广东技术师范学院计算机科学系 广州 510665)[2]

摘 要 计算机集群中的节点使用内存一般不均衡,往往有些节点使用太多内存,而其他节点又有较多的空闲内存。为了改进集群操作系统,将集群节点的内存作全局分布的资源使用,我们首先提出一个内存互操作高速缓存方案:通过使用集群范围内的内存作文件高速缓存,从其他节点的高速缓存中读文件,可以避免很多低速的磁盘访问,改进集群文件系统的总体吞吐量。然后利用我们提出的缓存页面代替策略 GCAR 来支持这种内存互操作的高速缓存方案。该算法与 CAR 相比,对缓存中被"经常"使用的页面的管理粒度更细,更适合集群协作缓存的计算环境。实验结果表明,GCAR 对本地缓存的命中率比 CAR 略好,在集群协作缓存下能取得更好的缓存命中率。

关键词 集群计算,缓存,互操作,页代替算法,命中率

Study of Interactive Caching in Cluster Computing

WEI Wen-Guo[1] CHEN Chao-Tian[2] YAN Jun-Hu[1]

(Department of Electronic Information Engineering, Guangdong Polytechnic Normal University, Guangzhou 510665)[1]

(Department of Computer Science, Guangdong Polytechnic Normal University, Guangzhou 510665)[2]

Abstract In generally there will be some nodes in a cluster that are short of memory while others have idle memory wasted. How to improve the cluster system to support the use of cluster-wide memory as a global distributed resource? In this paper, at first we propose an interactive caching scheme, which allows the operating system to avoid many expensive disk accesses by using cluster-wide memory for file reading, so improves the overall throughput of cluster file systems. Further we also integrate cache replacement policy GCAR with this interactive caching scheme, GCAR has smaller granularity to manage "frequency" pages of cache than CAR, so it is more suitable for cluster computing environment. Experiment results show that GCAR can improve cache hit-ratios both at local cache and interactive cache in cluster computing.

Keywords Cluster computing, Cache, Interactive operation, Page replacement algorithm, Hit-ratios

1 问题概述

现代的网络技术,例如 Myrinet 和 Gigabit Ethernet,数据传输率已经超过了磁盘,并且以更快的速率发展。磁盘的寻道和旋转延时(latency)与网络的初始化延时相比更是慢很多。据估计,通过高带宽、低延时的网络从远程内存中读取数据比本地磁盘中读数据大约快 10~20 倍[1]。另外,内存价格的连续下降使得内存不再是集群系统的性能瓶颈;另一方面,集群内存的使用率异常低,文[1]的实验研究表明:60%~68%的内存未充分使用。

使用远程内存比较激进的方法是 XMM subsystem[2]。XMM 将所有分布内存管理集成到单一的子系统,并且允许在虚拟内存级别完全跨节点透明地使用。系统中的所有进程被允许以相同的方式访问内存,就好像在单一系统上执行一样。我们的缓存互操作在设计决定和底层假设上不同于 XMM,因为维持文件一致性的系统开销会减弱它所带来的好处,所以我们只将远程内存用于只读的文件缓存。进一步假设除了文件服务器外其他节点都可能失效,并且缓存互操作策略能优雅地处理节点的加入、删除和失效。无论如何,XMM 没有这种机制,随着节点数量的增加节点失效的可能性加大,所以我们设计的系统比 XMM 的可靠性更高。

我们的远程页代替算法 GCAR(generalized CAR-like)从经典的缓存页面代替策略 CAR(clock with adaptive replacement)中取得灵感,对缓存中的页面区分为"最新的"页面和"最经常"使用的页面。对"最经常"使用的页面是通过一个引用计数来识别的,使得它更适合远程页面的管理。本文剩余部分的章节组织如下:第 2 节研究集群内的互操作高速缓存方案;第 3 节给出改进的缓存页面代替策略 GCAR;第 4 节通过 I/O traces 对比测试 LRU、CAR 和 GCAR 的性能,并在真实的集群计算环境下对比 GCAR 和 CAR 对缓存命中率的影响;最后总结全文。

2 互操作的高速缓存方案

在我们的互操作高速缓存方案中,假设集群中每个节点信任其他节点,节点任何时候都可能失效;进一步假设集群中有一个文件服务器,这个文件服务器使用缓存目录的数据结构来跟踪被每个客户缓存的文件。给定一个文件标示符和文件中的块号,在目录缓存中查找相应的数据块信息,例如数据块的当前版本号和在哪个节点上缓存了该数据块。我们扩展集群文件系统的读策略,但是不改变其写策略。写策略可以是直接写、关闭时写和延迟写。所有修改的数据块仍然写回服务器,否则就不能实现互操作缓存。

互操作的高速缓存方案如图 1 所示,详细解释如下。

情形 1:没有远程缓存数据。当客户 P 在它本地缓存中没有找到想要的数据块,它发送读请求到文件服务器,服务器查询它的目录缓存。如果没有客户需要的缓存数据,服务器

*)本课题受广东省自然科学基金(编号:06025383)资助。**魏文国** 副教授,研究方向:集群、计算机网络和高性能计算;**陈潮填** 博士,教授;**闫俊虎** 教授。

检查它的本地缓存。若不存在，服务器从磁盘中读数据，保存到本地缓存并且发送给 P。否则，服务器直接从本地缓存中发送数据。一旦 P 收到数据，服务器建立一个新的目录缓存项，表示 P 缓存了该数据块。

情形 2：有远程缓存数据。如果任何客户（例如客户 Q）缓存了被请求的数据块，并且服务器正是高负载状态，服务器转发读请求给该客户，客户 Q 发送数据给 P。一旦 P 接受数据，它发送一个包含数据块的版本号的 NOTIFY 给服务器，表示 P 已经收到数据。当服务器收到通知，它比较该数据块的当前版本号和收到数据的版本号。如果它们匹配，服务器更新它的目录缓存并且发送一个正确的确认包给 P。否则它发送错误的确认包给 P，因此 P 丢弃收到的数据并且重发数据读请求。为了避免客户的请求给服务器带来的高负载，服务器需要制定转发读请求到合适客户的策略，策略可以是轮流（poll）或基于客户负载的自适应策略。

情形 3：为了分担发送给服务器的读请求，不是每个请求都发送给服务器。在情形 2，当 Q 发送数据给 P 时，它同时告诉 P 它缓存的同一文件的其他数据块。当 P 需要读 Q 已经缓存的数据块时，P 直接发送读请求给 Q。一旦 P 接受了数据，类似情形 2，它通知服务器，服务器和情形 2 一样发送正确或错误的确认包给 P。当 Q 中缓存的数据因为内存页代替而不可用时，Q 转发请求给服务器。服务器查找自己的目录缓存，直接响应读请求或转发该请求。

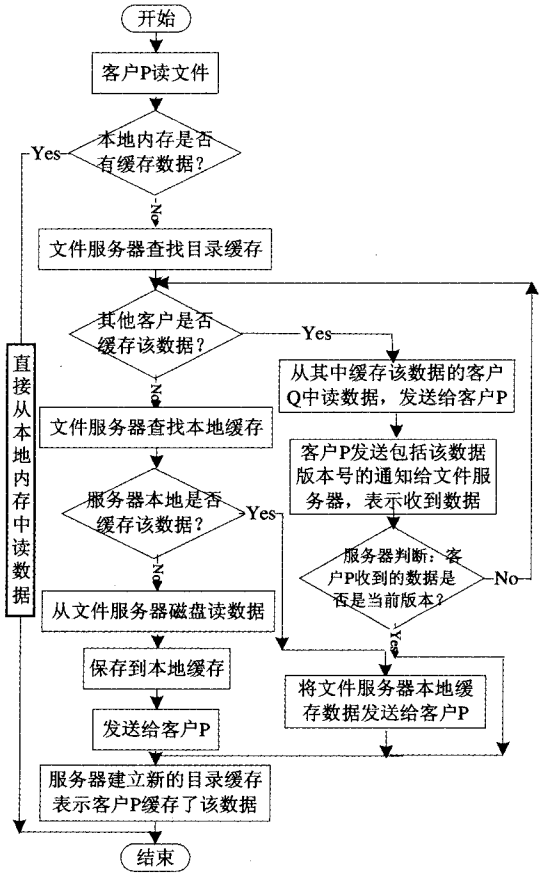

图 1　互操作高速缓存方案

文件服务器在它的目录缓存中保留了所有缓存数据的全局信息。当一个文件被任一客户删除或改变时，服务器也相应地更新目录缓存中的数据块记录。增加更多的客户不会给服务器增加更多的负载，因为增加的客户也会响应读请求。由于所有客户对文件缓存的互操作，大多数读服务器磁盘的访问被避免，服务器不需要从磁盘读大的数据块，只是发送小的信息和接受小的通知和确认包。服务器额外增加的工作是检索和维护目录缓存，操作系统可以很容易地与 I/O 操作交替进行，因此额外的工作不会太多地影响服务器的性能，但系统的整体吞吐量得到改进。

3　远程页代替策略

我们的远程页代替策略是为了使最有价值的页保留在集群范围内的远程内存中又不影响本地的性能。比较流行的缓存代替策略是 LRU：代替最近最少使用的页面。LRU 的好处是实现极其简单，具有常数的时间和空间复杂度，并捕捉了很多工作负载的“最近期”或者“聚集的引用位置”等特性。但是 LRU 的缺点有三：在每次命中缓存时，它必须移动到最多最近使用（MRU）的位置。在异步计算环境下，多个线程都试图将页面移动到 MRU 位置是不可接受的；LRU 捕捉工作负载的“最近期”特性，但是没有捕捉“频率”特性；LRU 容易被内存扫描影响，即一系列一次使用的页面导致系统的低性能。

CAR[3]（clock with adaptive replacement）是基于 clock 的自适应缓存代替算法，它克服了 LRU 的所有三个缺点。基本的理念是使用两个时钟 T_1 和 T_2，T_1 包含“最近期”或者“短期”页面，T_2 包含“经常使用”或者“长期”页面。新的页面首先插入 T_1，经过一个相对较长的时间段后被转移到 T_2。通过使用某种精确机制记住从 T_1 转移到 T_2 中的页面，从而自适应地决定这两个列表的大小，并且不需要用户指定的初始化参数。但是 CAR 的内存页面管理只是简单地通过是否被引用来区分“短期”页面和“长期”页面，这种“初线条”的内存管理策略不利于内存的优化使用。

我们提出 GCAR（generalized CAR-like）缓存代替策略：更一般的 CAR 变体。假设缓存容量是 c 个页面，GCAR 维持含有 $2c$ 个页面的缓存目录：c 个缓存页面，c 个历史页面。GCAR 的缓存目录维持两个链表：L_1 和 L_2。L_1 包含最近被使用一次的页面，L_2 包含最近被使用至少两次的按照引用计数降序排列的页面；L_1 被看作“短时”页面，L_2 被当作“经常性”页面。GCAR 策略决定如何在 $2c$ 个页面中选择 c 个页面到缓存，基本的理念是：将 L_1 划分成 T_1（头部，近期的页面）和 B_1（尾部，比较陈旧的页面）；将 L_2 划分成 T_2（头部，被经常引用的页面）和 B_2（尾部，较少被引用的页面）。T_1 和 T_2 存放在缓存中，历史页面 B_1 和 B_2 存放在缓存目录中。从 T_1（T_2）中被踢出的页面被放到 B_1 或者 B_2，算法设置列表 T_1 的大小为 p，代替策略如下：若 $|T_1|\geqslant p$，则代替 L_1 中的页面；否则代替 L_2 的页面。自适应特性来自于 p 的大小，由工作负载动态调整：若历史列表 B_1 中的页面被命中，则增加 p；若 B_2 的页面被命中，减少 p。算法试图保持 B_1 具有 T_2 的大小、B_2 近似于 T_1 的大小，算法也限制 $|T_1|+|B_1|$ 不超过缓存大小。

具体的算法如下：

```
初始化：设置 p=0，设置链表 T1，T2，B1 和 B2 为空
输入：被请求的页面 x
if (x∈T1∪T2) then /*缓存命中*/
    x 的引用计数增加 1
else /*缓存错失*/
    if (|T1|+|T2|=c) then /*缓存满，从缓存中代替一个页面*/
        replace()
        /*缓存目录代替*/
        if ((x∉B1∪B2) and (|T1|+|B1|=c)) then
            丢弃 B1 的尾部页面
        elseif((x∉B1∪R2) and(|T1|+|B1|+|T2|+|B2|=2c))
then                丢弃 B2 的尾部页面
        endif
    elseif
    if (x∉B1∪B2) then /*缓存目录错失*/
        将 x 插入到 T1 的头部，将 x 的引用计数设置为 0
    elseif (x∈B1) then /*缓存目录命中*/
```

```
        自适应地增加链表 T1 的大小 p:p=min(p+max(1,⌊|B2|/|B1|⌋),c)[1)]
        将 x 的引用计数增加 1,并插入到有序链表 B2
    else /*x∈B2*/
        自适应地减少链表 T1 的大小 p:p=max(p-max(1,⌊|B1|/|B2|⌋),0)
        将 x 的引用计数增加 1 并插入到有序链表 T2
    endif
endif
```

replace()函数如下:

```
found=0
repeat
    if (|T1|≥max(1,p)) then
        if (T1 尾部页面的引用计数为 0) then
        found=1
        将 T1 的尾部页面踢出,插入到 B1 的头部
        else
            将 T1 的尾部页面的引用计数减少 1,踢出它并插入到有序链表 B2
        endif
    else
        if (T2 尾部页面的引用计数为 0) then
            found=1
            将 T2 的尾部页面踢出,插入到有序链表 B2
        else
            将 T2 的尾部页面的引用计数减少 1,插入到有序链表 T2(即调整其位置)
    endif
endif
    until (found)
```

对 GCAR 的理论分析如下:GCAR 与 CAR 相似,将缓存中的页面区分为"短时"页面(L_1 中的页面)和"经常性"页面(L_2 中的页面),不同之处在于:GCAR 对"经常性"页面的管理粒度更细,通过每个页面的被引用计数实现,而不是简单地通过是否被引用来区分"短期"页面和"长期"页面。直观来说,这种缓存管理策略对于集群协作缓存(分布在远程节点的缓存)可能更适合一些,因为 GCAR 在缓存中既保留了"最近期"的页面(L_1 的头部页面,即 T_1 中的页面),同时对一些经常使用的页面(L_2 的头部页面,即 T_2 中的页面)也保留在缓存中,避免了跨越网络的远程磁盘读取。另一方面,该算法的复杂性与 CAR 相比没有增加多少:CAR 的链表 L_2 是无序链表,插入操作的时间复杂度是 O(1);GCAR 的链表 L_2 中的页面按照被引用计数降序排列,插入操作的时间复杂度是 $\log_2(|L_2|)$。算法 GCAR 的实际效果如何,是否适合集群协作缓存的特殊环境?我们在第 5 节详细讨论。

4 实验结果

首先根据读磁盘操作的跟踪文件(I/O traces)作模拟试验,比较 LRU,CAR,GCAR 对本地内存的缓存命中率。本文所用的 I/O traces 没有经过上层缓存的处理,可以看作存储控制器或者磁盘的工作负载,并且我们只考虑读请求,测试指标是缓存命中率。选择两个例子做 3 种缓存代替策略的性能比较测试:第一个测试用例是运行 ERP 应用的商业数据库服务器[4],收集了该服务器 7 天的磁盘访问数据,其中读请求的数量是 43704979,有 10516352 个不同的读请求,每个缓存页面大小是 8kB,我们把该测试用例简称为 DS。第二个测试用例是使用 Vtrace 工具从 14 台 Windows NT 工作站收集的几个月的磁盘访问数据[5],其中读请求的数量是 490139585,有 47003313 个不同的读请求,每个缓存页面大小是 512 字节,我们把该测试用例简称为 Merge(P)。三种缓存代替算法 LRU,CAR,GCAR 的缓存命中率如图 2 所示。从中可以看出:随着缓存容量的增加,三种缓存代替策略的缓存命中率都呈增加的趋势,并且 GCAR 的缓存命中率在大多数情况下都比其他两种缓存代替策略好。该实验说明了作为本地缓存策略,GCAR 具有比 CAR 更好的缓存命中率(平均提高了 5.7%)。

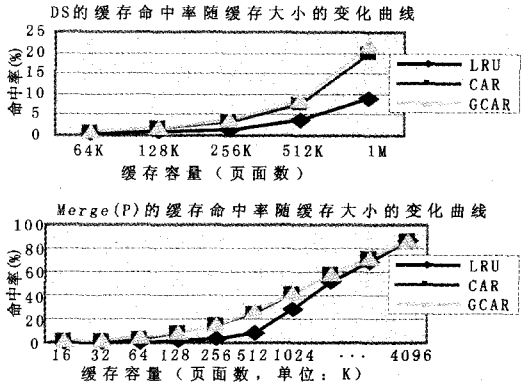

图 2 缓存代替策略 LRU,CAR 和 GCAR 的缓存命中率的比较

进一步,在 8 台 Linux(Fodera 2)的 PC 组成的集群环境下,有一个节点作为文件服务器,并且对 Fodera 2 的文件系统 Ext3 修改其读协议:分别将 CAR、GCAR 的缓存代替策略运用到第 3 节的互操作高速缓存方案中,对比测试 GCAR 和 CAR 的缓存命中率情况,结果如图 3 所示。从中可以看出,在集群高速协作缓存方案下的 GCAR(对应曲线 GCAR for cluster)和 CAR(对应曲线 CAR for cluster)比没有协作的单机模式下的 GCAR(对应曲线 GCAR)和 CAR(对应曲线 CAR)的缓存命中率都有明显的提高(分别平均提高了 16.6%和 13.2%)。

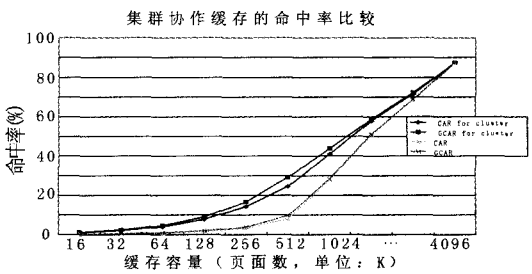

图 3 CAR 和 GCAR 在集群协作缓存和本地缓存的命中率比较

结论 本文提出了一个集群节点之间的内存互操作高速缓存方案,并利用我们提出的缓存页面代替策略 GCAR 来支持这种内存互操作的方案。实验结果表明,GCAR 在集群协作缓存下能取得较好的缓存命中率。将来的工作包括进一步完善该系统,在 I/O 的吞吐量等性能指标方面取得更好的结果。

参 考 文 献

1 XU YING, et al. Cooperative Caching in Linux Clusters. In: Proceedings of the Cluster World Conference and Expo 2003, San Jose, CA, San Jose, CA, Jun 2003. 23~25

2 Milojicic D S, Dean R W, Dominijanni M, et al. Extended Memory Management (XMM): Lessons Learned. Software Practice and Experience, 1998,28:1011~1031

3 Bansal S, Modha D S. CAR: Clock with Adaptive Replacement. In: Proceedings of the USENIX Conference on File and Storage Technologies (FAST), March 2004. 187~200

4 Hsu W W, Smith A J, Young H C. Characteristics of I/O traffic in personal computer and server workloads: [Tech Rep]. Computer Science Division, Univ California, Berkeley, July 2001

5 Hsu W W, Smith A J, Young H C. The automatic improvement of locality in storage systems: [Tech Rep]. Computer Science Division, Univ California, Berkeley, Nov 2001

[1)] ⌊|B2|/|B1|⌋表示不超过|B2|/|B1|的最大整数。