文章编号:1007-5321(2017)03-0091-06 **DOI**:10.13190/j.jbupt.2017.03.013

面向数据密集型应用的细粒度内存管理方案

郝晓冉, 倪 茂, 王力玉, 陈 岚
(中国科学院 微电子研究所 EDA 中心, 北京 100029)

摘要: 提出并实现了一种细粒度内存管理方案. 该方案以用户数据对象为粒度对内存进行管理,当需要将部分内存数据换出以获得空闲内存空间时,不再像传统操作系统以页为单位换出,而是将多个冷数据对象换出到底层存储设备中,最大限度地保证了被换出的数据近期内不会再次被访问,从而有效减少了额外的 I/O 开销. 测试结果表明,本方案与操作系统页交换机制相比,系统响应速度平均提高了 37.5%.

关 键 词: 计算机系统结构; 页交换; 数据密集型应用; 虚拟内存管理; 物理内存管理
中图分类号: TP302.1 **文献标志码**: A

Fine-Grained Memory Management Scheme for Data Intensive Applications

HAO Xiao-ran, NI Mao, WANG Li-yu, CHEN Lan
(Institute of Microelectronics of Chinese Academy of Sciences, the EDA Center of Chinese Academy of Sciences, Beijing 100029, China)

Abstract: For data intensive applications, a fine-Grained memory management scheme was proposed. The object-granularity data exchange between dynamic random access memory (DRAM) and swap device scheme was achieved. When DRAM is to be exhausted, the proposed scheme will swap data objects with low access frequency out to swap device, which reduces extra system I/O effectively. Compared to Linux-swap system, the proposed scheme improves system performance up to 37.5% in average.

Key words: computer system architecture; page swapping; data intensive application; virtual memory management; physical memory management

数据密集型应用在运行过程中往往需要占用大量的物理内存空间[1-2],当系统空闲物理内存小于某一阈值时,页面交换机制把被访问概率较小的冷内存页临时换出至交换分区中[3]. 1 个 4 KB 的内存页中往往包含多个冷热度不同的用户数据,当被换出页中的某一数据再次被访问时,系统需要将该内存页从交换分区中换入内存,增加了系统的 I/O 开销. 由于底层存储设备的访问速度与动态随机存取存储器内存(DRAM, dynamic random access memory)相比相差甚远,因此对于数据密集型应用,内存与交换分区之间的数据交换将严重影响系统的响应速度.

越来越多的企业,如 Facebook[4]、Google[5]、MySpace[6]等,使用基于闪存的固态硬盘(SSD, solid state drives)部分或全部替代传统的硬盘. 针对闪存的特性,提出了 SSD 交换分区的优化方案. Sohyang Ko[7-8]、Guangxia Xu[9]等通过优化 SSD 垃圾回收算法,减少对 SSD 的擦除和数据拷贝的次数. Yunjoo Park[10]等将新型非挥发存储器件相变存储器(PCM, phase change memory)作为交换分区的存储介质,通过减小页的尺寸,关闭 read-ahead 选项,提高了基于 PCM 的交换分区的性能. 上述研究根据存储介质特性,通过优化其自身的管理过程,从一定程度上克服了将其作为交换分区存储介质时存在

收稿日期: 2016-10-10
作者简介: 郝晓冉(1980—), 女, 助研, E-mail: haoxiaoran@ime.ac.cn;陈 岚(1968—), 女, 教授, 博士生导师.

的问题.

笔者针对数据密集型应用,从优化 DRAM 与 SSD 交换设备之间的数据交换过程入手,提出了一种细粒度内存管理方案(FG_MM, fine grained_memory management). 定义用户 1 次申请的数据空间为 1 个数据对象,FG_MM 以数据对象为粒度对内存进行管理[11],尽可能保证被换出的数据全部为冷数据对象,有效防止了因部分热点数据一并被换出而导致的底层存储设备访问次数的增加.

1 FG_MM

1.1 整体架构

FG_MM 的整体架构如图 1 所示. FG_MM 首先向系统申请一段连续内存空间供其自行管理,当用户通过 FG_MM 中的内存分配接口申请内存空间时,FG_MM 将从这段内存空间中为其分配所需的空间,当申请的这段内存空间用尽时,FG_MM 将再次向系统申请一段内存空间进行分配. 当用户需要访问内存数据时,如果所访问的数据在 DRAM 内存中,则系统将所需数据直接返回给用户;如果所访问的数据不在 DRAM 内存中,发生了系统缺页异常,将触发 FG_MM 中的缺页异常(FGMM_Page-Fault, fine grained_memory management page-fault)处理过程:若该数据空间由系统分配,将转向系统的缺页异常处理函数进行缺页异常处理;若该数据空间由 FG_MM 分配,则通过查询数据对象查找表(OT, object table),其中记录了虚拟地址到 SSD 逻辑地址的映射关系,根据被访问页的虚拟地址获得该页在 SSD 中的逻辑地址,然后根据该逻辑地址,将数据从 SSD 换入 DRAM 中,然后返回给用户.

1.2 虚拟页的管理及其与物理页的映射

要实现 DRAM 与 SSD 之间以数据对象为粒度的数据交换,在进行虚拟内存分配时每个虚拟内存页只保存 1 个数据对象[11]. 如果虚拟页与物理页仍然保持一一对应的关系,将造成大量物理内存的浪费. 因此采用多个虚拟页映射到同一个物理页的多对一的映射机制[12].

1 个物理页被划分成多个大小相同的细粒度页(FG_page, fine-grained page),划分的粒度 G 可以是 0.5、1、1.5、2、3.5、4 KB. 当 4 KB 不能被 G 整除时,4 KB 中剩余的部分将被划分到相应粒度大小的内存空间中,例如当 $G = 1.5$ KB 时,4 KB 中剩余的 1 KB 将被划分到 G = 1 KB 的内存空间中.

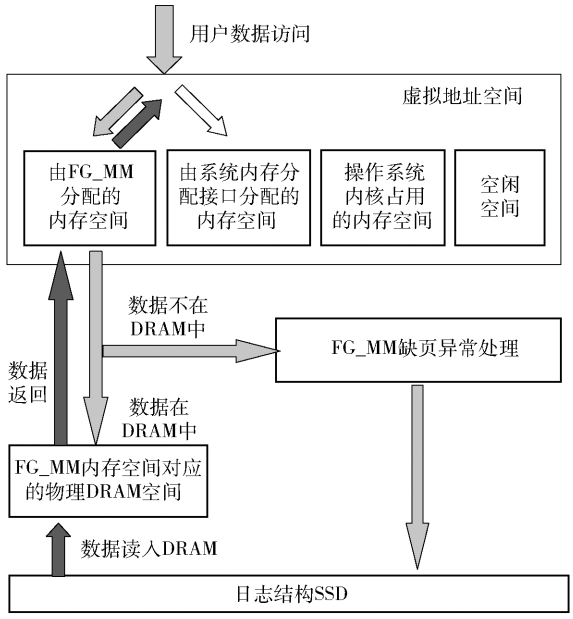

图 1 FG_MM 系统框图

如图 2 所示,要实现多个虚拟页向同一个物理页的映射,在分配虚拟页时,需要按照不同的页内偏移量保存数据对象,这样虽然 3 个虚拟页映射到同一物理页中,仍然可以根据不同的页内偏移量访问所需的数据对象. 通过手动填写页表(PTE, page table entry)中物理地址的方式,完成多个虚拟页向同一个物理页的映射.

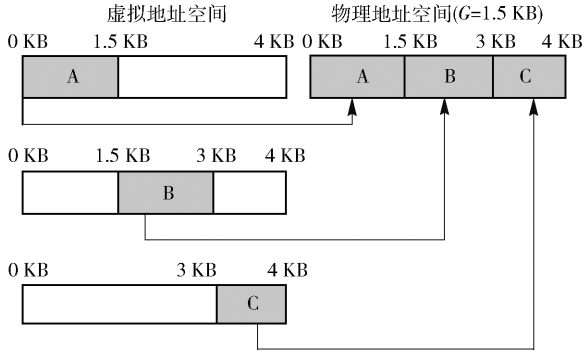

图 2 虚拟页的管理及其与物理页的映射

1.3 物理内存的管理

每当 FG_MM 管理的内存用尽时,FG_MM 会向系统申请 1 MB 连续的内存空间,称为 1 个内存块,并根据当次用户请求中数据对象的大小(size)将这一内存块划分成多个相同大小、不同偏移量(offset)的 FG_page,并称一种数据对象大小、偏移量的组合(size, offset)为 1 个内存分配模式(shape),每个模式都有一个空闲内存列表(free-list),划分好的 FG_page 被链入相应的空闲内存列表中. 每个 shape 还

对应两个先进先出(FIFO, first in first out)链表,活跃的 FIFO 链表和不活跃的 FIFO 链表,活跃的 FIFO 链表用于存放具有此种模式的活跃 FG_page,不活跃的 FIFO 链表则用来存放不活跃的 FG_page,当系统没有 1 MB 的连续空间可以分配给 FG_MM 时,FG_MM 将从不活跃 FIFO 中淘汰数据.

首次写入用户数据的 FG_page,根据 FG_page 的 shape,链入相应的活跃的 FIFO 的头部,并将其 PTE 中访问控制位置 1,表示该 FG_ page 刚刚被访问过,是活跃页. 当活跃的 FIFO 链表满时,将从其尾部开始检测每个 FG_ page 的访问控制位,若该位为 1,则将其置 0,并重新链入到活跃的 FIFO 的头部;若该位为 0,则将其淘汰至不活跃的 FIFO 中. 当不活跃的 FIFO 满时,检测其尾部 FG_ page 的访问控制位,若该位为 1,则将此 FG_ page 对应的访问控制位置 0,并重新链入到活跃的 FIFO 的头部;若该位为 0,则将其淘汰至 SSD 中. 图 3 以 size = 1 为例说明了物理内存的管理过程.

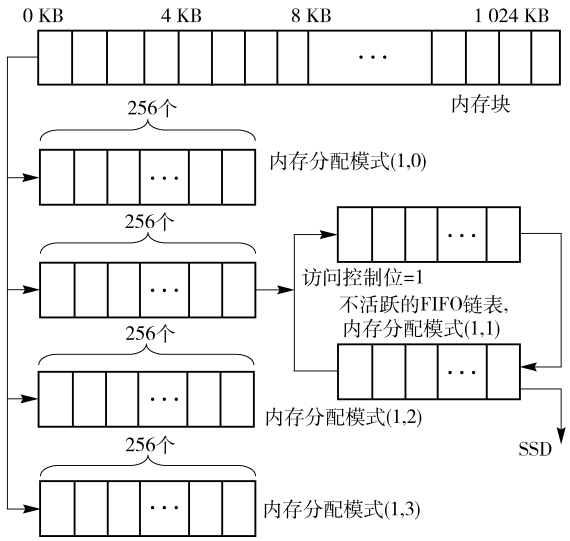

图 3　物理内存管理

1.4　SSD 的管理

为了建立虚拟地址与 SSD 逻辑地址之间的映射,FG_MM 定义了 OT,OT 以虚拟地址为索引,其内保存了数据对象的大小、页内偏移及在 SSD 中的逻辑地址. 当成功分配 1 个虚拟内存页时,FG_MM 会将该虚拟页中数据对象的大小及页内偏移量写入 OT 中. 当该数据对象被淘汰至 SSD 中时,FG_MM 将把它在 SSD 中的逻辑地址写入 OT 中.

基于闪存的 SSD 采用日志管理的方式管理[13-14],SSD 以 4 KB 为单位读取数据,而以块(如块大小为 256 KB)为单位写入数据,为了适应 SSD 这一读写特性,FG_MM 预留了 1 个块缓存,需要被淘汰至 SSD 的数据对象连同其虚拟地址先被保存在块缓存中,当块缓存存满后,会将整个块缓存中的数据一起写入 SSD 中. 根据每个块头部保存的其内所包含的数据对象的虚拟地址,FG_MM 将每个数据对象在 SSD 中的逻辑地址写入 OT 中.

因为 SSD 是按块写入,所以需要对 SSD 的逻辑地址空间进行垃圾回收. 当 SSD 可用的逻辑空间小于某个阈值时,FG_MM 将在后台启动垃圾回收过程,系统为整个 SSD 逻辑地址空间创建了垃圾回收表,表中保存了每个数据块中无效数据的总量,当 1 个数据块中所包含的无效数据量超过某个阈值时,该数据块将成为垃圾回收过程的备选数据块.

1.5　系统实现

为了方便用户使用以及与 Linux 系统的集成,本方案的设计遵循以下原则:FG_MM 提供的内存分配、释放接口,数据读、写接口完全独立于系统原有的接口,且两者管理的虚拟地址空间与物理地址空间也完全独立. 这样用户可以根据应用的特点(如用户数据的大小或数据的重要程度),灵活选择调用的接口. 例如,当用户数据对象大小大于 4 KB 或者用户数据对象为经常被访问的元数据时,用户可以调用系统原有接口进行内存的分配与管理.

本方案以库的形式对 FG_MM 内存管理系统进行封装,提供了以下 6 个接口供用户调用.

1) void sysInit( ), void sysQuit( ):sysInit( )用于初始化 FG_MM 系统中的元数据;sysQuit( )用于在系统退出前释放所用的内存空间.

2) void * nv_malloc(SIZE_t userReqSize):该函数用于为每个数据对象分配大小为 userReqSize 的内存空间.

3) void * nv_calloc(SIZE_t nmemb, SIZE_t userReqSize):该函数为 nmemb 个数据对象分别分配大小为 userReqSize 的内存空间.

4) void * nv_realloc(void * vaddr, SIZE_t userReqSize):该函数把由 vaddr 指向的内存空间的大小调整为 userReqSize 字节.

5) void nv_free(void * vaddr):该函数用于释放指针 vaddr 指向的内存空间.

1.6　适用的应用场景

FG_MM 适用于用户所需物理内存空间远大于系统所能提供的最大物理内存空间的应用场景,如

内存数据库、内存缓存系统等. 例如在内存数据库的应用中,为了提高数据的处理速度,用户数据被存储在DRAM中而非底层存储设备中. 这就需要系统能够提供足够的物理内存空间存放数据,但是由于DRAM在存储容量、功耗以及成本等方面的局限性,使得系统不可能无限制增加DRAM容量以满足用户的需求. 当内存中无法容纳更多用户数据时,就会发生DRAM与底层存储设备之间的数据交换. 在这种情况下,FG_MM的细粒度内存管理方案可以有效减少内存与底层存储设备间不必要的换入换出操作,从而减少数据交换过程对系统性能的影响.

注意,对于大部分数据对象大小都较小(远小于最小划分粒度,如256 B)的情况,本方案并不适用,因为这将造成大量物理内存空间的浪费以及元数据的大量增加,此时建议调用系统原有的内存管理函数进行内存空间的分配与管理.

2 测试结果及分析

选择不同的测试用例对FG_MM与Linux操作系统内存交换机制之间的性能进行比较. 包括随机访问测试程序、数据索引测试程序B-tree,以及快速排序测试程序Quicksort等.

实验中,测试用服务器为8核i7-4790 3.6 GHz处理器,存储系统使用6 GB的DRAM和120 GB的金士顿SSD. 服务器操作系统是以64-bit Linux 2.6.32为核心的CentOS系统. 该系统为FG_MM预留了一段物理内存空间供其自行管理. FG_MM与系统内存交换机制用于与DRAM进行数据交换的SSD存储区域相互独立. 为了测试大内存数据下系统的性能,设定由FG_MM自行管理的DRAM内存容量为64 MB,而测试程序的数据总量为10 GB,远大于FG_MM管理的内存空间,这样只有少部分数据驻留在DRAM中,而大部分数据需要保存在SSD中,在随机访问内存数据时就会频繁产生DRAM与SSD之间的数据交换. 由于FG_MM系统元数据(OT,free-list,活跃的/不活跃的FIFO)将占用部分系统DRAM空间,因此FG_MM实际占用的DRAM空间大于64 MB,为了比较的公正性,在测试内存交换机制系统性能时,也将为其设置与FG_MM实际占用内存总量相等的内存空间.

2.1 随机访问

对大量内存数据进行随机访问,为了使访问特性更符合实际情况,设计数据访问特性符合Zipf定律,即80%的访问集中在20%的数据中. 内存数据集总大小为10 GB,被访问的数据总量为1 GB. 不同的读写比例下,FG_MM与系统内存交换机制之间的性能比较如图4所示. 图中,纵坐标响应时间为测试程序的运行时间(以下各实验同). 从图4可以看出,在相同的运行环境下,FG_MM系统的数据响应速度与系统内存交换机制相比提高了42%~44%. FG_MM以数据对象为粒度对数据进行管理,当DRAM无可用空间,需要向SSD淘汰数据时,也将以数据对象为粒度进行淘汰,这样最大程度保证了所淘汰的数据为冷数据. 而系统内存交换机制以4 KB为单位进行数据的淘汰,如果被淘汰的页中包含热点数据对象,那么被淘汰的页可能会在短期内再次被换入内存,增加了系统访问底层设备的开销. 当读写比例为8:2时,在不同数据对象大小下,FG_MM与系统内存交换机制之间的性能比较如图5所示. 图中固定512 B、固定1 KB、固定2 KB分别表示数据对象大小固定为512 B、1 KB和2 KB;随机512 B、随机1 KB、随机2 KB分别表示数据对象大小随机,平均值为512 B,1 KB和2 KB. 由图5可以看出,FG_MM系统的数据响应速度与系统内存交换机制相比提高了30.6%~47.5%,无论数据对象是固定大小还是随机大小,随着数据对象大小与4 KB不断接近,FG_MM与系统内存交换机制之间的性能差异逐渐减小. 因为当数据对象大小逐渐增大时,其与原Linux系统中4 KB的交换粒度逐渐接近,因此FG_MM与原Linux系统之间的性能差异也将逐渐减小.

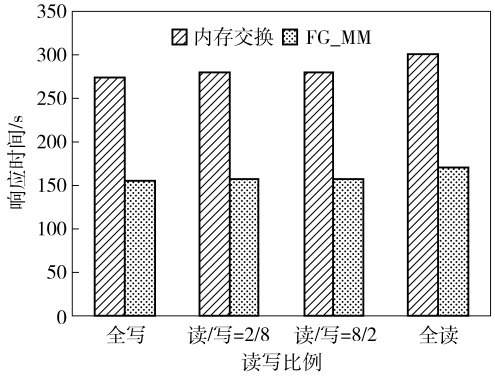

图4 不同读写比例下随机访部测试结果

2.2 B-tree

B-tree平衡多路查找树用于快速查找数据,树中每个节点根据实际情况可以包含大量的关键字信息和分支,这样树的深度降低了,能够快速访问到要

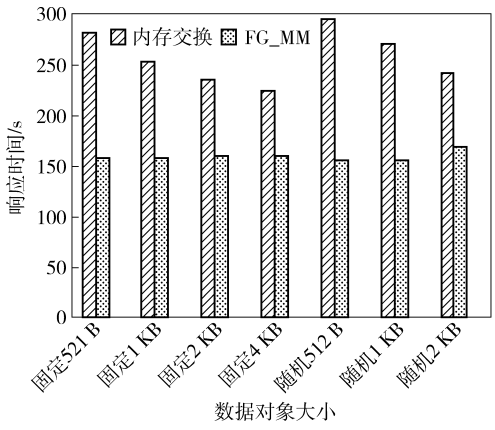

图 5　不同数据对象大小下随机访部测试结果

查找的数据. 在该测试程序中,B-tree 树结构占用的内存空间由 Linux 系统分配和管理,树结构中每个关键字对应的数据空间由 FG_MM 进行分配和管理. 测试中,内存数据总量为 10 GB,被访问的数据总量为 1 GB,读写比例为 8∶2.

不同数据对象大小下,FG_MM 与系统内存交换机制对测试程序 B-tree 的响应时间如图 6 所示. 从图中可以看出,当数据对象大小固定时,FG_MM 的响应速度与系统内存交换机制相比提高了 24% ~45%;当数据对象大小随机时,FG_MM 的响应速度与系统内存交换机制相比提高了 42% ~50%.

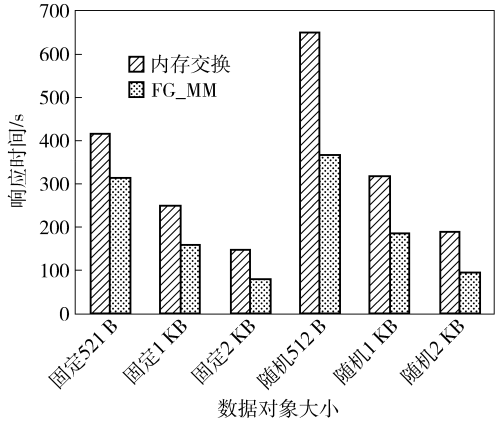

图 6　不同数据对象大小下 B-tree 测试结果

2.3　Quicksorting

Quicksorting 测试程序随机从 10 GB 的数据集中选取 1 GB 的数据,按照一定的规则进行排序. 此应用的读、写操作都较为频繁. 因进行排序的数据通常为同一类型的数据,因此,数据对象大小为固定大小. 在不同固定数据对象大小下,FG_MM 与系统内存交换机制的系统响应时间如图 7 所示. 由图 7 可见,由于 FG_MM 采用了以数据对象为粒度的内存管理方式,使得内存数据的换入、换出更加高效,有效减少了不必要的换入换出操作,从而响应速度提高了 27% ~57%.

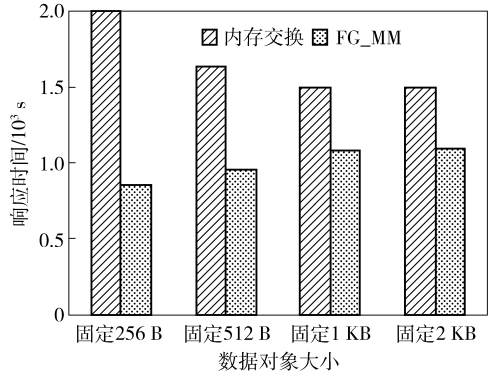

图 7　不同数据对象大小下 Quicksorting 测试结果

2.4　Bloom filter

Bloom filter 测试用例通过 8 个哈希表实现关键字的插入和索引. 当需要插入 1 个关键字时,该关键字最多可以映射到 8 个不同的数据对象中,这时就需要读取这 8 个数据对象并对相应的映射位进行修改,当修改后的数据对象需要被换出时,需要将其写回到 SSD 中. 因此,该测试用例属于读写操作都较多的应用. 由于 Bloom filter 测试用例本身的特性,测试只能选择固定的数据对象大小. 由图 8 可以看出, FG_MM 系统响应速度与系统内存交换机制相比提高了 16% ~31%.

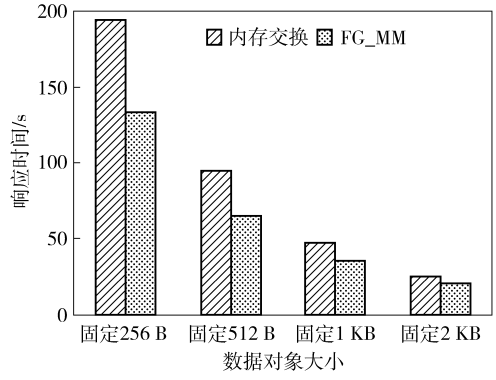

图 8　不同数据对象大小下 Bloom filter 测试结果

2.5　元数据开销

FG_MM 方案额外的元数据内存开销主要由两部分组成.

1）FG_MM 为每个细粒度物理页分配 1 个 24 B 的结构体对其进行管理. 在极端情况下,当用户请

求的数据对象大小都小于最小页粒度 $C_{\min}$ 时,所有物理页都按照最小页粒度的大小进行划分,此时,管理物理内存所需要的元数据开销 P_{meta} 也将达到最大值(假设 FG_MM 管理的物理内存大小为 M):

$$P_{\text{meta}} = \frac{M}{C_{\min}} \times 24$$

2) FG_MM 为用户虚拟地址和 SSD 逻辑地址之间建立了基于数据对象的映射表 OT. 该表的每个表项大小为 8 B,保存了数据对象的大小、偏移和 SSD 逻辑地址等信息. 该表的总大小 O_{meta} 与用户请求的数据对象个数 N_{obj} 有关,随着 N_{obj} 的增加,O_{meta} 的大小线性增长.

$$O_{\text{meta}} = N_{\text{obj}} \times 8$$

对于操作系统原有内存管理过程,当用户通过系统库函数 malloc( ) 等分配内存空间时,系统也需要利用一定大小的元数据记录数据大小等信息. 当物理页被换出到交换分区中时,物理页在交换分区中的索引信息被记录在页表项 PTE 中的物理地址字段中,无需额外建立映射表. 因此 FG_MM 的元数据内存开销主要来源于 OT. 但是,因为 FG_MM 采用细粒度的内存管理方案,减小了不必要的内存数据换入换出。上述实验结果表明,本方案能够有效减少程序运行时间,元数据增加对系统性能产生的不良影响小于本方案对系统带来的利好.

3 结束语

在 Linux 等传统的操作系统中,内存系统都是以页为单位进行管理的,当物理内存占用量超过一定阈值时,为保证有足够的内存供系统使用,系统将部分内存数据换出到交换分区中,当被换出的数据再次被访问时,需要将该数据从交换分区中换入内存以供系统访问. 1 个页中往往包含多个冷热度不同的用户数据,当被换出的页中包含热点数据时,该页就可能在短时间内再次被换入内存,造成了不必要的内存数据换入换出操作,影响了系统的响应速度.

笔者针对数据密集型应用,提出并实现了一种细粒度的内存管理方案 FG_MM,该方案不再以 4 KB 为粒度对内存空间进行管理,而是以数据对象为粒度分配、管理内存. 当内存用尽,需要与底层存储设备发生数据交换时,内存数据的换入、换出也将以数据对象为粒度进行管理,这样系统可以将多个冷的数据对象换出到底层存储设备中,减少了不必要的内存数据换入换出操作,提高了系统的响应速度. 实验结果表明,在不同的应用场景下,FG_MM 与系统内存交换机制相比,响应速度平均提高了 37.5%.

FG_MM 内存管理粒度为固定大小(256 B,512 B, 1 KB,…,4 KB),会在一定程度上造成物理内存的浪费. 例如,当用户申请 128 B 的内存空间时,系统也将为其分配 256 B 的内存空间,产生内存碎片,如何对产生的碎片内存进行合理的管理与利用将是本方案未来优化的重点.

参考文献:

[1] Chen C L P, Zhang Chunyang. Data-intensive applications, challenges, techniques and technologies: A survey on Big Data[J]. Information Sciences, 2014, 275(11): 314-347.

[2] Vibha B, Rahul J. Big data analysis: Issues and challenges[C]// EESCO 2015. Visakhapatnam: IEEE Press, 2015: 1-6.

[3] Mauerer W. Professional linux kernel architecture[M]. Beijing: Posts & Telecom Press, 2010: 821-824.

[4] Svwedlik Y. Facebook accelerates storage performance with flashcache update[EB/OL]. (2013-10-09)[2016-10-02]. http://www.datacenterdynamics.com/content-tracks/servers-storage/facebook-accelerates-storage-performance-with-flashcache-update/82690.fullarticle.

[5] Albrecht C, Merchant A, Stokely M, et al. Janus: Optimal flash provisioning for cloud storage workloads[C]// 2013 USENIX Technical Conference. San Jose: USENIX Press, 2013: 91-102.

[6] Armas C. MySpace replaces storage with solid-state drive technology in 150 standard load servers [EB/OL]. (2009-12-14)[2016-10-02]. https://www.infoq.com/news/2009/12/myspace-ssd.

[7] Ko S, Jun S, Ryu Y, Kwon C, et al. A new linux swap system for flash memory storage devices [C]// ICCSA 2008. Perugia: IEEE Press, 2008: 151-156.

[8] Ko S, Jun S, Kim K, Ryu Y. Study on garbage collection schemes for flash-based linux swap system [C]// ASEA 2008. Sanya: IEEEPress, 2008: 13-16.

[9] Xu Guangxia, Wang Manman, Liu Yanbing. Swap-aware garbage collection algorithm for NAND flash-based consumer electronics [J]. IEEE Transactions on Consumer Electronics, 2014, 60(1): 60-65.

(下转第 103 页)

puter Communication Review, 2013, 43(3): 59-67.
[6] Nam S W, Kim D, Yeom I. Content verification in named data networking[C]//2015International Conference on Information Networking. Cambodia: IEEE, 2015: 414-415.
[7] Kim D, Nam S, Bi J, et al. Efficient content verification in named data networking[C]//Proceedings of the 2nd ACM Conference on Information-Centric Networking. New York: ACM, 2015: 109-116.
[8] DiBenedetto S, Papadopoulos C. Mitigating poisoned content with forwarding strategy[C]//Computer Communications Workshops (INFOCOM WKSHPS). San Francisco: IEEE, 2016: 164-169.
[9] Li Yun, Zhao Ling, Liu Zhanjun, et al. N-Drop: congestion control strategy under epidemic routing in DTN [C]//International Conference on Wireless Communications and Mobile Computing: Connecting the World Wirelessly. Leipzig: ACM, 2009: 457-460.
[10] Wang Guoqing, Huang Tao, Liu Jiang, et al. Modeling in-network caching and bandwidth sharing performance in information-centric networking[J]. The Journal of China Universities of Posts and Telecommunications, 2013, 20(2): 99-105.
[11] 崔现东, 刘江, 黄韬, 等. 基于节点介数和替换率的内容中心网络网内缓存策略[J]. 电子与信息学报, 2014, 36(1): 1-7.
[12] Zhu Yi, Mi Zhengkun, Wang Wennai. A probability caching decision policy with evicted copy up in content centric networking [J]. Journal of Internet Technology, 2017, 18(1): 33-43.

(上接第 96 页)

[10] Park Y, Bahn H. Management of virtual memory systems under high performance PCM-based swap devices [C]//COMPSAC 2015. Taichung: IEEE Press, 2015: 764-772.
[11] Anirudh B, Vivek S P. SSDAlloc: hybrid SSD/RAM memory management made easy[C]//2011 Conference on Networked Systems Design and Implementation. Boston: ACM Press, 2011: 211-224.
[12] Anirudh B, Vivek S P. Better flash access via shapeshifting virtual memory pages [C]//SIGOPS 2013. Farmington: ACM Press, 2013: 1-14.
[13] Jian Xu, Steben S. NOVA: a log-structured file system for hybrid volatile/non-volatile main memories[C]//2016 USENIX Conference on File and Storage Technologies. Santa Clara: USENIX Press, 2016: 323-338.
[14] Mendel R, John K O. The design and implementation of a log-structured file system[J]. ACM Transactions on Computer Systems, 1992, 10(1): 26-52.