doi:10.3772/j.issn.1002-0470.2016.08 – 09.005

UGD:面向固态盘缓存系统的数据过滤技术①

张子刚②[*,**,***] 蒋德钧[*,**] 孙凝晖[*,**]

([*]中国科学院计算机体系结构国家重点实验室 北京 100190)

([**]中国科学院计算技术研究所 北京 100190)

([***]中国科学院大学 北京 100049)

摘 要 为了提高固态盘(SSD)缓存系统固态盘的使用寿命,研究了系统的数据过滤技术。针对现有固态盘缓存系统数据过滤技术没有考虑数据页重用距离,导致重用距离大于固态盘缓存大小的数据页进入固态盘缓存,造成不必要的缓存页替换和固态盘擦写,从而降低缓存命中率和固态盘使用寿命的问题,提出了一种协同的数据过滤技术 UGD。该技术综合考虑了数据页访问频率和数据页重用距离,让访问次数少和重用距离长的数据页不进入固态盘缓存,从而增长了固态盘的使用寿命。为了评测 UGD 技术的效果,实现了一个固态盘缓存模拟器,并使用开源负载和实际应用负载进行了大量实验。实验数据表明,与现有过滤技术相比,UGD 技术能够使固态盘缓存系统的平均命中率提高 10%,使固态盘平均写入量降低 42.5%。

关键词 固态盘(SSD),固态盘缓存系统,固态盘寿命,数据过滤

0 引 言

固态盘(solid state disk, SSD)[1-3]已被广泛应用在存储系统中。相对于磁盘,固态盘在吞吐率、带宽、延迟、并发性、功耗、抗震性和噪音等方面具有明显优势。但是,固态盘也有其不如磁盘的方面,比如价格、寿命、容量等。在大数据时代,数据量巨大并且呈现爆炸式增长,使用固态盘存储所有数据价格非常昂贵。在所有数据中,大部分数据是冷数据,访问频率低和访问次数少;少部分数据是热数据,经常被访问。从存储系统性价比和存储设备利用率角度考虑,没有必要将所有数据存储在固态盘上。由于固态盘在价格、性能方面处于内存和磁盘之间,因此,固态盘常常作为内存和磁盘中间的一层存储设备,当做磁盘的缓存设备[4-6]使用。由于固态盘的闪存介质具有固有的擦后写、擦写粒度不同和擦写次数有限等特性,固态盘使用寿命由闪存介质数据写入量决定。为了提高固态盘使用寿命,已有的固态盘缓存系统通常采用数据过滤的方式,数据过滤可避免只访问一次的数据页进入固态盘缓存,从而增加固态盘使用寿命。然而,已有固态盘缓存系统数据过滤技术没有考虑数据页重用距离,导致重用距离大于固态盘缓存大小的数据页进入固态盘缓存,造成不必要的缓存页替换和固态盘擦写,降低缓存命中率和固态盘使用寿命。针对这种情况,本文提出了一种全新的数据过滤技术,称为统一的过滤数据缓存(unified ghost cache and data cache,UGD),UGD 技术综合考虑了数据页访问频率和数据页重用距离,能避免访问次数少和重用距离过长的数据页进入固态盘缓存,从而增加固态盘使用寿命。为了评测 UGD 效果,本研究用实现的固态盘缓存模拟

① 863 计划(2015AA0153032),国家自然科学基金(61502448)和中国科学院先导课题(XDA06010401)资助项目。

② 男,1988 年生,博士生;研究方向:计算机系统结构;联系人,E-mail: zhangzigang@ict.ac.cn

(收稿日期:2016-03-16)

器,并使用开源负载和实际应用负载进行大量实验。实验数据表明,与现有过滤技术相比,UGD 能够显著提高固态盘缓存系统平均命中率,而且能够大大降低固态盘平均写入量。

1 相关研究

固态盘缓存系统有两种工作模式,内存扩展模式和磁盘扩展模式。在内存扩展模式中,将固态盘缓存页管理与内存缓存页管理结合,与应用耦合度高、实现复杂,适用范围有限,目前主要应用在特定领域[7-11]。在磁盘扩展模式中,固态盘缓存管理模块相对独立,与应用耦合度低、管理灵活、简单易用,适用范围广,已被广泛应用在数据中心[12]、云计算中心[13-15]等。本文技术适用于固态盘缓存的两种工作模式。但是,由于内存扩展模式适用范围有限,本文将针对磁盘扩展模式固态盘缓存进行论述和评测。

固态盘使用寿命由固态盘介质数据写入量决定。当前,提高固态盘使用寿命的研究工作主要集中在减少数据写入量。减少闪存介质写入的研究工作主要分为两类,第一类是降低固态盘内部数据存储和维护开销,第二类是减少进入固态盘的数据量,即采用数据过滤技术从源头上减少写入固态盘的数据量。

降低固态盘数据存储开销,最直接的方式是压缩和去冗余。Makatos 等[16]将压缩技术引入缓存管理系统,通过将数据页内容进行压缩减少固态盘缓存空间占用,减少数据写入量的同时,增加固态盘缓存有效缓存空间。Li 等[17]在压缩技术的基础上,将去冗余技术引入固态盘缓存系统中,进一步减少数据写入量。由于在固态盘或者固态盘系统中资源有限,实现压缩和去冗余技术复杂度高等原因,当前技术没有被广泛采用。

降低固态盘内部维护开销即降低固态盘内部垃圾回收开销。不间断的垃圾回收用于保障固态盘内部有充足的空闲页用于新数据写入。在垃圾回收时,需要将牺牲擦除块中的有效页拷贝到其他擦除块中。数据拷贝过程既消耗固态盘内部带宽资源,又引入固态盘擦写。因此,降低垃圾回收数据页拷贝开销,是提升固态盘使用寿命的重要途径。Saxena 等[18]提出一种专门为缓存系统定制的固态盘设备(称为 SSC)以及通信源语,将部分缓存管理功能从主机端移动到设备端。SSC 将固态盘中垃圾回收和缓存页管理结合,当垃圾回收时将擦除块中的所有缓存页从固态盘缓存系统中删除,不再将有效数据页拷贝到其他擦除块。Yang 等[19]提出将缓存页管理和闪存页管理结合来减少垃圾回收开销。Oh 等[20]将固态盘空间按照读、写划分,并且根据负载特征动态地调整 over-provision[1] 空间大小,降低垃圾回收数据页拷贝开销。虽然上述方法能够降低垃圾回收开销,但是需要定制的固态盘设备支持。Kang 等[21]提出采用日志写方式,即使用 FIFO 替换策略管理固态盘缓存页,减少固态盘内部垃圾回收中数据页拷贝开销。由于 FIFO 替换策略过于简单,不能利用访问模式特性,如重用距离、访问频率等,导致命中率过低。Tang 等[21]将日志写模式与高级缓存替换策略结合,提出使用大粒度数据块(512MB 或者 1024MB)进行缓存写入和替换的方法,降低固态盘内部数据页拷贝开销。但是,这种方式需要将固态盘缓存页在内存中进行聚合,不适合具有写请求的负载,需要在可靠性和性能之间做权衡。

数据过滤技术决定数据页是否可以进入缓存,从源头上减少写入固态盘的数据量。Canim 等[8]将存储系统数据进行热度统计,然后人工指定将数据库系统中热度高的文件放置到固态盘缓存中。Liu 等[22]采用类似方法,将去冗余系统中访问热度高的数据放置在固态盘缓存中。Pritchett 等[23]将存储设备地址空间进行分段,并统计不同分段热度,然后周期性自动地将热度高的数据放置到固态盘缓存中。上述方法由于需要人工参与或者周期性替换,灵活性低,适合访问特征非常稳定的负载。Huang 等[24]提出一种动态的过滤技术 LARC,增加一个数据过滤窗口,并根据负载命中率动态调整过滤窗口大小,能够有效地避免只访问一次的磁盘页进入固态盘。因其简单、易用、高效、适用范围广等原因,目前 LARC 最常使用。Santana 等[25]根据负载特征动

态地开启或者关闭数据过滤技术来加快工作集变化剧烈的负载固态盘数据替换。

由于数据过滤技术不需要特殊硬件,并且简单、高效,已被广泛应用在固态盘缓存系统中。然而,当前的数据页过滤技术只考虑数据页访问次数,而没有考虑数据页重用距离,允许重用距离过长的数据页进入固态盘缓存,造成不必要的数据页替换,不仅降低固态盘缓存命中率,并且引入不必要固态盘写入,降低固态盘使用寿命。本文提出了一种全新的数据过滤技术,该技术综合考虑了数据页访问频率和数据页重用距离,可避免访问频率低或者重用距离长的数据页进入固态盘,避免不必要的数据页替换和写入,提高缓存命中率和固态盘使用寿命。

2 固态盘缓存系统

通常,固态盘缓存系统中将固态盘作为磁盘扩展进行管理,在系统中的位置如图 1 所示。由于固态盘缓存系统对上层提供与磁盘相同接口,不需要上层应用做任何改变,并且上层应用也不需要感知底层是磁盘还是固态盘缓存系统。

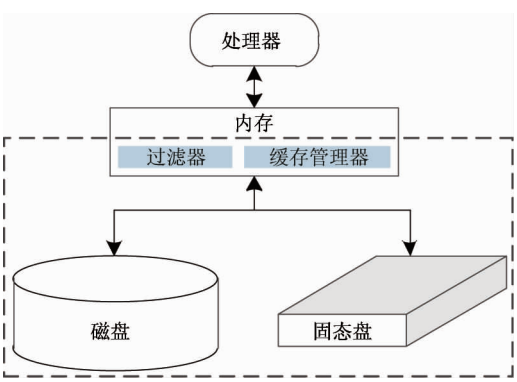


图 1 固态盘缓存系统示意图

固态盘缓存系统通常由磁盘、固态盘、缓存管理器和过滤器组成。缓存管理器控制固态盘空间分配,维护着哪些磁盘页在固态盘中、磁盘页与固态盘页之间的映射关系、磁盘页访问次数/重用距离等信息,并负责数据页替换。过滤器维护着未进入固态盘的磁盘页重用距离。在过滤器中只记录磁盘页的元数据(磁盘页号)而不将其数据保存在固态盘中。由于磁盘页非常多,维护全部磁盘页的访问次数需要大量内存资源并且没有必要,因为只有近期访问频率高的磁盘页才会被放入固态盘中。因此,过滤器中只维护部分近期被访问过的磁盘页信息。通常,过滤器中只维护固态盘缓存大小的数据页信息。

缓存管理器和过滤器各维护一条最近最少使用(least recently used, LRU)链表和哈希表。LRU 链表用于维护磁盘页重用距离;哈希表便于快速查询,通过磁盘页号计算获得哈希表索引。当应用请求一个磁盘页 P_D 时,该请求被固态盘缓存管理系统截获并解析,操作流程示意图如图 2 所示。

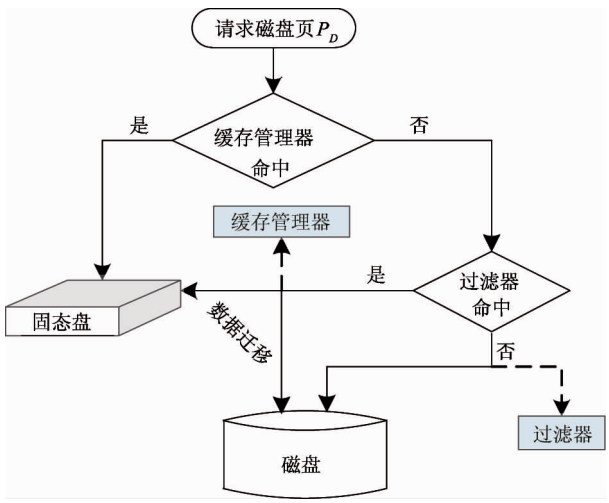


图 2 固态盘缓存系统操作流程示意图

固态盘缓存系统操作流程如下:首先,查询磁盘页 P_D 是否在缓存管理器命中。如果在缓存管理器中命中,则将 P_D 迁移到缓存管理器 LRU 链表最近最常使用(most recently used, MRU)端,获得 P_D 在固态盘中的固态盘页号 P_S,从固态盘 P_S 读取数据或者将数据写入固态盘 P_S。如果没有在缓存管理器命中,查询磁盘页 P_D 是否在过滤器命中。如果在过滤器命中,则将 P_D 从过滤器 LRU 链表和哈希表中删除。如果缓存管理器 LRU 链表已满,则淘汰缓存管理器 LRU 链表 LRU 端数据页 $P_{D'}$/ $P_{S'}$。如果固态盘 $P_{S'}$ 中数据页为脏,则将 $P_{S'}$ 数据写回磁盘 $P_{D'}$。否则,分配固态盘页 $P_{S'}$。将 $P_{D'}$ 从缓存管理器中删除,将磁盘页 P_D 从磁盘迁移到固态盘 $P_{S'}$ 中,将 P_D 添加到缓存管理器 LRU 链表 MRU 端,并且更新 P_D 与 $P_{S'}$ 的映射,从固态盘 $P_{S'}$ 读取数据或者将数据写入固态盘 $P_{S'}$ 中。如果没有在过滤器命中,则将磁

盘页 P_D 添加到过滤器 LRU 链表的 MRU 端,从磁盘 P_D 读取数据或者将数据写入磁盘 P_D。如果过滤器 LRU 链表已超大小阈值,则淘汰 LRU 链表 LRU 端 $P_{D'}$。算法 1 描述了固态盘缓存管理系统具体操作流程。

算法 1:固态盘缓存管理系统替换算法

初始化: 缓存管理器 LRU 链表 — LRU_M
缓存管理器哈希表 — H_M
过滤器 LRU 链表 — LRU_F
过滤器哈希表 — H_F

输入:磁盘页 P_D

替换算法:

```
If(P_D 在 H_M 中)
  将 P_D 迁移到 LRU_M MRU 端
  获得 P_D 对应的固态盘页号 P_S
  从固态盘 P_S 读取数据或者将数据写入固态盘 P_S
  If(写 P_D)
    将 P_S 标记为脏页
  EndIf
Else
  If(P_D 在 H_F 中)
    If(LRU_M 已满)
      将 LRU_M LRU 端 P_D'(P_S')淘汰
      If(P_S'为脏页)
        将 P_S'写回磁盘 P_D'
      EndIf
    Else
      分配固态盘页 P_S'
    EndIf
    将 P_D 从 LRU_F 和 H_F 删除
    If(读 P_D)
      读取磁盘 P_D
    EndIf
    将磁盘页 P_D 写入固态盘 P_S'
    将磁盘页 P_D(P_S')映射添加到 LRU_M MRU 端和 H_M
  Else
    If(LRU_F 已满)
      将 LRU_F LRU 端 P_D''淘汰
    EndIf
    将 P_D 添加到 LRU_F MRU 端和 H_F
    从磁盘读取 P_D 或者将 P_D 写入磁盘
  EndIf
EndIf
```

假设固态盘大小为 4,则缓存管理器和过滤器 LRU 链表长度限制为 4。按照算 1 执行缓存替换算法,得到固态盘缓存管理系统实例,如表 1 所示。

表 1 固态盘缓存管理算法示例

访问	缓存管理器 LRU 链表	过滤器 LRU 链表
3	{}	{3}
2	{}	{2→3}
7	{}	{7→2→3}
4	{}	{4→7→2→3}
2	{2}	{4→7→3}
6	{2}	{6→4→7→3}
3	{3→2}	{6→4→7}
5	{3→2}	{5→6→4→7}
7	{7→3→2}	{5→6→4}
9	{7→3→2}	{9→5→6→4}
4	{4→7→3→2}	{9→5→6}
8	{4→7→3→2}	{8→9→5→6}
6	{6→4→7→3}	{8→9→5}
0	{6→4→7→3}	{0→8→9→5}
5	{5→6→4→7}	{0→8→9}
3	{5→6→4→7}	{3→0→8→9}
9	{9→5→6→4}	{3→0→8}
2	{9→5→6→4}	{2→3→0→8}
8	{8→9→5→6}	{2→3→0}
4	{8→9→5→6}	{4→2→3→0}

从表中可以看出,磁盘页 3、7、4 被放入固态盘时的重用距离分别为 5、6、7,均超过固态盘大小,导致磁盘页 3、7、4 在固态盘命中之前被替换出固态盘的情况,造成没有必要的数据迁移和固态盘擦写,降低固态盘使用寿命。造成过滤器将重用距离超过固态盘大小的数据页放入固态盘的原因是过滤器只看到数据页在过滤器 LRU 链表中的重用距离,而没有考虑在固态盘中的数据页重用。

3 协同过滤固态盘缓存系统

本文提出了一种全新的固态盘缓存系统数据过滤技术,称为 UGD,其主要思想是综合考虑数据页访问频率和重用距离,避免访问次数少和重用距离长的数据页进入固态盘缓存,减少固态盘写入量,增

加固态盘使用寿命。

图3展示了传统固态盘缓存系统与协同过滤固态盘缓存系统的区别。与已有的固态盘缓存系统相比,协同数据过滤固态盘缓存系统具有以下不同:

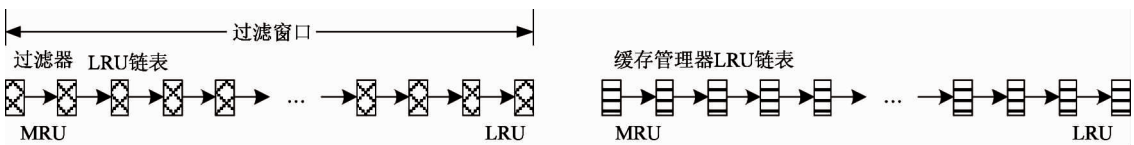
(1) 增加一个全局LRU链表,用于获得数据页的全局重用距离,而非过滤器LRU链表中的重用距离。因此,一个数据页同时处于两条LRU链表中:全局LRU链表、缓存管理器LRU链表或者过滤器LRU链表中。一个数据页不能同时处于缓存管理器LRU链表和过滤器LRU链表中。本文不需要获得确切的数据页重用距离,只要能够判断数据页重用距离是否在固态盘缓存准入窗口(下个条目详细解释)内即可。

(2) 维护一个固态盘缓存准入窗口和指示器,窗口大小为固态盘大小。只有在准入窗口内被再次命中的数据页才可能被从磁盘迁移到固态盘中。在已有的固态盘缓存系统中,过滤窗口内只有未进入固态盘的磁盘页;而在协同数据过滤的固态盘缓存系统中,准入窗口中包含最近被访问的所有数据页信息,既包含未进入固态盘的磁盘页,也包含已在固态盘中的数据页。指示器为准入窗口的边界数据页。

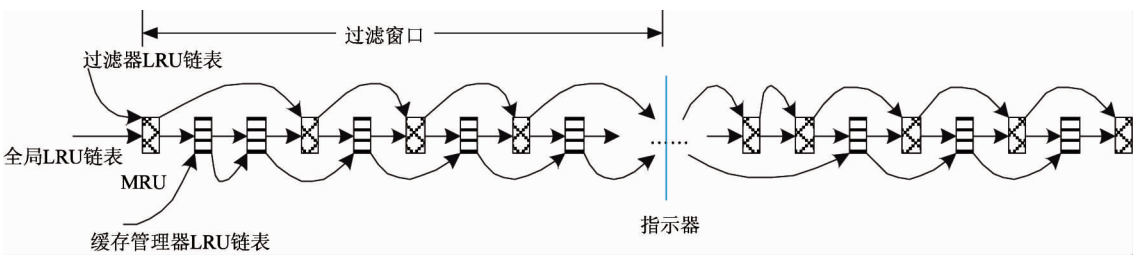
(3) 当在准入窗口中被访问数据页访问次数大于缓存管理器LRU链表LRU端的数据页访问次数时,磁盘页才被从磁盘迁移到固态盘中。

(4) 当进行缓存替换时,被替换的数据页从缓存管理器LRU链表迁移到过滤器链表,而在全局LRU链表中的位置不变。

(5) 维护一个全局序列号,顺序递增。缓存管理器和过滤器链表中的每个数据页都分配一个序列号。当数据页被访问时,全局序列号加1,并将全局序列号当前值赋值给数据页。使用序列号是为了判断数据页是否在准入窗口中。如果数据页的序列号大于指示器数据页序列号,则在准入窗口中;否则,不在准入窗口中。



(a) 已有固态盘缓存系统



(b) 协同过滤固态盘缓存系统

图3 已有固态盘缓存系统与协同过滤固态盘缓存系统对比

图4给出了协同过滤固态盘缓存系统替换算法的流程图。

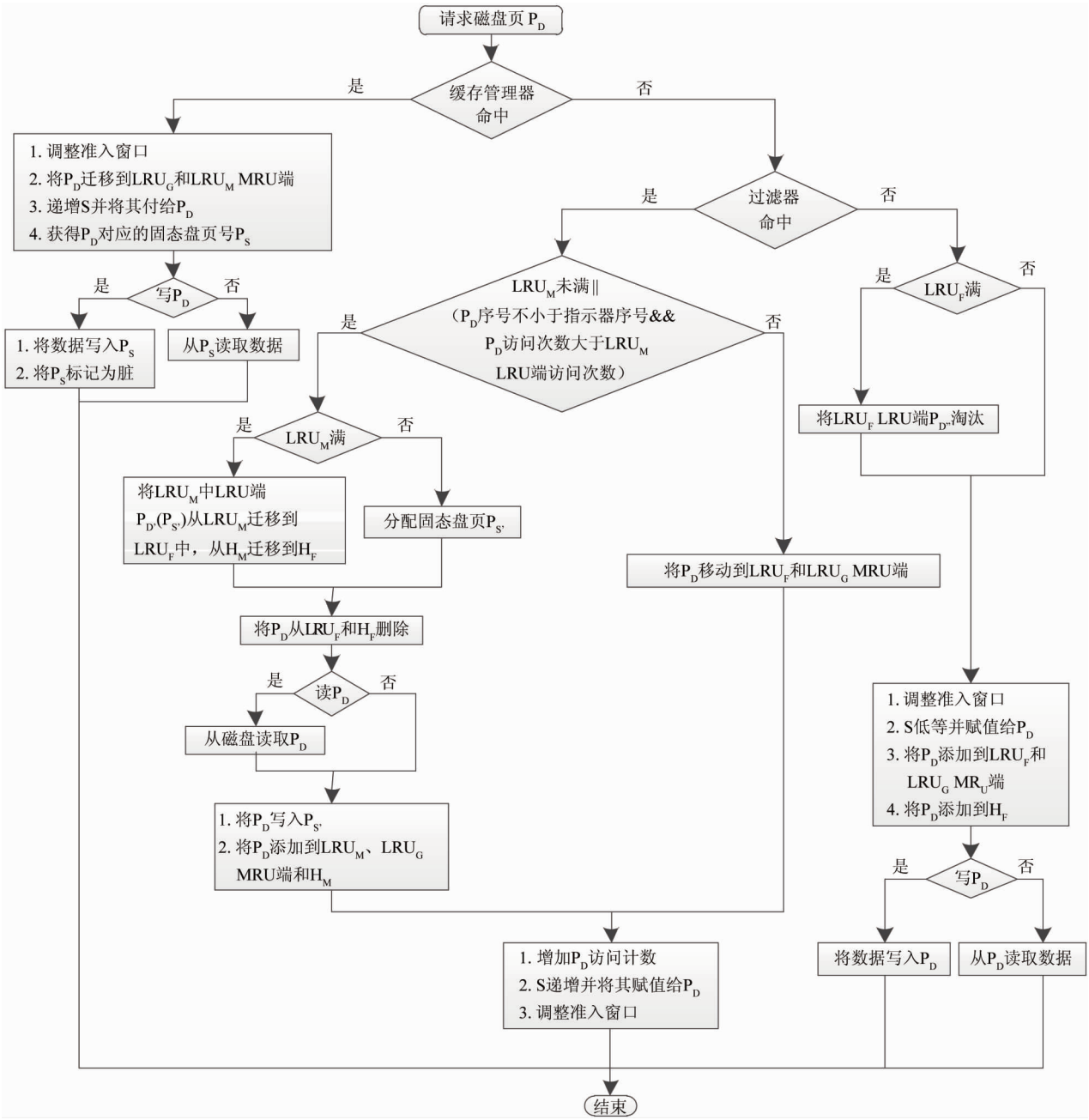


图 4　协同过滤固态盘缓存系统替换算法流程图

算法 2 描述了协同过滤固态盘缓存系统操作流动算法。

算法 2　协同过滤固态盘缓存系统替换算法

初始化： 全局 LRU 链表 — LRU_G
缓存管理器 LRU 链表 — LRU_M
缓存管理器哈希表 — H_M
过滤器 LRU 链表 — LRU_F
过滤器哈希表 — H_F
准入窗口边界指示器 — A
全局序列号 — *S*

输入： 磁盘页 P_D

替换算法：

1. If(P_D 在 H_M 中)
2. 　调整准入窗口
3. 　将 P_D 迁移到 LRU_G MRU 端
4. 　将 P_D 迁移到 LRU_M MRU 端
5. 　*S* 递增并将其赋值给 P_D
6. 　获 P_D 得对应的固态盘页号 P_S
7. 　从固态盘 P_S 读取数据或者将数据写入固态盘 P_S
8. 　If(写 P_D)

```
9.        将 P_S 标记为脏页
10.     EndIf
11.   Else
12.     If(P_D 在 H_F 中)
13.       If(LRU_M 未满 || (P_D 序号不小于指示器序号
          && P_D 访问次数大于 LRU_M LRU 端访问次数))
14.         If(LRU_M 已满)
15.           将 LRU_M 中 LRU 端 P_D'(P_S')从 LRU_M 迁
              移到 LRU_F 中,从 H_M 迁移到 H_F
16.           If(P_S'为脏页)
17.             将 P_S'写回磁盘 P_D'
18.           EndIf
19.         Else
20.           分配固态盘页 P_S'
21.         EndIf
22.         将 P_D 从 LRU_F 和 H_F 删除
23.         If(读 P_D)
24.           读取磁盘 P_D
25.         EndIf
26.         将磁盘页 P_D 写入固态盘 P_S'
27.         将磁盘页 P_D 添加到 LRU_M MRU 端和 H_M
28.         将磁盘页 P_D 添加到 LRU_G MRU 端
29.       Else
30.         将 P_D 移动到 LRU_F 和 LRU_G MRU 端
31.       EndIf
32.       增加 P_D 访问计数
33.       S 递增并将其赋值给 P_D
34.       调整准入窗口
35.     Else
36.       If(LRU_F 已满)
37.         将 LRU_F LRU 端 P_D''淘汰
38.       EndIf
39.       调整准入窗口
40.       S 递增并赋值给 P_D
41.       将 P_D 添加到 LRU_F 和 LRU_G MRU 端
42.       将 P_D 添加到 H_F
43.       从磁盘读取 P_D 或者将 P_D 写入磁盘
44.     EndIf
45.   EndIf
```

准入窗口调整算法:

输入:插入数据页或者命中数据页序列号 — S_Q

准入窗口边界指示器 — A

调整算法:

```
1.  If(S_Q 大于 S_A)
2.    指示器向 LRU_G MRU 端移动一次
3.  Else
4.    指示器向 LRU_G LRU 端移动一次
5.  EndIf
```

将表 1 中的访问实例按照算法 2 替换算法执行获得结果如表 2 所示。在执行过程中,指示器按照算法向前或者向后滑动,保持准入窗口大小不变。

表 2　协同过滤固态盘缓存替换算法示例

访问	全局 LRU 链表 — 准入窗口指示器	
	缓存管理器 LRU 链表	过滤器 LRU 链表
3	{3} — 3	
	{ }	{3}
2	{2→3} — 3	
	{ }	{2→3}
7	{7→2→3} — 3	
	{ }	{7→2→3}
4	{4→7→2→3} — 3	
	{ }	{4→7→2→3}
2	{2→4→7→3} — 3	
	{2}	{4→7→3}
6	{6→2→4→7→3} — 7	
	{2}	{6→4→7→3}
3	{3→6→2→4→7} — 4	
	{2}	{3→6→4→7}
5	{5→3→6→2→4} — 2	
	{2}	{5→3→6→4}
7	{7→5→3→6→2} — 6	
	{2}	{7→5→3→6}
9	{9→7→5→3→2} — 3	
	{2}	{9→7→5→3}
4	{4→9→7→5→2} — 5	
	{2}	{4→9→7→5}
8	{8→4→9→7→2} — 7	
	{2}	{8→4→9→7}
6	{6→8→4→9→2} — 9	
	{2}	{6→8→4→9}
0	{0→6→8→4→2} — 4	
	{2}	{0→6→8→4}
5	{5→0→6→8→2} — 8	
	{2}	{5→0→6→8}
3	{3→5→0→6→2} — 6	
	{2}	{3→5→0→6}
9	{9→3→5→0→2} — 0	
	{2}	{9→3→5→0}
2	{2→9→3→5→0} — 5	
	{2}	{9→3→5→0}
8	{8→2→9→3→5} — 3	
	{2}	{8→9→3→5}
4	{4→8→2→9→3} — 9	
	{2}	{4→8→9→3}

从执行结果来看,协同过滤技术将固态盘写入量从 8 减少为 1,效果明显。

由于在 UGD 中增加全局 LRU 链表和序列号,使得每个缓存条目占用的内存资源增加:两个指针和一个序列号。假设指针和序列号长度都为 8 字节,则每个缓存条目增加 24 字节。缓存页大小为 4kB,则增加的内存资源额外开销为,低于 0.6%。因此,本文可以认为 UGD 增加新的数据结构引入的额外内存开销很低。

UGD 是固态盘缓存过滤关键技术,应用在本地存储系统中,如使用固态盘缓存的文件系统等。分布式文件系统是由一系列运行本地文件系统的节点构成,本地节点文件系统性能的提升能够增加分布式文件系统性能。因此 UGD 也适用于分布式文件系统。

4 实验与评价

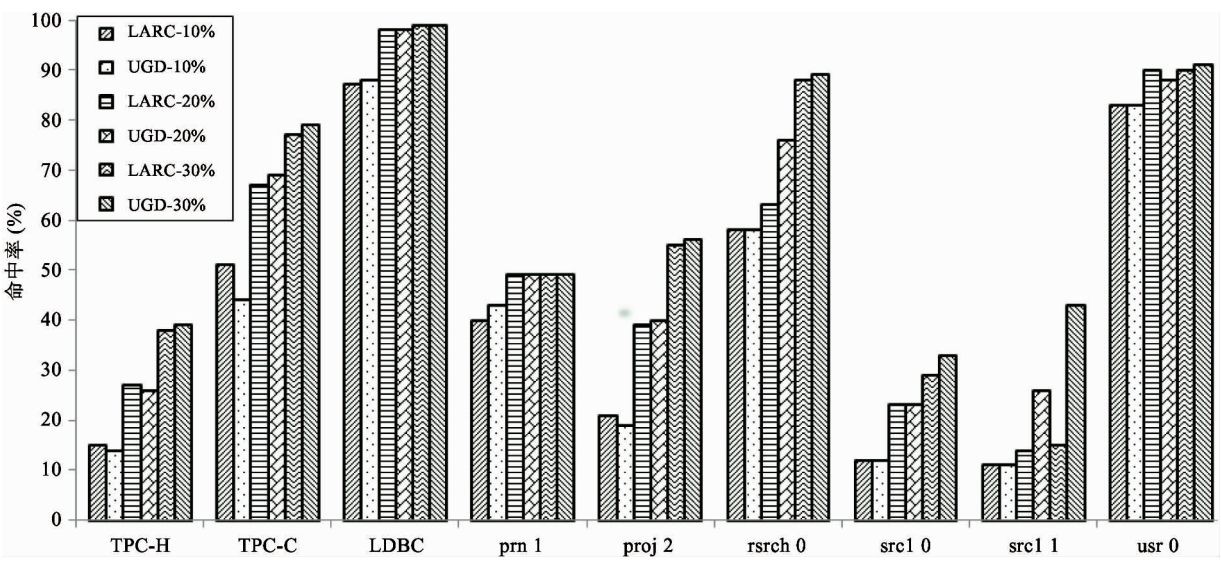
为了评测协同过滤固态盘缓存系统在数据过滤方面的效果,本文实现一个固态盘缓存系统模拟器。此外,本文还实现了当前应用最广、最高效的固态盘缓存系统过滤技术,LARC[24],用于性能对比。

本文使用负载回访的方法进行评测。负载包括来自 SNIA MSR Cambridge[26] 负载库中的部分负载,包括 prn _ 1、proj _ 2、rsrch _ 0、src1 _ 0、src1 _ 1 和 usr _ 0,和虚拟机实际应用运行负载,包括 OLAP、OLTP、社交网络。

虚拟机负载运行服务器包含两个 12 核 E5645 2.4GHz 处理器、96GB 内存、三块 1TB 7200RPM 希捷硬盘(一块用于运行操作系统,一块用于存储虚拟机镜像,一块用于存储 I/O 轨迹)。本文使用 KVM 作为虚拟机监管程序。虚拟机配置为 4 个虚拟处理器、2GB 内存、100GB 虚拟存储设备。在虚拟机中运行上述负载,在主机端使用 blktrace 工具抓取 I/O 轨迹。OLAP 和 OLTP 负载底层使用 MySQL 作为存储引擎,上层使用 HammerDB[27] 产生负载请求。OLAP 负载比例因子为 10,OLTP 使用 128 个仓库。社交网络负载使用 Virtuoso[28] 作为存储引擎,使用 LDBC[29] 产生用户请求负载,负载比例因子为 10。

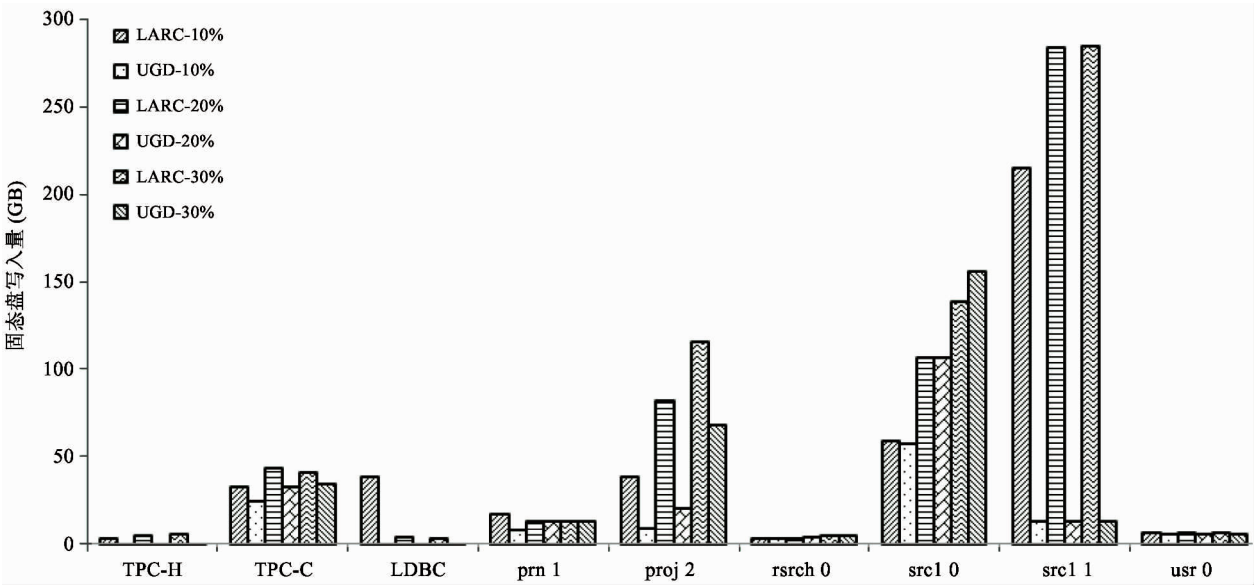
在测试时,本文分别将固态盘大小设置为负载工作集大小的 10%、20%、30% 进行测试。测试时,将负载前半部分 I/O 轨迹用于系统预热阶段,负载后半部分 I/O 轨迹用于测试阶段。实验数据来自测试阶段。

图 5 展示了 UGD 和 LARC 性能对比图。从测试数据来看,相对于 LARC,UGD 命中率平均提升 10%,最大提升 186%;固态盘写入量平均降低42.5%,最大降低 99%。



(a) 命中率

图 5 协同过滤固态盘缓存系统性能评测结果



(b) 固态盘数据写入量

图 5　协同过滤固态盘缓存系统性能评测结果

UGD 比 LARC 性能提升的主要原因是避免访问次数少并且重用距离长的数据页进入固态盘缓存,减少没有必要的缓存写入和替换,提升命中率,减少固态盘写入量。以 src1 _1 为例,在 LARC 中让 273GB 数据进入固态盘缓存,而 UGD 中只让不到 1GB 数据进入固态盘缓存,大大降低进入固态盘的数据量。对于 src1 _0,在缓存空间为工作集的 30% 时,由于 UGD 的命中率增加,写命中率增加,与 LARC 相比 UGD 增加了数据量。

由于 UGD 的过滤机制比 LARC 更加严格,当数据页访问次数相对较少时,UGD 不利于数据页替换,导致命中率比 LARC 要低,如 TPC-H 在缓存空间为工作集的 10% 和 20% 时。此外,相比于 LARC,UGD 算法稍显复杂。但是,测试结果表明,UGD 在保持命中率的情况下,能够有效地避免访问距离长的数据页进入固态盘,提升了固态盘的使用寿命。

5　结　论

本文通过分析现有固态盘缓存系统过滤技术,得出已有过滤技术只避免访问一次的数据进入固态盘,而访问次数少和重用距离长的数据页仍进入固态盘,然而这些数据页不能够增加命中率和固态盘性能,反而因为频率的数据页替换,降低了命中率和固态盘使用寿命。本文提出了一种协同的数据过滤技术 UGD,该技术综合考虑了数据页访问频率和重用距离,将缓存管理器和过滤器协同设计,让访问次数少和重用距离长的数据页不进入固态盘缓存。为了评测 UGD 优化效果,本文实现了一个固态盘缓存系统模拟器和已有的固态盘缓存数据过滤技术。为了测试 UGD 性能,使用公开的常用负载 I/O 轨迹和实际应用 I/O 轨迹进行了大量实验。实验结果表明,与 LARC 相比,UGD 将命中率平均提高了 10%,将固态盘平均数据写入量降低了 42.5%。

本文后续计划将协同过滤技术与其它相关技术(如压缩、去冗余、动态关闭/开启数据过滤等)进行结合,并将其应用到实际系统中,通过实际应用进行评测来进一步完善固态盘缓存系统。

参考文献

[1] Agrawal N, Prabhakaran V, Wobber T, et al. Design tradeoffs for SSD performance. In: Proceedings of ATC USENIX Annual Technical Conference, Boston, USA, 2008. 57-70

[2] Intel Solid-State Drives. http://www.intel.com/content/www/us/en/solid-state-drives/solid-state-drives-ssd.html:Intel, 2016

[3] Samsung Solid-State Drives. http://www.samsung.com/us/computer/solid-state-drives:Samsung, 2016

[4] XtremCache. http://www.emc.com/storage/xtrem/xtremcache.html:EMC, 2016

[5] FlashCache. http://www.netapp.com/us/products/storage-systems/flash-cache/flash-cache-tech-specs.aspx: NetApp,2016

[6] FlashSoft. https://www.sandisk.com/business/datacenter/products/flash-software/flashsoft:SanDisk, 2016

[7] Kgil T, Mudge T. FlashCache: A NAD flash memory file cache for low power web servers. In: Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embeded Systems, Seoul, Korea, 2006. 103-112

[8] Canim M, Mihaila G, Bhattacharjee B, et al. SSD bufferpool extensions for database systems. *Proceedings of the VLDB Endowment*, 2010, 3: 1435-1446

[9] Do J, Zhang D, Patel J, et al. Turbo charging DBMS buffer pool using SSDs. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, Athens, Greece, 2011. 1113-1124

[10] Kang W, Lee S, Moon B. Flash-based extended cache for higher throughput and faster recovery. *Proceedings of the VLDB Endowment*, 2012, 5: 1615-1626

[11] Liu X, Salem K. Hybrid storage management for database systems. *Proceedings of the VLDB Endowment*, 2013, 6: 541-552

[12] Albrecht C, Merchant A, Stokely M, et al. Janus: Optimal flash provisioning for cloud storage workloads. In: Proceedings of the USENIX Conference on Annual Technical Conference, San Jose, USA, 2013. 91-102

[13] Byan S, Lentini J, Madan A, et al. Mercury: Host-side flash caching for the data center. In: Proceedings of the IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST), Pacific Grove, USA, 2012. 1-12

[14] Luo T, Ma S, Lee R, et al. S-CAVE: Effective SSD caching to improve virtual machine storage performance. In: Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques, Edinburgh, Scotland, 2013. 103-112

[15] Meng F, Zhou L, Ma X, et al. vCacheShare: Automated server flash cache space management in a virtualization environment. In: Proceedings of the USENIX Conference on Annual Technical Conference, Philadelphia, USA, 2014. 133-144

[16] Makatos T, Klonatos Y, Marazakis M, et al. Using transparent compression to improve SSD-based I/O caches. In: Proceedings of the 5th European Conference on Computer Systems, New York, USA, 2010. 1-14

[17] Li C, Shilane P, Douglis F, et al. Nitro: A capacity-optimized SSD cache for primary storage. In: Proceedings of the USENIX Conference on Annual Technical Conference, Philadelphia, USA, 2014. 501-512

[18] Saxena M, Swift M, Zhang Y. FlashTier: A lightweight, consistent and durable storage cache. In: Proceedings of the 7th ACM European Conference on Computer Systems, Bern, Switzerland, 2012. 267-280

[19] Yang J, Plasson N, Gillis G, et al. HEC: Improving endurance of high performance flash-based cache devices. In: Proceedings of the 6th International Systems and Storage Conference, Haifa, Israel, 2013. Article No. 10

[20] Oh Y, Choi J, Lee D, et al. Caching lesss for better performance: Balancing cache size and update cost of flash memory cache in hybrid storage systems. In: Proceedings of the 10th USENIX Conference on File and Storage Technologies, Berkeley, USA, 2012. 25-25

[21] Tang L, Huang Q, Lloyd W, et al. RIPQ: Advanced photo caching on flash for Facebook. In: Proceedings of the 13th USENIX Conference on File and Storage Technologies, Santa Clara, USA, 2015. 373-386

[22] Liu J, Chai Y, Qin X, et al. PLC-cache: Endurable SSD cache for deduplication-based primary storage. In: IEEE 30th Symposium on Mass Storage Systems and Technologies (MSST), Santa Clara, USA, 2014. 1-12

[23] Pritchett T, Thottethodi M. SieveStore: A highly-selective, ensemble-level disk cache for cost performance. In: Proceedings of the 37th Annual International Symposium on Computer Architecture, Saint-Malo, France, 2010. 163-174

[24] Huang S, Wei Q, Chen J, et al. Improving flash-based disk cache with lazy adaptive replacement. In: Proceedings of the IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST), Lake Arrowhead, USA, 2013. 1-10

[25] Santana R, Lyons S, Koller R, et al. To ARC or not to ARC. In: Proceedings of the 7th USENIX Conference on

Hot Topics in Storage and File Systems, Santa Clara, USA, 2015. 14-14

[26] Narayanan D, Donnelly A, Rowstron A. Write off-loading: Practical power management for enterprise storage. *ACM Transactions on Storage*, 2008, 4: 10:1-10:23

[27] HammerDB. http://www.hammerdb.com/: Hammerdb, 2016

[28] LDBC Social Network Benchmark. http://ldbcouncil.org/developer/snb, 2016

[29] Virtuoso. https://github.com/openlink/virtuoso-open-source: Github, 2016

UGD: A novel filtering method for SSD-based cache system

Zhang Zigang[*,**,***], Jiang Dejun[*,**], Sun Ninghui[*,**]

([*]State Key Laboratory of Computer Architecture, Chinese Academy of Sciences, Beijing 100190)

([**]Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

([***]University of Chinese Academy of Sciences, Beijing 100049)

Abstract

To increas the lifetime of the solid state disk (SSD) in a SSD-based cache system, the data filtering for the system was studied, and the study revealed that existing filtering techniques do not take the data re-use distance into account, which makes the data pages with the large re-use distance enter into the SSD cache, causing the non-necessary page replacement and SSD writing, then the decline in the cache hit rate and the SSD lifetime. In view of this problem, a novel data filtering method, called unified ghost cache and data cache (UGD), was proposed. The UGD technique concurrently considers the page access count and the re-use distance to filter out the pages of large re-use distance and small access count to increase the SSD lifetime. A cache system simulator was implemented and number of extensive experiments were conducted by using public traces and traces of real-world workloads. The experimental results show that in comparison with LARC, the UGD improved the cache hit rate by 10% and reduced write amount by 42.5% on average.

Key words: solid state disk (SSD), SSD-based cache system, SSD lifetime, page filtering