高技术通讯 2013 年 第23卷 第1期:8~15

doi:10.3772/j.issn.1002-0470.2013.01.002

基于稳定集模型的大容量缓存管理①

郭明阳②*** 张永乐*** 刘振晗* 刘振军* 许 鲁*

(*中国科学院计算技术研究所 北京 100190)

(**中国科学院研究生院 北京 100039)

摘 要 指出了阶段间数据换入换出效率是大容量缓存管理最重要的问题,并对这一问题进行了研究。定义了一种用于预测新阶段的数据访问的新的数据访问宏观模型——稳定集模型(SSM),并基于该模型设计了一套缓存管理方法,包括缓存容量配置、缓存粒度选择、数据预取和缓存替换算法。该套算法能提高缓存在阶段间的数据换入换出效率,更有效地优化集中存储负载,并提高应用访问的性能。实验表明,基于SSM的缓存管理方法能够将集中存储负载降低到传统缓存管理方法的2.0%~15.8%,平均响应时间降低到0.8%~15.2%。

关键词 网络存储,大容量缓存,稳定集模型(SSM),阶段-转换行为,缓存需求估计,缓存粒度选择,数据预取,缓存替换

0 引 言

随着科技、社会的发展,需要保存和处理的数据飞速增长。传统的数据处理集群共享存储数据的主要方式是网络集中存储,它存储密度高,存储容量大,但性能和可扩展性逐渐不能满足越来越大的数据处理集群的需求。大容量客户端缓存可以降低集中存储负载,同时能提高应用I/O访问性能,从而提高了存储系统的性能和可扩展性。Zazen系统[1]和panache系统[2]的出现表明这一方案重新受到学术界关注,而固态硬盘(solid state disk, SSD)技术的逐渐成熟为这一方案的广泛应用提供了有利条件。来自Fusion IO、OCZ、Intel、LSI等公司的PCI-E接口SSD卡目前可以达到数TB的容量、数GB/s的带宽。EMC提出了"闪电计划",研究使用客户端的SSD卡提高盘阵上的访问性能,而Intel则致力于将SSD集成到主板上,来加速所有服务器的I/O访问性能。然而,缓存容量增大到一定程度后,阶段内缺失[3]逐渐消失,如何利用大容量缓存,最小化阶段间缺失引起的负载问题和性能问题,成为缓存管理最重要的问题。现有的替换算法、预取算法的研究[4-6],多是基于传统容量缓存,主要优化阶段内缺失的问题,缺乏对阶段间数据换入换出效率的考虑。为了解决该问题,本文定义了稳定集模型(stable set model, SSM),一种新的数据访问宏观模型。相比于阶段-转换模型,该模型在空间维度对数据访问进行了划分,把数据访问划分为多个稳定集上的访问,而阶段数据是由稳定集组成。这样新阶段的数据访问就可以通过探测稳定集状态来预测。本文基于SSM设计了一套大容量缓存的管理方法,包括缓存的容量配置、粒度确定、数据预取和替换,从而高效地利用大容量缓存优化阶段间数据换入换出效率,最终大大提高网络集中存储系统的性能和可扩展性。

1 稳定集模型(SSM)

1.1 模型定义

SSM像LRU堆栈模型、LFU概率模型一样,是对数据访问规律的一种假设。本文通过观察多种trace的访问图示,总结出两条可能的数据访问宏观规律:(1)与程序访问的阶段-转换模型[3]类似,存储系统上数据访问也呈现阶段-转换的特性。(2)数据访问除了在时间维度上可划分为一个一个阶段

① 973计划(2011CB302304),863计划(2011AA01A102),国家自然科学基金(61100012)和中国科学院战略性先导科技专项(XDA06010401)资助项目。

② 男,1983年生,博士;研究方向:文件系统,缓存系统;联系人,E-mail: guomingyang@nrchpc.ac.cn

(收稿日期:2012-01-09)

外,在空间维度上也是可划分的。数据可以分成多个组,每组数据总是同时被访问并且同时不被访问。基于这两条规律,本文给出 SSM 的以下定义:

定义 1　稳定集模型(SSM) SSM 认为,任何数据访问流可以分解为有限个稳定集上阶段的同时访问流的叠加,即 $R = \sum_{i=1}^{n}(S_i, \mathbf{T}_i)$,其中 $(S_i, \mathbf{T}_i)$ 表示稳定集 S_i 上的访问流。访问流 $(S_i, \mathbf{T}_i)$ 阶段性地同时访问 S_i 上的所有元素,并且与其他稳定集上的访问流相互独立。S_i 是这个访问流访问的数据集,并且与其他稳定集是不相交的。$\mathbf{T}_i$ 表示稳定集访问流 $(S_i, \mathbf{T}_i)$ 发生的时间段序列 $\mathbf{T}_i = (t_1^i, t_2^i)(t_3^i, t_4^i)(t_5^i, t_6^i)\cdots$

对比于阶段-转换模型,SSM 最大的不同是:除了在时间维度对数据访问序列做了划分以外,还在空间维度对访问涉及的数据做了划分。由于数据在存储和文件系统上的地址有更强的固定性,SSM 认为局部集里面的部分数据之间有着更持久的、可以跨越阶段间转换的同时访问关系。这种关系是 SSM 与阶段-转换模型的根本区别,也是 SSM 能够不通过程序行为分析[7]就能预测转换后数据访问行为的关键原因。

1.2　模型刻划定义

为了使用 SSM 指导缓存管理,必须首先对数据访问进行 SSM 刻划。这正如使用 LRU 堆栈模型管理缓存时,必须首先根据数据访问生成当前的 LRU 堆栈一样。SSM 需要的刻划结果分为 3 组:(1) 稳定集粒度(SSG);(2) 所有的稳定集 $\{S_1, S_2, S_3, \cdots, S_n\}$;(3) 各稳定集的最大活跃间距 $\{I_1, I_2, I_3, \cdots, I_n\}$。由上节可知,稳定集指的是总是被同时访问并且同时不访问的数据集。为了从已有数据访问中提取出稳定集,本文从 BLI[8] 和 Major phase[8] 中提取了同时访问的概念,并基于工作集重新定义了提取同时访问集(局部集)的方式——稳定访问集,以降低提取的时空开销,最后将所有的稳定访问集相交,以获得总是被同时访问并且同时不访问的数据集合——稳定集。

定义 2　同时访问(simultaneous access) 如果在最近的一段时间内,一个集合的所有元素都被反复访问过,并且该集合中任取两个元素 A 和 B,其访问都被充分交织在一起,即有 A 先于 B 的访问,又有 B 先于 A 的访问,则称该集合的元素在这一段时间里被应用同时访问。

同时访问的定义体现了数据访问对缓存容量的本质需求。这是因为只有反复被访问的数据才需要缓存,而只有访问被充分交织在一起,缓存才不能采用时分复用的方式满足应用访问的需求。

有了同时访问的定义,我们基于工作集模型给出了"稳定访问集"的定义,用于提取同时访问的数据集合(也即局部集)。

定义 3　稳定访问集(stable access set) 如果一个数据访问流中存在两个连续的 N 时长最小稳定工作集 W_i 和 W_{i+1},而集合 $A = W_i \cap W_{i+1}$,则称集合 A 为该数据访问流的 N 时长稳定访问集。

该定义有几个概念需要解释:N 时长稳定工作集是一种特殊的工作集,它对当前访问的数据涵盖得非常全面,以至于连续 N 个访问都没有增加这个工作集涵盖的内容,N 也被称为稳定度。而 N 时长最小稳定工作集是指第一次达到上述标准的稳定工作集。为了简单起见,下文有时会省略定语"N 时长"。连续的最小稳定工作集是指这两个最小稳定工作集所跨越的时间段是首位连接在一起的。

稳定访问集内的数据是同时被访问的,这是因为采用两个最小稳定工作集相交的方式,保证了稳定访问集里面的所有数据都是反复被访问的。并且,任取两个元素 1 和 2,由于它们既属于 W_i 又属于 W_{i+1},这就保证了它们的访问是充分交织在一起的。

有了稳定访问集之后,我们把所有的稳定访问集相交,就得到了总是同时访问的集合——稳定集。这样得到的稳定集,能够保证任取稳定访问集,或者包括稳定集里面的所有元素,或者不包括稳定集的任何元素。

定义 4　稳定集(stable set) 将一个数据访问流中所有的稳定访问集相交,得到的结果即为该数据访问流的所有稳定集 $\{S_1, S_2, S_3, \cdots, S_n\}$。

由于数据被划分为不同的稳定集,这些稳定集上的数据访问也就不再共用一个阶段划分。每一个稳定集都会有自己的活跃时间。如果一个稳定访问集 A 包含某一个稳定集 S,则认为 S 在 A 所跨越的时间段内是活跃的。但这种方法只能后验地判断稳定集的状态,不适合预测将来的状态。由此本文给出了最大活跃间距的定义,用于预测稳定集的状态及状态转换。

定义 5　最大活跃间距(maximum active interval, MAI) 稳定集 S_i 在活跃状态下的最大访问间距 I_i 称为该稳定集的最大活跃间距。

最大活跃间距利用了稳定集在活跃和不活跃状态下的访问间隔的巨大差异来预测稳定集的状态和

状态变化。如果一个稳定集上连续两次访问的距离小于已有的最大活跃间距,则该稳定集可被认定为活跃;如果一个稳定集超过最大活跃间距不被访问,则可以预测该稳定集转入了不活跃态。

稳定集访问是具有同时访问特性的访问流,而真实的数据访问中,必然会有一些访问不能被归到同时访问关系里去,这部分请求所占的比重就是 SSM 的漏报率,它的大小决定了真实数据访问与刻划结果的差异性。影响这个差异大小的主要有两个因素:稳定度和数据块大小。

稳定度 N 决定了最小稳定工作集所涵盖的时间范围。由于稳定集刻划中不区分一个最小稳定工作集内各个访问的时间差异,因此稳定度也决定了 SSM 这一宏观模型认知一个真实数据访问时,在时间维度上的概括程度。在本文中,我们通过漏报率的指导,自学习了一个最佳稳定度,使得真实访问和刻划结果的差异最小。

数据块大小决定了刻划过程中认知数据的最小粒度。由于稳定集刻划中不区分一个数据块内部的访问差异,只要一个访问落在某个数据块内,就认为这个数据块整体被访问,因此数据块大小也决定了 SSM 认知数据访问时在空间维度上的概括程度。本文通过缓存容量和后端负载两方面的限制,自学习了一个适宜的块大小,也就是稳定集粒度。

定义 6 稳定集粒度(stable set granularity, SSG) 一个数据访问流在采用 B 作为数据块大小时,缓存不会发生阶段内缺失,且其 footprint 所引起的后端负载最小,B 就是该数据访问流在此时的缓存容量和后端存储情况下的稳定集粒度。

2 基于 SSM 的缓存管理方法

为了利用大容量客户端缓存,最大限度地降低集中存储上的负载,同时降低应用 I/O 访问的时延,从而提高网络集中存储系统的性能和可扩展性,我们首先需要估计客户端所需要的缓存容量,以便配备合适的 SSD 卡。然后需要适宜的缓存管理粒度、高效的缓存替换算法和预取算法,从而最大限度地发挥客户端缓存的作用。接下来我们将分别论述 SSM 对缓存需求估计、缓存管理粒度、数据预取、缓存替换这 4 部分的作用。

2.1 缓存需求估计

为了最小化应用引起的集中存储负载,我们需要提供足够的缓存来消除所有的阶段内缺失。因此客户端缓存必须满足在使用最小粒度(本研究环境中是 4KB)访问缓存的情况下,缓存容量大于最大的阶段的数据量,这样才有可能做到消除所有阶段内缺失。由此我们给出了大容量缓存的定义。

定义 7 大容量缓存(large-capacity cache) 应用使用最小粒度访问该缓存系统时,如果它的容量足够容纳任何一个阶段的全部访问数据,则该缓存被称为大容量缓存。而刚好达到这一要求的缓存容量,被称为大容量点(large capacity point)。

但在实际应用中,除了容纳同时访问数据,缓存还需要容量来过滤一遍访问和偶发访问,容纳预取数据等。这部分容量的需求类似 LIRS[5] 的 hir 栈需求,因此我们借鉴了它的数据,预留 30% 的缓存容量。在后面的测试我们将看到,由于 SSD 的大容量性质,这部分的容量需求是易于满足的。因此本文基于 SSD 可以提供足够的缓存这一需求展开下一步的工作。

2.2 缓存粒度选择

使用较大的缓存粒度有三个优点:(1)减少后端负载;(2)降低 I/O 延迟;(3)减少缓存管理开销。但是较大的缓存粒度也可能带来无效 I/O 和缓存污染。为了保证正面效果大于负面效果,需要仔细选择缓存粒度。

本文使用稳定集粒度作为缓存管理基本粒度,这是因为:首先,我们对应用 trace 进行分析并选择引起后端负载最小的块大小作为稳定集粒度的最大值,这就保证了使用稳定集粒度为缓存带来的连续性的好处大于无效 I/O 带来的负面影响;其次,我们限制稳定集粒度不能引起阶段内缺失,这就保证了使用较大缓存粒度带来的缓存污染不会导致缓存缺失的急剧上升。

此外,由于 2.1 节提到的问题,我们在计算稳定集粒度时,要首先预留 30% 的缓存容量,用于处理一遍访问数据和预取数据。

2.3 基于稳定集的数据预取方法

基于稳定集的数据预取策略的核心是:预测稳定集即将到来的状态转换,在它将要活跃时推送整个稳定集的数据。采用的预测方法是监测不活跃稳定集上的访问间距,并与它的最大活跃间距相比。如果一个不活跃的稳定集上的访问间距小于它的最大活跃间距,我们将预测该稳定集即将转为活跃态,并推送该稳定集的所有数据。下面我们给出稳定集空间局部性的定义。

定义 8 稳定集空间局部性(stable set spatial

locality) 如果一个不活跃稳定集上的访问间距小于最大活跃间距,则该稳定集内的所有数据都有在最近被访问的可能,称为稳定集空间局部性。

稳定集空间局部性能够在上一步最小化后端负载的基础上,进一步降低前端应用的不命中率,降低后端 I/O 负载。已有的工作[9]已经证实了这一结论。与上述工作不同的是,由于采用稳定集粒度适应缓存容量变化,保证了阶段内缺失的不再出现。因此,本文不需要在 SSM 内嵌套一个微模型,用于指导稳定集内部数据管理。

2.4 基于稳定集的缓存替换算法

基于稳定集的缓存替换算法的核心是:预测稳定集即将到来的状态转换,在一个稳定集将要转换为不活跃态时将整个稳定集的数据降级。采用的预测方法是监测活跃稳定集上的访问间距,并与它的最大活跃间距相比。如果一个活跃稳定集上的访问间距大于它的最大活跃间距,我们将预测该稳定集即将转为不活跃态,并将该稳定集的所有数据降级到低级别的状态。下面给出稳定集时间局部性的定义。

定义 9 稳定集时间局部性(**stable set temporal locality**) 如果一个活跃稳定集上的访问间距大于最大活跃间距,则该稳定集内的所有数据在最近被访问的可能都比较小,称为稳定集时间局部性。

稳定集时间局部性可以用于增强任何替换算法,这是因为稳定集时间局部性正交于所有替换算法的理论基础,包括时间局部性、访问频度、频繁访问序列等。本节中,我们将稳定集(stable set)时间局部性增加到 LIRS 替换算法中,得到 SSLIRS 替换算法。选择 LIRS 是因为,ARC 和 LIRS 是目前效果最好的两种缓存替换算法[4],而 ARC 有专利问题的限制。我们直接与目前最好的缓存替换算法相比,可以节省大量的对比工作量。

3 测试和评价

3.1 评价方法

为了评价基于稳定集的大容量客户端缓存管理效果,我们实现了稳定集挖掘器 SS-mining 以及缓存模拟器 csim。稳定集挖掘器用于处理应用访问 trace,生成稳定集模型参数。缓存模拟器用于给出特定参数对缓存行为进行模拟。除了实现基于稳定集的管理方法外,模拟器还实现了 LIRS 替换算法[5]和自适应顺序预取算法[6]用于比较。总响应时间采用将缓存模拟器过滤后的 I/O 负载在 skysan iSCSI 盘阵上播放获得。该盘阵采用 8 块盘的 raid5 配置,以千兆网络导出至客户端。由于集中存储系统通常是多个应用并发访问的,应用的相互干扰导致了最后的访问是一种伪随机状态,因此我们采用后端存储对于不同长度的随机请求的吞吐能力来计算后端负载,其定义如下:

定义 10 后端负载(**backend load**) 如果后端集中存储对长度为 L 的请求的随机读吞吐能力为 $Thoughput_L$,则一个长度为 L 的请求引起的后端负载 $Load_L$ 可以以如下方式计算:

$$Load_L = \frac{L}{Thoughput_L}$$

后端负载的计算需要上述 skysan 盘阵在不同请求大小下的随机吞吐能力,具体数值见图 1。

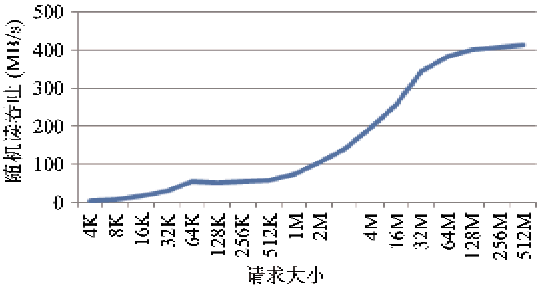


图 1 skysan iSCSI 盘阵在不同请求大小下的随机读吞吐

本文采用数据中心常见的几种真实负载进行评测,有代表搜索引擎类的 WebSearch2[10]、代表网页和 SQL 服务器类的 web_0[11]、代表互联网广告类的 DAP-DS[12]、代表媒体服务的 mds_0[11],以及代表虚拟机负载的 VMTRACE。前 4 种 trace 都是公开的、常见于相关研究中的,最后一种 trace 是代表了逐渐兴起的虚拟机计算中心应用,如 Amazon 的 EC2。由于没有找到公开的这类应用的 trace,我们采用两个虚拟机交替运行的方式抓取了 VMTRACE,两个虚拟机上的应用都是在 6 份 linux 内核源码里随机查找一些从/boot/System.map 里抽取的关键字。由于这种应用中稳定集大范围换出的情况比较普遍,特别适合于评价基于稳定集的替换算法的研究。

首先我们使用这些 trace 测试大容量点的数值,从而论证本文的缓存需求估计的合理性。然后我们在传统的缓存管理方法上逐步叠加上基于稳定集缓存管理的三步优化,用于论证这三步优化的有效性。

我们用 LA 代表采用 4K 粒度的目前最有效的缓存管理算法——LIRS[5] 替换算法和自适应预取算法[6]的测试结果;用 SL 表示采用稳定集粒度、LIRS 替换算法、没有自适应预取的测试效果;用 SLP 表示在 SL 基础上添加了基于稳定集预取的测试效果;用 SSP 表示在 SLP 基础上使用 SSLIRS 替换算法代替 LIRS 替换算法的测试效果。最后我们评测刻划前半段 trace 的开销,以及刻划所得结果的稳定性,以此证明稳定集刻划的开销是可承受的。

3.2 大容量点的评价

表 1 给出了各个 trace 的存储容量以及大容量点(包括 30% 的容量用于一遍访问和预取)。根据测试结果,我们可以得出结论:大于大容量点的 SSD 缓存在真实场景中是常见的,也就是说本文的缓存管理方法是广泛适用的。这是因为:(1)大容量点的绝对值从 3.8MB 到 23.9GB,远小于 SSD 可以达到的动辄数 TB 的容量;(2)大容量点相对于存储容量的比值从 0.01% 到 17.7%,大容量点远小于存储容量需求;(3)集中存储系统上通常有多个应用的数据,而一个客户端上通常只有少量的应用,这使得客户端的 SSD 更易于成为大容量缓存。

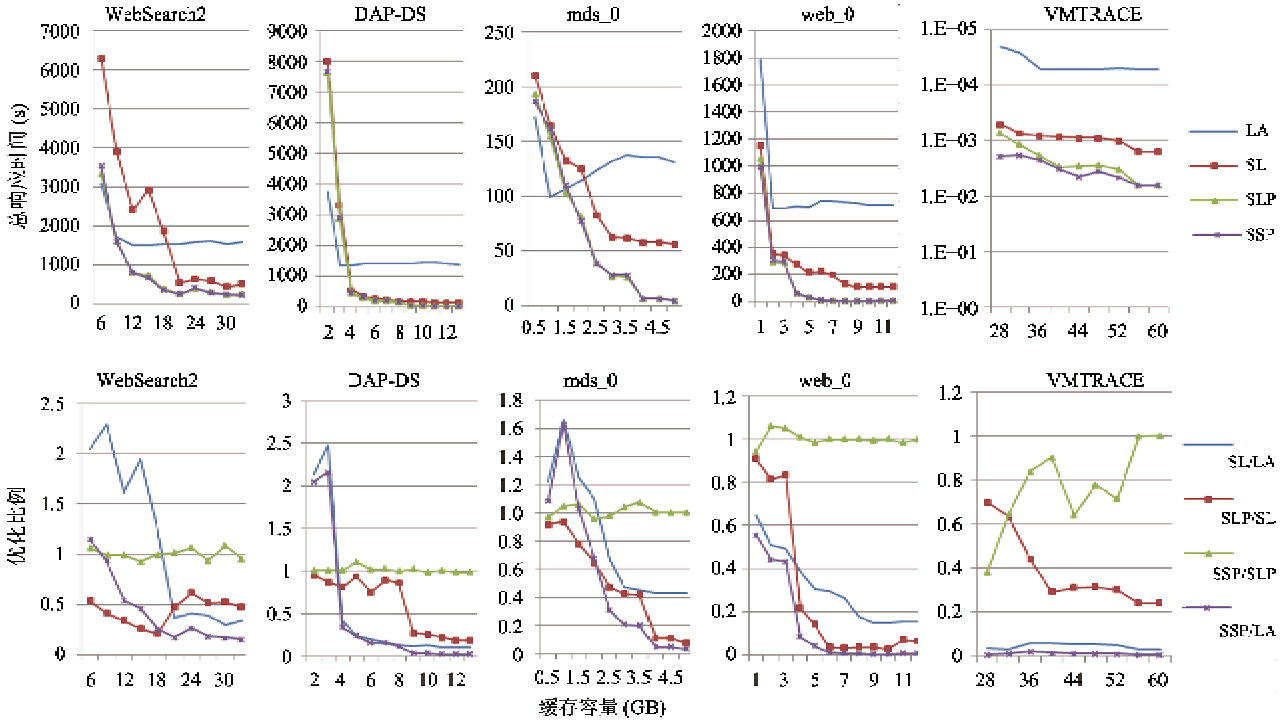
表 1 几种 trace 所需的存储容量、大容量点以及大容量点占存储容量比例

Trace	存储容量(GB)	大容量点	比例(%)
WebSearch2	91.1	5.83GB	6.4
web_0	33.9	250.4MB	0.7
DAP-DS	101.1	984.5MB	0.95
mds_0	33.9	3.8MB	0.01
VMTRACE	134.8	23.9GB	17.7

3.3 优化效果评价

图 2 给出了响应时间的测试结果,从图中我们可以得出如下结论:在缓存容量充足的情况下,稳定集粒度可以显著降低响应时间。在缓存容量达到 33GB、12GB、13GB、5GB 和 60GB 时,相对于 LA,稳定集粒度可以将总响应时间降低为原来的 33.5%、15.3%、9.7%、42.7% 和 3.3%。但当缓存容量变小时,稳定集粒度的优化效果逐渐减弱,甚至差于 LA。

基于稳定集的预取在大多数情况下显著地降低了响应时间。与 SL 相比,它最多可以把 5 个 trace



第一排图是 5 个 trace 的响应时间测试结果,第二排图是每一步及总的优化比例;
注意第一排中的 VMTRACE,由于不同方法的响应时间差距过大,使用了对数化的 Y 轴

图 2 总响应时间结果和优化比例

的响应时间分别降低为原来的 20.6%、18.3%、3%、10.9% 和 24.3%。值得注意的是预取没有带来任何负面效果,这主要是因为 LIRS 的两栈设计,低级别的预取数据很难替换高级别的反复访问数据,所以稳定集预取并没有带来缓存污染。

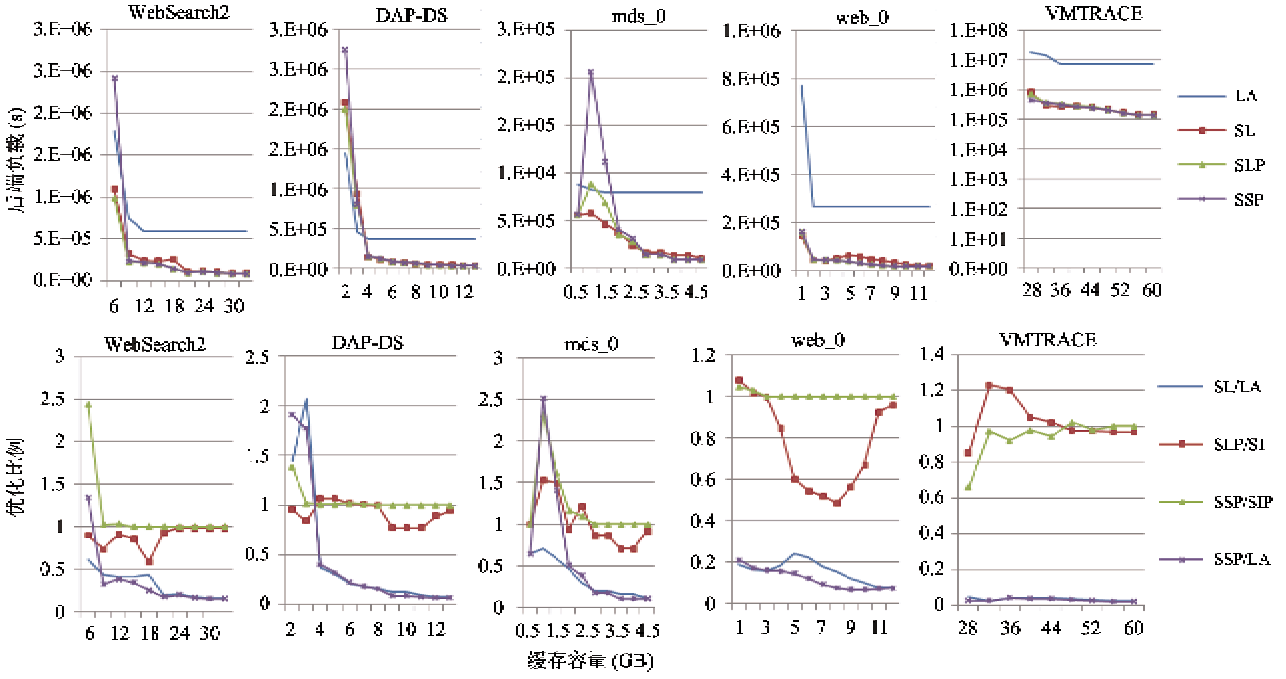
SSLIRS 显著地降低了 VMTRACE 的响应时间,尤其是在缓存容量相对不大的情况下。与 SLP 相比,它最多在缓存容量为 28GB 时降低响应时间到 38.1%。随着缓存容量增加,优化效果逐渐减小,直到缓存容量大到无需换出任何稳定集时优化效果不再增加。在其他 SSLIRS 没有优化效果的 trace 中,有时响应时间甚至会增加。

SSP 和 LA 的比值显示了总的优化率。在缓存容量很大时,本文的缓存管理算法带来了极大的改善,可以分别降低各个 trace 的响应时间到原来的 15.2%、1.7%、1.0%、3.3%、0.8%。但优化效果会随着缓存容量降低而降低。当缓存粒度很小且局部性比较差时甚至会带来负面效果。三个 trace 显示了最多可达 14%、116% 和 62% 的负面效果。

这里需要说明为什么我们不采用不命中率评价缓存性能。因为本文采用稳定集粒度作为缓存管理粒度,单个块的缺失代价与 4KB 粒度下的缺失代价差异很大,因此采用不命中率评价性能提升有失公允。但为了与近期的研究如 SieveStore[13] 直接相比,我们通过 LA 的不命中率估算了 SSP 的 4K 等价不命中率(是用 LA 的不命中率乘上 SSP 的响应时间优化比率)。在缓存容量充足的情况下,SSP 的 4K 等价不命中率为 1.4%,0.6%,0.4%,2.4% 和 0.2%。而 SieveStore 的不命中率几乎都在 50% 以上。本文的大容量客户端缓存性能远高于 SieveStore。带来的代价是每个客户端需要一块 SSD,而 SieveStore 大多数时间总共只需要一块,并且写开销小的多。由于 SSD 在性能、价格上的快速进步,性能、价格这两个因素的重要性逐渐减弱,而性能提高的重要性越来越强,因此本研究的出发点与 SieveStore 有很大区别。

图 3 给出了后端负载的测试结果,从图中我们可以得出如下结论:稳定集粒度在绝大多数情况下都能极大优化后端负载。并且,随着缓存容量的增大,SL 的优化能力总体上是越来越大的。在缓存容量很大,对 5 个 trace 分别达到 33GB、12GB、13GB、5GB 和 60GB 时,SL/LA 分别达到了 16.2%、11.9%、8.0%、7.8% 和 2.0%。

稳定集预取对于后端负载的优化效果大致是个“V”型。当缓存容量刚刚大于大容量点时,稳定集预取有可能会增加后端负载。随着缓存容量不断增大,在稳定集预取中对请求的聚合成为影响后端负载的主要方面,稳定集预取最多可以把后端负载降



第一排图是后端负载测试结果,第二排图是优化比例;注意对于 VMTRACE 的后端负载我们仍然采用了对数 Y 轴

图 3　后端负载以及优化比例

低为原来的 58.8%、48.5%、77.0%、70.8% 和 96.6%。但是随着缓存容量的进一步增大,稳定集粒度也会进一步增大,而 I/O 连续性对于后端负载的降低是随着基本粒度的增大而逐渐减弱的,因此稳定集预取对于后端负载的优化将会越来越小。此时 SLP 的后端负载只是稍小于 SL。

SSLIRS 对于 VMTRACE 的后端负载也有比较显著的优化:在优化效果最明显时,SSP 可以将 SLP 的后端负载降低到原来的 65.8%。但是对于其他 4 个 trace,SSLIRS 有最大 143.3%、4.6%、38.0%、133% 的后端负载增长,这说明 SSLIRS 的主动降级方式可能还是有些激进。

SSP 和 LA 的比值显示了稳定集缓存管理的总效果。当缓存容量很大时,基于稳定集模型的缓存管理方法可以将后端负载降至原来的 15.8%,7.5%,7.5%,10.9% 和 2.0%。随着缓存容量的减少,SSP 的优化能力总体上减少。在缓存容量接近大容量点的情况下,SSP 的后端负载甚至可能超过 LA。

3.4 挖掘时空开销分析

表 2 中列出了 5 种 trace 刻划部分的统计数据以及时空开销。由表 2 中的数据可知,稳定集参数挖掘的时间开销是可以接受的。这是因为:(1)每个 trace 的刻划时间占 trace 持续时间的百分比是 0.01% ~26.01%,刻划的进度不会慢于 trace 产生的进度。(2)在 web_0 trace 里,我们使用 1.4 天的 trace 刻划结果来指导接下来 5 天的缓存管理,并且根据上文的测试,我们取得了良好的效果。这证明了稳定集刻划的结果很稳定,稳定集刻划不需要频繁地进行。(3)刻划对 trace 持续时间的平均时间占比只有9.94%,远小于经典挖掘算法C-miner[14] 中的25%;其平均每秒钟处理的请求数是 2931K,远大于 C-miner 的 0.087K。这些都证明了稳定集刻划的时间开销是可接受的。

表 2 刻划时空开销分析

Trace	跨越时间(h)	条数(K)	时间开销(s)	空间开销(MB)
WebSearch2	1.93	2290	1807.2	7.6
web_0	33.53	303	187.7	1.8
DAP-DS	13.60	693	790.6	13.7
mds_0	75.48	72	30.4	1.5
VMTRACE	8.90	25466	7017.3	2.8

由表 2 中的数据可知,稳定集刻划的空间开销是可接受的,这是因为现代服务器一般都有数 GB 的内存,最大 13.7MB 的内存开销微不足道。

4 与相关研究的对比

已有工作如文献[13,15]对使用 SSD 作为存储系统缓存进行了深入研究。但这些研究都局限于当时的 SSD 的价格及性能状况,没有考虑到 SSD 在这两方面的快速发展。比如 SieveStore[13] 的设计依据的是 Intel X-25E 的容量、性能、价格等指标,因此采用了共享缓存及基于筛选的缓存载入等方式,用于减少 SSD 花费以及写 SSD 的次数。然而今天,每 GB SSD 的价格已经降到 1 美元以下[16],性能最高的 SSD 如 OCZ Z-Drive R5,其随机写性能已经达到 2GB/s 以上,远优于当时的 14 美元/GB 以及 13.2MB/s[13]。因此这两个因素的重要性也随之下降。本文立足于 SSD 的快速发展趋势,使用 SSD 的容量提高集中存储系统的性能和可扩展性,这两方面提升的效果远高于 SieveStore[13]。

已有的存储系统缓存管理方法[4-6] 都基于数据访问的微观模型,如 LRU 堆栈模型、LFU 概率模型、空间局部性等。为了更高效地管理大容量缓存,本文引入了数据访问的宏观模型指导缓存管理。宏观模型的发展已经历了两个阶段:工作集模型阶段和阶段-转换模型阶段[3]。工作集模型认为数据访问是柱状的,在一个数据集合上反复不变地访问。而阶段-转换模型发现了这一假设与事实的不符,并在时间维度对数据访问进行了划分。本文引入的稳定集模型(SSM)则更进一步,在空间维度上对数据也进行了划分,从而能够更好地预测数据访问在下一个阶段的行为。

5 结论

本文使用大容量客户端缓存技术提高集中存储系统的性能和可扩展性,并指出阶段间数据换入换出效率是大容量缓存管理最重要的问题。同时定义了 SSM,并使用 SSM 参数指导大容量客户端缓存管理。模拟实验表明,基于 SSM 的缓存管理方法适用大部分情况下的 SSD 客户端缓存,并在缓存容量充足的情况下将应用的 I/O 响应时间降低 1 到 2 个数量级,将集中存储负载降低 1 个数量级左右,从而大大提高了网络存储系统的性能和可扩展性。SSM 刻

划的开销是合理的,它很有希望应用于真实的网络存储系统中,在性能优化方面有巨大的潜力。

未来的工作将继续研究在缓存没有如此充足的情况下,如何利用 SSM 优化大容量客户端缓存的管理。另外,在线学习和使用 SSM 参数,以及多客户端信息的综合,都是下一步需要进行的工作。

参考文献

[1] Tu T, Rendleman C A, Miller P J, et al. Accelerating parallel analysis of scientific simulation data via Zazen. In: Proceedings of 8th USENIX Conference on File and Storage Technologies, San Jose, USA, 2010. 129-142

[2] Eshel M, Haskin R, Hildebrand D, et al. Panache: a parallel file system cache for global file access. In: Proceedings of 8th USENIX Conference on File and Storage Technologies, San Jose, USA, 2010. 155-168

[3] Denning P J. Working sets past and present. *IEEE Trans on Software Engineering*, 1980, SE-6(1): 64-84

[4] Jiang S, Chen F, Zhang X. Clock-pro: an effective improvement of the clock replacement. In: Proceedings of the Annual Conference on USENIX Annual Technical Conference 2005, Berkeley, USA, 2005. 35-35

[5] Jiang S, Zhang X. LIRS: an efficient low inter-reference recency set replacement policy to improve buffer cache performance. In: Proceedings of the International Conference on Measurements and Modeling of Computer Systems 2002, New York, USA, 2002. 31-42

[6] 吴峰光. Linux 内核中的预取算法:[博士学位论文]. 合肥:中国科学技术大学,2008.

[7] Zhong Y, Orlovich M, Shen X, et al. Array regrouping and structure splitting using whole-program reference affinity. In: Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation, New York, USA, 2004. 255-266

[8] Batson A P, Madison W. Measurements of major locality phases in symbolic reference strings. In: Proceedings of the International Conference on Measurements and Modeling of Computer Systems 1976, NY, USA, 1976. 75-84

[9] Guo M, Sha Z, Xu L. Using stable sets to enhance I/O pre-fetch. Information, 2011, 14(12): 4007-4016

[10] SPC-1 trace, http://traces.cs.umass.edu/index.php/Storage: UMASS, 2006

[11] Narayanan D, Donnelly A, Rowstron A. Write off-loading: practical power management for enterprise storage. In: Proceedings of 6th USENIX Conference on File and Storage Technologies, CA, USA, 2008. 17

[12] Kavalanekar S, Worthington B, Zhang Q, et al. Characterization of storage workload traces from production windows servers. In: Proceedings of IEEE International Symposium on Workload Characterization, Seattle, USA, 2008. 119-128

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

[14] Li Z, Chen Z, Srinivasan S M, et al. C-miner: Mining block correlations in storage systems. In: Proceedings of the 3rd USENIX Conference on File and Storage Technologies, CA, USA, 2004. 173-186

[15] Narayanan D, Thereska E, Donnelly A, et al. Migrating server storage to SSDs: Analysis of tradeoffs. In: Proceedings of the European Conference on Computer Systems, Nuremberg, Germany, 2009. 145-158

[16] Shilov A. As Latest SSD Prices Hit $0.65 per Gigabyte, OCZ Forecasts Further Drop of Costs. News in Xbit lab. http://www.xbitlabs.com/news/storage/display/20120502220953_As_Latest_SSD_Prices_Hit_0_65_per_Gigabyte_OCZ_Forecasts_Further_Drop_of_Costs.html

Using the stable set model to management large caches

Guo Mingyang*,**, Zhang Yongle*,**, Liu Zhenhan*, Liu Zhenjun*, Xu Lu*

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

(**Graduate University of the Chinese Academy of Sciences, Beijing 100039)

Abstract

This article points out that the efficiency of "data exchange" between phases is the most important problem in management of large caches, and focuses on studying the problem. The stable set model (SSM), a new macro model for data access, is defined to help predicting data access in new phases, and based on the model, a set of cache management methods, including cache demand estimation, cache granularity selection, data pre-fetch and cache replacement, are given to optimize the efficiency of data exchange between phases, so as to improve the I/O load and performance. The evaluation results show that the cache management methods based on SSM can decrease the storage load to traditional ones' 2.0% ~15.8%, and decrease the average response time to traditional ones' 0.8% ~15.2%.

Key words: network storage, large cache, stable set model (SSM), phase-transition behavior, cache demand estimation, cache granularity selection, data pre-fetch, cache replacement