高技术通讯 2009 年 第 19 卷 第 7 期:699 ~ 705

doi:10.3772/j.issn.1002-0470.2009.07.007

存储系统中的频繁访问模式挖掘①

朱旭东②*** 卜庆忠*** 柯 剑*** 那文武*** 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 研究、分析了影响经典的模式挖掘方法挖掘频繁访问模式的效率,使其难以被存储系统接受的主要因素——噪音的产生原因和表现类型,提出一种具有噪音过滤能力,适应存储系统频繁访问序列模式挖掘的新方法——Z-Miner。Z-Miner 使用全局分支裁剪和分支聚类方法来过滤噪音,对实际系统工作负载的模拟结果显示,Z-Miner 指导的预取可以使缓存失效率降低 40% ~ 66%,平均响应时间降低 26% ~ 66%。相对经典挖掘方法,Z-Miner 的挖掘开销有 1 ~ 2 个数量级的下降,而预取优化效果提高了 1 倍。

关键词 频繁访问模式,数据块关系,序列模式挖掘,聚类,预取

0 引 言

频繁访问模式指存储系统的访问流中多次出现的数据块访问序列,是数据块关系在存储系统 I/O 访问中的一种表现。数据块关系是一种常见的数据语义模式,例如数据库 b + 树上的父子节点、文件系统中同一文件的数据块和 inode 块之间都存在着语义相关。存在语义相关的数据块总是趋向于被连续访问,表现为频繁出现的访问序列。频繁访问模式挖掘用于指导存储系统中的预取、缓存替换、数据分布,可以显著提升存储系统的性能。频繁访问模式比空间局部性等简单访问模式[1]具有更多的数据块信息,不需要对数据块的时空分布做任何假设,同时也具有更好的稳定性[2,3],很多研究对两数据块相关进行了统计分析,SD[4,5]、MPADC[6]和 ALIS[7]都使用有向图方法挖掘两数据块关系,即 2-pattern 模式。存储系统可以对 2-pattern 模式进行高效的挖掘,但 2-pattern 模式不具有多数据块相关的信息,也就无法提供更为精确的优化指导。频繁访问模式可以表现复杂的多数据块语义相关,但是现有挖掘方法的时空开销难以忍受。最近的一些研究引入数据挖掘领域的方法来发现存储系统的频繁访问模式。C-Miner[2]对存储系统的工作负载进行分段,转化为多序列频繁模式挖掘问题,并使用经典模式挖掘方法进行挖掘。C-Miner * [3]在 C-Miner 的基础上解决了分段产生的模式损失问题。C-Miner 和 C-Miner * (C-Miner[*])在真正意义上对频繁访问模式进行了挖掘,获取多数据块之间的关系。由于经典模式挖掘方法[8-10]不能很好地适用于存储系统的频繁访问模式挖掘问题,其挖掘时间甚至接近工作负载实际运行的时间,同时也限制了挖掘精度的提升。

本文分析了影响频繁访问模式挖掘效率的主要原因——噪音,同时针对噪音的特征,提出了一种高效的数据挖掘方法 Z-Miner。Z-Miner 使用聚类方法进行全空间分支裁剪来过滤噪音,在保证挖掘精度的前提下限制了候选集合的增长规模,使得挖掘的时空开销大大减少。测试结果表明,Z-Miner 的时间开销比经典模式挖掘方法降低一到两个数量级,而预取优化效果是经典挖掘方法的 2 倍以上。对实际系统工作负载进行模拟可知,Z-Miner 指导的预取策略使失效率下降 40% ~ 66%,平均响应时间下降 26% ~ 66%。

1 频繁访问模式挖掘中的噪音

数据块关系产生的完整访问序列称为有效模式。我们定义模式挖掘中的噪音为不体现数据相关性,且对有效模式的挖掘产生影响的请求或请求缺失。噪音在工作负载中普遍存在,使挖掘结果产生大量的冗余模式(称为伪模式)或者淹没有效模式,是影响挖掘效率的重要原因。由于缓存的影响,I/O

① 863 计划(2007AA01Z402)和 973 计划(2004CB318205)资助项目。

② 男,1979 年生,博士生;研究方向:网络存储;联系人,E-mail: zhuxd@ict.ac.cn

(收稿日期:2008-04-01)

流中的写请求几乎不体现数据块关系,因此本文只考虑数据块读语义关系,即只对读请求进行挖掘。

1.1 产生噪音的原因和噪音的分类

根据产生的原因和对噪音的影响,噪音可以被分为以下五种类型。

请求冗余噪音:工作负载由多个应用的访问流聚合产生。存储系统多个应用的访问模式之间、模式与随机请求之间相互交错,会产生大量随机出现的伪模式。这些伪模式比有效模式多一些新的数据块请求,我们称之为请求冗余噪音。

请求间隔噪音:应用流的聚合使有效模式的请求间隔增大。由于挖掘算法需要限制模式相邻请求间的最大距离 max _ step,因此请求间隔的增大会导致有效模式被截断甚至淹没,我们称这类噪音为请求间隔噪音。例如访问流 A = {abab} 和 B = {cdcd} 聚合成{acbdcabd},当 max _ step = 1 时,表现数据块语义关系的{ab}和{cd}被请求间隔噪音淹没。

请求干涉噪音:同一个数据块往往存在多种访问语义,会在多个模式中出现,或者被随机地访问到。这些数据块的访问同样会对模式挖掘产生干扰。例如序列{abd..abd..abd..acd..acd},在 max _ step > 1 时,两个有效模式{abd}和{acd}相互干涉产生了伪模式{ad}。这类伪模式大都是正常模式的子模式。我们称这类噪音为请求干涉噪音。

请求缺失噪音:应用系统的缓存对具有时间局部性的数据块请求进行过滤,同时也过滤掉有效模式序列中的一些请求,导致同一数据块关系产生不同的访问序列。例如应用产生的访问序列{cabcd ...abcd ...abcd},由于 c 的第二次访问与第一次访问的间隔很小而被缓存命中,经过缓存后的序列为{cabd ...abcd ...abcd},因此在挖掘得到有效模式{abcd}之外,还产生了伪模式{abd}。这类因个别元素缺失而产生相似模式的噪音,称为请求缺失噪音。

请求顺序噪音:一些数据块之间具有非顺序敏感的访问语义关系。有的应用不限制相关数据块的访问顺序,例如数据库的一个表可以以不同的索引方式被遍历。还有一些应用,符合语义相关的数据块访问由多个结点协作完成,使得数据块访问请求的到达顺序随机产生。对相同数据块不同顺序的访问,会使挖掘算法认为存在多种频繁访问模式。我们称这种噪音为请求顺序噪音。

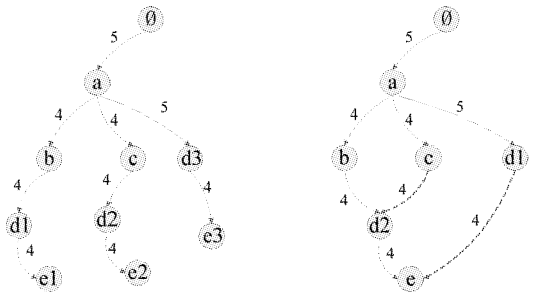
在噪音影响下,挖掘结果存在着大量相似的伪模式:这些相似模式由访问流的相同部分产生,或者具有共同的子序列,或者具有大量相同的元素。由于经典挖掘方法不具有较强的噪音过滤能力,挖掘结果中伪模式数量远远大于有效模式的数量,这是挖掘开销增长的重要原因。

1.2 噪音的过滤

经典挖掘算法限制模式序列出现的最少次数 min _ sup,可以有效过滤请求冗余噪音;设置合适的最大请求间隔 max _ step 可以过滤大部分请求间隔噪音。我们在经典深度挖掘方法 Clospan[9] 的基础上,讨论对其余三种噪音的过滤方法。

请求干涉或请求顺序噪音产生的不同模式前缀往往具有相同的后缀分支,可以被裁剪以减少挖掘的时空开销。如果两个前缀具有包含关系,经典的深度挖掘算法如 Clospan 对后缀分支进行裁剪。这样的裁剪可以减少对子模式的冗余搜索,但是对不具有包含关系的模式前缀不具有后缀分支裁剪能力,也就不具有过滤噪音的能力。

对任意两个后缀集合相同的分支进行合并称为全空间分支裁剪。{abcdef..abcdef..acbdef..acbdef..ad}由经典的深度优先挖掘算法得到模式树的一部分如图 1(a), max _ step = 3。由于 b 和 c 的请求顺序不一致,前缀{ab}和{ac}不互为子串,但是他们却具有完全相同的后缀集合。由于干涉噪音{ad},我们同样也无法将 e3 与 e1 或 e2 合并。图 1(b)的全局分支裁剪不对前缀进行限制,因此减少了冗余的分支搜索。



(a) 经典深度优先算法　　(b) 全空间分支裁剪

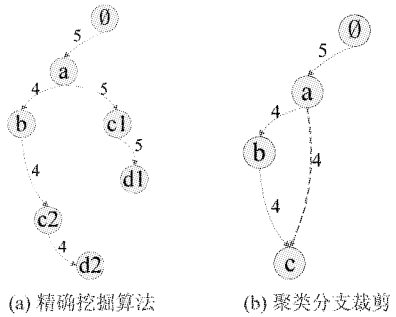
图 1　不同挖掘方法的模式树片段

全空间分支裁剪可以有效过滤请求干涉和请求顺序噪音,但是对请求缺失噪音无效。请求缺失噪音是产生伪模式的主要原因,最坏情况下, n 个请求缺失可以使一个有效模式产生 $n^2 - 1$ 个伪模式。而在多流的应用环境下,请求间隔噪音虽然对有效模式的挖掘影响很大,但是大多使有效模式被截断或者淹没,而产生的冗余伪模式反而数量不多。

序列{abcdef..abcdef..abcdef..abcdef..acdef}

可以挖掘得到{abcdef}和{acdef}。由图2(a)可知,由于其中一个子序列缺少请求b,因此结点c1和c2的后缀集合不同,产生了两个搜索分支。与大多数请求缺失的情况一样,请求b缺失的序列个数远小于完整序列的个数,c1和c2的后缀分支相近。

对相似后缀集合的分支进行裁剪,我们称为分支聚类,如图2(b)。由于合并的后缀分支接近,因此模式信息损失很小,也不会对系统优化产生明显的影响。相对于精确裁剪,分支聚类可以大量减少请求缺失噪音产生的冗余搜索开销。



(a) 精确挖掘算法　　(b) 聚类分支裁剪

图2　分支聚类

2　支持噪音过滤的挖掘算法

根据上文的分析,我们提出一种适应存储系统频繁访问序列模式挖掘的方法Z-Miner。Z-Miner以经典的深度优先挖掘算法Clospan为基础,使用聚类方法进行全空间分支裁剪对噪音进行过滤。

2.1　单序列挖掘算法

频繁访问模式是单序列挖掘,为区别经典的多序列挖掘问题,我们给出单序列挖掘的算法(表1)。

定义 $D = \{b_1, b_2, \cdots, b_n\}$ 为所有数据块集合。定义访问流源序列 $S = \{1, 2, \cdots, m\}$。设 $i \in S$,使得 $b(i)$ 表示 S 第 i 个请求访问的数据块,$b(i) \in D$。定义数据块b的请求全集 $Q_b = \{i \mid i \in S$ 且 $b(i) = b\}$。设 $1 \leqslant l \leqslant m$,$S$ 的子序列 $s = \{r_1, r_2, \cdots, r_l\}$,其中 $1 \leqslant r_1 < r_2, \cdots, r_l \leqslant m$。定义 $B(s) = \{b(r_1), b(r_2), \cdots, b(r_l)\}$ 为一个有序列表。定义 $t(s) = r_l$ 表示 s 最后一个请求。

定义模式 $p = \{s_1, s_2, \cdots, s_j\}$,满足 $B(s_1) = B(s_2) = \cdots = B(s_j)$。设 $T(p) = \{t(s_1), t(s_2), \cdots, t(s_j)\}$ 表示 p 的尾结点请求集合。易证 $T(p) \subseteq Q_{b(t(s))}$,其中 $s \in p$。定义L为 p 的集合。

算法1为单序列频繁模式挖掘算法。根据1.2节中的讨论,我们通过限制模式出现的频率min_sup来过滤请求冗余噪音,并通过设置合适的相邻请求在访问流中的最大间隔max_step来过滤请求间隔噪音。

表1　算法1

Algorithm 1 Z-Miner(S, max_step, min_sup, L)
Input: 访问流序列S, max_step, min_sup Output: 模式序列集合L
1. L←∅; 2. 获取S中所有数据块,加入集合D; 3. For each b∈D do 4. 将 Q_b 加入L; 5. MINING (Q_b, S, min_sup, max_step, L);

2.2　全空间分支裁剪

设数据块b的请求集合 $q_b \subseteq Q_b$,从 q_b 的每一个请求起向后max_step间隔内对S进行搜索,得到数据块 b' 的请求集合为 $q_{b'}$。易见,如果使用相同的max_step,则 $q_b = q_b' \Rightarrow q_{b'} = q_{b'}'$。

定理1　如果两个请求集合 $q_b = q_b'$,那么由 q_b 和 q_b' 可以挖掘到相同的后缀模式分支。

证明:假设 q_b 和 q_b' 挖掘得到不同的后缀模式集合 PL 和 PL',不妨设 $|PL| \geqslant |PL'|$。则必存在一个完整模式 p 满足 $p \in PL$ 且 $p \notin PL'$。我们设 PL' 中与 p 具有最长共同前缀的模式为 p',$p' \neq p$。p 和 p' 的最长共同前缀为 $p_m = p_m'$,可知 $T(p_m) = T(p_m')$。根据上文的讨论可知,$T(p_m)$ 和 $T(p_m')$ 向后max_step间隔内对任一数据块 b' 搜索得到的请求集合 $q_{b'}$ 和 $q_{b'}'$ 均满足 $q_{b'} = q_{b'}'$。如果 $p = p_m$,则 p 可以扩展 p' 中 p_m' 的后继数据块请求,这与 p 是一个完整模式矛盾。如果 $p \neq p_m$,则 p_m' 可扩展 p 中 p_m 的后继数据块请求,则 PL' 中必存在 p'' 使得 p 和 p'' 的共同前缀大于 p',这与假设矛盾。

推论1　如果两个模式前缀 p 和 p' 具有相同的尾结点请求集合 $T(p)$ 和 $T(p')$,那么有 p 和 p' 具有相同的后缀模式分支。

由此我们得到全空间分支裁剪算法,见算法2(表2)。当扩展得到一个新的模式前缀时,我们比较它和已有前缀的尾结点请求集合,如果存在相等的集合,则将两个前缀的尾结点合并。如果不存在,则将新扩展的模式前缀加入模式集合中,并保存该前缀的尾结点请求集合,然后对该前缀进行后缀分支搜索。

表 2 算法 2

Algorithm 2 MINING(p, S, min _ sup, max _ step, L)
Input: 模式前缀 p, 访问流序列 S, max _ step, min _ sup
Output: 模式集合 L
1. 对所有在 S 上出现在 $T(p)$之后 max _ step 间隔内的请求,按数据块分别产生请求集合 q_{b1}, q_{b2}…
2. for each $\mid q_b \mid \geqslant$min _ sup do
3. Match (q_b, L, p');
4. If $p' \neq \varnothing$
5. 将 p 指向 p' 的尾结点;
6. else
7. 将 p 按 q_b 扩展为 p_e ;
8. 将 p_e 加入 L;
9. MINING (p_e, S, min _ sup, max _ step, L);

2.3 分支聚类

根据 1.2 节中的讨论,如果两个前缀 p 和 p' 的后缀集合相似,即 $T(p) \approx T(p')$,则挖掘得到的后缀分支也近似。

为了减少尾结点请求集合的空间开销和比较操作的时间开销,我们在挖掘开始时,对源序列进行扫描,对每个数据块获得集合 Q_b 以及每个请求在 Q_b 中的序列号 $v(i)$, $i \in S$。每个结点都保存一个大小为 $\mid Q_b \mid$ 的 bitmap,如果请求 i 在某结点的请求集合 q_b 中,则该结点 bitmap 的第 $v(i)$ 位为 1,否则为 0。设 $Bm(p) = \{x_1, x_2, \cdots, x_n\}$, $x_1, x_2, \cdots, x_n \in \{0, 1\}$ 为 p 尾结点集合的 bitmap,易见,$Bm(p) = Bm(p') \Rightarrow T(p) = T(p')$。

我们用向量 Bm(p)来表示 T(p),通过修改 k-means[11]方法对尾结点请求集合进行聚类,来裁剪近似的后缀分支。

首先,两个尾结点请求集合的相似程度可以表示为 $\mid T(p) \cap T(p') \mid / \mid T(p) \cup T(p') \mid$,定义 Count(Bm($p$))为 Bm($p$)中 1 的个数。规定 T($p$)和 T($p'$)的距离为

$$Dist(p, p') = 1 - Count(Bm(p)\&Bm(p'))/ Count(Bm(p) \mid Bm(p'))$$

其次,请求集合的聚类不限制 k 的数量。由于请求集合的聚类对距离非常敏感,大的距离会导致对两个差异很大的后缀分支进行合并。因此我们只限制同一簇中向量离中心的距离,而不限制 k 的数簇的数量。第三,我们选用第一个出现的不能合并到其他簇的 Bm(p)作为新簇的 means。由于限制同一簇中向量的距离,我们在新元素加入到一个簇时,不修改该簇的 means 值,以免引起聚类中心的漂移。最后,为了提高挖掘的效率,当一个向量可以被聚类到多个簇时,我们选择加入第一个被我们找到的簇。综上,我们得到 Match 算法(表 3)

表 3 算法 3

Algorithm 3 Match (q_b, L, p)
Input: 尾结点请求集合 q_b, 模式集合 L
Output: 模式 p;
1. max _ d 为允许的最大聚类距离;
2. $p \leftarrow \varnothing$;
3. If 存在 $p' \in L$, 满足 Distance (q_b, p') $\leqslant$ max _ d;
4. $p \leftarrow p'$;
5. return;

Z-Miner 可以高效地挖掘频繁访问模式。Z-Miner 设置 min _ sup 和 max _ step 参数过滤请求冗余和请求间隔噪音,采用全空间分支裁剪过滤请求干涉和请求顺序噪音,采用聚类方法过滤请求缺失噪音。Z-Miner 提高挖掘效率的同时,采用合适的聚类距离 max _ d,避免在聚类过程中产生大的信息损失。

3 测试和评价

3.1 评价方法

为了评价 Z-Miner 挖掘频繁访问模式的效果,我们实现了存储系统的 cache 模拟器 ZM _ cache,并将 ZM _ cache 和 DiskSim 结合实现了自己的存储系统模拟器。ZM _ cache 支持模式预取和顺序预取。在本研究中,我们采用最常见的 LRU 算法。DiskSim 采用的磁盘模型为 10000 转的 IBM Ultrastar 36Z15。

模拟采用几种真实系统中收集到的磁盘 trace: Cello92, Cello96, Cello-99, TPC-C 和 OLTP。对每一种 trace,我们用 Z-Miner 挖掘其前半部分,然后对后半部分进行模拟播放,由挖掘得到的模式指导预取。

首先我们将比较 Z-Miner 与经典挖掘方法的挖掘效率,并讨论聚类距离对 Z-Miner 挖掘效率和预取效果的影响。其次我们对模式指导的预取(pattern-directed prefetching, PDP)与顺序预取的效果进行较为全面的比较。最后我们以 SD 为例,比较 Z-Miner 所代表的长模式指导预取与传统的 2-pattern 模式指导预取的精度和效率。

3.2 Z-Miner 的挖掘开销

我们在 Intel Xeon 2.4GHz 和 linux-2.6.11 的系统上运行 Z-Miner。表 4 为 Z-Miner 对不同的 trace 进行挖掘的时间和空间开销。与 Z-Miner 相比较的是

采用经典模式挖掘算法 Clospan 的 C-Miner[*],它们的运行环境均为 Intel Xeon 2.4GHz 和 linux-2.6.20。

由于经典挖掘算法在 min _ sup 减小时,其时空开销会爆炸式增长,甚至难以测试,因此 C-Miner[*]的 min _ sup 设为 20。Z-Miner 则可以将 min _ sup 设为最小值 2,尽可能多地获得模式信息。

由表 4 可知,Z-Miner 的时间开销比 C-Miner[*]要少一到两个数量级。对大多数 trace,Z-Miner 的空间开销比 C-Miner* 小,但大于 C-Miner。

由于对噪音的有效率过滤,Z-Miner 的时间开销远小于经典挖掘方法。这是由于 Z-Miner 减少了对相似分支的搜索时间。对分支的大量合并也使空间开销减小。由于挖掘效率高,Z-Miner 可以设置较低的 min _ sup,使挖掘到的有效信息远多于 C-Miner[*],这也是其空间开销大于 C-Miner 的原因。

表 4　ZM 和经典挖掘算法的开销比较(Z-Miner: min _ sup = 2, max _ d = 0.1; Cminer[*]: min _ sup = 20)

工作负载	Z-Miner		C-Miner		C-Miner*	
	时间(秒)	空间(MB)	时间(秒)	空间(MB)	时间(秒)	空间(MB)
Cello92 (3 天)	18	35.5	7800	3.1	513	599
Cello96 (1 天)	92	146	2089	4.6	979	355
Cello99 (1 天)	47	103	6060	8.5	2135	656
TPC-C (1 小时)	21	47.2	3355	9.2	1414	14
OLTP (2.5 小时)	3.9	13.4	174	173	40	455

我们以 cello92 为例讨论 max _ d 对时空开销和信息量的影响,并用挖掘得到的模式树结点数量表示信息量的规模。图 3 显示了不同聚类距离下 Z-Miner 的挖掘开销。在 max _ d = 0 即不进行任何分支聚类的情况下,与表 4 比较,Z-Miner 的挖掘效率依然高于经典挖掘算法,这是由于全空间的分支裁剪可以有效过滤请求干涉噪音和请求顺序噪音。

如图 3 所示,max _ d 越大,聚合的程度也越大,挖掘开销越小。当 max _ d 大于 0.2 时,挖掘时间的改变趋于平缓,这是由于请求缺失噪音产生的大部分模式分支的频度都和有效模式非常接近,当 max _ d 等于 0.2 时这些分支都已经被裁剪。图 4 中的用例可以根据 max _ step 值分为 3 组,越高 max _ step 值的组随 max _ d 增加而下降的速度越快。同组中不同 min _ sup 用例之间的行为类似。这是由于分支聚类可以很好地过滤请求缺失噪音,而对请求冗余噪音无效。

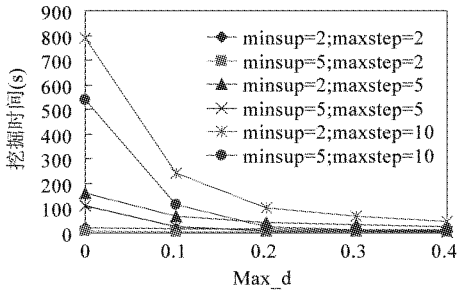
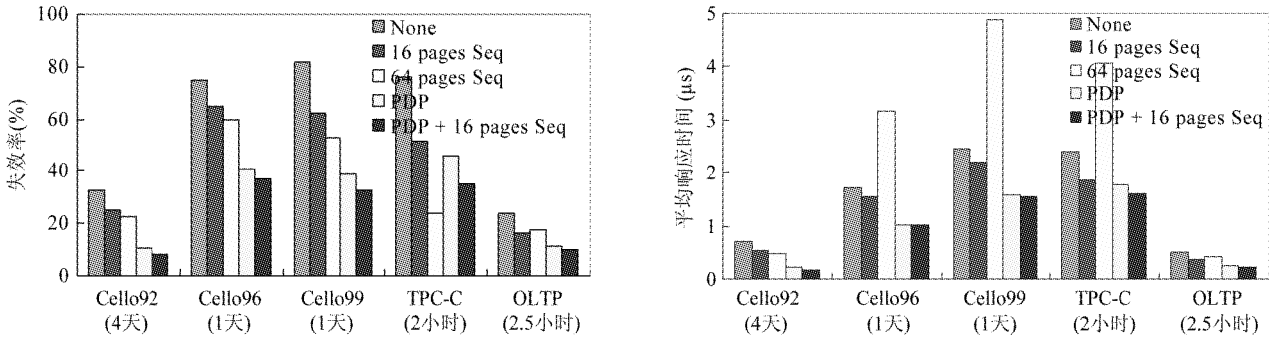


图 3　聚类距离对 Cello92 挖掘效率的影响



(a) 失效率比较　　(b) 响应时间比较

图 4　模式指导的预取和顺序预取的优化效果比较(None:无预取; Seq:顺序预取;PDP:模式指导的预取)

3.3　Z-Miner 指导预取的效果

我们对不同的预取策略设置相同的缓存总容量,对于需要进行预取的用例,在缓存中划出一部分作为预取缓存。在图 4 中,Cello 的缓存总容量为

64MB,预取缓存为 4MB。TPC-C 的缓存总容量为 1GB,预取缓存为 16MB。OLTP 的缓存总容量为 16MB,预取缓存为 1MB。

图 4 中 Z-Miner 的挖掘参数与表 4 相同,对所有测试的 trace 的响应时间有 26% ~ 66% 的提升。由文献[2,3]可知,C-Miner[*]对 Cello92、Cello99 和 TPC-C 的响应时间只有 12% ~ 30% 的性能提升,对 Cello96 和 OLTP 的优化效果不明显。Z-Miner 指导预取的优化效果达到 C-Miner[*]的 2 倍以上。这是由于 Z-Miner 的高效挖掘算法支持对低频模式的高效挖掘,比传统挖掘算法获得更多的模式信息指导预取优化。

如图 4(a)所示,顺序预取明显地降低了 cache 的失效率。顺序预取 64 个页比预取 16 个页的失效率略有降低。对大多数 trace,模式指导的预取优化效果对顺序预取有 14% ~ 57% 的下降,相对无预取的情况失效率有 40% ~ 66% 的下降。我们测试了顺序预取 16 个页的同时也进行模式指导预取的情况,失效率均明显低于单独使用顺序预取和模式预取的情况。

图 4(b)中显示,顺序预取对响应时间的优化效果有限,当预取的数据量较大时,响应时间反而增加。例如 64 页的顺序预取虽然使 TPC-C 的失效率下降到 23.8%,但是平均响应时间却比无预取增加了一倍。这是由于过多的预取操作增加了磁盘的负载,影响对正常请求的响应。Cello96 和 Cello99 的负载较高,因此顺序预取对响应时间优化效果也不理想。

模式指导的预取对响应时间有很好的优化效果,即使在 TPC-C、Cello96 和 Cello99 这样的高负载流中,依然有 26% ~ 40% 的性能提升。这是由于模式指导的预取只需要预取较少的数据量就可以明显减少失效率,对正常请求的影响很小。

我们对 Cello92 的每 2 万个请求作一次失效率的统计。如图 5 所示,系统在无预取情况下,失效率会周期性出现波峰,这是由于系统在更换数据集时产生大量的冷启动失效和容量失效。顺序预取只能在一定程度上减少这种失效率。使用模式预取则可以使失效率大幅降低。这是由于模式预取对数据集进行精确预取,大量减少冷启动失效和容量失效。

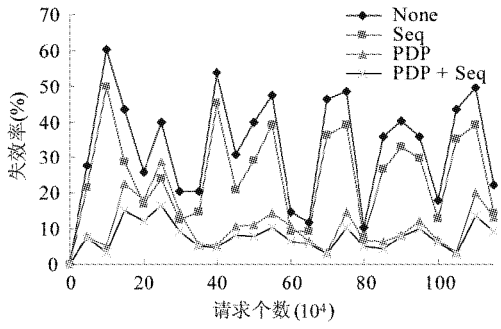


图 5 Cello92 的失效分布

我们以 cello92 和 OLTP 为例讨论了不同的聚类距离对预取效果的影响。在不考虑预取精度的情况下聚类的距离对预取效果的影响很小。由图 6 所示,max_d 上升几乎对失效率不产生影响。这是由于我们对模式前缀所有可能的后继数据块进行了预取,无论 max_d 为何值,其模式树对同一数据块的后继数据块集合近似,因此预取效果相近。由于大的聚类距离可以使挖掘时间大幅度下降,因此在系统负载较小的情况下,建议采用较大的 max_d 值。

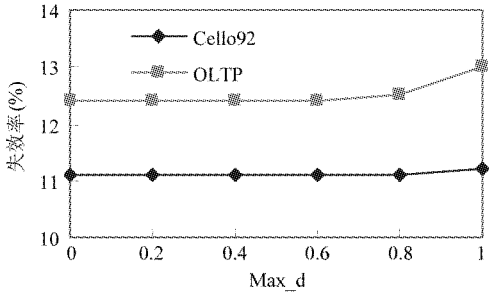


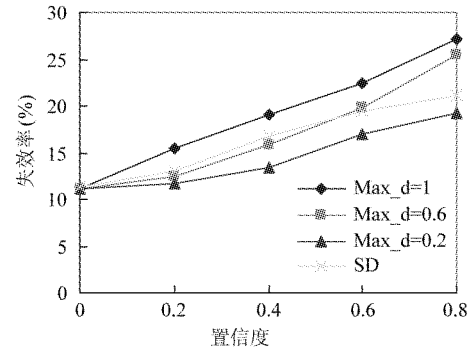
图 6 聚类距离对失效率的影响

3.4 Z-Miner 指导预取的精度

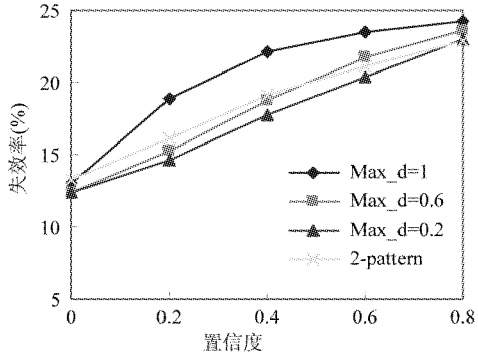
模式树保存了每个结点的频度,从 Z-Miner 的挖掘结果可以针对前缀计算其后出现某数据块的概率(相关度)。我们规定当相关度大于预先设置的置信度 conf 值时,才预取该数据块。通过 conf 可以控制预取的精度,从而测试聚类距离对预取精度的影响。

从图 7 可以看到,当 conf 增加时,max_d 较小的用例失效率上升比较平缓。这是因为小的聚类距离保存了有差异的后缀分支,可以提供更多的模式信息。当 conf 等于 0.8 时 Cello92 在不同 max_d 的情况下失效率相差最大,而 OLTP 的失效率差别缩小。这是由于 Cello92 的随机请求引起的请求干涉噪音较小,而 OLTP 中这类噪音较为明显。

在图 7 中我们将 Z-Miner 和 SD 进行了比较,当 conf 等于 0 时 Z-Miner 的预取效果和 SD 一致。随着预取精度的增加,较小的 max_d 获得的模式信息精度要明显高于 SD,这表明在合适的分支聚类距离下,Z-Miner 挖掘的多数据块关系可以提供比两数据块关系更为精确的预取指导。



(a) Cello92



(b) OLTP

图 7　聚类距离对预取精度的影响

4　结论

本论文分析了频繁访问模式挖掘中噪音的产生原因和表现类型,并提出了一种具有噪音过滤能力,适合存储系统的访问模式挖掘方法 Z-Miner。实验结果表明,通过对噪音的过滤可以用较小的代价,精确地挖掘多数据块语义关系,获得较好的优化效果。

我们未来的工作是,首先对模式分布特征进行分析,更好地了解应用的访问行为,其次,研究模式指导的缓存替换算法和磁盘数据分布,第三,研究实时的模式挖掘方法。

参考文献

[1] Coffman E G, Denning P J. Operating Systems Theory. New Jersey: Prentice Hall Professional Technical Reference, 1973. 0-331

[2] Li Z M, Chen Z F, 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. California: USENIX Association, 2004. 173-186

[3] Li Z M, Chen Z F, Zhou Y Y. Mining block correlations to improve storage performance. *ACM transactions on storage*, 2005, 1(2): 213-245

[4] Kuenning G H. The design of the SEER predictive caching scheme system. In: Proceedings of the Workshop on Mobile Computing Systems and Applications. New York: ACM, 1994

[5] Kuenning G H, Popek G J. Automated hoarding for mobile computers. In: Proceedings of the 16th Symposium on Operating Systems Principles. New York: ACM, 1997. 264-275

[6] Grimsrud K S, Archibald J K, Nelson B E. Multiple prefetch adaptive disk caching. *IEEE Transactions on Knowledge and Data Engineering*, 1993, 5(1): 88-103

[7] Hsu W W, Smith A J, Young H C. The automatic improvement of locality in storage systems. *ACM Transactions on Computer Systems*, 2005, 23(4): 424-473

[8] Han J, Pei J, Mortazavi-Asl B, et al. FreeSpan: frequent pattern-projected sequential pattern mining. In: Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. New York: ACM, 2000. 355-359

[9] Yan X, Han J, Afshar R. CloSpan: mining closed sequential patterns in large datasets. In: Proceedings of the 3rd SIAM International Conference on Data Mining, San Francisco, CA, USA, 2003. 166-177

[10] Pei J, Han J, Mortazavi-Asl B, et al. Mining sequential patterns by pattern-growth: the PrefixSpan approach. *IEEE Transactions on Knowledge and Data Engineering*, 2004, 16(10): 1-17

[11] MacQueen J B. Some methods for classification and analysis of multivariate observations. In: Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability. Berkeley, California: University of California Press, 1967. 281-297

Mining frequent access patterns in storage systems

Zhu Xudong[*,**], Bu Qingzhong[*,**], Ke Jian[*,**], Na Wenwu[*,**], Xu Lu[*]

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

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

Abstract

Based on the analysis of the effect mechanism of the noise, a major factor that lowers the efficiency of frequent access pattern mining and makes classic mining methods unacceptable for storage systems, this paper proposes a novel mining method ——Z-Miner. The Z-Miner employs a global-branch-cutting and branch-clustering approach for noise filtering. The simulation results under real workloads show that the prefetching directed by the Z-Miner could reduce the cache miss ratio by 40% ~ 66%, and the average response time by 26% ~ 66%. Compared with classic mining methods, the overhead of the Z-Miner is 1 to 2 orders of magnitude less, while the efficiency of the prefetching is two times more.

Key word: frequent access pattern, block correlations, sequential pattern mining, clustering, prefetching