一种改进大型存储系统离散小数据块读取性能的方法

赵 振 谢长生 李怀阳 吴 伟

(华中科技大学计算机科学与技术学院外存储国家重点实验室 武汉 430074)

摘 要 在大型存储系统中,改善离散小数据块读操作的性能已成为提高整个存储系统I/O性能的关键因素。针对这种情况,本文设计并实现了一种系统CBSS(correlative blocks speedup system)。该系统采用一种启发式算法,综合考虑数据访问时间的局部性和全局性,在文件系统和存储设备之间挖掘数据块的相关性,并根据取得的结果进行预取和数据块布局的物理调整,使整个存储系统性能能够平滑地不间断改善。实验结果显示,CBSS能有效改进系统的I/O性能,且不需要改变文件系统和存储设备的数据结构,具有广泛的适应性。

关键词 相关性,块设备,启发式

A Method of Improving the Performance of Continuously Reading Discrete Small Blocks in the Large Storage System

ZHAO Zhen XIE Chang-Sheng LI Huai-Yang WU Wei

(National Storage System Laboratory, College of Computer Science & Technology, Huazhong University of Science & Technology, Wuhan 430074)

Abstract In the large storage system, the operation of continuously reading discrete small blocks severely impacts the I/O performance. To solve this problem, this paper designs and implements a system, CBSS(correlative blocks speedup system), which implements precise prefetch and regulates the data distribution according the small blocks correlations, mined by a novel heuristic algorithm between the file system and block device. The system performance can be improved evenly and continuously without interruption and sudden state transitions. Furthermore, compared with other algorithms, this heuristic algorithm thinks about both the locality and the globality of the correlations. Through the experiments, it has been proved that CBSS and the algorithm are effective and the system I/O performance can be enhanced distinctly. Furthermore, the prototype can be used universally without modifying the file system and the storage devices.

Keywords Correlations, Block device, Heuristic

1 引言

当前,大多数存储系统的数据组织方式是固定的,数据分布不能适应外部访问特性的变化,而且系统整体性能呈现一种随使用而退化的趋势。例如磁盘性能随反复改写的次数增多而逐步下降,即众所周知的碎片问题。在单机上进行磁盘碎片整理已十分耗时,且整理时不可继续使用此硬盘。可想而知,在磁盘阵列上就更难使用整理程序。

现有的一些大型存储系统往往利用RAID1、RAID5进行数据存储,并根据负载特性进行数据分布的调整,如Auto Raid等。但这种调整经常是状态跃迁,即从一种状态转换为另一种状态,需要进行大量的数据迁移,这就需要以服务性能的暂时降低甚至服务的暂时停止为代价。而在电子商务等许多系统中,往往需要24*7的服务,这就需要系统结构的调整最好在不停止服务的前提下进行,同时能够做到连续平滑。

预取也是提高系统性能的重要手段之一。在特定的应用系统中,一定的预取方式能够取得很好的性能[1]。例如,在流媒体中对热点文件进行预读,可以提高系统的服务质量。但大多数预取方式只适合特定的访问模式,不具有广泛的适应性,特别是在大量读取离散小数据块的情况下。而这种访问模式下的系统是大量存在的[2]。

在大多数情况下,连续读取的离散小块数据是相互关联的。以'ls /home/'操作为例,首先要获得'/home/'的inode节点,然后要获取该目录的描述块。这两个数据块在语义上是相关的。又例如,有些应用一次读取小文件,或者读取被离散分布的单个文件,又或文件的跳跃访问,它们几次访问的小块数据都具有相关性。所以,获得这些离散分布的小块数据的相关性,具有广泛的用途,如数据预取、数据布局的调整等。特别是在系统使用一段时间后,系统数据碎片增多,性能逐渐下降,此时可以根据相关性信息采取一些措施,以提高系统性能。

获取数据块的相关性有多种方法,按照实现的层次区分,可以分为在应用层、文件系统层或者块设备层实现。

在应用层实现,往往须针对具体的应用。例如用于某数据库系统时,可取得丰富的数据块信息和数据的语义信息,同时预见数据块的被访问状况,进行精确的控制,从而加快查询速度。但同时,这种方式只能针对单个具体应用,广泛性受到了限制,而且增加了开发人员的负担。

在文件系统层实现,能够取得每个数据块的逻辑意义,可以采取针对性策略进行优化。例如,根据这些信息可以实现目录级或文件级的优化。但是采用这种方式需要对现有系统做大量修改,考虑到文件系统的多样性,该方式并不合适。

在文件系统层和设备层之间实现,在提高性能的同时,还具有良好的可扩展性和适用性,甚至可以将这种功能完全压缩到块设备驱动中。

挖掘相关性的算法可以采用多种方式,例如利用状态转换图等,但往往存在屏蔽无效信息不理想或不能综合考虑存储系统特性等问题。

根据以上分析,笔者在块设备层设计并实现了一套系统原型CBSS(correlative blocks speedup system),该系统利用

*)本课题受国家自然科学基金项目资助,编号:60273073。赵 振 博士研究生,主要研究方向为网络存储系统;谢长生 教授、博士生导师,主要研究方向为计算机系统结构、网络存储、超高密度、超高速存储技术等。

一种启发式算法，获取小离散数据块的相关性，并且根据这些相关性信息，对部分数据块进行预取并重新组织，以提高系统性能。实验证明：同现有系统相比，该套系统读写性能有大幅度提高。同时，该系统是以设备驱动模块的形式实现的，可作为一个外挂模块，不需要对操作系统内核进行修改，与具体文件系统无关，具有良好的通用性和扩展性。

2 CBSS 系统原型

客户端在连续读取多个离散小数据块时，需要给存储节点发送多个请求，同时存储节点中磁盘磁头需要做多次无序移动。如果将这些有一定访问频度、离散的并且相互关联的数据块进行预取处理并放置在相邻或连续的区域，可以提高系统的读性能，降低系统的响应时间。随着存储系统的使用，将不断大量出现这种类型的数据块，但有些关系将随着数据的删除、更改等操作而逐渐退化，不再属于此类型。在本系统中，在块设备层挖掘这类数据之间比较稳定的相关性集合。系统将根据这些信息，进行数据分布的平滑微量调整，使得系统性能尽可能保持在比较理想的状况，甚至相较于初始状态得到提高。

系统由两部分组成：BCRD(blocks correlations discover)和 CRKM(correlations keep module)，如图 1(1)所示。所有的 IO 请求被发送到 Storage proxy，然后直接发送到目标节点。同时将小块读取信息发送到 BCRD 中，这些信息将形成一个和时间相关的访问序列。而在 BCRD 中，进行块相关性的挖掘操作。当得到稳定的集合后，将该集合发送到目标节点中的 CRKM 中。在单个存储节点中，将依据 CRKM 中的信息进行优化。

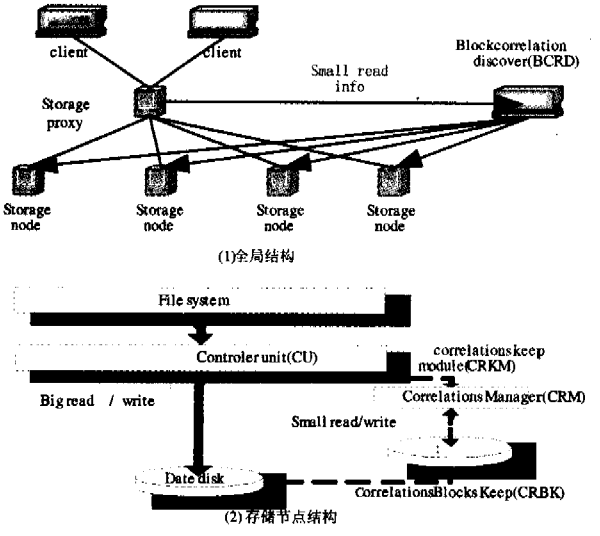

图 1 系统结构

CRKM 作为一个模块存在于驱动程序和文件系统之间，如图 1(2)所示。对于文件系统来说，它是透明的，如同一个块设备。它的主要功能是在控制单元 CU(Controller Unit)中负责请求的重组。关系被保存在 CRM(Correlations Manager)中，相关块被保存在 CRBK(Correlations Blocks Keep)中。CRBK 可以是磁盘的一部分，或者在内存中。如果在 CRKM 中，结合 DCD(disk Caching disk)技术[3]，还可提高系统的小写性能。

2.1 读取路径

数据的读取路径主要由请求大小决定。对于大的连续数据读取，控制模块将请求发送给数据盘 DD(data disk)，同时判断该范围内的数据块是否存在于 CRKM 中，并被修改。如果条件满足，则需要将该部分的请求发送给 CRKM。最终数据在 CU 中得到重新组织，并发返还给请求方。对于小块数据，首先检查它是否存在于 CRKM 之中。如果存在，则直接从此读取，否则从 DD 中读取。

当数据块是某个关系中的组成，并且首次从 DD 中读取后，该数据块同时被发送给 CRBK。这样，相关数据块便可逐渐转移到 CRKM 中。

2.2 写路径和数据的一致性

对于大块连续写入请求，CU 直接写入 DD 中。如果其中某些块在 CRKM 中已保存，将 CRM 中该块的有效位设为无效即可。对于小写数据，可以写至 DD，也可写至 CRKM，主要根据 CRKM 中 RBK 的结构。原因在于当 CRBK 或 DD 其中之一的数据被修改，而系统崩溃后，会导致数据不一致问题。如果 RBK 是非挥发性设备，例如磁盘或 NVRM，可以将需要修改的小写数据直接发至 CRKM；否则，需要同时修改 RDK 和 DD 中的数据，或者在修改 DD 的同时更改 CRM 中该数据块的有效位，这样才能保证系统恢复后的数据一致性。

2.3 数据的重组织

当 CRKM 中的数据达到一定数量且比较稳定时，可以考虑按照相关块的结构重新将这些数据组织到磁盘的空闲连续空间中，同时在 CRM 中保存这些数据块原始块号到新块号的映射关系，而 CDK 中的数据逐渐被新的相关块替换。当保存的磁盘中的这种类型数据达到一定的阈值时，可认为该存储节点结构的混乱程度太高，没有继续进行上述调整的意义，可通知系统，看是否需要做类似磁盘碎片整理的操作。

在存储节点处于空闲状态时，可以将那些在 CRKM 中被更改而 DD 中没有被更改的数据写回至 DD。这样，可以保证小读加速的同时，尽量不影响大的连续数据的读取。

2.4 storage proxy 的设计

在该系统，如果采用一个集中式的 storage proxy，由于每个请求都必须经过它，并且做请求信息的提取处理，该部分将成为系统瓶颈，同时有可能导致单点失效问题。如果 I/O 请求格式是固定并且规则的，例如在 SAN 环境下，或者存储节点是 iSCSI 类型设备，则整个存储系统 I/O 性能所受影响较低。同时，可以增加 storage proxy 的冗余度，以提高系统的可靠性。

将 storage proxy 功能分散也是一种选择，例如将它放置在每个客户端，这样将彻底消除它可能导致的系统瓶颈以及单点失效问题，但同时引起整个存储系统的可维护性降低，即每个客户端必须做单独配置。

3 挖掘相关性的算法

3.1 相关块的定义

当数据块 b_1 出现之后，b_2 出现的概率达到一定的值，并且这种情况多次出现，则称这两个块之间是相关的，记为 $b_1 \rightarrow b_2$。同时，关系也可能为 $b_1 b_2 \rightarrow b_3$，$b_1 \rightarrow b_2 b_3$。定义的左边称为前缀，右边称为后缀。

在块设备驱动层，除了数据块的块号和大小以外，任何关于该块的语义、逻辑信息都不得而知，相关块之间的联系完全依靠块号。现有的根据出现频率的挖掘算法，例如利用状态转换图记录信息或分裂树状图的算法，对少量信息的处理比较理想，但在大数据量的状况下，对无用信息的屏蔽效果不够理想，需要记录的信息过多。另外，存储系统中有些关系会逐渐退化，算法需要同时考虑这些关系的全局性和局部性，而这些算法没能综合考虑上述情况。

本文提出的这种启发式算法，既有效地屏蔽了无效信息，

又综合考虑了关系的局部性和全局性。

3.2 获取局部相关信息

对访问数据块进行一定时间 t 的记录后，形成一个记录集合 L。经过长期记录，形成 $L_0 \cdots L_n$ 序列。

对于每个集合 L_i，首先根据数据块的块号进行排序预处理。然后在 L 中查找出现次数超过 $\lfloor M/2 \rfloor$ 的集合 C_1。令 $M_1=C_1, C_2=M_1 \times M_1$，其中 C_2 是 M_1 的笛卡尔集。去除重复项，例如 aa、bb 等。接着判断 C_2 中的有效集合，例如 ab，该关系的 prefix_count 是 a 出现的次数。同时判断每个 a 出现位置之后的 s 步之内是否出现 b。如果出现，则 a→b 的 postfix count 增加 1。然后令 scale $=\frac{posfix\_count}{prefix\_count}$，如果它的 scale 小于 $\frac{Ms}{2}$，则该关系从 C_2 中剔除，最后得到 M_2。依次得到 $C_k=M_{k-1} \times C_1$，剔除不合适的信息，得到 M_k，直到 M_n 为空集。最终按照公式(1)合并各长度的关系集合。结果如表 1 所示。

$$T_m=M_2 \cup \cdots \cup M_{k-1} \cup M_k \tag{1}$$

表 1 获取局部信息

Serial number	prefix	postfix	Prefix count	Postfix count	scale
1	a	b	10	8	0.8
2	b	c	15	8	0.53
3	ab	c	8	4	0.5
…	…	…	…	…	…

(1)

5	a	bc	10	4	0.4

(2)

由于更希望根据少量的信息得到较多的信息，即通过短的前缀得到更长的后缀，例如，如果‘$abc \rightarrow d$’存在，那么‘$ab \rightarrow cd$’很可能存在，甚至‘$a \rightarrow bcd$’。根据定理(2)，得到推论(4)：

$$P(nm|m)=P(nm)/P(m) \tag{2}$$

$$P(k_1)>P(k_2)(k_1 \text{ 是 } k_2 \text{ 的前缀}) \tag{3}$$

$$P(n|k_1)< P(n|k_2) \tag{4}$$

如此可以得出结论，如果一个关系的 $P>m$，并且它的前缀大于等于 2，则可以尝试具有更短前缀的关系。同时，需要关注关系的合并，例如 $a \rightarrow bc$ 和 $a \rightarrow cb$，可以合并成一个关系。表 1(2)显示了这个过程。

整个流程完成后，得到了该时间段内的数据块的关系集合。非必要信息的屏蔽效果在开始两步，即得到 C_1 后，就基本完成。而且其后所操作的元素都属于 C_1，以其中的元素为种子，收敛速度也比较理想。

3.3 获取全局性相关信息

经过上一个步骤，对于每一个 t 得到一个集合 T。其中一些元素会随着时间逐渐退化，而算法的目标是取得长期的稳定关系。所以对于每一个时间 t，对以前的集合必须做一定的处理，使退化的关系逐渐被移出集合，而新的关系逐步加入。具体分下列 3 种情况：

$$T_{k1}=T_{k-1}T_k \tag{5}$$

$$T_{k2}=\overline{T}_{k-1} \cap T_k \tag{6}$$

$$T_{k3}=T_{k-1}\overline{T}_k \tag{7}$$

对于 T_{k1}，该集合元素的 prefix_count 和 postfix_count 是两个集合中该元素两项之和。对于 T_{k2} 中的各项，直接加入新集合之中即可。

而对于 T_{k3}，各项必须给予一定的惩罚。如果该元素的前缀出现在 T_k 中，则该项的 prefix_count 处理和 T_{k1} 一致，此时该项的 scale 已经被降低。而如果没有出现，则其 postfix count 减去一定的数值。这样，两种类型的关系分别得到惩罚，最终退化项可被逐渐从集合中剔除。

删除 3 个集合中不合要求的项。最终如公式(8)所示，得到新的 T_k。表 2 显示了该过程

$$T_k=T_{k1} \cup T_{k2} \cup T_{k3} \tag{8}$$

经过一段时期，将得到比较稳定的关系集合。在使用这一集合之前，可以移去一些冗余信息，例如($a \rightarrow b$、$b \rightarrow c$、$a \rightarrow c$)可以被($a \rightarrow bc$)取代，又例如($a \rightarrow b$，$b \rightarrow a$)可以表述为($a \rightarrow b$)。最终得到一个最小有效集。

表 2 获取全局信息

Serial number	prefix	postfix	Prefix count	Postfix count	scale
1	a	b	10	8	0.8
2	b	c	15	8	0.53
3	ab	c	8	3	0.375
…	…	…	…	…	…

(1) T_{k1}

1	a	b	4	2	0.5
2	d	c	15	8	0.53
…	…	…	…	…	…

(2) T_{k2}

1	a	b	10	10	0.715
2	b	c	15	7	0.487
3	d	c	15	8	0.53

(3) T_{k3}

3.4 合适的参数

在本算法中，影响系统性能的参数值主要是 t、M 和 Ms：不合适的 t 将影响关系的全局性和局部性，随着 t 的增加，计算负载也同步增长；M 和 Ms 将决定多少关系得以保留，且太小的选值将增大系统的负载。这些参数的合适与否将很大程度地影响系统性能。

同时，对上面提到的 T_{k3} 中的元素的惩罚力度，也需要仔细斟酌。因为惩罚力度将直接影响该集合中元素的退化速率，过大的惩罚将剔除相当一部分潜在的关系，相反则可能保留太多的失效信息。这些参数不宜选择固定的数值，应根据集合中元素的数量在一定的小范围内动态变迁。

4 性能测试

为了评估性能，笔者在 Linux 系统中实现了系统原型。CRKM 以驱动模块形式存在。

测试环境如下。两台机器的 CPU 为 Intel P4 1.7G，内存为 512MB，硬盘为 Maxton120G。这两台机器分别作为存储节点和 BCRD。Storage proxy 是集中式，并且和 BCRD 存在于同一台机器。

4.1 选择合适的 t 与 Ms

利用 Iometer for window 来测试不同的 t 及 Ms 对系统

的影响。结果如图 2 所示。

由图 2(1)可以看出，开始随着 t 的增加，响应时间逐渐下降，但是达到一定值之后，响应时间开始增加。原因是随着 t 的增加，可以获得更多的信息，但是信息退化速度将降低。换句话说，得到的信息可能已经退化，所以必须仔细地选择合适大小的 t。本系统中把它设置为在小范围内动态改变的数值，得到较好的效果。

由图 2(2)可以看出，系统的性能随着 Ms 的变化而变化，且对性能的影响亦相当大。太小的 Ms 屏蔽效果较低，将得到太多的信息，而相当部分关系的确定性不高；太大的 Ms 将丢失太多的信息。通过测试得到 60%是比较合适的选择。

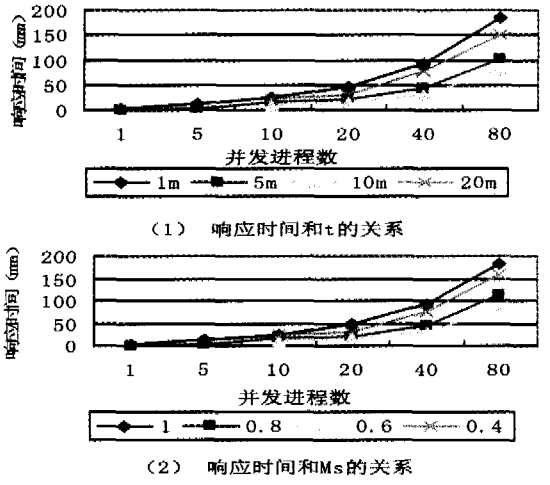

图 2 响应时间和 t、Ms 的关系

4.2 I/O 吞吐率测试

对于吞吐率，采用 bonnie^{++} 进测试，对比了加载和未加载该系统的情况，结果如表 3 所示。

表 3 I/O 吞吐率

测试类型		不加载	加载
Sequential Read 512M	Per(k/sec)	3228	3412
	Block(k/sec)	13925	13629
Sequential Write 512M	Per(k/sec)	3426	3398
	Block(k/sec)	15012	14465
	Rewrite(k/sec)	6701	6612
Sequential Create Files 16	Create(/sec)	1678	2078
	Read(/sec)	++++	++++
	Delete(/sec)	3345	3815
Random Create Files 16	Create(/sec)	1833	2418
	Read(/sec)	++++	++++
	Delete(/sec)	3559	4523

由测试数据可以看出，在顺序读写的过程中，由于加载模块的系统几乎没有做任何处理，系统性能几乎没有改变。但是在随机创建文件的测试中，系统性能提高了 40%，主要原因在于这些操作主要是针对元数据，往往是小块数据的读写，而且这些数据有较大的相关性。

4.3 峰值测试

峰值测试利用 HP 公司的 TRACE 文件来对加载和未加载该模块的系统进行对比。该 TRACE 文件是通过记录 HP 公司办公大楼内对服务器的 I/O 请求情况得来的，是一种典型的事务性处理的 I/O 请求记录。其中突发请求个数为 1024，每个请求的大小为 8kB，访问模式为并发访问。

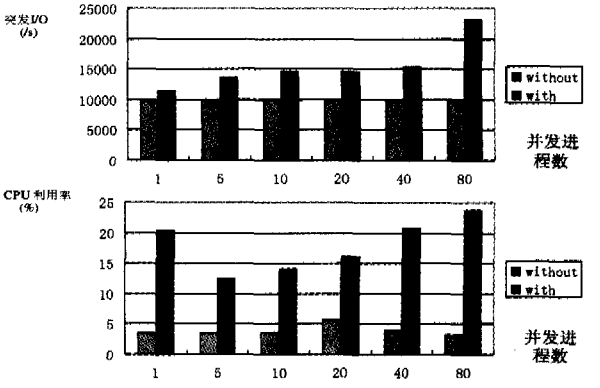

图 3 峰值测试

分析测试结果可知，加载了该模块的系统，请求响应时间降低了 30%左右，主要原因是系统对离散小数据块的加速作用。但同时整个系统的 CPU 被占用得更多，原因是相关关系匹配操作需要 CPU 开销。

从以上测试看出，该系统性能达到了设计的初衷。

总结 采用本文提出的启发式算法在块设备层挖掘离散块的相关性，并根据所得信息进行预取和数据分布的调整，提高了整个系统的 I/O 性能，使系统取得很好的可用性。同时，不需要特别的硬件支持，现有系统也几乎不需要更改，有着很强的适应性和通用性。但是，由于系统是在文件系统下层实现，对数据的逻辑意义毫无了解，不能实现单独目录或文件的优化。并且，在最坏情况下，由于没有别的取舍标准，可能得到一些干扰信息，例如频繁读取并更改的数据块。这些问题正是下一步工作所要解决的。

该系统在大型网络存储系统，如 san 环境的应用，以及改进本文提出的算法，使其适合面向对象的存储，也是下一阶段的努力方向。

参 考 文 献

1 Zhou Ke, Zhang Jiang-Ling. Cache prefetching adaptive policy based on access pattern. In: Proceedings of the first International Conference on machine leaning and cybernetics, Beijing, 4-5 November 2002

2 Brown A D, Mowry T C, Krieger O. Compiler-based I/O prefetching for out-of-core applications. ACM Transactions on Computer Systems, 2001, 19(2): 111～170

3 Hu Yiming, Yang Qing. DCD-Disk Caching Disk: A New Approach for Boosting I/O Performance. In: Proceedings of the 23rd international Symposium on Computer Architecture, May 1996. 169～178

4 Agrawal R, Srikant R. Mining sequential patterns. In Eleventh International Conference on Data Engineering, 1995

5 Gangory G R, Soules C A N. Soft Updates: A Solution to the Metadata Update Problem in File Systems. ACM Transactions on Computer Systems, May 2000

6 Soules C A N, Goodson G R, Strunk J D, et al. Metadata Efficiency in Versioning File Systems. 2nd USENIX Conference on File and Storage Technologies, San Francisco, CA, Mar 31 - Apr 2, 2003

7 Ayres J, Gehrke J E, Yiu T, et al. Sequential pattern mining using bitmaps. In: Proc. 2002 ACM SIGKDD Int Conf Knowledge Discovery in Databases(KDD'02), Edmonton, Canada, July 2002. 429～435

8 Schindler J, Griffin J, Lumb Cet al. Track-aligned extents: matching access patterns to disk drive characteristics. In: Proceedings of the First USENIX Conference on File and Storage Technologies, 2002

9 Seifert A, Scholl M H. A multi-version cache replacement and prefetching policy for hybrid data delivery environments. 28th International Conference on Very Large Data Bases(VLDB), 2002