高技术通讯 2012 年 第22 卷 第8 期:819 ~ 825

doi:10.3772/j.issn.1002-0470.2012.08.007

基于文件布局的文件碎化检测与评估研究①

朱亚东②*** 张景旺*** 张建刚* 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 100049)

摘 要 针对文件系统由于长期使用导致文件碎化,严重影响文件系统性能问题进行了文件碎化的检测与评估研究,提出并验证了一种新颖的碎化评估模型——碎化因子模型。该模型能根据文件的磁盘几何布局判断文件碎化情况,同时能较为精确地模拟出这种碎化情况对文件性能如顺序读的具体影响,为评估文件碎化的状况及其对文件性能的影响提供了一种更为新颖实用的策略。

关键词 文件碎化,顺序读性能,文件碎化因子

0 引 言

文件系统随着其长期使用,磁盘空间会形成众多的不连续片段(碎片),这种现象称为文件系统的"碎化"[1]。碎片增多会导致文件系统性能恶化,尤其是大型文件系统。因而对文件碎化的检测与评估具有重要意义。文件被分裂成多个碎片后,其读性能损失很大,原因在于读数据时的大量磁盘寻道。所以,为了避免碎片的产生和提高 I/O 性能,尽可能地分配连续资源块是非常重要的。因此,各种减少碎片产生的机制被引入,如文件系统 XFS 中的延迟分配[2],ext4 中的块保留[3]等。尽管如此,多文件的并发写仍然会导致碎化的产生,进而导致文件读性能的损失[4]。事实上,由于 I/O 密集型应用的流行,对文件系统 I/O 性能的要求更为严格。比如广电数字领域,对视频文件的播放延迟是有严格限制的,即当文件碎化导致的顺序读延迟超过一定限制时,都需要及时进行处理。当前,碎化的检测与评定已成为文件系统中的非常重要的一个环节[5-7]。如在 ext4 中,针对文件碎化情况,引入了文件的在线碎化整理特性[4],同时相应地提出了一个用户态的碎化检测工具,用以检测单个文件的碎化情况,并以此得出文件是否碎化及是否需要碎化整理的判定[8]。然而,ext4 采取的碎化检测方式是较粗粒度的,它只能大致给出文件是否需要进行碎化整理的建议,并不能反映文件的碎化状况对其性能造成的具体影响,如对文件的顺序读造成的性能损失等。因此,我们需要一种更为精确的细粒度的评测。比如能根据文件的碎化布局(layout)来量化出其对性能(顺序读)的具体影响。当前这个领域的相关研究还很少。本文主要针对单个文件碎化的检测评估进行研究,提出了一种新颖的基于文件布局的文件碎化检测评估模型——碎化因子模型,并对其有效性进行了验证与分析。

1 相关工作

国内外很多学者从各个角度展开了相关研究。Smith 采用 layout score 测量整个磁盘的碎化程度[1,9]。如 score 为 1,表示文件系统中的所有文件在磁盘上分布最佳,即任意指定文件的所有块在磁盘上都连续地相邻分布;若为 0,则表示文件的任何两块在磁盘上都不相邻。文献[10]从系统的角度研究了文件系统在长期使用(大量并发读写)后,碎化带来的平均性能损失。通过一系列研究分析,推导出公式 $L = 1/(2m - s)$,其中,m 为系统空闲空间比率,s 为平均单个文件占磁盘空间的比率,L 为文件的平均碎片个数。这样在文件系统的长期使用过程中,通过对系统参数(m 和 s)的分析,便可估

① 863 计划(2009AA01A403, 2009AA01Z139)资助项目。

② 男,1985 年生,博士生;研究方向:网络存储;联系人,E-mail: zhuyadong1985@126.com

(收稿日期:2011-06-21)

算出系统中文件的平均寻道次数(L),从而进一步研究由此带来的平均读写性能损失。ext4 引入 extent 这一概念来代替 ext2 - ext3 使用的传统块映射方式。"extent"是一个大的连续的物理块区域,它的引入加快了处理大文件的性能,减少了碎片。当块大小为 4KB 时,ext4 中的一个 extent 最大可以映射 128M 的连续物理存储空间[3]。尽管 ext4 中引入很多避免碎片产生的技术,但随着其长期使用,磁盘空间易形成众多的不连续片段,系统产生"老化"[1]。

Ext4 提供了专门的用户程序检测某个特定文件的碎化状况,通过研究相关源代码[8],ext4 的单个文件碎化检测总体上是基于文件的 extent 数目多少进行大致评定。首先计算 *best _ ext _ count*, 其目的为了描述最佳情形下某个文件的 extent 分布。计算方式为 $best\_ext\_count = \frac{blockcount - 1}{blocks\_per\_group} + 1$。事实上,由于 ext4 支持"FLEX _ BG"特性,ext4 上的计算方式为 $best\_ext\_count = \frac{blockcount - 1}{blocks\_per\_group} \times flex\_bg\_num + 1$, 其中 *blockcount* 为文件实际占据的 block 数目, *blocks _ per _ group* 为每个 group 的 blocks 数目, *flex _ bg _ num* 为 FLEX _ BG group 的大小。FLEX _ BG 这个特性放宽了对存储介质中每一个块组数据放在哪里的检查限制。它允许 bitmaps 或 inode tables[11]的分配在块组边界的外边。这也允许新的元数据分配方案,目的是提高性能及可扩展性。然后计算文件的碎化率即 *files _ ratio*, 计算公式为 $files\_ratio = (extents\_before\_defrag - best\_ext\_count)/file\_block\_count$, 其中 *extents _ before _ defrag* 为文件当前的 extent 数目, *file _ block _ count* 为文件实际占的 block 数目。最后计算相应的 *score*, 同时会引入一个阈值,与之比较,得出文件是否需要碎化整理的建议。计算公式为 $score = (files\_ratio) > 10?(80 + 20 \times \frac{files\_ratio}{100}):(8 \times files\_ratio)$。代码中给定的阈值为 55。

Ext4 采取的碎化检测方式是以文件 extent 数目多少为检测基础的,是一种较粗粒度的检测方式。从理论上讲,它只考虑了 extent 数目,而没有考虑 extent 之间的距离因素。事实上,即便 extent 数目较少而 extent 距离间隔较大时,长距离的磁盘寻道也会带来大量的读性能损失[10]。另一方面,它也只能大致给出文件是否需要进行碎化整理的建议,并不能反映文件的碎化状况对其性能造成的具体影响,如对文件顺序读造成的性能损失等。

2 碎化因子模型

本文提出的碎化检测模型的目标是,根据文件的 layout 分布来判断其碎化情况对性能(顺序读)的具体影响。具体来说,给定一个文件,首先从文件系统的元数据信息中获得其相应的 layout 信息,然后根据 layout 本身包含的文件相关的物理磁盘位置信息建立一个更加具体的碎化检测模型,并对碎化引起的读延迟时间进行计算和估计。

根据文件读写的逻辑和磁盘模型[12],文件读写时间可分为三部分:(1) 元数据操作时间,即根据文件逻辑块号确定其物理块号的时间。(2) 磁盘寻道时间。若此文件物理块不连续,则对文件连续读时,有一部分时间将用在磁盘寻道之上。(3) 数据读取时间。当磁头定位在指定位置之后,则开始真正的数据传输过程,这段时间即数据读取的时间。

单个文件的碎化包含元数据的碎化和数据分布的碎化。事实上,由于磁盘转速固定不变,数据传送速率可认为是常数,数据传送时间只与文件大小成正比。由此带来的读性能损失就由上面的元数据操作时间及磁盘寻道时间引起。下文会从这两方面出发进行测试分析。同时根据文献[10]的研究可知,文件碎化引起的性能损失主要耗费在磁盘寻道上,我们后续的测试将会证明这点。所以我们会重点考虑文件碎化引起的磁盘寻道时间。

在假设元数据已获得的情况下,根据磁盘的物理结构及其访问逻辑[12],数据访问时间分为排队时间、寻道时间、数据传送时间之和,由于我们考虑的是磁盘布局对访问时间的影响,因此不考虑排队时间,因为此项数据与磁盘布局无关。所以我们重点考虑的就是寻道时间。显然,磁盘寻道是由于文件磁盘布局间隔引起的,从而有公式

$$T = S_t + \sum_{i=1}^{M} S_i \tag{1}$$

其中 T 表示读取文件所花的总时间。S_t 是数据传送时间,与文件大小成正比, S_i 表示第 i 个间隔引起的寻道时间,而 M 是文件中间隔的总段数,也就是不连续的段的个数。当 $M = 0$ 时, T 取最小值 S_t, 即文件物理连续时,性能最佳。显然,由数据的碎化分布引起的文件性能损失可按式

$$F = T - S_t = \sum_{i=1}^{M} S_i \quad (2)$$

计算。我们将这个性能损失 F 命名为文件的“碎化因子”。根据文献[13,14],寻道时间的估算公式为

$$S \approx a + b\sqrt{d} \quad (d > 0) \quad (3)$$

其中 S 为寻道时间,a 为臂加速时间,b 为寻道因子,d 为寻道距离。本项目中也用此公式来近似计算寻道时间。将式(3)带入式(2)中,最终文件的“碎化因子”的表现公式为

$$F = aM + b\sum_{i=1}^{M} \sqrt{d_i} \quad (4)$$

其表示的物理意义为:对此文件顺序读,估计会比其理想性能(文件物理连续)多出的时间。其中 M 是文件中间隔的总段数,d_i 是第 i 个间隔的长度,a、b 是经验参数,跟文件分布所处的具体物理环境相关。具体取值可由文件的 layout 信息及实际测试的延迟时间数据进行最小二乘拟合获得。

由此我们在假设元数据已获得的前提下,推导出了文件的碎化布局与其引起的磁盘寻道时间的对应关系。从理论上很大程度地接近了我们的目标:根据文件的 layout 来判断其碎化情况对性能(顺序读)的具体影响。下面我们将对元数据碎化的情况进行测试分析,并对物理环境中 a, b 参数的实验取值进行拟合。

3 参数拟合

在后续的实验中,我们会首先考虑元数据碎化造成的性能损失;然后再通过实验测试,对式(4)进行验证分析;最后再对二者进行综合考虑,得到文件的碎化情况对其性能的具体影响。

3.1 性能测试环境

测试环境由一台元数据服务器(MDS)和一个存储服务器组成,然后在上面配置典型的 BWFS(蓝鲸分布式文件系统)。该文件系统的一个主要特点是元数据与数据的分离。之所以采用这样的测试系统,一方面由于海量存储的发展,这种类型的分布式文件系统逐渐成为一种趋势,具有很好的现实意义;另一方面,也简化了碎化分析的模型,不用考虑由于需要读元数据,而引入的数据分布与元数据分布之间的来回寻道。元数据服务器采用 64 位 Intel Xeon E5405,金士顿 1GB * 4,双 Broadcom 千兆网卡,Qlogic2460 HBA 卡,操作系统为 64 位的CentOS5.3。存储服务器为 SS4000,用多块 250GB SATA 磁盘组成的存储阵列。

后续测试的方案大致流程如下:

(1) 构造待测文件。满足不同测试的要求。

(2) 使缓存等失效。读大量不相关数据,丢弃本地 page cache。

(3) 测试文件顺序读性能。多次测量取平均值。

3.2 元数据碎化分析

对于元数据碎化带来的影响,需从两个方面进行分析。一方面需要考虑元数据操作的时间,即 GETBLKS(根据文件逻辑块号获得对应物理块号的操作)的耗时,另一方面需要分析不同元数据组织对整个顺序文件读性能的影响。文件在磁盘上的 layout 都是一样的,即定长的连续的磁盘块段,而相应的元数据组织不同,这种情况对读性能的影响需要考虑。

3.2.1 元数据操作时间

此处的元数据操作时间是用 GETBLKS 的 IOCTL(获取文件 layout 信息)执行时间来代表的。

由于三级间址结构的特点[15],可以认为给定文件大小后,三级间址的操作时间为常数,并且与文件是否碎化没有关系,因此这里只考虑 extent 形式[3]的文件。测试流程如下:

(1)将一个 64GB 的文件分为等长的 extent,设每个 extent 有 N 个块。

(2)失效 MDS 上的所有 page cache,从盘阵上读取与待测文件不相关的 4GB 数据,以使盘阵端的缓冲失效。

(3)对整个文件顺序调用 GETBLKS 的 IOCTL,每个块(4KB 大小)调用一次,即共调用 16M 次(16777216 次)。

(4)记录 3 的时间,之后再次执行步骤 3,并记录时间。

测试结果如图 1 所示。从图中可以看出,当块段长度为 1 时,第一次 GETBLKS 的性能比第二次 GETBLKS 性能差很多,说明这时读磁盘的影响还是很大的(第二次读基本上都可以在缓存中直接命中)。但随着块段长度到达 16 之后,第一次和第二次读的性能就基本上一样了。说明这时读磁盘的影响已经不明显。即主要时间花在陷入内核态以及查找 B + 树之上了。

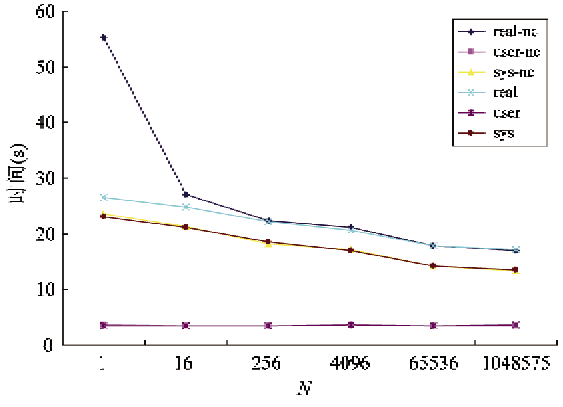


图 1 元数据操作时间

图示说明：▷ 数据是 3 次测试的平均值；▷ -nc 后缀是指 no cache，即做过缓存失效操作之后的第一次测试的数据；▷ 无此后缀的数据是第二次执行测试的时间数据（可认为元数据块基本都在 MDS 缓存中）；▷ real，user，sys 是 time 命令的输出，测量的是 16M 次 GETBLK IOCTL 的时间。

3.2.2 不同元数据组织的影响

此测试是为了测试不同的元数据组织对顺序文件读性能的影响。测试结果可反映不同元数据对性能的影响，可推算元数据操作在总时间中占的比例。测试方法如下：

（1）将一个 4GB 的文件分为等长的 extent，设每个 extent 有 N 个块。

（2）失效 MDS 上的所有 page cache，从盘阵上读任意其他的 8GB 数据，以使盘阵端的缓冲失效。

（3）对这个文件调用 dd if =/bwfs/test of =/dev/null bs =4096 count =1048576。

（4）记录 3 中 dd 显示的带宽，之后再次执行步骤 3，并记录 dd 显示的带宽。

其中，读盘阵裸盘的带宽为 76 ~80MB/s，平均约 78MB/s 左右，测试结果如图 2 所示。

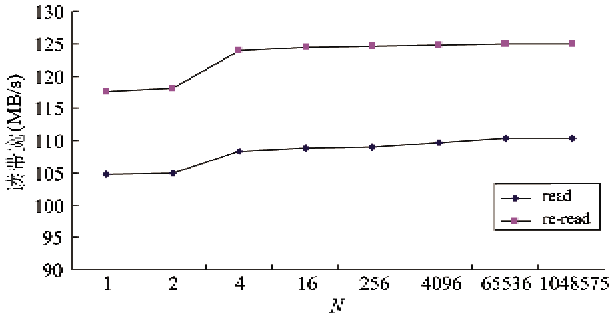


图 2 不同元数据组织下文件顺序读带宽

图示说明：▷ 数据是 3 次测试的平均值；▷ read 是失效 cache 之后第一次读文件的带宽；▷ re-read 是在 read 之后马上进行的第二次读的带宽。

如图 2 所示，当块段长度到 4 块之后，顺序读的性能已基本没有差别了。这说明当块段到一定长度后，元数据操作的时间对整体吞吐率的影响已经不明显了。而即使在元数据最多的情况下（即块段长度为 1），其测定的平均带宽分别为：read 104.8MB/s，re-read 118MB/s。而图示中对应的最好带宽（即块段长度为 1048575）是：read 110.4MB/s，re-read 125MB/s。显然，由于元数据太多，带来如下影响：

read：(110.4 - 104.8)/110.4 =0.0507≈5%

re-read：(125 - 118)/125 =0.056≈5.6%

所以即便在元数据最多的情况下，元数据操作带来的性能损失大约在 5% 左右。

3.2.3 元数据操作总结

由 3.2.1 和 3.2.2 的分析可得到如下结论：当块段长度大于 16 时，不必考虑元数据操作的影响；当块段长度小于 16 时，假设其为 N，则可简单估计其带来的性能影响幅度为：5% × 1/N，即

$$time\_penalty = \begin{cases} 0, & N \geqslant 16 \\ 5\% \times \dfrac{1}{N}, & N < 16 \end{cases}$$

显然，当 N = 1 时，性能影响幅度为 5%。

3.3 文件数据的碎化分析

文件的数据布局如图 3 所示。其中灰色的为文件的数据块（有 M 个），白色的为间隔（N 个）。测试这样的文件的读性能，就可以估计不同间隔对不同数据块大小的文件的读带宽的影响。测试方法如下：

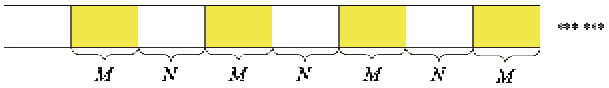


图 3 文件的数据布局

（1）按照图 3 构造待测文件，如 M =1，2，4，16，64，…，N =0，1，2，4，16，64，…文件大小为 4GB；

（2）从盘阵上读任意其他的 8GB 数据，以使盘阵端的缓冲失效；

（3）对这个文件调用 dd if =/bwfs/test of =/dev/null bs =4096 count =1048576；

（4）记录 3 中 dd 显示的带宽。

测试结果如图 4 所示。

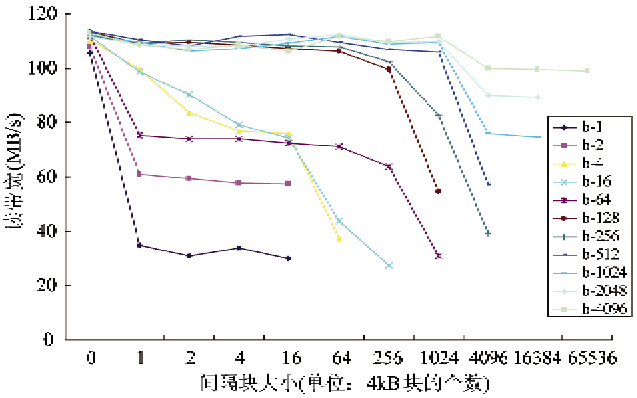


图 4　不同数据布局的文件顺序读带宽

图示说明:▷ 其中 b-数字中的数据表示了数据块的大小,比如 b-16 表示数据块为 16 个,即 M = 16。

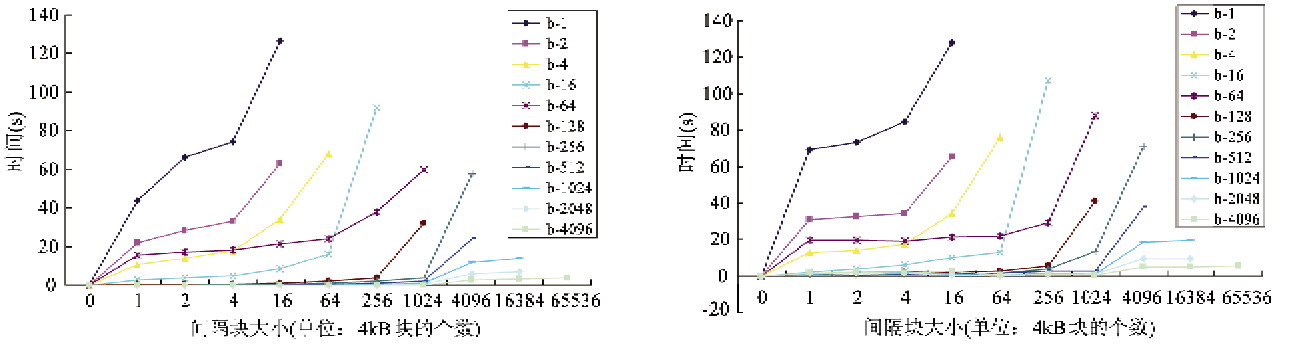
从图 4 可以看出,间隔对 4kB 基本块文件(b-1)的影响非常明显,使之性能下降超过50%。但是当文件基本块变大之后,间隔对性能的影响就不明显了。比如当文件块为 2048 个(8MB)时,文件性能随间隔的增长而下降的幅度已经明显变小了,只是下降到连续文件的 80% 的性能。

将实验数据换算成对应的读延迟时间,带入式(4)中进行最小二乘拟合,得到本次测试中 $a = 1.21074434 \times 10^{-5}$, $b = 2.94897522 \times 10^{-5}$。

4　验证与评估

4.1　拟合对比

我们采用与 3.1 节中相同的物理实验环境,对我们的碎化检测模型进行验证与评估,结果如图 5 所示。



(a) 采用拟合公式(4)计算出的延迟时间　　(b) 实际测试中的延迟时间

图 5　文件碎化检测模型的验证

图示说明:▷ 纵轴表示:对此文件顺序读,估计会比其最佳性能(物理上连续)多出的时间;▷ b-数字中的数据表示数据块的大小,比如 b-16 表示数据块为 16 个,即 $M = 16$;

显然,图 5 中的拟合曲线很好地反映了实际数据延迟的情况,从而表明文件的碎化因子计算公式(4)较好地描述了文件碎化对文件性能的影响。另外可以观察到,图中 b-1,b-2,b-4 拟合的数据略小于实际测试的数据,事实上由前面3.2.3节中的分析可知:当 b < 16 时,我们还需要考虑元数据碎化带来的性能影响,同时这个影响在 5% 以内。这也从另外的角度很好地验证了3.3节中分析的正确性。由此,我们便可根据文件的 layout 判断其碎化情况对文件性能(顺序读)的具体影响。

4.2　与 ext4 方法的实验对比评估

由相关工作中的分析知,ext4 采取的碎化检测方式是以文件 extent 数目多少为检测基础的,是一种较粗粒度的检测方式。从理论上讲,它只考虑了 extent 的数目,而没有考虑 extent 之间的距离因素。事实上,即便 extent 数目较少,而 extent 距离间隔较大时,长距离的磁盘寻道也会带来大量的读性能损失。由此,如果采用 ext4 的方法,就很容易带来“误判”。

取 3.3 章节中构造的待测文件如下:文件 A,平均块长 $M = 2$, 平均间隔 $N = 1$, 文件大小为 4GB;文件 B,平均块长 $M = 4$, 平均间隔 $N = 64$, 文件大小为 4GB。其中,每个基本块的大小为 4kB。文件 A 和 B 是具有同样大小的两个待测文件,对于文件 A,其包含的 extent 数目为 4GB/(4K × 2) = 512K 个;对于文件 B,其包含的 extent 数目为 4GB/(4K × 4) = 256K 个。

(1) 采用 ext4 的检测方法进行评估,显然,由于文件 A 包含更多的 extent 数目(512K > 256K),那么文件 A 的碎化程度要高于文件 B,即文件 A 比

文件 B 要更为碎化。

(2) 采用我们的碎化公式进行评估,从图 5(a)中观察可知,碎化对文件 A 带来的性能影响(延迟读时间)为21.9s,碎化对文件 B 带来的性能影响为68.1s。那么显然文件 A 的碎化状况要好于文件 B,即文件 B 比文件 A 更为碎化。

(3) 对 A、B 的实际顺序读情况进行测试,从图 5(b)观察可知,碎化对文件 A 带来的性能影响(延迟读)为 30.7s,碎化对文件 B 造成的性能影响为 76.0s。显然碎化对文件 B 造成的性能影响更为严重,文件 B 要比文件 A 更为碎化。这与我们根据碎化公式计算得出的结论是一致的,而根据 ext4 的检测方法则会得出恰恰相反的结论。

由上述实验对比发现,ext4 的检测方法并不能准确地反映一个文件当前的碎化状况。究其原因,则在于 ext4 的检测方法仅仅考虑了一个文件内部碎化片段的个数,而没有考虑这些片段之间的距离影响,以及由此带来的大量的磁头移动,而我们的方法较好地对二者进行了统一的权衡与考虑,能更为精确地对文件的碎化状况进行检测与评估。

4.3 实际应用

我们依此开发对应的碎化检测工具,以满足实际应用的需要。碎化检测工具的形式为指定待测目录,遍历目录下所有文件,列出碎化最为严重的文件的列表及其碎化程度。考虑到效率和效果问题,做如下两个限定:

(1)仅检测比给定大小更大的文件,因为小文件即使碎化严重,其总共的读写时间也十分有限,而且小文件访问中,page cache 会发挥比较大的作用。因此该工具忽略较小文件以加快检测速度。

(2)输出的碎化因子的大致含义是,对此文件顺序读,估计会比其理想性能(文件物理连续)多出的时间。因此碎化因子是偏向于大文件的,选择这样做的原因是认为对大文件整理碎片的收益更高,所以更倾向于整理大文件。

碎化因子的计算方法为第 2 节中的碎化因子计算公式(4),同时考虑 3.2.3 节中总结的元数据影响因素。主要算法描述如下:

根据碎化因子计算公式(4)求出碎化因子 F;
求出文件的平均块段长度 $AvgL$;
If ($AvgL < 16$)

$$F = F \times (1 + \frac{1}{AvgL} \times 5\%)$$

我们也进行了其执行效率测试。测试硬件环境与 3.1 节中所述相同。对于有 7T 空间(使用率超过 90%),28 万文件的 BWFS 文件系统,进行全面遍历花费了 22s,而单独的文件遍历一遍的时间为 19.5s,所以算法带来的开销很小,仅占时间总开销的 11% 左右。整个工具的执行速率是比较稳定的,即便在生产环境中也有很好的适用性。

5 结 论

本文提供了一种新颖的更为精确的碎化检测思路。对推导出的碎化因子的计算公式进行了严格的验证与评估,能很好地根据文件的 layout 来量化其碎化情况对文件性能(顺序读)的具体影响,为文件碎化的检测提供了应用级的依据。

当前“碎化因子”的概念只针对单个文件的碎化检测,并且它只描述了一个文件总体的碎化延迟时间。在后续的工作中,我们会尝试同时考虑连续文件自身顺序读的时间,然后将其与文件碎化延迟的时间相结合,从一个“相对延迟”的角度提供文件碎化的检测,具体策略仍需进一步的分析验证。同时,我们也会考虑将单个文件的“碎化因子”概念应用到“相关文件碎化检测”的场景[5],具体还需与特定的应用场景结合,进行理论与实验的详细论证,使碎化因子具有更好的扩展性。

参考文献

[1] Smith K, Seltzer M I. File system aging—increasing the relevance of file system benchmarks. In: Proceedings of the 1997 ACM International Conference on Measurement and Modeling of Computer Systems, Seattle, USA, 1997. 203-213

[2] Hellwig, Chrisoph. XFS for Linux. In: Proceedings of Linux 2003 Conference and Tutorials, Edinburgh, Scotland, July 2003

[3] “ext4.” http://zh.wikipedia.org/zh-cn/Ext4: Wikipedia, 2012

[4] Sato T. ext4 online defragmentation. In: Proceedings of the Linux Symposium 2007, Ottawa, Canada, 2007. 178-186

[5] Russinovich M, Solomon D. Windows XP: Kernel Improvements Create a More Robust, Powerful, and Scalable OS. http://msdn.microsoft.com/en-us/magazine/cc302206.aspx:MSDN, 2001

[6] Robb D. Defragmenting really speeds up Windows NT machines. *Spectrum*, *IEEE*, 2000, 37(9): 74-77

[7] Koester M, Kalte H, Porrmann M. Relocation and defragmentation for heterogeneous reconfigurable systems. In: Proceedings of the 2006 International Conference on Engineering of Reconfigurable Systems and Algorithms, Las Vegas, USA, 2006. 70-76

[8] "ext4 filesystem defragmenter." http://e2fsprogs.sourceforge.net/: Sourceforge, 2010

[9] Agrawal N, Arpaci-Dusseau A C, Remzi H. Generating realistic impressions for file-system benchmarking. In: Proceedings of the 7th Conference on File and Storage Technologies, San Francisco, USA, 2009. 125-138

[10] Giel de Nijs, Biesheuvel A, Dension A, et al. The effects of filesystem fragmentation. In :Proceedings of the Linux Symposium, Ottawa, Canada,2006. 193-208

[11] Steve D P. UNIX Filesystems: Evolution, Design and Implementation. New York: Wiley Publishing Inc, 2003

[12] Ruemmler C, Wilkes J. An introduction to disk drive modeling. *Computer*, 1994, 27(3):17-28

[13] 周可,张江陵,冯丹. 带 cache 的磁盘阵列 I/O 响应时间及吞吐量分析. 微电子学与计算机,2003, 8: 66-68

[14] 杨德志, 黄华, 张建刚等. 大容量、高性能、高扩展能力的蓝鲸分布式文件系统. 计算机研究与发展,2005, 6: 1028-1033

[15] Tweedie S, Theodore Y Ts'o. Planned extensions to the Linux ext2/3 file system. In: Proceedings of the 2002 USENIX Annual Technical Conference, Berkeley, USA, 2002. 235-244

Detection and estimation of file fragmentation based on file layout

Zhu Yadong[*,**], Zhang Jingwang[*,**], Zhang Jiangang[*], Xu Lu[*]

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

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

Abstract

The paper analyses the file fragmentation problem of a file system caused by its long use and the file fragmentation's serious influence on the performance of the file system, and then focuses on file fragmentation detection and estimation, and proposes a novel fragmentation measurement model: file fragmentation grade model. This model can accurately estimate the file fragmentation situation through file layout, and simulate the detailed effects of fragmentation on I/O performance such as sequential reading, which provides a novel practical policy for the detection of file fragmentation.

Key words: file fragmentation, sequential read performance, file fragmentation grade