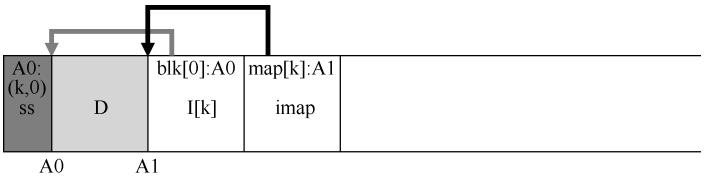

LFS 走了一些捷径，可以更有效地确定死活。例如，当文件被截断或删除时，LFS 会增加其版本号（version number），并在 imap 中记录新版本号。通过在磁盘上的段中记录版本号，LFS 可以简单地通过将磁盘版本号与 imap 中的版本号进行比较，跳过上述较长的检查，从而避免额外的读取。

43.11 策略问题：要清理哪些块，何时清理

在上述机制的基础上，LFS 必须包含一组策略，以确定何时清理以及哪些块值得清理。确定何时清理比较容易。要么是周期性的，要么是空闲时间，要么是因为磁盘已满。

确定清理哪些块更具挑战性，并且已成为许多研究论文的主题。在最初的 LFS 论文[RO91]中，作者描述了一种试图分离冷热段的方法。热段是经常覆盖内容的段。因此，对于这样的段，最好的策略是在清理之前等待很长时间，因为越来越多的块被覆盖（在新的段中），从而被释放以供使用。相比之下，冷段可能有一些死块，但其余的内容相对稳定。因此，作者得出结论，应该尽快清理冷段，延迟清理热段，并开发出一种完全符合要求的试探算法。但是，与大多数政策一样，这只是一种方法，当然并非“最佳”方法。后来的一些方法展示了如何做得更好[MR+97]。

43.12 崩溃恢复和日志

最后一个问题：如果系统在 LFS 写入磁盘时崩溃会发生什么？你可能还记得上一章讲的日志，在更新期间崩溃对于文件系统来说是棘手的，因此 LFS 也必须考虑这些问题。

在正常操作期间，LFS 将一些写入缓冲在段中，然后（当段已满或经过一段时间后），将段写入磁盘。LFS 在日志（log）中组织这些写入，即指向头部段和尾部段的检查点区域，并且每个段指向要写入的下一个段。LFS 还定期更新检查点区域（CR）。在这些操作期间都可能发生崩溃（写入段，写入 CR）。那么 LFS 在写入这些结构时如何处理崩溃？

我们先介绍第二种情况。为了确保 CR 更新以原子方式发生，LFS 实际上保留了两个 CR，每个位于磁盘的一端，并交替写入它们。当使用最新的指向 inode 映射和其他信息的指针更新 CR 时，LFS 还实现了一个谨慎的协议。具体来说，它首先写出一个头（带有时间戳），然后写出 CR 的主体，然后最后写出最后一部分（也带有时间戳）。如果系统在 CR 更新期间崩溃，LFS 可以通过查看一对不一致的时间戳来检测到这一点。LFS 将始终选择使用具有一致时间戳的最新 CR，从而实现 CR 的一致更新。

我们现在关注第一种情况。由于 LFS 每隔 30s 左右写入一次 CR，因此文件系统的最后一致快照可能很旧。因此，在重新启动时，LFS 可以通过简单地读取检查点区域、它指向的 imap 片段以及后续文件和目录，从而轻松地恢复。但是，最后许多秒的更新将会丢失。

为了改进这一点，LFS 尝试通过数据库社区中称为前滚（roll forward）的技术，重建其中许多段。基本思想是从最后一个检查点区域开始，找到日志的结尾（包含在 CR 中），然后使用它来读取下一个段，并查看其中是否有任何有效更新。如果有，LFS 会相应地更新文件系统，从而恢复自上一个检查点以来写入的大部分数据和元数据。有关详细信息，请参阅 Rosenblum 获奖论文[R92]。

43.13 小结

LFS 引入了一种更新磁盘的新方法。LFS 总是写入磁盘的未使用部分，然后通过清理回收旧空间，而不是在原来的位置覆盖文件。这种方法在数据库系统中称为影子分页（shadow paging）[L77]，在文件系统中有时称为写时复制（copy-on-write），可以实现高效写入，因为 LFS 可以将所有更新收集到内存的段中，然后按顺序一起写入。

这种方法的缺点是它会产生垃圾。旧数据的副本分散在整个磁盘中，如果想要为后续使用回收这样的空间，则必须定期清理旧的数据段。清理成为 LFS 争议的焦点，对清理成本的担忧[SS+95]可能限制了 LFS 开始对该领域的影响。然而，一些现代商业文件系统，包括 NetApp 的 WAFL [HLM94]、Sun 的 ZFS [B07]和 Linux btrfs [M07]，采用了类似的写时复制方法来写入磁盘，因此 LFS 的知识遗产继续存在于这些现代文件系统中。特别是，WAFL 通过将清理问题转化为特征来解决问题。通过快照（snapshots）提供旧版本的文件系统，用户可以在意外删除当前文件时，访问到旧文件。

提示：将缺点变成美德

每当你的系统存在根本缺点时，请看看是否可以将它转换为特征或有用的功能。NetApp 的 WAFL 对旧文件内容做到了这一点。通过提供旧版本，WAFL 不再需要担心清理，还因此提供了一个很酷的功能，在一个美妙的转折中消除了 LFS 的清理问题。系统中还有其他这样的例子吗？毫无疑问还有，但你必须自己去思考，因为本章的内容已经结束了。

参考资料

[B07]“ZFS: The Last Word in File Systems”Jeff Bonwick and Bill Moore
关于 ZFS 的幻灯片。遗憾的是，没有优秀的 ZFS 论文。

[HLM94]“File System Design for an NFS File Server Appliance”Dave Hitz, James Lau, Michael Malcolm
USENIX Spring ’94

WAFL 从 LFS 和 RAID 中获取了许多想法，并将其置于数十亿美元的存储公司 NetApp 的高速 NFS 设备中。

[L77]“Physical Integrity in a Large Segmented Database”
R. Lorie
ACM Transactions on Databases, 1977, Volume 2:1, pages 91-104
这里介绍了影子分页的最初想法。

[M07]“The Btrfs Filesystem”Chris Mason
September 2007
最近的一种写时复制 Linux 文件系统，其重要性和使用率逐渐增加。

[MJLF84]“A Fast File System for UNIX”
Marshall K. McKusick, William N. Joy, Sam J. Leffler, Robert S. Fabry ACM TOCS, August, 1984, Volume 2, Number 3
最初的 FFS 文件。有关详细信息，请参阅有关 FFS 的章节。

[MR+97]“Improving the Performance of Log-structured File Systems with Adaptive Methods”
Jeanna Neefe Matthews, Drew Roselli, Adam M. Costello,
Randolph Y. Wang, Thomas E. Anderson
SOSP 1997, pages 238-251, October, Saint Malo, France
最近的一篇论文，详细说明了 LFS 中更好的清理策略。

[M94]“A Better Update Policy”Jeffrey C. Mogul
USENIX ATC ’94, June 1994
在该文中，Mogul 发现，因为缓冲写入时间过长，然后集中发送到磁盘中，读取工作负载可能会受到影响。因此，他建议更频繁地以较小的批次发送写入。

[P98]“Hardware Technology Trends and Database Opportunities”David A. Patterson
ACM SIGMOD ’98 Keynote Address, Presented June 3, 1998, Seattle, Washington
关于计算机系统技术趋势的一系列幻灯片。也许 Patterson 很快就会制作另一个幻灯片。

[RO91]“Design and Implementation of the Log-structured File System”Mendel Rosenblum and John Ousterhout
SOSP ’91, Pacific Grove, CA, October 1991
关于 LFS 的原始 SOSP 论文，已被数百篇其他论文引用，并启发了许多真实系统。

[R92]“Design and Implementation of the Log-structured File System”Mendel Rosenblum
关于 LFS 的获奖学位论文，包含其他论文中缺失的许多细节。

[SS+95]“File system logging versus clustering: a performance comparison”
Margo Seltzer, Keith A. Smith, Hari Balakrishnan, Jacqueline Chang, Sara McMains, Venkata Padmanabhan
USENIX 1995 Technical Conference, New Orleans, Louisiana, 1995
该文显示 LFS 的性能有时会出现问题，特别是对于多次调用 fsync()的工作负载（例如数据库工作负载）。

该论文当时备受争议。

[SO90]“Write-Only Disk Caches”Jon A. Solworth, Cyril U. Orji
SIGMOD ’90, Atlantic City, New Jersey, May 1990
对写缓冲及其好处的早期研究。但是，缓冲太长时间可能会产生危害，详情请参阅 Mogul [M94]。

[Z+12]“De-indirection for Flash-based SSDs with Nameless Writes”
Yiying Zhang, Leo Prasath Arulraj, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau FAST ’13, San Jose, California, February 2013
我们的论文介绍了构建基于闪存的存储设备的新方法。由于 FTL（闪存转换层）通常以日志结构样式构建，因此在基于闪存的设备中会出现一些 LFS 中相同的问题。在这个例子中，它是递归更新问题，LFS 用 imap 巧妙地解决了这个问题。大多数 SSD 中存在类似的结构。

第 44 章　数据完整性和保护

除了我们迄今为止研究的文件系统中的基本进展，还有许多功能值得研究。本章将再次关注可靠性（之前在 RAID 章节中，已经研究过存储系统的可靠性）。具体来说，鉴于现代存储设备的不可靠性，文件系统或存储系统应如何确保数据安全？

该一般领域称为数据完整性（data integrity）或数据保护（data protection）。因此，我们现在将研究一些技术，确保放入存储系统的数据就是存储系统返回的数据。

关键问题：如何确保数据完整性

系统应如何确保写入存储的数据受到保护？需要什么技术？如何在低空间和时间开销的情况下提高这些技术的效率？

44.1　磁盘故障模式

正如你在关于 RAID 的章节中所了解到的，磁盘并不完美，并且可能会发生故障（有时）。在早期的 RAID 系统中，故障模型非常简单：要么整个磁盘都在工作，要么完全失败，而且检测到这种故障很简单。这种磁盘故障的故障—停止（fail-stop）模型使构建 RAID 相对简单[S90]。

你不知道的是现代磁盘展示的所有其他类型的故障模式。具体来说，正如 Bairavasundaram 等人的详细研究[B+07，B+08]，现代磁盘似乎大部分时间正常工作，但是无法成功访问一个或几个块。具体来说，两种类型的单块故障是常见的，值得考虑：潜在扇区错误（Latent-Sector Errors，LSE）和块讹误（block corruption）。接下来分别详细地讨论。

当磁盘扇区（或扇区组）以某种方式讹误时，会出现 LSE。例如，如果磁头由于某种原因接触到表面（磁头碰撞，head crash，在正常操作期间不应发生的情况），则可能会讹误表面，使得数据位不可读。宇宙射线也会导致数据位翻转，使内容不正确。幸运的是，驱动器使用磁盘内纠错码（Error Correcting Code，ECC）来确定块中的磁盘位是否良好，并且在某些情况下，修复它们。如果它们不好，并且驱动器没有足够的信息来修复错误，则在发出请求读取它们时，磁盘会返回错误。

还存在一些情况，磁盘块出现讹误（corrupt），但磁盘本身无法检测到。例如，有缺陷的磁盘固件可能会将块写入错误的位置。在这种情况下，磁盘 ECC 指示块内容很好，但是从客户端的角度来看，在随后访问时返回错误的块。类似地，当一个块通过有故障的总线从主机传输到磁盘时，它可能会讹误。由此产生的讹误数据会存入磁盘，但它不是客户所希望的。这些类型的故障特别隐蔽，因为它们是无声的故障（silent fault）。返回故障数据时，磁盘没有报告问题。

Prabhakaran 等人将这种更现代的磁盘故障视图描述为故障—部分（fail-partial）磁盘故障模型[P+05]。在此视图中，磁盘仍然可以完全失败（像传统的故障停止模型中的情况）。然而，磁盘也可以似乎正常工作，并有一个或多个块变得不可访问（即 LSE）或保存了错误的内容（即讹误）。因此，当访问看似工作的磁盘时，偶尔会在尝试读取或写入给定块时返回错误（非无声的部分故障），偶尔可能只是返回错误的数据（一个无声的部分错误）。

这两种类型的故障都有点罕见，但有多罕见？表 44.1 总结了 Bairavasundaram 的两项研究[B+07，B+08]的一些结果。

表 44.1 LSE 和块讹误的频率

项目	廉价	昂贵
LSEs	9.40%	1.40%
讹误	0.50%	0.05%

该表显示了在研究过程中至少出现一次 LSE 或块讹误的驱动器百分比（大约 3 年，超过 150 万个磁盘驱动器）。该表进一步将结果细分为“廉价”驱动器（通常为 SATA 驱动器）和“昂贵”驱动器（通常为 SCSI 或 FibreChannel）。如你所见，虽然购买更好的驱动器可以减少两种类型问题的频率（大约一个数量级），但它们的发生频率仍然足够高，你需要仔细考虑如何在存储系统中处理它们。

关于 LSE 的一些其他发现如下：

- 具有多个 LSE 的昂贵驱动器可能会像廉价驱动器一样产生附加错误。
- 对于大多数驱动器，第二年的年度错误率会增加。
- LSE 随磁盘大小增加。
- 大多数磁盘的 LSE 少于 50 个。
- 具有 LSE 的磁盘更有可能发生新增的 LSE。
- 存在显著的空间和时间局部性。
- 磁盘清理很有用（大多数 LSE 都是这样找到的）。

关于讹误的一些发现如下：

- 同一驱动器类别中不同驱动器型号的讹误机会差异很大。
- 老化效应因型号而异。
- 工作负载和磁盘大小对讹误几乎没有影响。
- 大多数具有讹误的磁盘只有少数讹误。
- 讹误不是与一个磁盘或 RAID 中的多个磁盘无关的。
- 存在空间局部性和一些时间局部性。
- 与 LSE 的相关性较弱。

要了解有关这些故障的更多信息，应该阅读原始论文[B+07，B+08]。但主要观点应该已经明确：如果真的希望建立一个可靠的存储系统，必须包括一些机制来检测和恢复 LSE 并阻止讹误。

44.2 处理潜在的扇区错误

鉴于这两种新的部分磁盘故障模式，我们现在应该尝试看看可以对它们做些什么。让我们首先解决两者中较为容易的问题，即潜在的扇区错误。

关键问题：如何处理潜在的扇区错误

存储系统应该如何处理潜在的扇区错误？需要多少额外的机制来处理这种形式的部分故障？

事实证明，潜在的扇区错误很容易处理，因为它们（根据定义）很容易被检测到。当存储系统尝试访问块，并且磁盘返回错误时，存储系统应该就用它具有的任何冗余机制，来返回正确的数据。例如，在镜像 RAID 中，系统应该访问备用副本。在基于奇偶校验的 RAID-4 或 RAID-5 系统中，系统应通过奇偶校验组中的其他块重建该块。因此，利用标准冗余机制，可以容易地恢复诸如 LSE 这样的容易检测到的问题。

多年来，LSE 的日益增长影响了 RAID 设计。当全盘故障和 LSE 接连发生时，RAID-4/5 系统会出现一个特别有趣的问题。具体来说，当整个磁盘发生故障时，RAID 会尝试读取奇偶校验组中的所有其他磁盘，并重新计算缺失值，来重建（reconstruct）磁盘（例如，在热备用磁盘上）。如果在重建期间，在任何一个其他磁盘上遇到 LSE，我们就会遇到问题：重建无法成功完成。

为了解决这个问题，一些系统增加了额外的冗余度。例如，NetApp 的 RAID-DP 相当于两个奇偶校验磁盘，而不是一个[C+04]。在重建期间发现 LSE 时，额外的校验盘有助于重建丢失的块。与往常一样，这有成本，因为为每个条带维护两个奇偶校验块的成本更高。但是，NetApp WAFL 文件系统的日志结构特性在许多情况下降低了成本[HLM94]。另外的成本是空间，需要额外的磁盘来存放第二个奇偶校验块。

44.3 检测讹误：校验和

现在让我们解决更具挑战性的问题，即数据讹误导致的无声故障。在出现讹误导致磁盘返回错误数据时，如何阻止用户获取错误数据？

关键问题：尽管有讹误，如何保护数据完整性

鉴于此类故障的无声性，存储系统可以做些什么来检测何时出现讹误？需要什么技术？如何有效地实现？

与潜在的扇区错误不同，检测讹误是一个关键问题。客户如何判断一个块坏了？一旦知道特定块是坏的，恢复就像以前一样：你需要有该块的其他副本（希望没有讹误！）。因此，我们将重点放在检测技术上。

现代存储系统用于保持数据完整性的主要机制称为校验和（checksum）。校验和就是一

个函数的结果，该函数以一块数据（例如 4KB 块）作为输入，并计算这段数据的函数，产生数据内容的小概要（比如 4 字节或 8 字节）。此摘要称为校验和。这种计算的目的在于，让系统将校验和与数据一起存储，然后在访问时确认数据的当前校验和与原始存储值匹配，从而检测数据是否以某种方式被破坏或改变。

提示：没有免费午餐

没有免费午餐这种事，或简称 TNSTAAFL，是一句古老的美国谚语，暗示当你似乎在免费获得某些东西时，实际上你可能会付出一些代价。以前，餐馆会向顾客宣传免费午餐，希望能吸引他们。只有当你进去时，才会意识到要获得“免费”午餐，必须购买一种或多种含酒精的饮料。

常见的校验和函数

许多不同的函数用于计算校验和，并且强度（即它们在保护数据完整性方面有多好）和速度（即它们能够以多快的速度计算）不同。系统中常见的权衡取决于此：通常，你获得的保护越多，成本就越高。没有免费午餐这种事。

有人使用一个简单的校验和函数，它基于异或（XOR）。使用基于 XOR 的校验和，只需对需要校验和的数据块的每个块进行异或运算，从而生成一个值，表示整个块的 XOR。

为了使这更具体，想象一下在一个 16 字节的块上计算一个 4 字节的校验和（这个块当然太小而不是真正的磁盘扇区或块，但它将用作示例）。十六进制的 16 个数据字节如下所示：

```
365e c4cd ba14 8a92 ecef 2c3a 40be f666
```

如果以二进制形式查看它们，会看到：

```
0011 0110 0101 1110    1100 0100 1100 1101
1011 1010 0001 0100    1000 1010 1001 0010
1110 1100 1110 1111    0010 1100 0011 1010
0100 0000 1011 1110    1111 0110 0110 0110
```

因为我们以每行 4 个字节为一组排列数据，所以很容易看出生成的校验和是什么。只需对每列执行 XOR 以获得最终的校验和值：

```
0010 0000 0001 1011    1001 0100 0000 0011
```

十六进制的结果是 0x201b9403。

XOR 是一个合理的校验和，但有其局限性。例如，如果每个校验和单元内相同位置的两个位发生变化，则校验和将不会检测到讹误。出于这个原因，人们研究了其他校验和函数。

另一个简单的校验和函数是加法。这种方法具有快速的优点。计算它只需要在每个数据块上执行二进制补码加法，忽略溢出。它可以检测到数据中的许多变化，但如果数据被移位，则不好。

稍微复杂的算法被称为 Fletcher 校验和（Fletcher checksum），命名基于（你可能会猜到）发明人 John G. Fletcher [F82]。它非常简单，涉及两个校验字节 s1 和 s2 的计算。具体来说，假设块 D 由字节 d1,…, dn 组成。s1 简单地定义如下：s1 = s1 + di mod 255（在所有 di 上计算）。s2 依次为：s2 = s2 + s1 mod 255（同样在所有 di 上）[F04]。已知 fletcher 校验和几乎

与 CRC（下面描述）一样强，可以检测所有单比特错误，所有双比特错误和大部分突发错误[F04]。

最后常用的校验和称为循环冗余校验（CRC）。虽然听起来很奇特，但基本想法很简单。假设你希望计算数据块 D 的校验和。你所做的只是将 D 视为一个大的二进制数（毕竟它只是一串位）并将其除以约定的值（k）。该除法的其余部分是 CRC 的值。事实证明，人们可以相当有效地实现这种二进制模运算，因此也可以在网络中普及 CRC。有关详细信息，请参见其他资料[M13]。

无论使用何种方法，很明显没有完美的校验和：两个具有不相同内容的数据块可能具有相同的校验和，这被称为碰撞（collision）。这个事实应该是直观的：毕竟，计算校验和会使某种很大的东西（例如，4KB）产生很小的摘要（例如，4 或 8 个字节）。在选择良好的校验和函数时，我们试图找到一种函数，能够在保持易于计算的同时，最小化碰撞机会。

校验和布局

既然你已经了解了如何计算校验和，接下来就分析如何在存储系统中使用校验和。我们必须解决的第一个问题是校验和的布局，即如何将校验和存储在磁盘上？

最基本的方法就是为每个磁盘扇区（或块）存储校验和。给定数据块 D，我们称该数据的校验和为 C（D）。因此，没有校验和，磁盘布局如下所示：

有了校验和，布局为每个块添加一个校验和：

因为校验和通常很小（例如，8 字节），并且磁盘只能以扇区大小的块（512 字节）或其倍数写入，所以出现的一个问题是如何实现上述布局。驱动器制造商采用的一种解决方案是使用 520 字节扇区格式化驱动器，每个扇区额外的 8 个字节可用于存储校验和。

在没有此类功能的磁盘中，文件系统必须找到一种方法来将打包的校验和存储到 512 字节的块中。一种可能性如下：

在该方案中，*n* 个校验和一起存储在一个扇区中，后跟 *n* 个数据块，接着是后 *n* 块的另一个校验和扇区，依此类推。该方案具有在所有磁盘上工作的优点，但效率较低。例如，如果文件系统想要覆盖块 D1，它必须读入包含 C（D1）的校验和扇区，更新其中的 C（D1），然后写出校验和扇区以及新的数据块 D1（因此，一次读取和两次写入）。前面的方法（每个扇区一个校验和）只执行一次写操作。

44.4 使用校验和

在确定了校验和布局后，现在可以实际了解如何使用校验和。读取块 D 时，客户端（即文件系统或存储控制器）也从磁盘 Cs（D）读取其校验和，这称为存储的校验和（stored checksum，因此有下标 Cs）。然后，客户端计算读取的块 D 上的校验和，这称为计算的校验和（computed checksum）Cc（D）。此时，客户端比较存储和计算的校验和。如果它们相等 [即 Cs（D）== Cc（D）]，数据很可能没有被破坏，因此可以安全地返回给用户。如果它们不匹配 [即 Cs（D）!= Cc（D）]，则表示数据自存储之后已经改变（因为存储的校验和反映了当时数据的值）。在这种情况下，存在讹误，校验和帮助我们检测到了。

发现了讹误，自然的问题是我们应该怎么做呢？如果存储系统有冗余副本，答案很简单：尝试使用它。如果存储系统没有此类副本，则可能的答案是返回错误。在任何一种情况下，都要意识到讹误检测不是神奇的子弹。如果没有其他方法来获取没有讹误的数据，那你就不走运了。

44.5 一个新问题：错误的写入

上述基本方案对一般情况的讹误块工作良好。但是，现代磁盘有几种不同的故障模式，需要不同的解决方案。

第一种感兴趣的失败模式称为“错误位置的写入（misdirected write）”。这出现在磁盘和 RAID 控制器中，它们正确地将数据写入磁盘，但位置错误。在单磁盘系统中，这意味着磁盘写入块 Dx 不是在地址 x（像期望那样），而是在地址 y（因此是“讹误的”Dy）。另外，在多磁盘系统中，控制器也可能将 Di，x 不是写入磁盘 i 的 x，而是写入另一磁盘 j。因此问题是：

关键问题：如何处理错误的写入

存储系统或磁盘控制器应该如何检测错误位置的写入？校验和需要哪些附加功能？

毫不奇怪，答案很简单：在每个校验和中添加更多信息。在这种情况下，添加物理标识符（Physical Identifier，物理 ID）非常有用。例如，如果存储的信息现在包含校验和 C（D）以及块的磁盘和扇区号，则客户端很容易确定块内是否存在正确的信息。具体来说，如果客户端正在读取磁盘 10 上的块 4（$D_{10,4}$），则存储的信息应包括该磁盘号和扇区偏移量，如下所示。如果信息不匹配，则发生了错误位置写入，并且现在检测到讹误。以下是在双磁盘系统上添加此信息的示例。注意，该图与之前的其他图一样，不是按比例绘制的，因为校验和通常很小（例如，8 个字节），而块则要大得多（例如，4KB 或更大）：

可以从磁盘格式看到，磁盘上现在有相当多的冗余：对于每个块，磁盘编号在每个块中重复，并且相关块的偏移量也保留在块本身旁边。但是，冗余信息的存在应该是不奇怪。冗余是错误检测（在这种情况下）和恢复（在其他情况下）的关键。一些额外的信息虽然不是完美磁盘所必需的，但可以帮助检测出现问题的情况。

44.6 最后一个问题：丢失的写入

遗憾的是，错误位置的写入并不是我们要解决的最后一个问题。具体来说，一些现代存储设备还有一个问题，称为丢失的写入（lost write）。当设备通知上层写入已完成，但事实上它从未持久，就会发生这种问题。因此，磁盘上留下的是该块的旧内容，而不是更新的新内容。

这里显而易见的问题是：上面做的所有校验和策略（例如，基本校验和或物理 ID），是否有助于检测丢失的写入？遗憾的是，答案是否定的：旧块很可能具有匹配的校验和，上面使用的物理 ID（磁盘号和块偏移）也是正确的。因此我们最后的问题：

关键问题：如何处理丢失的写入

存储系统或磁盘控制器应如何检测丢失的写入？校验和需要哪些附加功能？

有许多可能的解决方案有助于解决该问题[K+08]。一种经典方法[BS04]是执行写入验证（write verify），或写入后读取（read-after-write）。通过在写入后立即读回数据，系统可以确保数据确实到达磁盘表面。然而，这种方法非常慢，使完成写入所需的 I/O 数量翻了一番。

某些系统在系统的其他位置添加校验和，以检测丢失的写入。例如，Sun 的 Zettabyte 文件系统（ZFS）在文件系统的每个 inode 和间接块中，包含文件中每个块的校验和。因此，即使对数据块本身的写入丢失，inode 内的校验和也不会与旧数据匹配。只有当同时丢失对 inode 和数据的写入时，这样的方案才会失败，这是不太可能的情况（但也有可能发生！）。

44.7 擦净

经过所有这些讨论，你可能想知道：这些校验和何时实际得到检查？当然，在应用程序访问数据时会发生一些检查，但大多数数据很少被访问，因此将保持未检查状态。未经检查的数据对于可靠的存储系统来说是个问题，因为数据位衰减最终可能会影响特定数据的所有副本。

为了解决这个问题，许多系统利用各种形式的磁盘擦净（disk scrubbing）[K+08]。通过定期读取系统的每个块，并检查校验和是否仍然有效，磁盘系统可以减少某个数据项的所有副本都被破坏的可能性。典型的系统每晚或每周安排扫描。

44.8 校验和的开销

在结束之前，讨论一下使用校验和进行数据保护的一些开销。有两种不同的开销，在计算机系统中很常见：空间和时间。

空间开销有两种形式。第一种是磁盘（或其他存储介质）本身。每个存储的校验和占用磁盘空间，不能再用于用户数据。典型的比率可能是每 4KB 数据块的 8 字节校验和，磁盘空间开销为 0.19%。

第二种空间开销来自系统的内存。访问数据时，内存中必须有足够的空间用于校验和以及数据本身。但是，如果系统只是检查校验和，然后在完成后将其丢弃，则这种开销是短暂的，并不是很重要。只有将校验和保存在内存中（为了增加内存讹误防护级别[Z+13]），才能观察到这种小开销。

虽然空间开销很小，但校验和引起的时间开销可能非常明显。至少，CPU 必须计算每个块的校验和，包括存储数据时（确定存储的校验和的值），以及访问时（再次计算校验和，并将其与存储的校验和进行比较）。许多使用校验和的系统（包括网络栈）采用了一种降低 CPU 开销的方法，将数据复制和校验和组合成一个简化的活动。因为无论如何都需要拷贝（例如，将数据从内核页面缓存复制到用户缓冲区中），组合的复制/校验和可能非常有效。

除了 CPU 开销之外，一些校验和方案可能会导致外部 I/O 开销，特别是当校验和与数据分开存储时（因此需要额外的 I/O 来访问它们），以及后台擦净所需的所有额外 I/O。前者可以通过设计减少，后者影响有限，因为可以调整，也许通过控制何时进行这种擦净活动。半夜，当大多数（不是全部）努力工作的人们上床睡觉时，可能是进行这种擦净活动、增加存储系统健壮性的好时机。

44.9 小结

我们已经讨论了现代存储系统中的数据保护，重点是校验和的实现和使用。不同的校验和可以防止不同类型的故障。随着存储设备的发展，毫无疑问会出现新的故障模式。也许这种变化将迫使研究界和行业重新审视其中的一些基本方法，或发明全新的方法。时间会证明，或者不会。从这个角度来看，时间很有趣。

参考资料

[B+07]“An Analysis of Latent Sector Errors in Disk Drives”

Lakshmi N. Bairavasundaram, Garth R. Goodson, Shankar Pasupathy, Jiri Schindler SIGMETRICS ’07, San Diego, California, June 2007

一篇详细研究潜在扇区错误的论文。正如引文[B+08]所述，这是威斯康星大学与 NetApp 之间的合作。该

论文还获得了 Kenneth C. Sevcik 杰出学生论文奖。Sevcik 是一位了不起的研究者，也是一位太早过世的好人。为了向本书作者展示有可能从美国搬到加拿大并喜欢上这个地方，Sevcik 曾经站在餐馆中间唱过加拿大国歌。

[B+08]“An Analysis of Data Corruption in the Storage Stack”Lakshmi N. Bairavasundaram, Garth R. Goodson, Bianca Schroeder, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau
FAST ’08, San Jose, CA, February 2008
一篇真正详细研究磁盘讹误的论文，重点关注超过 150 万个驱动器 3 年内发生此类讹误的频率。Lakshmi 做这项工作时还是威斯康星大学的一名研究生，在我们的指导下，同时也与他在 NetApp 的同事合作，他有几个暑假都在 NetApp 做实习生。与业界合作可以带来更有趣、有实际意义的研究，这是一个很好的例子。

[BS04]“Commercial Fault Tolerance: A Tale of Two Systems”Wendy Bartlett, Lisa Spainhower
IEEE Transactions on Dependable and Secure Computing, Vol. 1, No. 1, January 2004
这是构建容错系统的经典之作，是对 IBM 和 Tandem 最新技术的完美概述。对该领域感兴趣的人应该读这篇文章。

[C+04]“Row-Diagonal Parity for Double Disk Failure Correction”
P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, S. Sankar FAST ’04, San Jose, CA, February 2004
关于额外冗余如何帮助解决组合的全磁盘故障/部分磁盘故障问题的早期文章。这也是如何将更多理论工作与实践相结合的一个很好的例子。

[F04]“Checksums and Error Control”Peter M. Fenwick
一个非常简单的校验和教程，免费提供给你阅读。

[F82]“An Arithmetic Checksum for Serial Transmissions”John G. Fletcher
IEEE Transactions on Communication, Vol. 30, No. 1, January 1982
Fletcher 的原创工作，内容关于以他命名的校验和。当然，他并没有把它称为 Fletcher 校验和，实际上他没有把它称为任何东西，因此用发明者的名字来命名它就变得很自然了。所以不要因这个看似自夸的名称而责怪老 Fletcher。这个轶事可能会让你想起 Rubik 和他的立方体（魔方）。Rubik 从未称它为“Rubik 立方体”。实际上，他只是称之为“我的立方体”。

[HLM94]“File System Design for an NFS File Server Appliance”Dave Hitz, James Lau, Michael Malcolm
USENIX Spring ’94
这篇开创性的论文描述了 NetApp 核心的思想和产品。基于该系统，NetApp 已经发展成为一家价值数十亿美元的存储公司。如果你有兴趣了解更多有关其成立的信息，请阅读 Hitz 的自传《How to Castrate a Bull: Unexpected Lessons on Risk, Growth, and Success in Business》（如何阉割公牛：商业风险，成长和成功的意外教训）”（这是真实的标题，不是开玩笑）。你本以为进入计算机科学领域可以避免“阉割公牛”吧？

[K+08]“Parity Lost and Parity Regained”
Andrew Krioukov, Lakshmi N. Bairavasundaram, Garth R. Goodson, Kiran Srinivasan, Randy Thelen, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau

FAST ’08, San Jose, CA, February 2008
我们的这项工作与 NetApp 的同事一起探讨了不同的校验和方案如何在保护数据方面起作用（或不起作用）。我们揭示了当前保护策略中的一些有趣缺陷，其中一些已导致商业产品的修复。

[M13]“Cyclic Redundancy Checks”Author Unknown
不确定是谁写的，但这是一个非常简洁明了的 CRC 说明。事实证明，互联网充满了信息。

[P+05]“IRON File Systems”
Vijayan Prabhakaran, Lakshmi N. Bairavasundaram, Nitin Agrawal, Haryadi S. Gunawi, An- drea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau
SOSP ’05, Brighton, England, October 2005
我们关于磁盘如何具有部分故障模式的论文，其中包括对 Linux ext3 和 Windows NTFS 等文件系统如何对此类故障作出反应的详细研究。事实证明，相当糟糕！ 我们在这项工作中发现了许多错误、设计缺陷和其他奇怪之处。其中一些已反馈到 Linux 社区，从而有助于产生一些新的更强大的文件系统来存储数据。

[RO91]“Design and Implementation of the Log-structured File System”Mendel Rosenblum and John Ousterhout
SOSP ’91, Pacific Grove, CA, October 1991
一篇关于如何提高文件系统写入性能的开创性论文。

[S90]“Implementing Fault-Tolerant Services Using The State Machine Approach: A Tutorial”Fred B. Schneider
ACM Surveys, Vol. 22, No. 4, December 1990
这篇经典论文主要讨论如何构建容错服务，其中包含了许多术语的基本定义。从事构建分布式系统的人应该读一读这篇论文。

[Z+13]“Zettabyte Reliability with Flexible End-to-end Data Integrity”
Yupu Zhang, Daniel S. Myers, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau MSST ’13, Long Beach, California, May 2013
这是我们自己的工作，将数据保护添加到系统的页面缓存中，以防止内存讹误和磁盘讹误。

第 45 章　关于持久的总结对话

学生：哇，文件系统看起来很有趣，但很复杂。

教授：这就是我和我妻子从事这个领域研究的原因。

学生：坚持下去。你是写这本书的教授之一吗？我认为我们都只是虚构的，用来总结一些要点，并可能在操作系统的研究中增加一点点轻松气氛。

教授：呃……呃……也许吧。不关你的事！你认为是谁写的这些东西？（叹气）无论如何，让我们继续吧：你学到了什么？

学生：嗯，我认为我掌握了一个要点，即长期（持久）管理数据比管理非持久数据（如内存中的内容）要困难得多。毕竟，如果你的机器崩溃，那么内存内容就会消失！但文件系统中的东西需要永远存在。

教授：好吧，我的朋友 Kevin Hultquist 曾经说过，“永远是一段很长的时间”。当时他在谈论塑料高尔夫球座，对于大多数文件系统中的垃圾来说，尤其如此。

学生：嗯，你知道我的意思！至少很长一段时间。即使简单的事情，例如更新持久存储设备，也很复杂，因为你必须关心如果崩溃会发生什么。恢复，这是我们在虚拟化内存时从未想过的东西，现在是一件大事！

教授：太对了。对持久存储的更新向来是，并且一直是一个有趣且有挑战性的问题。

学生：我还学习了磁盘调度等很酷的东西，以及 RAID、校验和等数据保护技术。那些内容很酷。

教授：我也喜欢这些话题。但是，如果你真的深入进去，可能需要一些数学。如果你想伤脑筋，请查看一些最新的擦除代码。

学生：我马上就去。

教授：（皱眉）我觉得你是在讽刺。那么，你还喜欢什么？

学生：我也很喜欢构建有技术含量的系统的所有想法，比如 FFS 和 LFS。漂亮！磁盘意识似乎很酷。但是，有了 Flash 和所有最新技术，它还重要吗？

教授：好问题！这提醒我开始写 Flash 的章节……（自己草草写下一些笔记）……但是，就算使用 Flash，所有这些东西仍然相关，令人惊讶。例如，Flash 转换层（FTL）在内部使用日志结构，以提高基于闪存的 SSD 的性能和可靠性。考虑局部性总是有用的。因此，尽管技术可能正在发生变化，但我们研究的许多想法至少在一段时间内仍将继续有用。

学生：那很好。我刚花了这么多时间来学习它，不希望它完全没用！

教授：教授不会那样对你，对吗？