摘 要

近几年来，随着计算机软硬件的多元化和用户数据规模的急速增长，数据存储的安全性和可靠性已成为亟待解决的重要问题。数据错误类型的增多，错误数据造成损失的增大，单一原始的数据保护技术已经不能满足现有数据存储的保护和恢复要求。因此，迫切需要为文件系统设计有效的复合型数据保护和纠错技术。然而，由于受存储系统层次、文件系统之间的差异和数据管理复杂等因素的影响，致使数据存储的可靠性与一致性一直难以保证。

麒麟加密文件系统是麒麟操作系统中开发的文件系统，它在内核层实现数据存储加密保护，能有效防止直接对磁盘进行物理访问所带来的安全问题，但是，目前，该系统在文件数据块自动纠错方面做得还不够，有待进一步完善。本文首先对两种典型文件系统所采用的数据纠错技术进行了分析和比较，在此基础上结合麒麟加密文件系统的特点，分析了其中数据纠错的基本需求与目标，为数据纠错的设计与实现提供了理论基础。

为了达到全面保护存储数据的目的，本文提出了麒麟加密文件系统数据纠错软件框架结构，其中包含三个功能模块，分别是数据检测模块，数据恢复模块和快照恢复模块。它们从不同方面来增强文件系统存储的可靠性，其中数据检测模块能够发现文件系统中的数据错误，然后通过数据恢复模块对错误数据做纠错处理。快照恢复模块则使用户误删，误改的数据恢复成为可能。在纠错系统框架的基础上，本文研究了框架内各模块所采用的数据纠错技术，分析了其中的不足。重点介绍了 Checksum 技术，奇偶校验技术和快照技术。在增强型 Checksum 技术中，主要针对现有 Checksum 技术的实施方法进行改进，将 Checksum 值的存储和数据块的地址信息绑定，使得 Checksum 值包含有更多的数据信息，进而使数据存储的安全性进一步提高。对于奇偶校验技术，本文为文件系统设计了基于文件数据块的奇偶校验技术，每一个文件都将与一个奇偶校验块相关联，从而能够对文件中错误的数据块进行恢复。在快照技术中，文件系统对修改和未修改的快照数据块作出了拷贝数据和记录引用的不同处理，有效降低了快照的开销。之后，面对具体的加密文件系统现状，本文设计了相关纠错技术的实施方案和实现方法。最后，本文在麒麟加密文件系统上实现了相关的数据纠错技术，并且对文件系统进行了性能测试，验证了数据纠错系统的有效性和可用性。

主题词：数据纠错，可靠性，文件系统

ABSTRACT

In recent years, with the increase in the diversity of hardware and software and the rapid growth of user's data, the data storage security and reliability has become an important problem which needs to be resolved urgently. But with the increasing of data corruptions and the damages caused by it, a single data protection technology has been unable to meet existing data storage protection and recovery requirements. Therefore, it has practical significance to design and implementation of a complex data protection and correction technology for the file system. However, due to the hierarch of storage systems, the different between file systems and complicated data management, it is difficult to ensure reliability and consistency in data storage.

Kylin Encrypting File System is developed in the Kylin operating system file system, which protects the stored data by encrypting with the implementations in kernel. In this paper, we first made a analysis and comparison between two kinds of typical file system's data correction techniques, on this basis, combining the characteristics of Kirin Encrypting File System, we analyzed the basic needs and goals for data correction system which provides a theoretical basis design and implementation for the data correction of the system.

In order to achieve the purpose of fully protection of stored data, this paper presents Kylin Encrypting File System Data correction software framework, which includes three functional modules, respectively, data detection module, data recovery module and snapshot recovery module. They enhance the file system storage reliability from different aspects, the data detection module can detect any read and write errors in the file system, and then through the data recovery module, file system can correct the bad data. Snapshot recovery module made it possible to recover the wrong data which were made by user's mistakes. Based on the design of data correction system framework, the paper analysis the data correction techniques which are used by the modules within the framework, and highlighting the Checksum technology, parity technology and snapshot technologies. In the reinforced Checksum, we improved the implementation of the existing methods. By binding the Checksum storage with the data block address information, we makes the Checksum value contains more data information, thereby making the data storage further more reliable. Checksum technology aimed at the implementation of the existing methods of improving the value of the Checksum storage and data block address information binding, makes the Checksum value contains more data and information, thereby enabling the security of data storage further improved. For the parity techniques, this paper design a parity techniques based on file data block. One parity block is allocated for each file. And thus the wrong data block in file can be recovery. In the snapshot technology, the file system

make the different treatments between modified and unmodified data block which reducing the overhead effectively. Then, this paper design and implement the related technologies on Kylin Encrypting File System. Finally, we make a testing of the file system performance and function to verify the validity and availability of the data correction system.

Key Words: Data Recovery, Reliability, File System.

表 目 录

图 目 录

独 创 性 声 明

本人声明所呈交的学位论文是我本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表和撰写过的研究成果，也不包含为获得国防科学技术大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

学位论文题目：麒麟加密文件系统数据块纠错技术的研究与实现

学位论文作者签名：郑思 日期：2009年12月30日

学位论文版权使用授权书

本人完全了解国防科学技术大学有关保留、使用学位论文的规定。本人授权国防科学技术大学可以保留并向国家有关部门或机构送交论文的复印件和电子文档，允许论文被查阅和借阅；可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密学位论文在解密后适用本授权书。）

学位论文题目：麒麟加密文件系统数据块纠错技术的研究与实现

学位论文作者签名：郑思 日期：2009年12月30日

作者指导教师签名： 日期：09年12月31日

第一章 绪论

1.1 课题研究背景

随着信息技术的不断发展，计算机越来越多地参与到人们的工作与生活中，在社会和生活中扮演着日益重要的角色。越来越多的企业、商家、政府机关和个人通过计算机来获取信息、处理信息，同时将自己重要的信息以数据文件的形式保存在计算机中。计算机或者软件系统不可避免的会出现差错从而导致数据丢失，在这种情况下，如何迅速而正确地进行数据恢复也就成了至关重要的问题。而如果没有相应的技术来支持数据存储保护和错误数据恢复，那么文件系统将无法保证数据存储的安全性，从而导致用户遭受巨大的损失[2]。这就使得数据纠错技术不论对个人、企业还是国家都显得日益重要起来。根据近期的一项研究表明，数据破坏平均成本2008年达到了每份记录202美元，相对于2007年上升了2.5%。这项研究是由Ponemon和PGP做出的，Ponemon是一家隐私权和数据保护的研究小组，PGP则是数据加密技术供应商。这项调查基于43个组织的实际数据破坏综合而成。根据该报告，每家公司每次遭遇破坏的总平均成本超过660万美元，这相较2007年的630万美元和2006年的470万美元进一步增高。43家组织中的最高总成本为3200万美元。平均每记录202美元的成本，其中有139美元归因于数据丢失导致的商务问题。其相较于总额比例达到了创纪录的69%，2007年为67%，2006年为54%。用户对数据安全失去了信心，并开始采取其它措施。

1.1.1 数据安全存储的现状

安全可靠的存储和访问数据应当是每一个文件系统所重点关注的方面，不幸的是伴随着软硬件复杂程度的提升，由于各种原因，现在的文件系统往往无法完全保证用户存储数据的一致性，而当这些错误的数据一旦交给上层的应用后，会对用户数据造成巨大的损失和破坏。比如，在读取某一文件inode的位图时即使出现很小的一位错误也会造成文件系统重要的数据破坏。特别是在当前，由于用户存储的数据量越来越大，所以要求文件系统所能提供的数据的一致性[2]和安全性的能力也越来越高。来自Identity Theft Resource Center的一个案例报告显示：和2007年相比，2008年数据破坏增长了47%。在这些破坏中，20.7%的事故与在笔记本或磁带上"迁移数据"相关，41%的事故是来自于黑客，内部人员和经销商破坏。而这些对数据的破坏统称为数据损坏[1][5]（Data Corruption），也就是计算机数据由于一些软硬件的原因而导致的损坏。有关数据存储的危险不仅仅包括了基于计算机的问题，例如病毒和硬件或者软件的、不兼容、缺陷或崩溃，还包括了环境的威

胁，例如电源消耗、灰尘、水和极端的温度。为了保护数据免于损坏，系统应该不仅确保只有经过授权的用户才能访问到数据，并且也要保护物理环境是安全的，以防止环境危险。在数据出现了不一致现象后，即出现了正确数据的丢失。提及数据丢失，许多人会立刻想到误删除、误格式化，而现实当中这些单纯的数据灾难正逐渐变得越来越少，取而代之是复杂的、叠加的复合型数据灾难。比如，删除、格式化、重分区、格式转换、后期覆盖等极为复杂的情况交织一起。而且，随着使用年限的增加，硬盘发生硬损伤的比例正在逐年提高，随着磁盘存储密度的逐年增加，硬盘制造精密度的逐年提高，发生硬故障从而导致数据丢失的硬盘数据恢复难度也在逐年增大。虽然数据恢复技术与水平也在逐年提升，但如今硬盘硬故障数据恢复的整体成功率，却并没有得到相应明显的提升，这是如今数据恢复领域不争的事实[4]。

目前造成数据被破坏的原因有很多，主要有以下三类：

（1）硬件和软件错误。

由于日益增长的磁盘技术复杂性，在硬件的每一个部分都有可能出错，这样使得硬件错误变得越来越频繁，比如 misdirect write 和 silent data corruption，这种硬件错误一般不易被察觉。而同时软件也可能造成数据的错误，比如硬件驱动的错误和操作系统本身存在的错误而导致了存储数据的破坏，从而造成了数据的不可访问或不一致的情况。

（2）恶意攻击。

如果数据没有被很好的安全机制所保护，网络中存在的攻击者就可能取得非法的数据访问权限，进而对存储数据进行恶意修改，这样将对用户造成巨大损失。

（3）由于用户疏忽造成的错误操作。

用户自身由于疏忽产生的操作错误也可能在应用层上造成数据的错误，这些数据虽然是合法的，但却不是用户所希望的。例如无意间的删除数据和修改了正确数据而造成了重要数据的丢失都是可能发生的。

正是由于上述出现的各种数据错误，所以用户对数据存储一致性和可靠性的要求越来越高，对数据存储保护和恢复技术的要求也越来越迫切。

1.1.2 文件系统对安全存储的保护

当前，为了保护数据存储的安全性和一致性，文件系统也采取了许多相关技术。这些保护技术选择在文件系统层上实现主要有两点原因，一是文件系统是操作系统内核中管理数据的最高层次，因此它可以发现下面任何层次出现的错误。二是文件系统拥有大量数据在磁盘上的组织信息，因此它可以执行数据语义层次的一致性检测。更主要的是由于文件系统处于内核当中，因此能够提供比用户层

更大范围的数据检测[3]。

但遗憾的是随着数据量的增多和磁盘技术复杂程度的增大，目前多数文件系统并没有提供足够的技术及策略来检测和恢复错误数据，使得上述的错误数据得不到修正，这样只会对用户数据造成更大的损失，并且这些损失可能是致命的。贝弗利山的《计算机技术评论》出版商 WestWorld 公司的调查数据显示：多数公司认为，100 兆的数据价值超过 100 万美元。如果公司的数据没有很好的保护和备份措施的话，在两年内，43%的公司经历数据灾难后无法重新开张，29%的公司倒闭。每 500 个数据中心中，就有 1 个每年要经历一次灾难。而如果数据存储量以现在年均 80%的速度增长，破坏的程度将进一步增加。

由于上述的原因，数据检测和恢复的技术开始逐渐的得到人们的重视。在数据恢复技术中，恢复时间目标（RTO）和恢复点目标（RPO）是决定数据恢复级别的两个关键指标，它们可以由用户需求和应用来确定。并且这两个指标和所要投入的成本是成反比的：要让 RTO 和 RPO 接近于 0 就需要投入更多的预算，如果恢复时间可以是几天或者几周，那么成本就将会大大减少。确定必要的 RTO 和 RPO 指标是满足应用需求所必须的，这样才能确定正确的数据恢复级别以避免浪费更多的投资。RTO 和 RPO 取决于应用流程和应用的影响分析，如果系统变得不可用，那么将带来极大的价值损失和预期财务影响。很显然，不同的业务流程和应用间有着不同的 RTO 和 RPO 要求。可用的预算和恢复目标的要求是两个相互竞争的力量，因此确定 RTO 和 RPO 将是一个反复的过程。而文件系统不仅要能够为数据的安全存储提供保护，也要考虑到系统性能的平衡问题。

1.1.3 麒麟加密文件系统中的数据存储

本文将在上述背景下详细讨论基于麒麟加密文件系统的数据块纠错技术的研究和实现。目的是在麒麟加密文件系统中实现和改进各种数据纠错技术，使之达到增强文件系统数据存储的一致性和可靠性的目的，并且为数据纠错问题的解决提供参考。

本次课题来源于国家 863 课题“分布加密存储软件结构及其关键技术”。本课题将致力于研究 Internet 环境中的分布式加密存储技术，研究分布式加密存储体系、高效分布数据存取、密钥管理、安全共享等关键技术。从私密性、可用性、可靠性等方面提高我国的海量信息安全存储系统的研制能力，为信息系统和其它的技术革新提供技术储备。加密存储能够为用户能够提供更加安全可靠的数据存储，但同时也对数据安全存储带来了新的挑战。

麒麟加密文件系统是麒麟操作系统中开发的文件系统，它使用内核实现方式，对存储数据加密保护，并且在核外使用改进的 SSH 认证系统，使得对用户身份的

认证更加安全，同时保证了传输数据的安全性。麒麟加密文件系统是以麒麟操作系统为平台，在虚拟文件系统层实现的一个栈式文件系统，不需要过多修改就可运行在多个系统之上。同时麒麟加密文件系统采用了三层加解密结构，支持文件和目录的共享；对文件数据和文件结构信息进行加密保护。

目前加密文件系统的数据存储主要存在以下几个问题：

- 麒麟加密文件系统采用链式对称加密算法对文件数据进行保护，为了提高安全性，系统自动为每个文件生成一个对称密钥。由于该系统采用了链式加密方式，因此哪怕是只有 1bit 数据出错，都会导致整个数据块解密出错，从而最终导致用户遭受比采用普通文件系统更大的损失。
- 由于麒麟加密文件系统是在数据块层次上进行的链式加密，因此数据检测和恢复的粒度也相应地要求在数据块层次上执行。否则会带来很多额外的加密解密开销以及检测与恢复开销。
- 对于用户的疏忽操作造成的数据丢失没有防范措施，因而无法对用户的原始数据作出恢复，对于这部分数据造成了永久性的丢失；
- 对于写操作一致性的保护还很薄弱，一旦出现系统错误，数据一致性无法保证，加密前后的数据无法一致。

1.2 课题研究内容

本课题研究了基于麒麟加密文件系统的数据块纠错技术。目的在于为麒麟加密文件系统提供全面的数据存储保护机制，并使其能够对错误的存储数据做出恢复。为此，本文为麒麟加密文件系统设计了数据纠错系统框架以及其中的各个数据纠错和保护功能模块，在各功能模块的具体目标和要求下，为各模块设计和改进了相关纠错技术，实现了麒麟加密文件系统保护数据存储和恢复错误数据的目的。课题的工作主要包括以下几个方面：

- 研究并分析了常见文件系统数据纠错技术；
- 分析了麒麟加密文件系统数据存储安全需求，并在此基础上设计了数据纠错系统框架；
- 对数据纠错系统框架中的各个模块进行了功能性的区分与分析，并对文件系统读写数据的保护流程以及数据检测与恢复流程做出了详细阐述。
- 对各功能模块中采用的数据保护技术进行了研究与分析，对于技术的不足进行了改进。
- 设计了麒麟加密文件系统数据块纠错技术以及应对错误数据的策略，为文件系统设计有效的遇错处理机制；
- 实现了数据块纠错技术，并对新的文件系统进行性能测试，为用户选择相

关技术给出了参考标准。

本课题的完成为麒麟加密文件系统设计数据块纠错技术提供了理论基础和初步实现，提高了文件系统数据存储的可靠性，具有很好的参考价值和实用价值。

1.3 论文的组织

本文是对课题研究工作的总结，全文共分为 6 章，结构如下：

第一章介绍了数据块纠错技术的背景，以及本课题的基本情况。

第二章详细阐述了数据块纠错技术的分类以及基本组成，分析了 NTFS 以及 Linux3 文件系统的相关技术特色，随后介绍了麒麟加密文件系统。

第三章提出了麒麟加密文件系统数据纠错系统框架的设计思想、总体结构，并且对各模块的功能和作用做出了详细地论述。

第四章在数据检测模块和数据恢复模块功能要求的指导下，研究了数据检测和恢复技术，对各技术的不足做出了相应的改进，设计并实现了增强型 Checksum 技术和基于文件的数据块的奇偶校验技术。

第五章针对快照恢复模块的功能给出了文件系统快照的创建方式和维护方法。然后给出了自动快照的实现方式，最后设计了自动快照中快照文件的淘汰算法。

第六章对数据纠错文件系统性能进行了测试，分析了实验结果，从而验证了纠错系统的有效性和可用性。

第七章对本文的工作进行了总结并对下一步工作进行了展望。

第二章 相关研究

数据存储的一致性和可靠性应当成为文件系统首要关注的问题[11]。能够安全的存储和获取正确的数据是每一个文件系统应当提供的基本功能。从文件系统的设计之初到现在，许多文件系统都希望能够提供简单、有效、可靠、一致的数据存储功能。本章首先对当前的数据块纠错技术和数据存储保护技术做了系统的介绍，然后对常见的文件系统的数据纠错技术进行了分析与研究，对其各自的优缺点做出了阐述和比较。为麒麟加密文件系统数据纠错系统框架的设计提供了参考和借鉴。

2.1 数据纠错技术概述

2.1.1 数据纠错技术分类

目前有很多种方式来保证数据存储的可靠性和一致性。有些技术是通过校验的方式来检测存储的数据是否一致，有些技术则是通过各种保护策略来尽量避免出现数据不一致的情况发生。根据这些技术保护措施不同的出发点，可以将数据纠错技术分为以下三类：

（1）避免技术

避免技术通过制定和实施相关的保护策略，来尽量避免数据出现不一致的情况发生，从而保证了数据存储的一致性和安全性。这种技术具有以下好处：第一不会对系统带来过大的额外开销（与下面两类技术相比），第二避免了其它技术发现错误后相对漫长的数据恢复过程，因而降低了系统数据恢复的代价。这一类技术的应用有：只读的存储系统、日志文件系统、加密文件系统，以及事务性文件系统等。

（2）检测技术

检测技术运用了简单的对比思想，通过将正确的数据和现有的数据，或者现有数据的某个特征进行比较来判断数据是否一致。而这一特征信息一般为信息摘要，它可以简要地描述一份较长的信息或文件，或者被看作一份长文件的“数字指纹”。信息摘要主要用于创建数字签名，对于特定的文件而言，信息摘要是唯一的。这一类技术的应用主要有校验值，副本镜像等。

（3）恢复技术

当检测技术检测出存储数据不一致时，即出现了数据错误的情况，需要通过恢复技术[6]将数据恢复到正确的状态。这就需要一些含有原始数据的冗余信息来帮助数据进行恢复，这些冗余信息必须包含了某些原始数据的信息，比如通过线性

运算来恢复数据的 RAID[6]奇偶校验技术。

2.1.2 数据纠错技术应用

由于采用单个技术只能保护部分类型的数据或者数据部分属性上的一致性，所以一个完整的数据保护策略应当采用多个技术的组合方式来达到全方位保护数据存储的目的。考虑到文件系统运行环境的多样性（比如从个人笔记本到服务器），以及文件系统所担负的不同的应用类型和负载量，在为文件系统[7]选择数据纠错技术时应考虑到以下两点：

（1）从文件系统运行环境上考虑，数据比较易于发生什么类型的错误，什么样的错误又将会对系统造成灾难性的破坏，通过上述考虑来决定文件系统应重点采取什么样的技术来避免哪一类的错误的发生。

（2）从文件系统的应用上考虑，该文件系统所能承受的最大额外系统开销是多少，而相关的数据纠错技术又会带来多少开销，文件系统应在可容忍的系统开销范围内来选择适合的数据纠错技术。

文件系统应当基于以上两个方面的考虑来对具体的数据纠错技术进行选择。

除了相关的技术之外，发现数据错误之后文件系统应当采取有效的动作来处理错误的数据，即策略，也应当被文件系统所考虑。针对不同类型的数据错误采取合适的策略，不仅能有效地保证存储数据的可靠性还能保证文件系统的高效性，比如 reboot，retry，stop 等。

2.2 数据纠错技术现状

本节将对当前数据纠错技术的研究现状进行介绍，对几种常见文件系统中采用的数据纠错技术进行分析和比较。

2.2.1 麒麟加密文件系统

麒麟操作系统（英文名称：Kylin Operating System）是由国防科技大学计算机学院“863 服务器操作系统内核”项目组研制的操作系统，Kylin 操作系统分为标准版和安全版。

麒麟加密文件系统是麒麟操作系统中开发的文件系统，其目标是在保证安全性的同时还保证系统的运行效率和系统的易用性。麒麟加密文件系统在 UFS 文件系统基础上，在内核的文件系统中应用加密技术保证数据的安全存储和透明访问，为用户提供了透明的加解密支持。如图 2.1 所示：

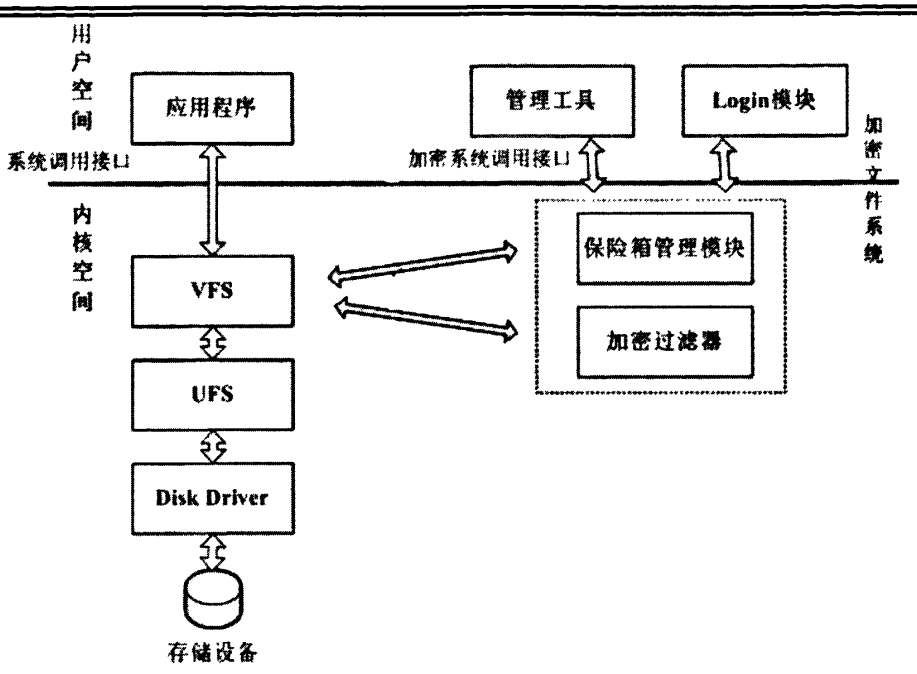

图 2.1 麒麟加密文件系统结构图

目前麒麟加密文件系统对数据存储一致性的保护还很薄弱，对于数据存储保护主要是采用 fsck[19]命令。fsck 命令通过检查步骤来发现文件系统中的数据不一致的情况，对个别类型错误还可以进行修复[19]。但是 fsck 的检测和修复过程漫长，并且不能自动的发现数据错误，只能由用户来启动检测或每一次系统启动时执行 fsck 命令。在用户使用 fsck 修复文件系统时，文件系统必须处于非活动状态。若在 fsck 修复过程中发生的文件系统更改，可能导致文件系统损坏，甚至可能导致系统的崩溃。当文件系统的超级块损坏时，文件系统必须恢复它。在文件系统中存储有超级块的多个副本，可以将超级块替换为其中一个副本。这样通过信息的冗余麒麟加密文件系统达到了保护超级块的目的。

2.2.2 Windows 文件系统

Windows 作为全球最大商用操作系统对于数据纠错建立起自己的一套标准。下面以 NTFS 为例，介绍 Windows 文件系统主要采用的几种数据纠错技术。

1) 读写数据的错误检测技术

NTFS 通过相关的检测技术能够发现文件系统对读和写数据的错误，再通过相应的策略，将相关的数据活动停止，避免了错误数据造成的更大破坏。

2) 元数据保护技术

NTFS 对元数据采取 Sanity check 技术进行保护，如果元数据块出现错误将造成文件系统的无法挂载，对于用户的损失较大，因此采用专门的技术对元数据进

行保护。

3）日志记录技术

NTFS 文件系统通过基于事务处理模式的日志记录技术(transaction logging and recovery techniques)，保证 NTFS 卷的一致性，实现文件系统的可恢复性。在 NTFS 下，系统会维护针对目录、分配和文件表等组件的事务日志。因此，CHKDSK 只需将事务回滚到上一个提交点就可以恢复文件系统中的一致性。事务日志方法需要的系统开销是很小的。NTFS 并不是直接在日志文件中存取记录，而是通过 LFS 来读写。LFS 提供了包括打开、写入、向前、向后、更新等操作来帮助 NTFS 处理日志文件。

4）多次重试和及时停止技术

当用户数据出现错误时，文件系统会及时停止，并且阈值范围内的重试多次无效之后，通知用户错误。

2.2.3 Linux 文件系统

Linux 缺省情况下使用的文件系统为 Linux ext2，Linux ext2 文件系统具有高效稳定的特点。但是，随着 Linux 系统在关键业务中的应用，Linux 文件系统的弱点也渐渐显露出来：其中系统缺省使用的 ext2 文件系统是非日志文件系统。为了解决这一问题，Linux ext3 文件系统[16]从 Linux ext2 文件系统发展出来。Linux ext3 文件系统的数据纠错技术特点如下。

1）通过日志实现了事务性文件系统

Linux ext3 比 Linux ext2 增加了日志技术，从而保证了文件系统的事务性和数据的一致性。Ext3 有多种日志模式，一种工作模式是对所有的文件数据及 metadata（定义文件系统中数据的数据,即数据的数据）进行日志记录（data=journal 模式）；另一种工作模式则是只对 metadata 记录日志，而不对数据进行日志记录，也即所谓 data=ordered 或者 data=writeback 模式。系统管理人员可以根据系统的实际工作要求，在系统的工作速度与文件数据的一致性之间作出选择。

2）对重要数据结构的检测技术

Linux ext3 并非对所有的数据块都做错误检测，而是对相对重要的数据结构的数据块做检测，例如超级块和日志块，对重要性相对弱一些的数据块则检测频率降低，如 bitmap，目录块。

3）非完全的保护策略

Linux ext3 对部分数据错误是无法感知和处理的，比如 rmdir 和 truncate 命令出错时，Linux ext3 是不会通知用户的。再例如，若日志数据出现了错误，Linux ext3 也是无法感知而是继续执行错误的事务，造成了数据的不一致。

2.3 本章小结

数据块纠错技术从其分类来看，主要分为三类，避免技术，检测技术，恢复技术；从其组成来看，主要是根据文件系统自身和运用的特点来对不同技术进行组合。

本章分别对麒麟加密文件系统、Windows 文件系统和 Linux 文件系统在数据纠错方面的功能进行了简要的介绍。总体来说，三种文件系统在数据纠错方面都提供了一定的功能支持，均在一定程度上保证了文件系统及数据的可靠性。Windows 的 NTFS 文件系统和 Linux ext3 文件系统作为目前文件系统中的佼佼者，其技术特点是十分鲜明的。Linux ext3 文件系统的数据纠错技术是简单却又高效的，问题是它对部分写数据错误的忽略可能会对数据和系统造成更大的破坏。Windows NTFS 文件系统与 Linux ext3 文件系统相比，数据的保护是更加高效的，它通过更多次的重试才通知用户出现的错误。麒麟加密文件系统在文件系统一致性检查和恢复方面提供了一定的功能，但是在一致性的检查和恢复过程中缺乏较强的可靠性，在检查和恢复过程中如果文件系统处于活动状态，可能出现检查错误或者恢复失败甚至导致文件系统崩溃的情况。本课题在其基础上进一步的研究和开发，致力于改进和完善其数据纠错的功能，从而保证系统数据存储的可靠性和系统高效性。

第三章 麒麟加密文件系统数据纠错框架

尽管数据纠错技术能够保证文件系统中部分数据的可靠性，但是由于软硬件复杂程度的增加，出现数据错误的种类和复杂度也在增加，这就要求不断的改进和发展数据纠错技术。为了能够更加有效和全面的保护存储数据，提高存储数据的可靠性[16]，本节针对麒麟加密文件系统设计了一个数据纠错框架，该框架由 3 个不同的数据保护功能模块构成，分别为数据检测模块，数据恢复模块和快照恢复模块，从而形成了一个保护体系，对文件系统存储数据的各个方面进行全方位的保护。3.1 节首先对麒麟加密文件系统的可靠性作出分析，提出了文件系统数据纠错的总体需求。3.2 节提出了数据纠错模块的整体设计思想，并且设计出了整体框架。后几节针对框架中的各模块进行了深入的功能分析和设计。

3.1 麒麟加密文件系统数据可靠性分析

麒麟加密文件系统采用链式加密技术，文件系统对数据可靠性的要求进一步增加，因为很小的数据错误就会导致整个数据块无法解密，从而引发数据损失。但是目前的文件系统本身并没有增加相应的数据保护技术，使得当文件系统数据块发生了错误时，系统却无法感知，若错误的数据导致系统破坏时，也缺乏相关的数据恢复技术来纠正错误的数据，使得用户和系统遭受到更大的损失[3]。为了能够全方位的保护数据的可靠性，文件系统需要具备以下功能：

1) 能够发现每一个数据的错误，这样要求数据纠错技术应当渗入文件系统的每一次 I/O 请求和答复中。虽然，数据块纠错技术可能会带来额外的系统开销。但这些开销应当在用户不可感知和可以承受的范围内。
2) 对重点数据进行重点的保护。由于不同的数据块可能属于不同的数据结构，因此彼此之间存在着重要性不同的差异，比如元数据和普通用户数据。所以在针对不同的数据块时应采取不同的数据纠错技术，对于那些相对重要的数据应当重点保护。
3) 对系统和数据破坏严重的错误要采取相关技术尽量避免。在造成数据出错的各种系统软硬件错误中，它们彼此之间对数据和系统造成的破坏程度也不同，对于那些会对系统和数据造成严重破坏的错误要采取重点防护措施。

3.2 数据纠错目标

针对麒麟加密文件系统对数据的可靠性要求，本节提出纠错系统的数据存储保护目标有以下几点：

- 能够发现每一次 I/O 操作的读写数据错误。
- 能够保证写数据时的数据一致性要求。
- 能够保证错误数据不被交给上层应用对用户造成损害。
- 在发现错误后能够通过相关技术使错误的数据得到恢复。
- 能够根据用户要求和指令来恢复某一时刻文件系统的状态。
- 对重要数据重点保护，重要错误重点防护。

3.3 数据纠错框架设计

本节从文件系统数据纠错目标入手，依据目标来设计相关的数据保护模块和恢复模块，通过不同模块的组合形成一个数据纠错的系统框架结构。然后对各模块之间的工作原理进行了说明，叙述了其对数据读写错误的检测和恢复，从而达到了数据纠错的功能，增强了文件系统数据存储的可靠性。

3.3.1 基本思想

对于数据纠错，系统首先要有能够发现数据错误的模块，这个模块需要能够感知数据的每一次 I/O 操作，并且在数据返回给上层应用之前就能够对读取数据的正确与否做出准确的判断，然后触发数据纠错的动作。

发现数据错误后，系统需要有相应的数据恢复模块来对数据进行纠错和恢复。在这里首先需要区分产生数据错误的两类主要原因：1）由于软硬件或系统错误所造成的数据错误。2）用户由于疏忽造成的错误操作导致了正确数据的修改或丢失。对于这两类不同的错误，第一种错误是可以由数据检测模块来发现的，而第二种错误，由于它对存储数据的修改是合法性的，所以数据检测系统是无法感知的，所以第一类数据错误的恢复可以由数据检测系统来触发，而第二类数据错误只能由用户自身来发现并且通过应用层的命令来触发恢复过程。所以针对这两类不同的数据错误，本文设计两类不同的数据恢复子模块来分别进行数据恢复。

3.3.2 模块组成

整体数据纠错系统的模块框架如图 3.1 所示：

如图所示，最上层是用户层，用户通过它向文件系统层发出各种指令，包括数据的读和写操作等。下一层是文件系统层，用来接收用户的各种数据并且执行用户的各种命令。

位于文件系统层下面的是数据保护层，在这一层中，拥有三个不同的模块，分别为快照恢复模块，数据检测模块，数据恢复模块，它们各自分别用来实现发

现错误数据，恢复错误数据的功能。

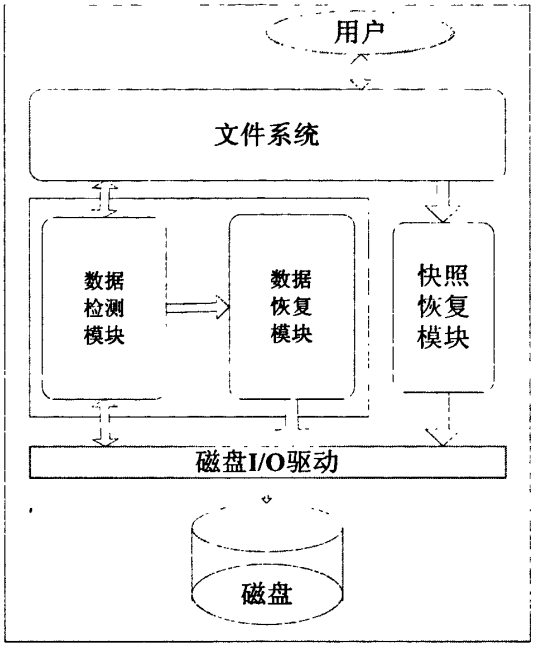

图 3.1　数据纠错系统框架图

对于文件系统的读操作，从磁盘获得的数据首先被送到数据检测模块，在这里，数据将通过相关技术来验证其正确性，只有被证明是正确的数据才能够交给文件系统，然后进一步的上交给用户层的应用。但当数据检测模块发现所读取的数据出错时，该模块会将错误数据的相关信息交给数据恢复模块，在此模块中，通过相关的数据纠错和恢复技术对错误的数据进行数据恢复，将恢复了正确性的数据重新写入磁盘，并且将恢复后的正确数据返回给文件系统，从而完成数据自动纠错的功能。

最后一个快照恢复模块是专们针对操作错误来恢复数据，该模块专门用于第二类错误数据的恢复，即用户由于疏忽造成的错误操作导致了正确数据的修改或丢失，这类错误由于其操作的合法性，所以在数据检测模块是无法识别的，只能够通过用户自身发现并且请求恢复过程。当用户意识到自己的错误操作或者希望文件系统恢复到某一时刻的正确状态时，用户首先向文件系统提出请求，文件系统再将请求传送到快照恢复恢复模块，在此模块中，文件系统将被恢复到用户所请求时刻的正确状态。此模块的数据恢复和纠错是针对文件系统来实现恢复的，而系统数据检测和数据恢复模块是针对数据块来实现恢复的。

表 3.1 数据纠错系统功能模块表

模块名称	触发条件	功能描述
数据检测模块	文件系统的读操作，磁盘向文件系统返回数据	对磁盘返回数据的正确性进行检验，若数据正确则返回给文件系统，若数据错误则触发数据恢复模块
数据恢复模块	数据检测模块发现所读取的数据不正确	对错误的数据通过相关技术进行自动恢复，将恢复后的正确数据写入磁盘，同时返回给上层应用
快照恢复模块	由用户通过向文件系统层发送命令来启动数据恢复过程	将文件系统恢复到前面某一时刻正确的状态

3.3.2.1 数据检测模块

下面将首先介绍数据检测模块的功能。对于读数据时的错误，系统首先要通过数据检测模块来发现数据错误，只有首先发现了错误才能够及时的避免错误数据对用户应用的破坏，然后对其进行数据纠错和恢复。

对于数据检测模块，它用于判断读取的数据是否是正确的。数据的正确性包含两层意思，一是取到了正确位置上的数据，二是取得正确位置上的数据是正确的。因此数据模块所提供的数据检测功能也应该包含这两个方面。在数据检测模块中，主要由数据检测技术构成，检测技术的原始思想也十分简单，通过与正确数据的对比来判断取得的数据是否正确。比如简单的通过与数据的复制副本来进行对比，但是这种方式的对比是不现实的，一是副本的空间开销大，二是无法保证数据副本的正确性，从而当两个数据不相同时无法判断是谁的数据出现的错误。于是文件系统希望能够运用数据的某些特征来作为数据比较的标准，这样首先降低了冗余信息在文件系统中的空间开销，比如为数据计算 Checksum 值[16]，不同的数据拥有不同的 Checksum 值。通过为读取来的数据计算 Checksum 值，然后与所存储的 Checksum 值进行比较来判断出数据是否正确。因此，在每一次的写数据块时，不仅仅要写入数据块的内容，还要计算出数据块的 Checksum 值并且正确的保存。在读取数据块时，同时读出它的 Checksum 值，以便于计算和判断数据的正确性。

表 3.2 数据检测模块

名称	功能细节	相关技术
数据检测模块	通过对数据进行计算得出数据的特征和正确的数据特征进行比较来判断数据的正误	Checksum，副本镜像，奇偶校验 Cyclic Redundancy Check (CRC),

3.3.2.2 数据恢复模块

在上一节的数据检测模块中，如果发现所读取的数据是错误的，下一步的工作就是将这些错误的数据恢复到正确的状态。此时数据将被送到数据恢复模块中来，这时模块会根据当初数据存储时所采用的数据保护技术所提供的冗余信息来对错误的数据进行恢复。这些用于恢复数据的冗余信息，应在数据写入时便存储下来，虽然这样会为系统带来一定额外的计算开销和空间开销，但是为了达到数据存储的可靠性，这些开销是无法避免的，但可以通过相关技术的改进来尽量减少这些开销。对于重要性较高的数据，相应对冗余量的要求可以适当的提高。而相对于重要性较低的数据，为了系统性能的平衡，可以使用可靠性稍弱或数据冗余量较低的数据保护技术。由于存在冗余信息，所以在这里文件系统对磁盘的写操作也会发生改变，写的内容和磁盘数据的分布都有可能会发生变化。另外，数据的恢复过程也会带来时间上的额外开销，因为数据恢复是一个重新读取相关数据计算的过程，而这个过程一般比冗余值的计算更加复杂。数据恢复到正确的状态后，该模块还要负责将正确的数据重新写入磁盘并且返回给文件系统。

表 3.3 数据恢复模块

名称	功能细节	相关技术
数据恢复模块	对数据检测模块送来的错误数据，通过错误数据保护技术所提供的冗余信息进行自动恢复	各种 RAID 技术，副本镜像，ECC. Error Correction Codes FEC. Forward Error Correction

3.3.2.3 快照恢复模块

对于快照恢复模块，由于数据检测模块无法感知用户合法操作带来的错误数据，所以该模块只能由用户通过命令来触发。当用户意识到自己的错误操作带来了数据错误并且希望文件系统能够恢复到之前某一时刻的正确状态时，快照恢复模块将通过相关技术把整个文件系统的状态恢复到用户指定时刻的状态。为了能够达到通过快照恢复模块恢复数据的目的，系统将定时的对文件系统做备份，这种备份可以是全数据备份，快照，克隆等。而这些备份技术的选择，备份的时机，都是由用户来指定的。

表 3.4 快照恢复模块

名称	功能细节	相关技术
快照恢复模块	根据用户要求将文件系统恢复到某一时刻的正确状态	快照，副本镜像，克隆

这里，我们首先给出功能模块与相关技术对应关系，如下表：

表 3.5 功能模块与纠错技术对应表

模块名称	主要技术
数据检测模块	校验和技术
数据恢复模块	奇偶校验技术
快照恢复模块	快照技术

3.4 本章小结

本章节首先对麒麟加密文件系统的可靠性做出了分析，发现了文件系统中的许多潜在的数据错误隐患。在分析的基础上提出了对文件系统数据纠错的目标，即为文件系统提供高可靠的数据存储保护。然后根据目标提出了纠错系统的基本设计思想，最后在设计思想的指导下设计出了文件系统数据纠错模块的系统框架。在系统框架模型中，主要由 3 个子模块组成，数据检测模块负责对读写的数据验证其正确性，数据恢复模块用于对错误的数据进行纠正，快照恢复模块则能够挽回用户错误操作带了的数据损失。本章各小节分别对各个子模块的设计思路、具体位置、功能和触发条件做出了详细介绍，并且对相关技术做出了简介。

第四章　麒麟加密文件系统数据检测与恢复

由于软硬件复杂程度的增加，数据错误的种类不断增加，现有的数据检测技术已无法满足需要。为了增强麒麟加密文件系统数据存储的可靠性，本章设计了数据纠错系统框架中的数据检测与数据恢复模块。首先列举和分析了数据检测和恢复模块的相应技术，然后针对其中的不足之处，分别改进了 Checksum 技术和 Parity 技术，进一步提高了文件系统数据存储的可靠性和一致性。

4.1　基于 Checksum 的数据检测错技术

文章首先对数据检测中的 Checksum 技术进行了细致分析，研究了传统 Checksum 技术的实现方案，并且针对加密文件系统设计了可靠性增强的 Checksum 技术实现方案。

4.1.1 Checksum 技术

4.1.1.1 Checksum 技术

判断一个数据是否完整，是否错误，即需要一个正确的数据作为标准来进行比较和衡量，Checksum 技术正在越来越多的被文件系统所采纳用来检测数据的一致性，当文件系统读取了某个数据后，通过为数据计算 Checksum 值，再与原来存储的该数据的 Checksum 值进行比较来验证取得的数据与原来存储数据是否是一致。这种 Checksum 值是基于 Hash 函数实现的，比如 MD5[16]和 SHA1,其中一个重要的原则是拒绝冲突，即找到两个不同的数据却具有相同的 Checksum 值是不可能的。

4.1.1.2 Checksum 效验的粒度

首先讨论引言中的第三个问题，即在什么粒度上对数据计算 Checksum 值在系统的可靠性和效率上平衡的。理论上来说，从小到大，可以选择从小到几 K（甚至更小）的数据或几 Byte，Block，Page，大到整个文件系统来计算 Checksum 值[16]。选择合适的粒度对不同的文件系统会带来性能和可靠性都合理的选择。一方面，如果选择较大的粒度，当在读取很小的数据量时，由于要整体计算 Checksum 值，所以会导致很多不必要的 I/O。另一方面，如果选择较小的粒度，会带来更多的计算开销和存储 Checksum 值的空间开销，因此，选择什么样的粒度，取决与文件系统本身的特性。

- Block 级别

Block 级别就是以数据块为单位，为每个数据块计算 Checksum 值并且存储。

在这个级别上，能够较准确的定位出现错误的数据块，在小数据量的读操作时不会引起大量的 I/O 操作，从而影响系统的性能，并且很多的数据恢复技术，比如 RAID[9]都是以 Block 为单位实施的，这样有利于文件系统采取复合的数据保护策略（比如 Checksum+RAID5）。但同时，相对于更高层次的抽象，比如 Page，文件系统是看不见块级别的，在大数据量读取时会增加许多额外的计算开销。现实中，ZFS[16]文件系统目前就采用的是 Block 级别的 Checksum。

- Page 级别

Page，进程中的单位，内存中的读取磁盘数据的最小单位。从性能上来说，它比 Block 级别更加平衡，因为数据都是以 Page 为单位读入内存中的。但是，Page 本身作为一个动态单位（即，组成 Page 的块是不恒定的），所以可能需要系统固定组成页的块，但这样对于数据读取的效率又会有影响，因为系统可能会为了一个数据而读入很多其他无关的数据。同时，相对于 Block 级别来说，在可靠性上 Page 级别相比于 Block 级别仍有缺陷，本文将在下一章讲到。I^3FS[16]采用的就是 Page 级别的效验。

- File 级别

为每一个文件计算 Checksum 值。他的优点是显而易见的，计算开销的减少和空间开销的减少，因为每一个文件仅需一个 Checksum 值,但是它的缺点也是明显的，过多的 I/O 开销。有一些应用级别的效验工具就是在这个粒度上实施的，比如 Tripwire[28]。

4.1.2 可靠性增强的 Checksum 技术

4.1.2.1 可靠性增强的 Checksum 值存储

- Checksum 值的分散存储

关于 Checksum 值的存储位置，传统文件系统采用的方法就是将 Checksum 值和存储的数据放在一起，比如存储在一个 Block 中，这样在取得了数据块的同时也就取得了 Checksum 值，从而进行一致性的检验。这种方法易于实施，但方法自身仍然有缺陷，它不能够发现错误位置的读和写操作（比如 misdirected write）。比如文件系统需要读取 A 块，但由于错误的发生导致系统取得了 B 块，但是进过数据块校验，B 块的数据是一致的（因为 B 块本身没有错），如图 4.1 所示。此时上层应用会得到错误的数据块 B 并且没有发现错误，从而对上层用户的应用造成了破坏。所以，首先 Checksum 值所存储的位置要求能够体现出该数据的位置信息，以保证取得的数据块就是所希望的块。

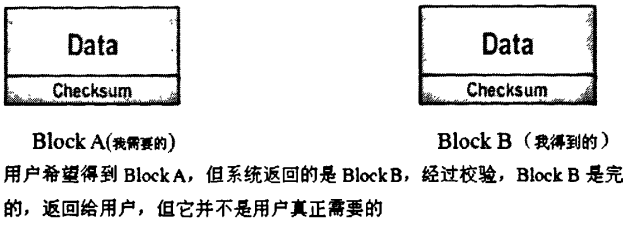

图 4.1 Checksum 值和数据集中存储示意图

在 ZFS 文件系统[8]中，数据和数据的 Checksum 值就是分离存储的，ZFS 将每一个数据块的 Checksum 值存储在其父节点块上，这样当读取了一个数据块时，从它的父节点中读取 Checksum 值，这种逻辑关系保证了系统取得了正确位置上的块，如图 4.2 所示，而由于 Page 不具备这种逻辑结构从而这种方法不适 用于 Page 级别。

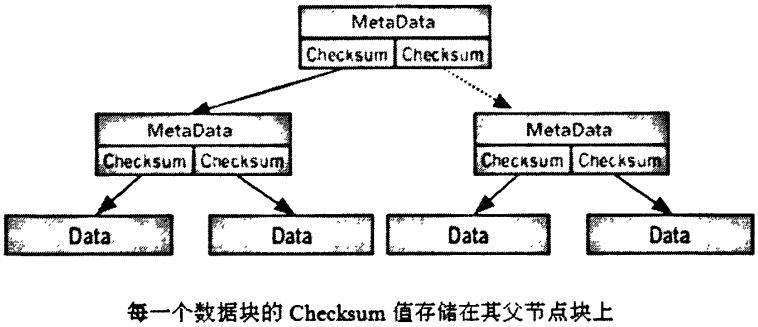

图 4.2 ZFS 中 Checksum 分离存储示意图

再考虑另外一种情况，Block A 是 Block B 和 C 的父节点，用户想取得 Block B，却取得了 Block C，这种情况也是无法检测出来的，解决的方法是将 Checksum 值和具体的块地址进行绑定，或者将 Checksum 值作为地址索引的一部分，这样也可以将 Checksum 值和地址绑定，如图 4.3 所示。

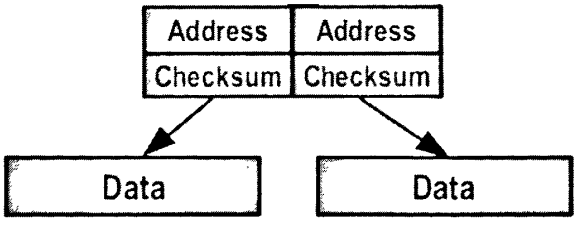

图 4.3 Checksum 值作为地址索引示意图

- Checksum 值的集中存储

本文提出的另外一种方法也是基于将 Checksum 值和数据值分离存储的思想，但是系统将所有的 Checksum 值另外专门开辟的特定空间内集中存储，这种做法有些类似于为数据的 Checksum 值做 Bitmap，而每一个位置上的 Checksum 值对应固

定位置上的数据或者在计算数据的 Checksum 值时就包含了数据的位置信息（将地址信息作为计算 Checksum 值的一部分），这样文件系统可以将 Checksum 值和数据的位置信息进行了绑定。如图 4.4 所示。比如用户需要读取数据块 A，文件系统从两个地方分别读取数据块 A 的内容和数据块 A 的 Checksum 值。当计算值和存储值一致时，说明数据是正确的，当出现不一致时，首先检查存储 Checksum 值所对应的地址信息，若和数据的地址一致，则说明数据出现了错误，若地址不一致，则说明寻址出现了错误，取了错误地址的数据。

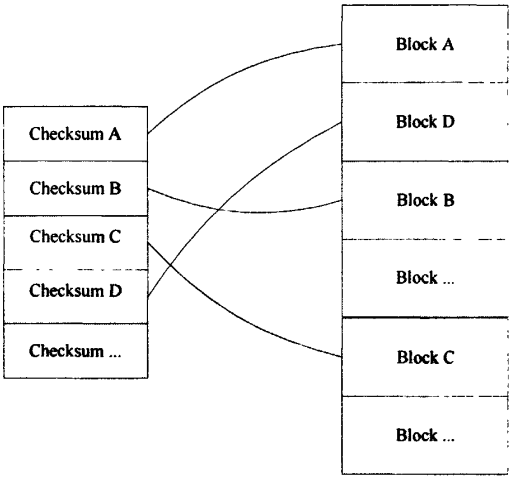

图 4.4　Checksum 值集中存储示意图

这种集中存放的方法的缺点是一旦存放 Checksum 值的数据块出现了错误，那么损失将会非常大，但是可以通过备份的方式来减少这种损失，并且集中存放的方式是便于备份的，因为不会像将 Checksum 值分散备份那样浪费过多的空间。

4.1.2.2 可靠的 Checksum 值验证

进过上述的方案后，随之而来的是如何保证 Checksum 值的正确性，即遇到了计算值和存储值不一致时如何确定是数据错误还是 Checksum 值本身出现了错误，因为 Checksum 值的存储也可能出现不一致的现象。接着上一章的方案，将每一个数据块的 Checksum 值存放在父节点中，所以每当读取一个数据块时既包含了一个自上而下的索引过程又包含了一个自上而下的校验过程，只有最顶层的超级块是自我效验的。这样，当一个数据块的计算值和所存储的 Checksum 值出现不一致时，文件系统就可以确定是存储的数据错误而 Checksum 值是可以被相信的，因为它的 Checksum 值是上一层的一部分，而上一层的正确性是已经被校验过的。而对于超级块自身，考虑到它的重要性和数据量较小的特点，拥有 2~3 个备份几乎是没有额外开销的并且能够大大提高系统的可靠性，提供恢复能力。

而对于 Checksum 值的 Bitmap 这种方式则需要集中存储 Checksum 值的数据块

提供自我校验和备份的方式来保证 Checksum 值的可靠性和可恢复性。由于 Checksum 值本身数据量较小，这部分数据的读取频率又较为频繁，可以考虑以页大小为单位连续整体的存储这部分 Checksum 值，并设置较高的优先级使其可以较长时间的停留在内存中不被替换或能够驻留 Cache 中，这样能够提高读取 Checksum 值的效率从而提高读写效率。

- Checksum 技术总结：

通过上述的讨论和提出的各种方案，本文总结出在实施 Checksum 技术的过程中应注意以下问题：

Checksum 值要存储在已确定正确的位置上或拥有机制能够确定其可信。通过对比来判断一致性，必须建立一个可信的标准。在这里必须确保 Checksum 值的正确性才能够将其作为标准，否则所有的对比和判断均没有意义和参考价值。

Checksum 值要能够体现出位置信息。为了保证文件系统的寻址正确，要求 Checksum 值要能够与地址信息有所联系来保证其得到的是正确位置上的数据，比如 Bitmap 和 Checksum 值与数据分离存储都是基于这样的思想。

Checksum 值需要备份。由于无法保证所有的 Checksum 值都是能够完整正确存储的，比如超级块和 Bitmap。所以需要一种简单并且可靠的方法来确定 Checksum 值是否完整并对出错的 Checksum 值作出恢复，简单的备份方式是合适的，因为 1）空间开销不会很大，2）易于实现且恢复高效，所以现在许多文件系统对文件元数据的备份策略就包含了对 Checksum 值的备份。

4.1.3 增强 Checksum 在文件系统中的实现

在介绍增强 Checksum 技术的实现之前，先要了解数据块的文件系统中的组织方式和索引方式，所以先介绍文件系统 inode 节点。

4.1.3.1 inode 节点数据结构

在文件系统中，一个文件数据块的分布信息都记录在了 inode 节点中，inode 节点又称为文件索引节点，一般一个文件对应一个 inode 索引节点，它用来保存文件名以外的其他所有文件信息，而文件名存放在目录中，这些文件信息包括：

- 文件类型：
- 文件的模式（读-写-执行权限集）
- 指向文件的硬链接数
- 文件属主的用户 ID
- 文件所属的组 ID
- 文件中的字节数
- 包含 15 个磁盘块地址的数组

- 上次访问文件的日期和时间
- 上次修改文件的日期和时间

通过文件的 inode 索引节点，可以访问到该文件数据块的磁盘地址。首先每一个 inode 索引节点包含一个 inode 头，在这个 inode 索引节点头中记录文件的相关信息，然后 inode 索引节点使用 15 个磁盘块地址（0 到 14）的数组指向存储文件内容的数据块。前 12 个地址是直接地址。即，它们直接指向文件内容的前 12 个逻辑存储块。如果文件大于 12 个逻辑块，则第 13 个地址指向间接块，该块包含直接块地址而不是文件内容。第 14 个地址指向双重间接块，该块包含一级间接数据块的地址。第 15 个地址用于三重间接地址。图 4.6 描述了 inode 节点与这些地址块之间的关系链。

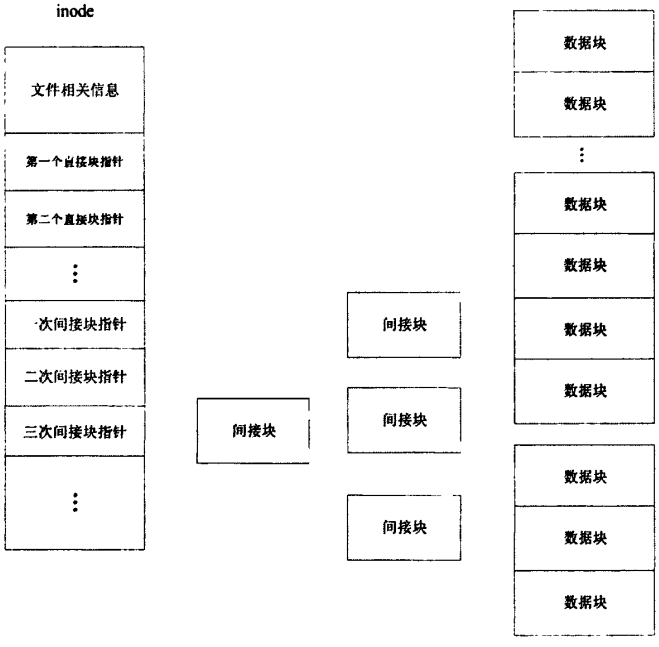

图 4.5　inode 索引节点结构图

4.1.3.2 Checksum 数据结构

通过前面的分析，传统的 Checksum 技术将数据与 Checksum 值存储在一个数据块内，这种方式不能发现错误地址的读取数据，本文采用了 Checksum 值与数据分离存储的方式，将 Checksum 值存储在其父节点上，并且与数据块的地址信息进行绑定，通过这种方式不仅保证了数据的正确性，而且保证了数据地址信息的正确性。

首先，要确定系统计算 Checksum 值的粒度，本文选择了在数据块粒度上计算 Checksum 值。采取这样设计的好处有：能够更细粒度的定位出数据错误的位置，

二是能够与后面基于数据块的恢复相对应。设计的第二步是确定 Checksum 值存储的位置以及如何与数据之间建立联系？

这里本文采用了将 Checksum 值和地址索引信息相关联的存储方式，简单的来说就是通过设计一个新的数据结构，使得这个数据结构中不仅包含了数据块的索引地址指针，还包含了用于存储数据块 Checksum 值的空间，称这个新的数据结构为数据索引校验项。具体如图 4.5 所示：

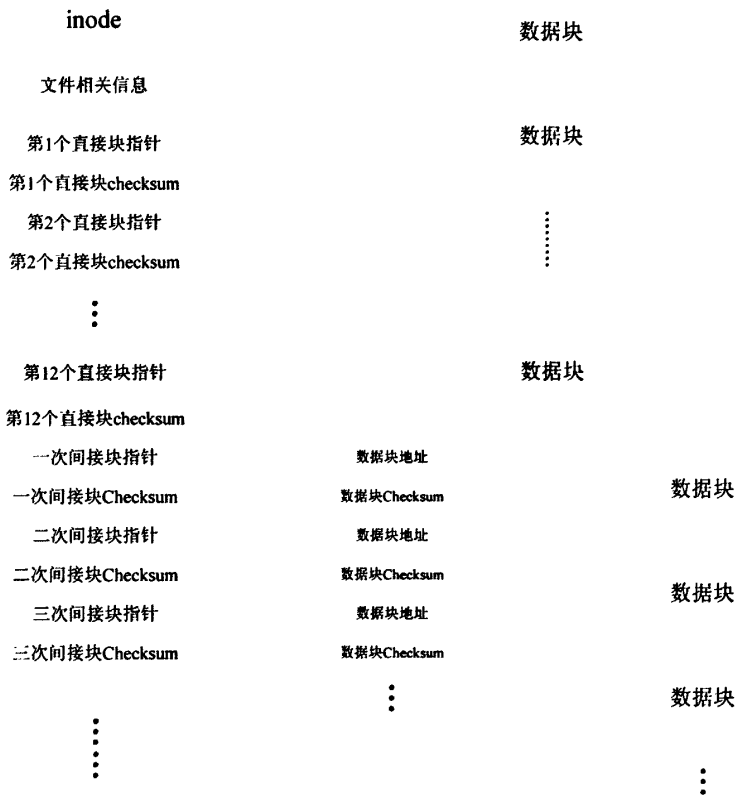

图 4.6　增强型 Checksum 值存储示意图

如上图所示，对于每一个地址索引指针，紧跟着这个数据块的 Checksum 值，这种将地址信息和 Checksum 值绑定的方法不仅能够检验出数据的错误，还能够保证获得正确位置上的数据。为了能够实现这种绑定，本文将数据索引校验项设计为 128 位，其中前 64 位依然用于存放数据地址指针，后 64 位用来存放数据的 Checksum 值。这样文件在读取数据时，在读取了数据地址信息后能够很方便的寻址读取到数据的 Checksum 值。

由于目前加密文件系统的一个 inode 节点大小为 256 字节，它只有 5 个未使用的 64 位字段，其中还有两个被用做了扩展属性，所以只剩下了 3 个 64 位字段可

以使用。如果在不修改 inode 节点大小的情况下，将 64 位地址指针转换为 128 位地址指针，意味着直接块的数量将从 12 个减为 6 个，而剩下的 3 个 64 位字段也将用于间接索引地址的增加。但是，由于加密文件系统中只有直接块能够使用碎片（fragment），减少直接块会大大增加空间的浪费，所以唯一的方法就是将 inode 节点的大小由 256 字节扩展到 512 字节。

将 inode 节点由 256 字节扩展到 512 字节后，inode 节点中的 12 个直接地址指针，3 个间接地址指针都有足够的空间可以从 64 位变为了 128 位，同时 inode 节点中仍然还有 144 个字节的空间未被使用，这部分空间可以作为以后增加文件属性的预留空间。

具体在文件系统中的修改，首先要创建创建新的数据结构，数据索引校验项，在代码中命名为 ksfs_daddr_t_c，其中前 64 位为原来的地址指针，后 64 位存储 Checksum 值。代码如下：

```
struct ksfs_daddr_c_t {
int64_t   daddr ;          /*64 位地址指针空间*/
unsigned char   cs[8];   /*64 位数据 Checksum 值空间*/
};
```

第二步是修改 inode 节点结构，这包括两部分修改，一是将原来的 64 位地址指针更换为 128 位的索引校验项，二是将 inode 节点的空间扩充到 512 字节。具体的代码如下：

```
ksfs_daddr_c_t   di_extb[NXADDR];/*   96: External attributes block. */
ksfs_daddr_c_t   di_db[NDADDR];  /* 128: Direct disk blocks. */
ksfs_daddr_c_t   di_ib[NIADDR];    /* 320: Indirect disk blocks. */
int64_t   di_spare[17];   /* 368:将 inode 节点的空间扩充到 512 字节*/
```

4.1.3.3 Checksum 的计算

在修改了相关数据结构后，文件系统便将 Checksum 值的大小限定在了 64 位，也就是 8 个字节大小，所以在设计 Checksum 值算法的时候，需要考虑到以下三点：一是算法要避免不同数据出现相同 Checksum 值的情况，即避免冲突。二是对于长度变化的数据输入，Checksum 算法要能够提供定长为 64 位的输出。这样才能方便在 64 位空间中的存储。第三个方面是算法的高效性，即能够尽量避免额外的时间开销。基于上述三点的考虑，下面首先介绍 Checksum 的计算方法。

Checksum = 目标数据各字节相互异或的结果

采用这种方法的最大原因是简单高效，由于计算过程并不复杂，所以能减少额外的计算开销，并且计算出来的 Checksum 值仅占 1 字节。这里针对 Checksum 算法首先考虑的是算法的高效性而不是“冲突性”是因为文件系统本身是加密文

件系统，即使存在数据 Checksum 值的冲突，非法权限希望利用这个冲突对用户数据的修改而 Checksum 值不变的情况也是不可能的。

目前文件系统采用的默认数据块大小为 8KB，在文件系统创建时还可以选择 4KB，16KB 大小等。如果数据块大小是 8KB，那么每 1KB 的数据计算 Checksum 值，这样就会有总共有 8 个字节的 Checksum 值大小，正好可以存放入 64 位的 Checksum 值空间。同理，如果是 4KB 大小的数据块，那么每 512 字节的数据计算 Checksum 值。总之，将数据块的大小分为相等的 8 份计算 Checksum 值，并且按顺序存储入 Checksum 值空间。

如果遇到数据块的数据小于数据块的容量时，比如文件的最后一个数据块可能存在未使用的碎片，那么空闲碎片的数据按全零的方式处理，也即计算 Checksum 值时不考虑这部分数据。

计算 Checksum 值的功能由函数 KSFS_GenCheckSum 实现，该函数将输入参数 pData 指向的长度为 iDataLen 的数据中的每个字节依次进行异或运算，并将结果作为 Checksum 值返回。具体实现如下：

```
unsigned char
KSFS_GenCheckSum(unsigned char *     pData,
                 int         iDataLen)
{
    int i = 0;
    unsigned char cs = 0;

    for (i = 0; i < iDataLen; i++)
        cs ^= *pData++;

    return cs;
}
```

4.1.3.4 文件系统数据读写的修改

由于加入了新的数据检测功能，原文件系统读写流程也要做出相应的修改。首先是读操作，在数据寻址过程中，不仅要将数据的地址信息读入内存，连数据的 Checksum 值也要一起读入。在将获取到的数据值返回给用户之前，对数据计算 Checksum，然后判断计算出的 Checksum 值和读取的 Checksum 值是否一致，若一致，将读取的数据结果返回给用户，若不一致，数据将不被返回，并通知用户数据错误。

4.2 基于 Parity 的数据恢复技术

上一节本文设计并实现了增强型 Checksum 技术，该技术能够及时的检测出系统中的数据错误，但是不能恢复错误，所以在发现错误数据后，系统希望能够自动根据数据信息的冗余来恢复错误的数据，为此本文引入了 Parity 技术。

4.2.1 高效的 Parity 计算

4.2.1.1 奇偶校验原理

为了提高系统的可靠性，通过采用奇偶校验[12]的方法，利用冗余信息来恢复错误数据。其基本思想是，奇校验通过增加一位校验位的逻辑取值，用于存储数据中“1”的位数的奇偶性，在数据源端将数据二进制编码中为 1 的位数形成奇数，然后在接收端使用该数据时，连同校验位一起检查为 1 的位数是否是奇数来判断数据的正误，做出进一步操作的决定。单纯的奇偶校验只能检查一位错误，且没有纠错的能力，因为它不能判断出是哪一位错误。偶校验道理与奇校验相同，只是将校验位连同原数据二进制编码中为 1 的位数形成偶数。

4.2.1.2 校验过程

校验分为编码和译码过程，下面以偶校验为例解释整个过程：

1．编码：编码就是当 8 位代码 D7~D0 写入存储器时，同时将它们进行偶校验逻辑运算以产生偶校验位（偶形成），若 D7~D0 中有偶数个 1，则 D7⊕D6⊕D5⊕D4⊕D3⊕D2⊕D1⊕D0=0, 即“偶形成”=0，若 D7~D0 中有奇数个 1，则 D7⊕D6⊕D5⊕D4⊕D3⊕D2⊕D1⊕D0=1, 即“偶形成”=1，然后将 D7~D0 和“偶形成”一起写入存储器。

2．译码：译码即读出时的校验，将读出的代码与 1 位校验位同时进行偶校验运算，若“偶校错”为 0，表示数据正确（无奇数个错），若“偶校错”为 1，表示数据有错（奇数个错）。

4.2.1.3 Parity 的改进计算

在文件系统中实现基于数据块的奇偶校验技术，可以通过为若干数据块增加一个奇偶校验块来实现对数据的冗余，从而实现对错误数据的恢复。比如 N 个数据块对应 1 个奇偶校验块，奇偶校验块的数据由 N 个数据块的数据通过计算得来。这意味着每修改一个数据块，文件系统都要读取 N-1 个数据块的数据，然后再进行计算，这显然不是一个高效的方法，并严重影响了文件系统的读写性能。这里，使用一种更加高效的计算方法，通过老的奇偶校验值计算新的奇偶校验值。比如文件系统修改了一个数据块的数据，那么新的奇偶校验值等于旧的奇偶校验值异或旧的数据值再异或新的数据值，公式为 New_Parity=Old_Parity⊕Old_Data⊕

New_Data。这样，在修改每一个数据块时只需要最多读取两个数据块。

4.2.2 高效 Parity 在文件系统中的设计与实现

4.2.2.1 Parity 数据结构

由于 inode 索引节点能够拥有文件所有数据块的索引信息，所以通过对 inode 结构做相应的修改就可以达到为文件系统实现数据奇偶校验的功能。为了不给文件系统带来更多的额外开销，本文为每一个文件分配一个奇偶校验块，并且使这个奇偶校验块与文件相关联，这样在数据检测模块发现数据错误后，通过读取奇偶校验块和文件其他数据块的内容对错误数据块进行恢复。采取这种设计的最主要原因是基于文件系统时间和空间开销上的考虑，每个文件只有一个奇偶校验块使得在数据修改时只需要修改一个校验块内容，并且一个额外数据块的空间开销也是很小的。当然，在节省开销的同时，系统的容错性会相应的降低，因为这样文件中最多允许一个数据块错误。由于在上一节中文件系统将 inode 索引节点的大小从 256 字节扩大到了 512 字节，因此有足够的空间来存放奇偶校验块并且与文件相关联。如图 4.7 所示：

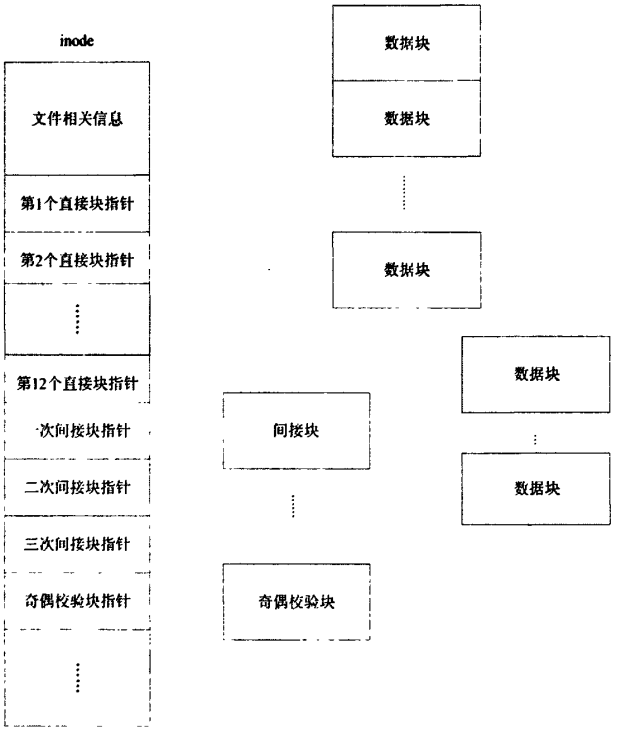

图 4.7　修改后的 inode 索引节点结构图

如上图所示，由于在上一节 inode 索引节点扩展到 512 字节后，有 144 个字节的空间未被分配，所以将其中的 16 字节用于存放奇偶校验块的索引校验项，其中

前 8 个字节存放奇偶校验块的地址指针，后 8 个字节存放奇偶校验块的 Checksum 值。inode 索引节点代码中，首先再添加一个数据索引校验项，再将剩余空间补齐 512 字节

```
ksfs_daddr_c_t  di_fpb; /*368:file parity block*/
int64_t   di_spare[15];   /*384:将 inode 节点的空间扩充到 512 字节*/
```

4.2.2.2 奇偶校验值的计算

当文件开始写数据时，如果是第一次向一个新文件写数据，奇偶校验块的内容等于新写的数据块与全零的数据块进行逻辑异或计算，比如文件只有一个数据块时，奇偶校验块的内容为这个数据块与全零数据块的异或值。

当文件的写操作为文件新增加了数据块时，新的奇偶校验值等于老的校验值异或新的数据块数据。

Parity=Old_Parity ⊕ New_Data

当文件的写操作修改了一个数据块的内容时，新的奇偶校验值由下面的公式计算：

Parity=Old_Parity ⊕ Old_Data ⊕ New_Data

当文件删除了一个数据块时，新的奇偶校验值等于老的校验值异或老的数据块数据。

Parity=Old_Parity ⊕ Old_Data

以上计算过程如图 4.8 所示。

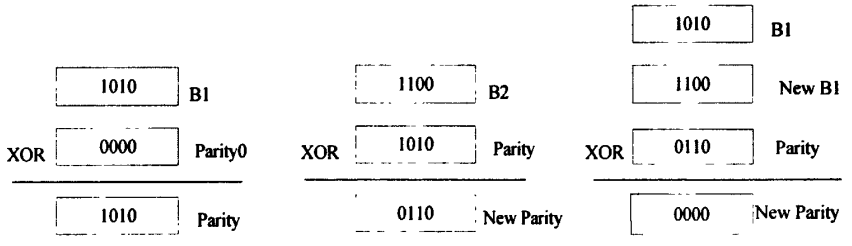

图 4.8　Parity 计算过程图

B1：文件产生的第一个数据块。B2：文件新加入的数据块。New B1：新的 B1 块数据。

4.2.2.3 文件数据恢复过程

当文件系统通过 Checksum 技术判断出某一数据块的内容错误时，文件系统将自动启动对错误数据块的写操作，写操作的数据内容由文件数据块和奇偶校验块计算得来，也就是说，一个数据块出错时，系统会读取文件内其他数据块的内容，然后计算出错误数据块的内容并写入。恢复过程如图 4.9 所示：

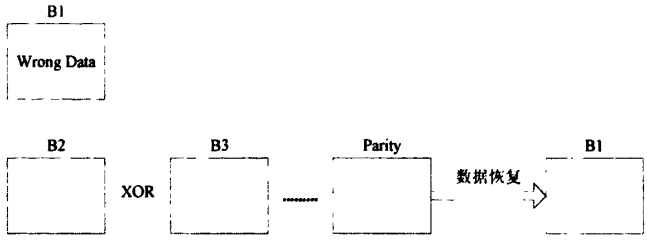

图 4.9 Parity 恢复数据图

4.3 本章小结

本章节首先根据数据检测和恢复模块功能的具体要求，提出了各模块的具体技术。然后通过分析和改进相关技术进一步提高了技术的可靠性，使得这些技术可以检测并且恢复系统带来的数据错误。同时根据加密文件系统自身的特点并结合以上的技术设计出了数据检测和恢复的实施方法，并按照设计思路修改相关的数据和函数实现了数据检测和恢复功能。从而完成了达到了提高数据存储可靠性的目的。

第五章　麒麟加密文件系统的自动快照

继上一章对数据检测模块和恢复模块的设计和实现，本章节将给出基于麒麟加密文件系统快照恢复模块的设计方案，该模块可用于恢复被用户误删，误改的数据，或者将文件系统恢复到某一时刻的原始状态。快照恢复模块主要采用了基于文件系统的快照技术，快照是对文件系统的一种备份方式，建立快照以后对文件系统的修改不会影响快照中的内容。因而快照能够使文件系统恢复到原始状态。在快照基本功能的基础上，本章研究了麒麟加密文件系统的自动快照技术，设计了快照淘汰算法。

5.1　文件系统布局

5.1.1 加密文件系统布局

麒麟加密文件系统主要由超级块，柱面组描述符，inode 节点表，柱面组摘要和数据区组成，大致的结构分布如图 5.1 所示：

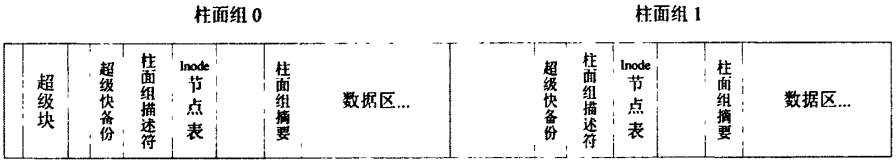

图 5.1　文件系统布局图

柱面组：加密文件系统将文件系统的磁盘分区划分为一个或多个区域，每一个区域称为柱面组，它作为一种逻辑上管理数据块的手段，类似于 Ext3 文件系统中“块组”的概念。每一个柱面组必须能够放入到一个文件系统中，文件系统为每一个柱面组按顺序编号，如 0 号柱面组，1 号柱面组。柱面组的一切相关信息保

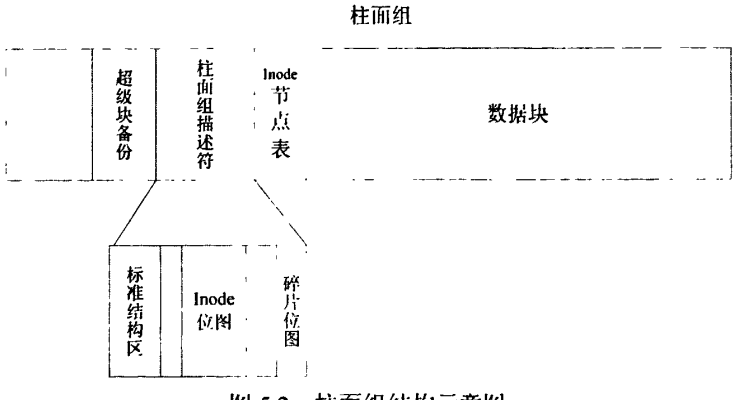

图 5.2　柱面组结构示意图

存在柱面组描述符和柱面组摘要中。柱面组结构如图 5.2 所示。

超级块：超级块存储有关文件系统参数的大量信息，包括柱面组大小，数据块大小，碎片的大小，还包括文件系统最后挂载时间等相关信息。通常位于文件系统起始位置的 64KB 处。由于它的重要性较高，所以在每一个柱面组中都拥有超级块的备份副本。

inode 节点表：每一个柱面组都拥有一个 inode 节点表，它用于存放本柱面组内的 inode 节点。每一个柱面组的 inode 节点表都位于柱面组的固定位置，一般为第 32 个碎片处，它的大小由 inode 节点大小和为柱面组分配的 inode 节点数决定。

柱面组摘要：柱面组摘要主要用于存放柱面组的统计信息，包括该柱面组的目录数，空闲块数，空闲 inode 节点数，空闲碎片数。

5.2 快照文件的创建与维护

5.2.1 快照基本概念

快照技术是数据备份的一种重要解决方案。文件系统的快照是一个文件系统在某个时刻获得的冻结映像。它通过保存文件系统即时数据块分配视图的方法来保证备份数据的完整性。文件系统快照具有以下几项重要的功能：能够提供文件系统在一天里几个不同时刻的备份，能够对活动的文件系统做可靠地转储，还能够在一个活动的系统上运行文件系统检查程序，回收丢失的数据块和 inode。

5.2.2 创建快照文件

创建快照文件的基本步骤如下：

1）创建一个快照文件，并将快照文件的大小初始化为文件系统分区的大小，其中快照文件 inode 中每一个地址指针都对应着文件系统分区上的一个数据块，这样快照文件就代表了整个文件系统分区的平面结构。该快照文件被用来跟踪文件系统未来发生的变化，它的文件块指针首先都被标记为“0”，代表该数据块未被复制。

2）预先扫描一遍分区中的每个柱面组，将超级块信息和柱面组信息复制到空闲数据块，再将快照文件 inode 节点中对应位置的地址指针指向复制的数据块。此外扫描每个柱面组里的映射表，即柱面组图，判断出那些数据块是空闲块，在快照文件中标记为数据块未被使用。

3）将文件系统标记为“等待挂起”状态。在这种状态下，对于那些能够修改文件系统的系统调用来说，如果有进程想调用它们，那么就会被挂起，保证这些系统调用不会再修改文件系统。而那些已经调用了这些系统调用的进程，则获许

执行完毕。

4）文件系统从“等待挂起”变为“彻底挂起”状态。当前文件系统中所有修改文件系统的系统调用都执行完成后，文件系统将被彻底挂起不再修改任何数据。

5）文件系统向磁盘做同步操作。

6）对于第 2 步里复制过的柱面组信息，如果此后执行的系统调用又修改过任何的柱面组，则需要将修改过的柱面组信息再次复制到先前给它们分配的数据块中。重新扫秒映射表，判断出哪些块被修改过。新分配的数据块标记为未被复制，而新释放的数据块则被标记为未使用的。

7）快照文件就位后，文件系统上的活动继续进行。

至此，快照文件的创建工作完成。一个快照文件的结构如图 5.3 所示：

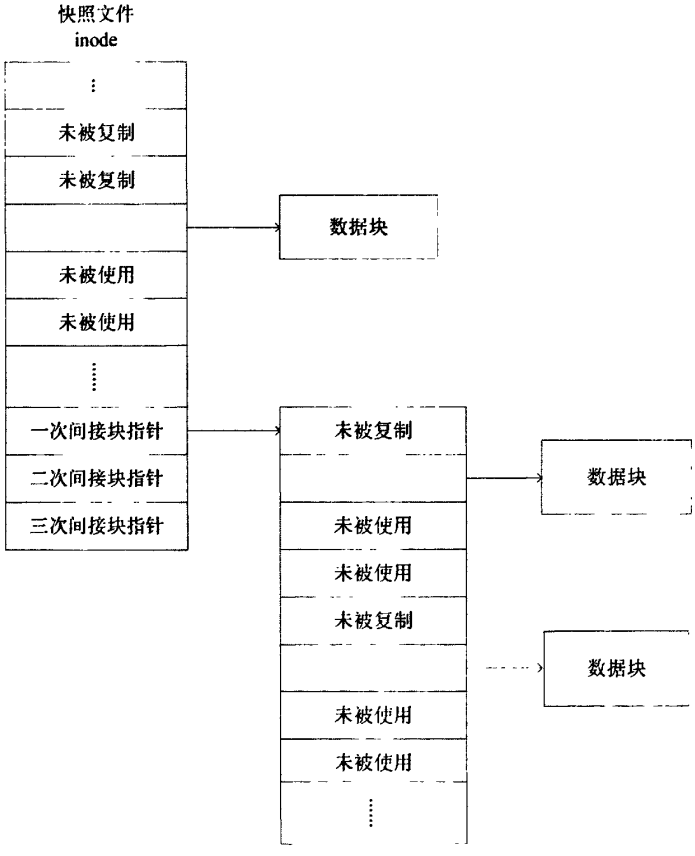

图 5.3　快照文件结构图

如图中所示，已分配的数据块存储了被复制的文件系统数据信息，未被复制代表该数据块还未被修改，未被使用代表该数据块还未分配。

5.2.2 快照文件的维护

5.2.2.1 快照文件数据块的维护

当文件系统中的数据块发生变化时，快照文件中的数据块也会发生变化。首先对于在快照文件中被标记为未使用的数据块，如果在文件系统中发生了修改（如写入了数据），那么快照文件是不需要发生变化的，因为在创建快照时这些数据块还是空闲块，所以它的变化不需要被记录。

对于那些已复制的数据块，也是不需要变化的，因为它本身就存储的当时的数据内容。

只有那些未被复制的数据块在文件系统中发生变化时需要被记录。对这类数据块的操作只有两种，覆盖写和删除。如果是覆盖写操作，文件系统首先会从空闲块中分配一个数据块，然后将老数据块的内容复制到新的数据块中，然后将新的数据块加入到快照文件中去，将原来未被复制的标记修改为新数据块的地址。之后，数据块的覆盖写操作得以在老数据块上进行，因为老数据块的内容已经被复制，并且记录在了快照文件中。

对于删除一个数据块，文件系统首先会遍历快照文件链表，因为文件系统可能同时拥有多个快照文件。当找到一个快照文件，这个要删除的数据块在该快照文件中是“未被复制”时，将这个要删除的数据块加入到该快照文件中，快照中的“未被复制”修改为该数据块的地址。遍历快照链表的过程结束。对于其他的快照文件，由于该数据块并未被释放给空闲空间，而做为了快照文件中的一个数据块，所以即使这个数据块在其他快照文件中也是活动块，依然标记为“未被复制”。

5.2.2.2 快照文件的文件维护

由于文件系统可能同时拥有多个快照文件，在创建快照时需要对快照文件的总数加以限定。限定快照数量的原因主要有两点：一是基于快照链表空间大小的考虑，在创建快照文件时，快照文件的 inode 节点号会被记录在超级块中，而超级块中数组的大小决定了快照链表的大小，目前这个数组能够最多存放 20 个快照文件。第二个原因是基于系统性能的考虑，由于对每一个数据块的修改都需要遍历快照文件来记录数据块的变化，如果快照文件过多的话会增大文件系统的额外开销。所以目前加密文件系统只支持最多 20 个快照文件。

快照文件的文件维护主要包括快照文件的增加和删除，以及对其它快照文件带来的变化。由于多个快照文件的共存，时间点后面的快照文件可能会包含时间点前面的快照。如果删除一个前面的快照，它的数据块并不会返还给空闲链表，而是加入到后面的快照文件中。这意味着，除非删除最新的快照文件，否则数据块永远不会返回给文件系统。久而久之会对系统带来巨大的空间开销。为了解决这个问题，后面的快照文件会把前面的快照文件看作是长度为 0 的文件，这样，

删除一个前面的快照时就会释放它所占用的空间。

5.2.2.3 文件系统的恢复

当通过快照技术恢复文件系统时，文件系统通过读取快照文件来恢复数据，当读到的是地址指针时，就返回指针所指数据块的内容，因为这个数据块上的内容是对原来数据的复制，如果读到的是“未被复制”，就返回文件系统中与之对应的数据块内容。

5.3 自动快照的设计

5.3.1 自动快照技术

实现快照功能首先需要掌握快照的相关命令，并且需要确定为哪个文件系统做快照、快照文件的存放位置、创建快照文件的名称等。通过相关的快照命令就可以执行创建快照的操作。自动快照是指由文件系统自动执行创建快照的命令，为文件系统创建快照。它一般由用户设置创建快照文件命令参数，通过设定触发条件由文件系统来代理用户自动执行创建快照指令。实现自动快照的功能，就需要系统能够通过用户提供的参数自动执行创建快照命令。所以这里要介绍一下麒麟操作系统中的自动脚本技术 cron。利用 cron，可以让系统定时执行一些指令或某个脚本程序，这样可以实现比如自动数据备份、清理等功能。cron 服务是麒麟操作系统内置的服务。默认情况下，cron 是开机自动启动。通过程序来自动修改 cron 脚本，将用户对自动快照的参数传递给脚本，修改脚本文件，然后由 cron 自动执行。cron 脚本实际为一个文本文件，可以通过应用程序修改自动快照的相关参数，包括做快照的频率，时间，做快照的文件系统，快照存放的位置，快照的名称。通过与用户的交流来获得自动快照的相关参数，再把相关的参数写入 cron 自动脚本的文本文件中，最后由 cron 脚本来执行自动快照的任务。

5.3.2 快照淘汰算法

由于文件系统对快照文件的总数加以限定，所以在自动创建快照文件时，如果快照文件数量已达到最大值，就需要删除前面的快照文件来为新的快照文件腾出空间。如何选择删除的快照，而更能保证文件系统的可靠性便成了自动快照淘汰算法所要考虑的问题。

5.3.2.1 淘汰算法思想

在设计快照文件淘汰算法时，首先要区分快照文件中的自动快照文件和用户手动创建的快照文件。对于那些由用户手动创建的快照文件，意味着用户对这些

快照文件有着特殊的要求，它们有着特殊的用途，对这种用户所要求的快照文件不应该属于被自动删除的范畴，因此在扫描快照链表时，不需要考虑这部分快照。由于目前加密文件系统只支持最多 20 个快照文件，所以加密文件系统限定了用户手动执行的快照数量为 5 个，自动快照的数量为 15 个。

其次，淘汰快照文件时需要考虑的参数。最主要的参数是快照文件创建时间，每一个快照文件都有一个创建时间，这个时间被记录在快照文件的 inode 节点中。如果自动快照是按时间顺序执行的话，不需要读取 inode 节点就能计算出快照的创建时间。比如自动快照每天在 12 点，18 点，24 点分别创建三个快照文件，系统就可以按顺序推断出任何快照的创建时间。一般如果两个快照文件创建时间间隔越短，文件系统的变化就越小，但是也不排除文件系统短时间内变化巨大的情况。文件系统变化的越小，快照文件能够恢复的粒度就越小，因为文件系统大部分数据没有改变。反之，快照时间间隔越长，数据发生的变化就越大。用户的一般要求是离现在时间点越近的快照数量应该越多，因为用户可能会意识到最近的错误操作，而离时间点越远的快照数量应该较少，因为长时间的间隔证明错误操作的几率很小，或不存在，除非用户希望恢复系统的早期原貌才会恢复长时间段以前的快照。所以，综上所述，快照文件在时间上的分布应该类似于指数函数，离当前时间点越近，快照文件的数量越多，如下图所示：

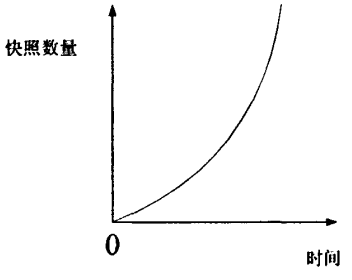

图 5.4　快照时间与数量关系图

如上图所示，距离最新快照创建时间越近，时段内拥有快照的数量应该越多，距离最新快照时间越远，时间段内的快照数量越小。但是由于快照建立时间和快照数量是一些分布离散的点，而上面的图是连续函数的分布图，所以快照文件在时间段内的实际分布应该是离散分布的数据点，如图 5.5 所示：

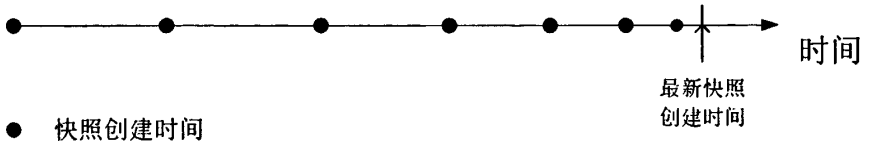

图 5.5　时间段与快照点分布图

如上图利最新快照文件创建时间越近，快照文件时间点的分布越密集，反之，

越远，快照文件的分布越稀疏。

5.3.2.2 时间段与快照数量的分布

为了达到快照在图 5.5 中的分布效果，在具体的执行该过程中，系统将快照数量按时间段由小至大进行分配。首先是一天之内的时间段，文件系统为一天之内分配 3 个快照，例如每天的 12 点，18 点，24 点创建 3 个快照。这样在第二天的 12 点创建快照时，会删除前一天 12 点的快照，以此类推，第二天的 18 点时删除前一天 18 点的快照，由此，文件系统始终能够保证最近 24 小时内的快照。对于时间段星期，文件系统也为其分配 3 个快照，分别为每周的周一，周三，周五。每个月也将分配三个快照，分别是每个月的 10 日，20 日，最后一日（可能为 28 日，30 日或 31 日）。每年也有三个快照，分别是 4 月，8 月，12 月。这样划分后还剩下 3 个快照，所以以每九年为一个时间段，其中每三年一个快照。通过上面的划分后，达到了离当前时间越近，快照数量越多的效果。具体如表 5.1 所示：

表 5.1 时间段与快照数量分配图

距离当前时间间隔	快照数量	快照时间点
一天（24 小时）	3	12 点，18 点，24 点
一周（7 天）	3	周一，周三，周五
一个月	3	10 日，20 日，最后一日
一年	3	4 月，8 月，12 月
9 年	3	每三年一个快照

- 时间段内的快照代表：

在确定了时间段与快照数量的关系后，文件系统将一个系统快照最长时间确定为 9 年，意味着快照文件时间若超过 9 年就会被自动删除。这样设定的原因是时间过早的快照文件一般占用的空间开销较大，二是时间越久，快照被用于恢复的可能性就越小，所以无限期的保留快照文件的意义不大。当快照数量被用完，每一个时间段与每一个快照的关系确定后，系统需要从每一个时间段内的快照中选取一个快照作为这个时间段的代表。例如，若一天中有三个快照，那么这一天中最后的快照作为这一天的代表，同理，每个月的最后一个快照作为这个月的代表。以此类推，在为某一个时间段选取快照时，这个时间段内最后一个快照将作为这个时间段的代表。

5.3.2.3 快照文件的淘汰算法

下面将具体介绍快照文件的淘汰过程，对于一天来说，每天三个快照创建的时间分别是 12 点，18 点，24 点。在下一个 12 点的快照创建前，前一个 12 点的快照将被删除，下一个 18 点的快照创建前，前一天的 18 点快照会被删除，以此类推。但是 24 点的快照在删除时需要考虑该快照是否作为了另外时间段的快照，

比如，作为了某一天的快照，或者某一个月的快照，如果这个快照是这个时间段的最后一个快照就不能被删除。对于一个月的时间段快照也一样处理，保留每个月 10 日，20 日，最后一日的最后一个快照作为这个月的三个快照，当下一个月的 10 日创建快照时，就可以删除前一个 10 日的快照，下一个月的 20 日快照创建时，删除前一个 20 日的快照，但是 4 月，8 月，12 月的最后一日快照可能会被保留做为这一年的快照记录。

以上过程可以表述为：

准备创建一个新的快照，如果这个新的快照是 12 点或 18 点，删除前一天 12 点或 18 点的快照，如果这个快照是 24 点的快照，判断这一天是否是周一，周三或者周五，如果是，删除上一个周一，周三或者周五的快照，判断这一天是否是这个月的 10 日，20 日或最后一日，如果是，删除上一个月的 10 日，20 日或最后一日的快照，如果这个快照是最后一日快照，并且是 4 月，8 月或 12 月的最后一日，则删除上一个 4 月，8 月或 12 月的最后一日快照。如果这个快照是 12 月最后一日的快照，则删除 9 年前 12 月最后一日的快照。

Delete(*x*)：删除文件*x*；
Time(*x*)：文件*x*创建的时间；

```
BEGIN
对于新的自动创建快照文件 NewSnapshot
1       Switch（Time(NewSnapshot)）
2       Case 12 点或 18 点；
3           Delete（前一天的 12 点或 18 点的快照）；
4       Case 周一，三，五的 24 点；
5           Delete（上周一，三，五的 24 点快照）；
6           Delete（上一天的 24 点快照）；
7       Case 本月的 10 日或 20 日的 24 点
8           Delete（上个月的 10 日或 20 日的 24 点快照）；
9       Case 2，3，4，6，7，8，10，11 月的最后一日 24 点
10          Delete（上个月最后一日的 24 点快照）；
11      Case 4，8 月最后一日 24 点
12          Delete（上一个 4 月或 8 月或 12 月最后一日 24 点快照）；
13      Case 12 月 31 日的 24 点
14          If （3 年循环内的年份）
15              Delete（去年 12 月 31 日 24 点的快照文件）；
16  End
```

5.4 本章小结

本章节首先介绍了加密文件系统的数据布局和有关的数据结构与功能。然后说明了文件系统创建步骤和相关数据结构的变化，并且结合快照文件结构的变化说明了快照文件中数据块的维护和文件的维护。在此基础上给出了基于 Cron 自动快照的实现方式，随后又设计了快照文件自动淘汰算法，该算法特点是基于时间段的不平均分配，因此更能够保证数据的一致性。全章节围绕快照的相关功能一步步递进，全面的展示了快照的功能与实现。

第六章　麒麟加密文件系统数据块纠错测评

在实现了数据纠错功能的基础上，本文将对文件系统的性能做出测试，以证明数据纠错系统的可行性，并且为用户在选择数据保护技术时提供性能方面的参考。本章首先介绍了测试的各种软硬件环境，然后采用标准的文件系统测试方法，测试数据纠错技术对文件系统性能的影响。通过测试，为用户在技术的选择上提供了重要的参考，使得用户可以在自身应用的要求下（如时空开销），有选择性的使用相关技术来保护自己的数据。

6.1　测试环境

本文所有测试工作均在麒麟天机服务器上完成。麒麟天机服务器是分布式加密文件服务器，采用麒麟加密文件系统，对用户数据采取链式加密方法，有效地保护用户数据的安全。

麒麟天机服务器配置如下：

表 6.1　服务器配置表

名称	配置
CPU	INTEL E5450/3.00G/12M/1333MHZ*2
内存	Samsung 2G/ECC/REG/DDR2/FBD/667 *2
硬盘	富士通 300G/15K/SAS * 15
主板	BB5000PSLSASR/8DIMM/SAS *1
阵列卡	LSI MegaRAID SAS 84016E *1

6.2　文件系统性能测试

本节将给出文件系统读写性能测试结果，通过结果分析文件系统性能的前后变化来证明纠错技术的可行性。下面首先介绍测试工具：iozone。

iozone 是一个文件系统的 benchmark 工具，专门用于测试操作系统和文件系统的读写性能。它可以测试 read，write，re-read，re-write，read backwards，read strided，fread，fwrite，random read，pread，mmap，aio_read，aio_write 等等不同的模式下的读写性能。

首先是文件的写操作性能，这里测试文件的大小在 64K 到 64M 之间，记录块的大小从 4K 到 16M，测试出系统的写数据速率。对比如图 6.1：

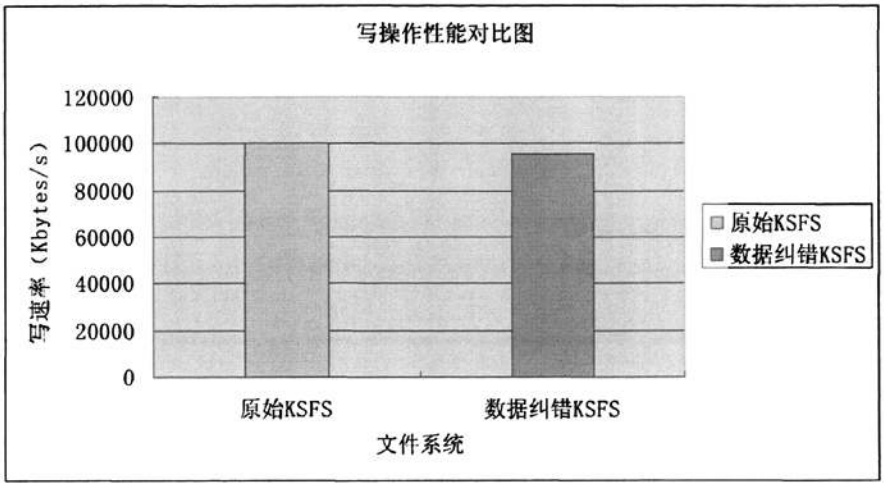

图 6.1　文件系统写操作性能对比图

如上图所示，由 iozone 测试得到原始麒麟加密文件系统平均写速率是 100469Kbytes/s，具有数据纠错功能的麒麟加密文件系统平均写速率是 95922Kbytes/s。该平均速率是由文件大小从 64K 到 64M，记录块的大小从 4K 到 16M 所测出的。通过上图可以说明，从系统整体写性能考虑，与原始的 KSFS 相比，数据纠错 KSFS 带来的额外开销并不大，主要原因是 Checksum 值计算方法简单高效，本文的 Checksum 值相对于其它 Checksum 值计算方法简单，相对于加密文件系统加密功能算法的开销几乎可以忽略。

但是将小文件和大文件区分考虑，文件系统的额外写开销相对还是明显的，如图 6.2 和图 6.3 所示：

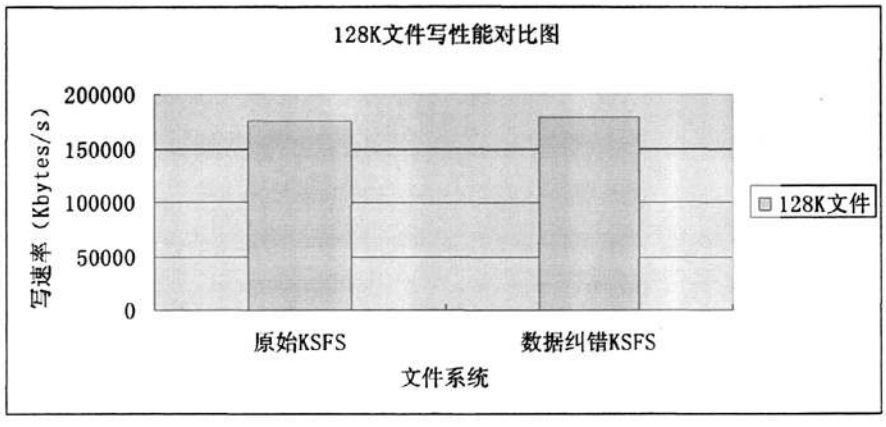

图 6.2　128K 文件写性能对比图

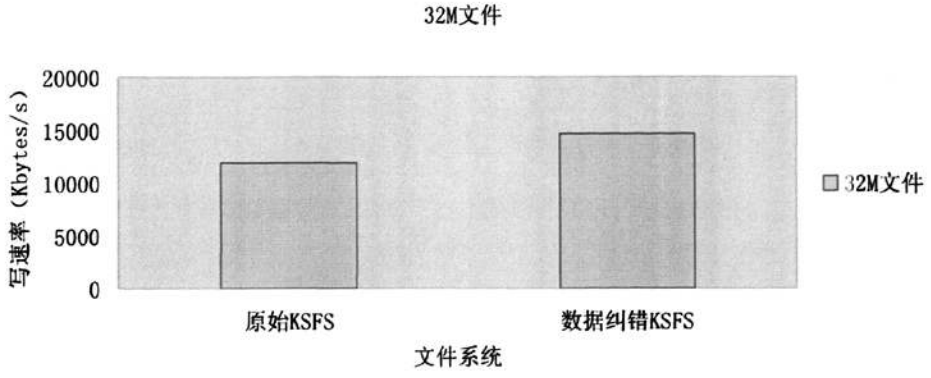

图 6.3　32M 文件写性能对比图

从图中可知，对于小文件，系统额外开销增加并不大，这是因为小文件的数据一般能够整体的存在于内存甚至 Cache 中，能够快速访问，并且由于数据量小导致计算量小。对于大文件的写，由于数据量大而导致计算开销相对较大。

对于读操作，原始加密文件系统平均读速率是 8620Kbytes/s，数据纠错文件系统平均读速率是 8513Kbytes/s，通过下图比较，二者之间的差别不大。对于大小不同的文件，小文件的额外开销相对于大文件增加较明显，但整体上变化基本不大。如图 6.4，6.5，6.6 所示。

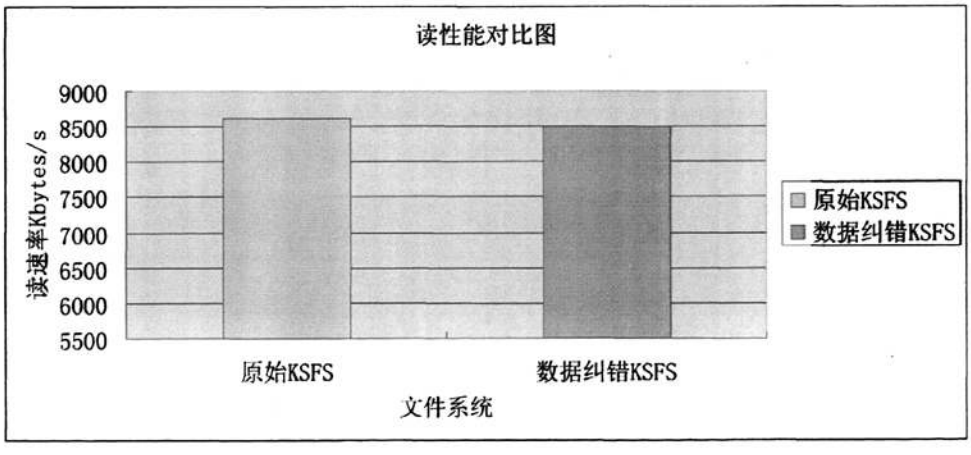

图 6.4　文件系统读性能对比图

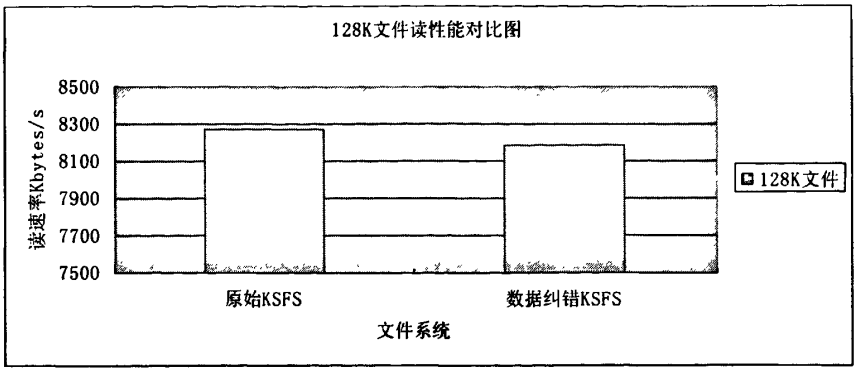

图 6.5　128K 文件读性能对比图

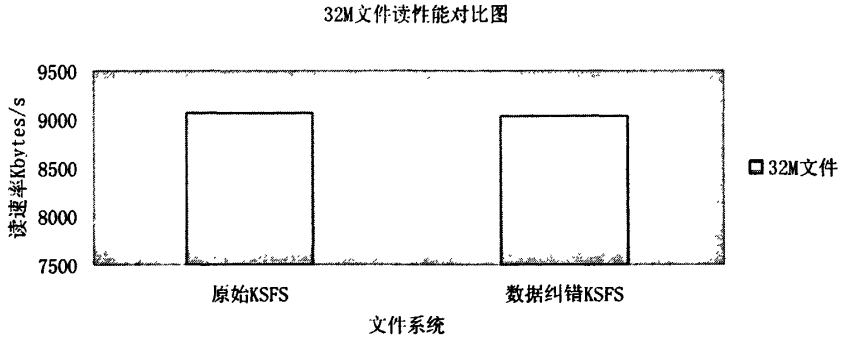

图 6.6　32M 文件读性能对比图

在使用 iozone 获得大量测试数据后，为了直观上更加清晰，我们将对实验数据做简单的处理，通过平均值的方法测试出文件系统读写数据的平均速度，然后对原始数据取为 1，对数据纠错系统带来的额外开销计算增加的百分比。这样能够直观比较出系统开销的增加。比较如下表：

表 6.2　性能测试比较表

操作	原麒麟加密文件系统	数据纠错系统
写操作	1.00	1.05
读操作	1.00	1.01
重复写操作	1.00	1.10
重复读操作	1.00	1.02

从实验结果可以看出相关的技术对文件系统带来的额外系统开销并不会增加多少。特别是对读操作的影响基本上是很小的。而对于重复写操作带来的额外系统开销最大。

6.3 本章小结

本章首先给出了对于文件系统测试的软硬件环境，然后通过 iozone 工具对文件系统的性能做出了测试，得到的测试结果有助于我们有目的和有选择的实施具体的数据保护技术。

第七章 结束语

前面的章节详细描述了课题研究中所做的工作，本章中将进一步总结，客观评价课题研究中取得的成果，分析存在的不足，对下一步工作进行指导。

7.1 全文工作总结

伴随着用户数据量的增加和对数据安全性要求的提高，文件系统对数据的保护也将更加全面和可靠。正是在这样的背景下，如何提高数据存储的安全性和可靠性被越来越多的文件系统所重点关注。本文也是在这种前提下，提出了基于麒麟加密文件系统数据块纠错技术的研究与实现。本文的工作主要有以下几部分组成：

介绍了数据块纠错技术的概况，包括数据块纠错技术的分类以及组成；通过对 NTFS 以及 Linux3 的介绍，分析了目前主流文件系统在数据块纠错上的一些技术特色，并对麒麟加密文件系统特点作了介绍。

为了达到全面保护存储数据的目的，本文提出了麒麟加密文件系统数据纠错软件框架结构，其中包含三个功能模块，分别是数据检测模块，数据恢复模块和快照恢复模块。它们从不同方面来增强文件系统存储的可靠性，其中数据检测模块能够发现文件系统中数据的读写错误，然后通过数据恢复模块对错误数据做纠错处理。快照恢复模块则使用户误删，误改的数据恢复成为可能。

在纠错系统框架的基础上，本文研究了框架内各模块所采用的数据纠错技术，分析了其中的不足。重点介绍了 Checksum 技术，奇偶校验技术和快照技术。在增强型 Checksum 中，主要针对现有 Checksum 技术的实施方法进行改进，将数据块的地址指针和 Checksum 值绑定，进而使数据存储的安全性进一步提高。对于奇偶校验技术，本文为文件系统设计了基于文件数据块的奇偶校验技术，每一个文件拥有一个奇偶校验块，使得能够对文件中错误的数据块进行恢复。在快照技术中，结合了传统快照技术和写时拷贝快照技术的特点，对于修改和未修改的快照数据块作出了拷贝数据和记录引用的不同处理，有效降低了快照的开销。之后，面对具体的加密文件系统现状，本文设计了相关纠错技术的实施方案和实现方法。

最后，本文在麒麟加密文件系统上实现了相关的数据纠错技术，并且对文件系统进行了性能测试，验证了数据纠错系统的有效性和可用性。

7.2 下一步研究工作展望

首先本文的所有工作都是基于数据安全考虑的，并没有对一些具体的应用背景做出分析，而有一些数据和应用为了高效性是不需要很高要求数据保护的。目前这些技术和策略都是静态的，即一旦实现，对所有应用都将一视同仁。但实际中文件系统希望能够按照用户的选择来实现存储数据的保护级别。其次，由数据纠错技术带来的额外系统开销目前看似是不可避免的，但是通过一些技术改进能够将数据保护功能和文件系统常规功能相结合，达到一个技术复合多种功能，从而降低系统的开销和数据的冗余量。另外出于性能的考虑，在文件系统的 Parity 技术恢复数据时，由于只有一个奇偶校验块，所以在遇到大数据量文件恢复时，效率是低下的，解决的方法是增加奇偶校验块的个数，比如每 7 个数据块对应一个奇偶校验快，但是这样又会带来更多的额外系统开销，所以具体的实施方法要针对具体的应用背景而定。

在实验环节中，本文发现在写操作性能测试中，当文件大小是 64M 时，文件系统的写速率变化不大，原始的文件系统为 10215Kbytes/s，数据纠错文件系统为 10250 Kbytes/s，它们之间的差距相对较小。在个别点上的数据甚至优于原始文件系统，造成原因目前仍在研究，进一步的实验结果仍待验证。将 64K 到 64M 文件变化大小性能对比变化如图 7.1 所示：

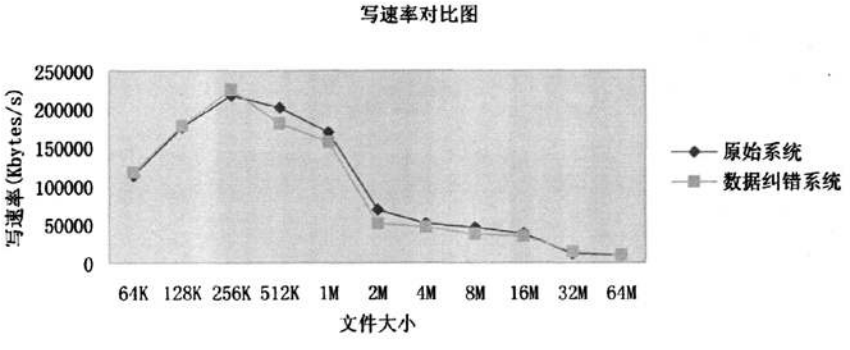

图 7.1 文件大小与速率对比图

新的性能实验应该考虑到更大的文件，比如 8G，16G。另外，文件系统逻辑块大小对文件系统性能影响也应当被考虑进来，需要进一步的实验来验证。

致 谢

在我的课题和硕士论文完成之际，谨向在我攻读硕士学位的过程中曾经指导过我的老师，关心过我的朋友，永远关怀我的父母，以及所有帮助过我的所有人致以崇高的敬意和深深的感谢！

首先，要衷心感谢我的导师廖湘科研究员！感谢老师在我的学习、工作和生活中所给予的无私的关心、悉心指导和严格要求。廖老师平时工作繁忙，但在百忙之中仍抽出时间对我进行全面的指导。廖老师渊博精深的学识、严谨求实的作风、敏锐独到的洞察力、勇于开拓的思维使我受益匪浅。廖老师高尚的品质和敬业的精神也是我学习的楷模。

感谢何连跃和付松龄老师。本次课题的选择和展开是和何老师的指导分不开的。付老师经常从自己繁忙的工作中抽出时间和我探讨课题的研究和具体的实现问题；在我的课题遇到问题时，总是能给我有益的启示，使我得以顺利的完成课题任务，对我论文的撰写也帮助了很多。在付老师身上使我看到了做工程应有的执着和严谨。

感谢李姗姗老师，李老师总是严格要求我们，并在课题的进程中对我们给予了必要的指导、帮助和督促。并且在百忙之中抽出时间指导我的论文撰写工作，并给出了指导性的建议和具体要求，对我的论文严加要求，对本课题的展开有很大的帮助。每一次与李师姐的谈话都能使我有所启迪，让我感受到了思想的力量。

感谢我的父母，感谢我父亲对我的培养，是您一直在指引我前进的方向，让我在这条道路上不断地坚持下来，使我明白这一定不是唯一的出路，但却是最有可能成功的道路。感谢母亲对我的包容和关怀，让我时刻感觉到家人的关心与爱护。

感谢 620 教研室的所有老师以及各位工作人员，他们工作细致周密，为人温和，在课题的研究过程中给予我很多关心和帮助。

最后，再次向所有在我攻读硕士学位的过程中曾经指导我的老师，关怀我的领导，关心我的朋友和帮助过我的人致以真挚的谢意！

思想驱动行为，好的思想引导好的行为。我将继续培养自己品格和思想，来指导自己的行动。年轻时最大的投资就是对自己的投资。

参考文献

[1] LakshmiN. Bairavasundaram, GarthR.Goodson, BiancaSchroeder AndreaC, Arpaci-Dusseau, RemziH. Arpaci-Dusseau. An Analysis of Data Corruption in the Storage Stack. FAST '08: 6th USENIX Conference on File and Storage Technologies, USENIX Association.

[2] W. Bartlett and L. Spainhower. Commercial Fault Tolerance: A Tale of Two Systems. IEEE Transactions on Dependable and Secure Computing, 1(1):87~96, January 2004.

[3] Gopalan Sivathanu, Charles P. Wright, and Erez Zadok: Ensuring Data Integrity in Storage. Techniques and Applications StorageSS '05 FairFax, Virginia USA. Ensuring Data Integrity

[4] D. Gambetta. Can We Trust Trust? In Gambetta, Diego (ed.) Trust: Making and Breaking Cooperative Relations, electronic edition, Department of Sociology, University of Oxford. 1990: 213~237

[5] Muthian Sivathanu, LakshmiN, Bairavasundaram, AndreaC, Arpaci-Dusseau, RemziH, Arpaci-Dusseau. Life or Death at Block-Level. OSDI '04:6th Symposium on Operating Systems Design and Implementation, USENIX Association.

[6] Michael.M.Swift, Muthukaruppan Annamalai, Brian.N.Bershad and Henry.M. Levy. Recovering Device Drivers. OSDI '04: 6th Symposium on Operating Systems Design and Implementation, USENIX Association.

[7] G.Weinberg. The Solaris Dynamic File System. http://members.visi.net/thedave /sun/DynFS.pdf, 2004.

[8] Sun Microsystems, Inc. Solaris ZFS storage solution. Solaris 10 Data Sheets, 2004.

[9] TPM Main Part 1 Design Principles Specification V1.2. Trusted Computing Group, 2003.

[10] R.L.Rivest. RFC 1321: The MD5 Message-Digest Algorithm. In Internet Activities Board. Internet Activities Board, 1992.5.

[11] K. J. Biba. Integrity Considerations for Secure Computer Systems. ESD-TR-76-372, Electronic Systems Division, Air Force Systems Command, Hanscom Air Force Base, Bedford, MA, USA, 1977

[12] AndrewKrioukov, LakshmiN, Bairavasundaram, GarthR, Goodson, Kiran Srinivasan, Randy Thelen, AndreaC, Arpaci-Dusseau, RemziH, Arpaci-Dusseau. Parity Lost and Parity Regained. FAST '08: 6th USENIX Conference on File and Storage Technologies, USENIX Association.

[13] D.Patterson, G.Gibson, R.Katz. A case for redundant arrays of inexpensive disks

(RAID). In Proceedings of the ACM SIGMOD.1988.6.

[14] DavidBigelow, ScottA.Brandt, CarlosMaltzahn, SageWeil. Adapting RAID Methods for Use in Object Storage Systems. FAST '08: 6th USENIX Conference on File and Storage Technologies, USENIX Association.

[15] A. Jøsang and S. J. Knapskog. A metric for trusted systems. Global IT Security, 1998: 541~549

[16] Peter Loscocco and Stephen Smalley. Integrating Flexible Support for Security Policies into the Linux Operating System. In: Proc of the FREENIX Track of the 2001 USENIX Annual Technical Conference, 2001.

[17] R. Watson. TrustedBSD: Adding Trusted Operating System Features to FreeBSD. In: Proc of 2001 USENIX Annual Technical Conference, Boston, Massachusetts, USA, June 2001:15~28

[18] Robert W, Brian F, Adam M, et al. Design and Implementation of the TrustedBSD MAC Framework. In: Proc of the 3rd DARPA Information Survivability Conference and Exposition, Washington DC: IEEE Press, 2003:38~49

[19] Marshall Kirk McKusick, Gregory R. Ganger. Soft Updates: A Technique for Eliminating Most Synchronous Writes in the Fast Filesystem. Proceedings of the FREENIX Track: 1999 USENIX Annual Technical Conference Monterey, California, USA, 1999.

[20] Chris Wright, Crispin Cowan, Stephen Smalley. Linux Security Modules: General Security Support for the Linux Kernel. In: Proc of 11st USENIX Security Symposium, San Francisco, CA, USA. 2002.

[21] A. Jøsang and S. J. Knapskog. A metric for trusted systems. Global IT Security, 1998: 541~549

[22] Peter Loscocco and Stephen Smalley. Integrating Flexible Support for Security Policies into the Linux Operating System. In: Proc of the FREENIX Track of the 2001 USENIX Annual Technical Conference, 2001.

[23] G. Sivathanu, C.P.Wright, E. Zadok: Enhancing File System Integrity Through Checksums. Technical Report FSL-04-04, Computer Science Department, StonyBrook University, 2004.5.

[24] Gopalan Sivathanu, Charles P. Wright, and Erez Zadok: Ensuring Data Integrity in Storage. Techniques and Applications StorageSS '05 FairFax, Virginia USA.

[25] Sun Microsystems, Inc. Solaris ZFS storage solution. Solaris 10 Data Sheets, 2004.

[26] SunMicrosystems, Inc: 系统管理指南：设备和文件系统. 819 - 7062 - 10, 2006.9.

[27] A.Kashyap, S.Patil, G.Sivathanu, E.Zadok. I3FS: An In-Kernel Integrity

Checker and Intrusion Detection File System. In Proceedings of the 18th USENIX Large Installation System Administration Conference (LISA 2004), pages 69-79, Atlanta, GA, USENIX Association. 2004.11.

[28] G.Kim, E.Spafford. Experiences with Tripwire: Using Integrity Checkers for Intrusion Detection. In Proceedings of the Usenix System Administration, Networking and Security (SANS III), 1994.

[29] HaryadiS Gunawi, Vijayan Prabhakaran, Swetha Krishnan, AndreaC. Arpaci-Dusseau, RemziH. Arpaci-Dusseau: Improving File System Reliability with I/O Shepherding. SOSP'07, 2007, Stevenson, Washington, USA.

[30] Vijayan Prabhakaran, LakshmiN, Bairavasundaram, Nitin Agrawal, HaryadiS, Gunawi AndreaC, Arpaci-Dusseau and RemziH.Arpaci-Dusseau. IRON File Systems SOSP'05, 2005, Brighton, United Kingdom.

作者在学期间取得的学术成果

[1] 郑思,杨尹.Checksum 技术在文件系统中应用的研究.第 20 届全国计算机技术与应用(CACIS)学术会议论文集. 2009:817~821.

[2] 杨尹,韩伟红,郑思.基于时序分析的木马规模预测技术.全国计算机安全交流会论文集. 2009:204~209.