

本科生毕业设计

|  |
| --- |
| XXX系统的设计与实现 |

|  |  |
| --- | --- |
| 院 系 | 计算机科学与技术 |
| 专业班级 | 卓越1901 |
| 姓 名 | 小岳岳 |
| 学 号 | U201915102 |
| 指导教师 | 郭德纲 |

2023年06月01日

**学位论文原创性声明**

本人郑重声明：所呈交的论文是本人在导师的指导下独立进行研究所取得的研究成果。除了文中特别加以标注引用的内容外，本论文不包括任何其他个人或集体已经发表或撰写的成果作品。本人完全意识到本声明的法律后果由本人承担。

作者签名： 2023年06月01日

**学位论文版权使用授权书**

本学位论文作者完全了解学校有关保障、使用学位论文的规定，同意学校保留并向有关学位论文管理部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权省级优秀学士论文评选机构将本学位论文的全部或部分内容编入有关数据进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本学位论文属于 1、保 密□，在 年解密后适用本授权书

2、不保密☑。

（请在以上相应方框内打“√”）

作者签名： 2023年06月01日

导师签名： 2023年06月01日

摘 要

随着数据量不断呈现爆炸性的增长，压缩技术被广泛的使用以减少数据存储的成本。相比于会抢占CPU资源的主机端压缩和因块设备跟文件系统协议不匹配导致性能下降的文件系统端压缩，存储设备端压缩具有更好的性能。而存储设备硬件层上的压缩会引入“内部碎片”问题，即压缩后的数据页跟存储的物理页的大小不匹配而造成存储空间利用率的下降。为了降低“内部碎片”问题所带来的负面影响，需要一种合适的数据布局管理方案来管理压缩后的数据页。

为了研究合适的数据布局管理方案，首先进行对于RE-BP和COMPACTION这两个现有方案的复现。但这两种方案都难以兼顾读写性能和存储空间利用率。RE-BP方案由于破坏了压缩页的空间局部性并且在关键路径上引入了排序和查新插入的步骤而降低了顺序读取的性能。COMPATION方案则因允许一个压缩页存储在两个物理页上而降低随机读取的性能。为了减少上述因素对于性能的制约，设计了FOLLOW方案。首先，在存储当前压缩页时，优先满足当前压缩页同前一个压缩页存储在同一个页里，充分利用数据的空间局部性原理。其次，保证一个压缩页存储在一个物理页上，降低了读取时较高的尾延迟。最后，设计了一个读缓冲区来存储解压缩之后的数据页以减少访存次数。

实验结果表明。

**关键词**：数据布局；空间利用率；尾延迟；缓冲区

Abstract

Failure resilience is of pivotal importance in practical network function virtualization (NFV) systems, but has been mostly absent in the existing ones. The absence is mainly due to the challenge of patching source code of the existing NF software for extracting important NF states, a necessary step toward flow migration and replication to provide failure tolerance.

This paper proposes NFActor, a novel NFV system that uses the actor programming model to provide transparent resilience, high scalability and low overhead in network flow processing. In NFActor, a set of efficient APIs are provided for constructing NFs, with inherent support for scalability and resilience. A per-flow management principle is advocated, different from the existing practice, which provides dedicated micro service chain services for individual flows, enabling decentralized flow migration and scalable flow replication. We implement NFActor and show that it achieves good scalability, prompt flow migration and failure recovery with large numbers of concurrent flows. We also show that NFActor can enable applications such as live NF update and correct MPTCP subflow processing, which cannot be efficiently achieved in previous systems.

**Keywords:** Network Function Virtualization (NFV), Failure resilience, Actor model, Flow migration, Service Chain

目 录

[摘 要 I](#_Toc126600794)

[Abstract II](#_Toc126600795)

[1 绪 论 1](#_Toc126600796)

[1.1 课题背景 1](#_Toc126600797)

[1.2 国内外研究现状 2](#_Toc126600798)

[1.3 研究目的和主要内容 4](#_Toc126600799)

[1.4 论文结构 4](#_Toc126600800)

[1.5 课题来源 5](#_Toc126600801)

[2 方案论证（或具体背景技术概述） 6](#_Toc126600802)

[2.1 系统需求分析 6](#_Toc126600803)

[2.2 系统需求分析 6](#_Toc126600804)

[2.3 系统可行性分析 7](#_Toc126600805)

[2.4 开发工具分析及选择 7](#_Toc126600806)

[2.5 关键技术分析 7](#_Toc126600807)

[2.6 基本方案制定 8](#_Toc126600808)

[2.7 本章小结 8](#_Toc126600809)

[3 XXX系统设计 9](#_Toc126600810)

[3.1 功能需求 9](#_Toc126600811)

[3.2 系统总体设计 10](#_Toc126600812)

[3.3 功能模块设计 14](#_Toc126600813)

[3.4 本章小结 16](#_Toc126600814)

[4 XXX系统实现 17](#_Toc126600815)

[4.1 过滤器实现 17](#_Toc126600816)

[4.2 属性管理模块实现 18](#_Toc126600817)

[4.3 数据迁移模块实现 19](#_Toc126600818)

[4.4 本章小结 22](#_Toc126600819)

[5 性能测试与分析 23](#_Toc126600820)

[5.1 测试环境 23](#_Toc126600821)

[5.2 功能测试 23](#_Toc126600822)

[5.3 系统界面 23](#_Toc126600823)

[5.4 性能测试 23](#_Toc126600824)

[5.5 本章小结 25](#_Toc126600825)

[6 总结与展望 26](#_Toc126600826)

[7 毕业设计撰写要求 27](#_Toc126600827)

[7.1 图的格式 27](#_Toc126600828)

[7.2 表的格式要求 31](#_Toc126600829)

[7.3 公式的格式要求 33](#_Toc126600830)

[7.4 流程图的格式要求 34](#_Toc126600831)

[7.5 常见格式问题 34](#_Toc126600832)

[致 谢 35](#_Toc126600833)

[参考文献 36](#_Toc126600834)

[附录：大学期间发表或提交的论文 39](#_Toc126600835)

# 绪 论

随着现代社会技术的不断发展，海量数据的产生对存储技术带来了新的挑战。本章首先介绍了SSD内部数据压缩技术的重要性，然后分析SSD内部压缩在数据布局管理方面上的技术与挑战，接着介绍了国内外在研究SSD内部数据布局管理的现有工作，最后对于本工作的主要目的和意义作了具体说明。

## 课题背景

### 研究背景和趋势

随着物联网、人工智能等计算机技术的不断进步，数据的产生途径变得多样化，同时数据的产生量也随之不断增长。现如今数据量已经呈现爆炸性的增长，现有SSD的容量不足以满足现在的需求。因此需要一种提升数据存储量的技术来帮助存储海量数据。

数据压缩技术[ips-4][ips-5z]作为一项通过降低数据冗余度来减少数据存储空间的辅助存储技术，该技术的出现在一定程度上缓解了海量数据的存储问题。现有的压缩通常被实现在三个层面，从顶层往下分别是应用程序层（数据库[1][2][3]）、文件系统层[4][5][6]、存储设备硬件层[7][8]。而现如今的两个技术趋势使得SSD上的压缩越来越普遍。一个是闪存的物理特性。反复的擦除和写入会逐渐对闪存单元形成不可逆转的磨损。这一工作原理决定了闪存会限制固态硬盘的寿命。另一个是摩尔定律，随着技术的不断发展，逻辑门尺寸的持续改进导致计算的成本越来越低。这两个趋势使得压缩技术成为SSD的一个明智选择，可以带来一些积极影响。首先，压缩技术减少了数据所需存储空间，这相当于增加了存储设备的暴露容量，同时也可以降低垃圾回收的开销。其次，压缩技术减少了NAND闪存单元的磨损。这不仅仅增加了SSD的寿命，而且还延缓了由于闪存老化而导致的性能下降。最后，压缩技术还可以减少I/O延迟。另一方面，压缩需要额外的硬件资源的支持，而且上述的积极影响在数据的不同时可能有较大的差异。

研究证明实现在存储设备的硬件上的压缩效果是最好的[9]。对于主机端压缩来说，本层次的压缩会抢占CPU资源；对于文件系统的压缩来说，块设备跟文件系统的接口协议固定了传输数据块的大小，因此压缩产生的变长数据块跟协议规定的数据块大小之间的不匹配会造成性能的下降，而且文件系统上频繁的压缩和解压也会提高读写的延迟，影响性能；对于压缩卡实现的硬件压缩，压缩卡有限的带宽是影响性能的瓶颈。这些问题都会随着压缩实现在SSD内部而解决。

### 面临的问题和挑战

**“内部碎片”问题。**与现有的用于NOR型闪存[10]、[11]的压缩技术不同，NOR型闪存提供字节I/O和快速读取操作[12]。而NAND型闪存中的数据压缩很难利用，因为它只支持页面I/O[13][ips ours 2-7][]。如果按照原来的数据存放方式（一个压缩页存储在一个物理页上）进行数据的存储，那么压缩带来的益处将会在很大程度上被削减。这是因为当压缩后的数据页大小小于NAND中一个物理页的大小，NAND闪存上的物理页在存储压缩后的数据时会产生内部碎片（没有被写入数据的部分），这严重降低了压缩的有效性。而一种合理的数据布局管理方案可以充分利用压缩带来的用户更少的空间使用、更小的写放大、更快的I/O速率等优点。

在此背景下，本课题复现了现有的两种效果较好的数据布局管理方案（RE-BP方案和COMPACTION方案），并且在复现过程中对各自的局限性进行分析和优化，提高其I/O性能和闪存的空间和利用率，设计并实现了一种新的数据布局管理方案。

## 国内外研究现状

现有的SSD内嵌压缩的数据布局管理方案有如下几种，CaFTL中CHUNK-BASED方案[14]、IPS方案[15]和COMPACT方案[16]。

### CHUNK-BASED方案

CaFTL的CHUNK-BASED方案通过增加一次压缩时页面的数量来降低“内部碎片”问题所带来的影响。该方案使用大小为4个物理页大小的CHUNK来作为I/O基本单元，同时其设置了一个大小CHUNK大小的缓冲区来缓存写请求的数据。当上层应用发起写请求时，写请求的数据会按照I/O次序存放在CHUNK中。只有当缓冲区被填满时，缓冲区里的数据才会被一起压缩，然后被刷新到FLASH之中。该布局方案旨在使一个CHUNK内的四个压缩页在闪存中是连续存放的，以此来减少存储四个压缩页的物理页数，提高空间利用率。但由于一个CHUNK里面可能含有多个被压缩之后的数据页，因此该方案中除了页映射级别的FTL表之外还需要一个管理表Data CHUNK Table来管理一个CHUNK里面的数据页信息。

CaFTL的CHUNK-BASED方案通过增加一起压缩的数据页的数量来提高对于闪存的空间利用率。但是其存在三个缺点，一是在读取数据时需要对包含所请求页面的整个CHUNK进行解压，增加了额外的开销；二是对大数据块的压缩需更多的硬件资源；三是需要额外的管理表Data CHUNK Table来管理数据页信息，这相比于传统的页级别映射FTL需要更多的内存空间。

### IPS方案

IPS方案通过将压缩后的数据页分组，并按组进行存储来减小内部碎片。IPS方案中设置若干个闪存物理页大小的缓冲区（下文称为组），对于一个I/O单元（一个物理页代销）进行一次压缩，并且在写缓冲区中临时维护一组压缩后的页。当上层应用发起写请求时，写请求的数据被压缩，然后采用三种策略（first-fit、best-fit、worst-fit）中的一种对压缩后的页面进行分组。当由于容量限制而无法将传入的压缩页存储到任何写缓冲组中时，内部打包方案将具有最小内部碎片的组之一刷新到FLASH之中。上述的三种分组策略的标准如下：first-fit策略将压缩的页面存储到第一个有足够空间的组中；best-fit策略贪婪地将压缩的页面放到一个组中，该组在执行该分组策略后会产生所有组中最小的内部碎片；worst-fit策略将压缩的页面放到一个组中，该组在执行该分组策略后会产生所有组中最大的内部碎片。现有的研究证明best-fit策略是三种策略中的最优策略。

IPS方案通过将压缩后的数据页进行策略分组来达到更高的闪存空间利用率，但是其分组策略会破坏文件的数据页连续特征，使得同属于一个文件的数据页被分散存储在多个物理页上，降低了读带宽。

### COMPACTION方案

COMPACTION方案直接将压缩后的数据页连续存储，同时允许一个压缩后的数据页存储在两个物理页上。该方案使用了一个小的页大小的NV-RAM缓冲区。该方案对每个主机写入的数据块分别进行压缩，然后将压缩块复制到NV-RAM缓冲区末尾。如果在复制过程中缓冲区溢出，它会将完全满的缓冲区刷新到FLASH中，然后清除缓冲区，并将压缩块剩余的末尾部分复制到缓冲区的开头。

COMPACTION方案通过将压缩后的数据页连续存放来达到极高的空间利用率，然而该布局方案使得一个数据页可能存储在两个物理页上，而且这两个物理页通常是不连续的。这会导致读取操作的平均延迟和尾延迟变长，同时也需要设计和使用更加复杂的FTL映射表。

### RE-BP方案

为了进一步提升IPS方案的存储空间利用率，RE-BP方案作为IPS的优化方案被提出。RE-BP方案的优化点在于进行排序再分组。当IPS方案的缓冲组无法容纳当前的压缩页而要将具有最小内部碎片之一的组刷新到FLASH时，RE-BP方案按照其数据量从大到小的顺序对缓冲组内所有的压缩页进行排序，而后按照同样的次序使用first-fit策略进行重新分组。如果重新排序之后产生足够的空间来容纳所有的压缩页则继续对下一个压缩页进行分组。如果没有足够的空间则将具有最小内部碎片的组刷新到FLASH中。

虽然RE-BP方案进一步提升了闪存的空间利用率，但是和IPS方案一样，RE-BP方案依然没有利用文件的数据页连续特征，而且RE-BP方案的重新排序在分组操作是发生在关键的I/O路径上，这增加了写开销。

### 其他方案

有对IPS方案的另一种优化方案，IPS{our}方案。该方案类似COMPACT方案，将压缩后的数据连续存储。每两个压缩后的数据块之间没有碎片空间。两者方案的区别在于对缓冲区的设置。

COMPACT方案由于其具有极小的碎片空间（总是小于一个页大小）和方案较为简单的特性被较为广泛的使用。国防科学技术大学的刘芳等人设计的FCTL方案[9]中也是使用了跟COMPACT一样的思想，将压缩后的数据连续存放。FCTL中对于存在于一个物理闪存页的数据使用一个三元组来记录，而对于存在于两个物理闪存页的数据采用两个三元组来记录。Yifan Qiao等人在2022年中利用透明压缩提升B+ tree性能的相关工作[10]中使用的也是COMPACT思想的数据布局管理方案。ScaleFlux研究团队在研究透明压缩来简化数据管理软件的工作中也使用了COMPACT思想的方案[11]。

## 研究目的和主要内容

### 课题研究的意义

对于SSD内嵌压缩的数据布局方案研究方案有助于充分利用有限的闪存空间，充分发挥压缩带来的用户更少的空间使用、更小的写放大、更快的I/O速率等优点。

### 课题研究的内容

1. 学习和理解SSD模拟器FEMU，了解FEMU项目对于I/O的模拟细节、对于FTL的实现方式、对于闪存页的管理方式等。
2. 理解RE-BP方案和COMPACTION方案，在FEMU项目上增加压缩/解压缩模块，实现对应的I/O操作、FTL表、闪存页的管理的修改，最后复现上述两个对于压缩后数据页的布局管理方案。
3. 设计并实现对RE-BP方案和COMPACTION方案优化的FELLOW方案。
4. 验证上述三个方案的实现的正确性并测试读写延迟、空间利用率等性能指标。

### 课题研究的目标

本次的课题研究中首先需要学习并深入理解FEMU项目，了解FEMU项目对于闪存页的管理方案。然后理解SSD内部的表现较好的两种数据布局管理方案——RE-BP方案和COMPACTION方案。接着在FEMU项目中复现上述的两种现有方案。然后设计针对上述两种方案的局限性进行优化的FELLOW方案并且在FEMU项目里进行实现。然后还需要对三种方案进行正确性验证和性能测试，进行三种方案的性能对比和数据分析。最终目的是实现课题要求。

## 论文结构

本文的主要内容如下：

第一章我们首先介绍了当前存储系统面临的挑战和技术发展趋势，然后分析了对象存储技术的产生及发展现状，介绍了国内外在存储策略和智能存储领域的相关研究工作，并对本文的主要研究内容及工作意义作了具体说明。

第二章首先介绍几种存储技术、特别是存储区域网（SAN）在安全性、数据共享和设备管理等方面的存在问题。然后详细论述了基于对象的存储如何解决当前的问题。最后讨论了对象存储的实现途径，OSD标准和Lustre文件系统代表了两种发展方向。本文将重点以OSD为基础进行有关研究。

第三章以磁盘阵列作为研究对象，描述了属性控制的数据放置策略。我们建立了基于闭环fork-join排队网络（CFJQN）的RAID-5读写模型，对最优分条单元大小进行了定量分析，并归纳得到可以用来准确描述一个负载的若干属性（包括存储需求和行为）。

第五章比较详细的描述了一个原型系统的设计和实现。iSCSI-OSD-RAID成功地对传统的RAID控制器进行了扩展，增加了支持iSCSI和OSD协议的访问接口。除了软件方面的工作之外，还实现了基于Intel IOP321处理器的控制器硬件，本章对硬件结构和交叉开发环境也进行了介绍。最后还给出了部分测试结果。

第六章总结了所做的工作，并计划了下一步的工作以及展望了基于对象存储的前景。

# 相关技术基础

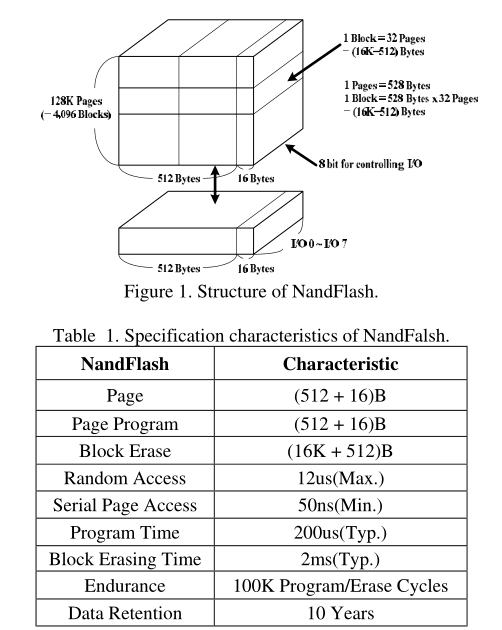
本课题研究的“内部碎片”问题本质上是因NAND FLASH的结构特性导致其只支持页面级别I/O所引起的。因此，为了开展本课题的研究，我们需要了解NAND FLASH相关的基本概念，还需要了解数据压缩技术（压缩发生的层次位置和压缩算法）。最后，由于时间和能力上的限制，本课题目前无法在真实的SSD上开展研究，而是基于SSD模拟器（FEMU）而开展。

## NAND FLASH的基本概念

文献（layer-fo-nand）[2][8][11-14]中讨论了Flash的标准结构。根据应用功能的不同，Flash可以分为Code Flash和Data Flash。根据内部存储结构的不同，Flash可以分为Nor Flash和Nand Flash。Nand Flash通过序列来读取数据，其仅使用8位I/O端口来访问页面单元的数据。因此，在读取数据或清理文件时，特别是对于非常大的文件，Nand Flash的访问速度比Nor Flash快。较低的随机访问速度是Nand Flash的主要缺点，因为它使用页面单元来访问数据。此外，与Nor Flash相比，Nand Flash具有更小的尺寸、更高的容量和便宜的价格，因此被广泛的使用。

文献（layer-fo-nand）[2][8][11-14]中讨论了Nand Flash的一些特征。Nand Flash的一个示例结构如图一所示。一个Nand Flash由很多个块（Block）组成的，块的大小一般是512KB、1MB、2MB。图中的是1MB。每一个块里面又包含了很多个页。以前的Nand Flash的页大小为256B、512B。这类Nand Flash被称为Small Block。现在常见的Nand Flash的页大小为2KB、4KB,甚至16KB。Nand Flash的读写操作以页面（4KB）为基本单位，而擦除操作以块（16KB）为基本单位。每一个页除了存储数据的区域之外还有另一块区域，叫做空闲区域（spare area）/冗余区域（redundant area），而Linux系统中，一般叫做**OOB**（Out Of Band），这个区域，是最初基于Nand Flash的硬件特性：数据在读写时候相对容易错误，所以为了保证数据的正确性，必须要有对应的检测和纠错机制，此机制被叫做EDC(Error Detection Code)/ECC（Error Code Correction,或者Error Checking and Correcting），所以设计了多余的区域，用于放置数据的校验值。通常来说一个大小为4KB的页搭配有128B大小的冗余区域。

Nand Flash具有两个功能，即快速写入和擦除操作。Nand Flash中块的擦除时间约为2ms[11]-[14]。由于用于实现顺序读取的数据总线和地址总线其实是同一总线，因此随机访问数据较慢，并且不能以字节为基本单位作为随机写入。但由于固态存储器的芯片尺寸小、引脚数量少，因此其位成本最低。



Nand Flash无法以一个字或一个字节为单位进行写入或清除。Nand Flash的清除操作的基本单元是块。对Nand Flash同一个地方进行重新编程代码之前，必须先对该区域所属的块进行清除操作。而由于清除操作和写入操作速度较慢，并且清除操作所清除的区域较大，因此清除操作和写入操作时间将更长。Nand Flash的另一个限制就是擦除时间长的问题[11][14]。由于这两个原因，传统的各种高效管理算法无法应用于Nand Flash。因此，Nand Flash文件管理系统发挥着重要作用，文献【6】对此展开研究分析，并已经取得一定成果。



## SSD内嵌压缩

SSD内部压缩技术已称为现今各大厂家和技术人员研究的一个热点技术。SSD内嵌压缩有另外一个名称——“透明压缩”。透明压缩是结合了存储和压缩的功能，在数据存储的I/O路径上完成对数据的压缩，而用户是感知不到数据压缩操作的发生，所以称之为"透明压缩"。存储中的透明压缩技术是使得计算存储驱动器商业化的最佳选择，具有两个主要原因：(1)适配性强：透明压缩不需要对现有的 I/O 软件堆栈（例如，文件系统、驱动程序和块层）和I/O接口协议（例如，NVMe和SATA）进行任何更改。这使得用户在无需修改任何代码和更改任何配置的情况下，仅仅使用实现透明压缩的计算存储器就可以将透明压缩技术集成和部署到现有的设备当中。(2)高效的性能：除了具有非常广泛的适用性之外，该技术还能在一定程度上提升设备性能。无损数据压缩技术（例如，LZ77算法及其变体）涉及大量随机访问，而这会导致较高的CPU Cache未命中率进而降低性能。透明压缩技术则可以通过减少数据所占用的存储空间来增加Cache所含有的数据数量，进而提高了CPU Cache命中率，提升性能。

现如今，具备透明压缩技术的计算存储驱动器已经被成功研发并且进行生产。ScaleFlux团队发布了世界上第一个PCIe CSD（CSD 2000），该存储器可以在内部对I/O路径上的数据块进行zlib压缩和解压缩，该压缩对于用户是不可感知的。经过不断且深入的研究，该团队发现该存储内透明压缩技术除了可以透明地降低存储成本和透明地提高 IOPS等优势之外，透明压缩在系统级别的创新方面也具有巨大的潜力。

数据在I/O路径上需要通过标准块 I/O 接口协议（例如，NVMe 和 SATA）以固定大小的块（例如，4KB）为单位对数据存储驱动器（例如 SSD、HDD、光盘和磁带）进行访问。在使用没有内置透明压缩普通存储驱动器时，计算机系统为了最小化数据的存储成本而需要使用有用的数据来填充每个固定大小的 4KB 块以满足I/O接口协议的约束。而固定大小的块I/O在一定程度上限制了上层的数据管理应用的设计（例如，关系数据库、键值存储、对象存储和文件系统）。举个例子，为了满足固定大小的块，B+树中所有树节点的大小必须相同且节点大小需为块I/O大小的倍数。这样的限制条件在理论上会其降低性能。而通过存储内透明压缩技术，我们不再需要使用有用的数据来填充数据块以降低数据的存储成本。因为无效的部分可以被简单又高效的压缩，而且只占有极小的存储空间。

因此，在使用现有的标准块 I/O 接口协议的条件下，现在固定大小的I/O块中具有的有效数据量是任意而可变的（例如，200B、1KB）。同时，数据真实的物理存储成本基本不受影响。这在本质上打破了现有的固定大小块 I/O 约束。

在过去的时间里，数据管理应用都或多或少的受到了固定大小的块I/O的限制，而存储中透明压缩技术支持可变大小的块I/O打破了这个限制，这带来了数据管理应用创新的可能。

## FEMU

传统存储系统中数据的可靠性通常是采用冗余技术或者备份技术来实现的，如果存储节点出现了如风扇损毁，磁盘温度过高、误码率过高代、性能下降等问题，系统通常不会进行主动处理，而是等待设备或者磁盘完全故障后才通过数据重建或热切换到镜像节点的方式保持存储系统的持续工作，这大大增加了数据风险性。本文提出了主动监控的思想，它对存储系统进行及时的控管，定时对系统中的各个存储节点工作温度、节点能耗当健康指标超出正常阈值时，及时产生预警信息，系统随后自动启动数据迁移，将数据迁移到适当的节点或磁盘，这就是存储设备健康预警及数据迁移技术。

## X-RL压缩算法

传统存储系统中数据的可靠性通常是采用冗余技术或者备份技术来实现的，如果存储节点出现了如风扇损毁，磁盘温度过高、误码率带过高、性能下降等问题，系统通常不会进行主动处理，而是等待设备或者磁盘完全故障后才通过数据重建或热切换到镜像节点的方式保持存储系统的持续工作，这大大增加了数据风险性。本文提出了主动监控的思想，它对存储系统进行及时的控管，定时对系统中的各个存储节点工作温度、节点能耗、数据误码率、传输性能等健康指标进行分析，当健康指标超出正常阈值时，及时产生预警信息，系统随后自动启动数据迁移，将数据迁移到适当的节点或磁盘，这是存储设备健康预警及数据迁移技术。

## 本章小结

本章提出了一种可以提高存储系统数据可靠性的技术，即存储设备安全预警及数据主动迁移技术，它包括磁盘数据自愈、磁盘数据主动迁移、阵列级数据自愈及系统级数据主动迁移四个层次。对磁盘数据自愈与数据主动迁移进行了详细描述。（第二章到倒数第二章每章都必须要有一个小结，用于阐述本章干了什么，篇幅不用太长，要与本章最开始的引言相呼应，除小结外，每章至少有3个小节）

# XXX系统设计

前面论述了存储接口从基于“数据块”发展到“对象”的必要性、重要性和可行性。基于对象的接口拥有更强的表达力，其基本手段是通过“属性”描述数据的信息。不同的属性内容服务于不同的目的。这里我们将以磁盘阵列为研究对象，重点是考虑需要获得什么样的负载属性信息，从而为存储对象选择合适的RAID级别和分条单元大小。

## 功能需求

（1）JGMSA30型加固磁盘阵列在磁盘存在一部分坏扇区时能够继续工作，当磁盘出现一部分扇区时，阵列并不立即使用新盘替换并重建数据，该磁盘通过自愈可以继续正常工作，为数据迁移提供时间窗口；

（2）能够将磁盘坏扇区数据恢复，根据磁盘所在RAID阵列对应的RAID算法，通过获取其它成员盘的同一Stripe上的数据由，RAID算法计算恢复出坏扇区数据；

（3）能够检测磁盘健康状况，并在磁盘健康状况不佳时将数据主动迁移到阵列中其它空闲磁盘中，迁移过程不影响正常业务流程，对系统性能影响应该降低到最低限；

（4）能够通过WEB界面进行统一监控，用户可通过WEB界面对磁盘自愈与数据主动迁移进行相关配置操作，例如可以配置磁盘健康参数、使用数据迁移的目标磁盘替换坏扇区磁盘等；

**（5）一般的系统都必须要有需求分析，有功能需求，界面需求，接口需求，性能需求等等。具体参照软件工程，这是系统设计的重要依据，一定要将自己做什么说清楚，如果第二章写了需求这章就不需要了**

**。。。。**

## 系统总体设计

### 系统硬件平台

JGMSA30型磁盘阵列采用嵌入式技术*[25-27]*实现，其硬件结构主要是在支持64位PCI-X接口的主板增加相应的芯片构成，例如网卡等相关芯片。将光纤芯片加入到该平台中，以支持光纤访问该阵列，同时还具有4块SATA芯片，能够连接多达16个SATA硬盘。为了提高系统计算校验的速度，该系统还加入了XOR硬件，使得RAID5及其它RAID级别计算校验信息的速度大大提高。给出了JGMSA30型加固磁盘阵列硬件结构图。**（有硬件系统需要提供硬件环境示意图）**



图3‑1磁盘阵列硬件平台

JGMSA30型加固磁盘阵列有三种方式与主机交互，分别为网络连接、光纤连接及串口连接，其中光纤是数据通路，即存储数据到阵列系统时，可以通过光纤连接传输数据，网络连接既是数据通路，也是配置信息通路，可以通过WEB浏览器配置RAID系统，此时通过网络连接交互，而当阵列配置成ISCSI访问时，网络连接也是数据通路，串口连接输出该阵列的调试信息，在实际使用时一般不连接。

### 系统软件架构

JGMSA30型加固磁盘阵列控制器上运行嵌入式Linux操作系统*[26-29]*，整个系统的软件部分是基于Linux系统开发的。为了提高该系统核心程序的可移植性，采用Linux内核模块的方式实现。给出了RAID系统核心程序的模块划分图，其中DH模块是需要设计实现的模块，它位于RAID核心程序与Linux操作系统提供的读写磁盘接口之间，RAID核心程序调用DH模块提供的读写磁盘接口，DH模块调用操作系统提供的读写磁盘接口真正读写磁盘，这样DH模块就能够过滤RAID系统发出的磁盘读写请求，分析请求执行状态，以实现健康状况的监测及对读写出错的扇区进行磁盘自愈，并在磁盘健康状况偏离正常时启动数据迁移。（软件整体结构图必须提供，有些系统还需要提供详细的动态运行分析图）



图3‑2 XXX软件架构

JGMSA30型加固磁盘阵列系统*[30-35]*软件部分主要分为6个模块，分别为RC阵列配置信息管理、CM模块、RAID核心算法模块、 DH模块以及用户配置程序接口模块。下面对这几个模块功能做简单介绍。

（1）RC阵列配置信息管理模块

阵列配置信息管理模块管理整个阵列配置信息，这些信息包括RAID系统中所接磁盘信息，以及这些磁盘所属的RAID组信息(RAID1、RAID5等)，还包括每个RAID组包含的逻辑卷的详细信息。

（2）AGST模块

AGST模块与光纤设备进行交互，负责接收用户发来的读写请求，并传输数据，它是整个RAID系统存储数据的窗户。

（3）CM模块

该模块负责管理RAID系统Cache，这部分用于加速用户访问逻辑卷的速度，将用户经常访问的速度存于Cache中，用户访问时，只需要与AGST模块及CM模块交互，不需要真正访问磁盘，能够提高I/O访问速度。计算公式如式3-1所示。

(3‑1)

(3‑2)

传统存储系统中数据的可靠性通常是采用冗余技术或者备份技术来实现的，如果存储节点出现了如风扇损毁，磁盘温度过高、误码率过高、性能下降等问题，系统通常不会进行主动处理，而是等待设备或者磁盘完全故障后才通过数据重建或热切换到镜像节点的方式保持存储系统的持续工作，这大大增加了数据风险性。本文提出了主动监控的思想，它对存储系统进行及时的控管，定时对系统中的各个存储节点工作温度、节点能耗、数据误码率、传输性能等健康指标进行分析，当健康指标超出正常阈值时，及时产生预警信息，系统随后自动启动数据迁移，将数据迁移到适当的节点或磁盘，这就是存储设备健康预警及数据迁移技术。

### 开发环境

整个系统的软硬件设计是同步进行的。在自己定制的控制器硬件可以使用之前，我们使用了IQ80321[[1]](#footnote-1)评估板作为软件开发平台。该硬件平台是目前开发网络存储设备比较好的选择。

操作系统的选择将对系统性能和开发过程产生重要影响。已经移植到80321的操作系统包括Nucleus，MontaVista Linux和VxWorks等。我们选择Linux作为嵌入式操作系统。创建嵌入式Linux系统的基本任务是在目标板上运行Linux内核。参考如图3‑3。所示的“主机/目标机”交叉开发环境，建立一个嵌入式ARM Linux系统需要下面4个步骤。



图3‑3 交叉开发环境

1. **准备交叉开发工具链**

工具链（toolchain）实际上包含许多部件，其中最重要的一个是gcc编译器，另一个是binutils（一组操作二进制文件的工具）。有了这些工具就可以编译内核了，不过其实通常情况下还需要C库glibc，以支持编译更多的应用。

1. **编译ARM Linux内核**

为了编译一个可以在IOP321上运行的Linux内核，需要为官方Linux内核打上一系列补丁。我们为linux-2.4.21官方内核源代码打rmk1，ds0和dj9补丁包，从而得到linux-2.4.21-rmk1-ds0-dj9版本内核。然后使用make命令即可调用arm-linux工具链生成内核映像文件zImage。后来我们又将内核升级到了linux-2.6.10-iop1。

1. **编译ARM Linux内核**

### 功能模块划分

磁盘自愈与数据主动迁移模块提供读写磁盘接口供其它模块调用，然后再调用操作系统提供的读写磁盘接口，需要对磁盘坏扇区、保留扇区及磁盘健康状况等信息进行管理，在磁盘健康状态不满足标准时开启数据迁移，同时该模块还需要提供相关配置接口，以提供配置及状态查询等功能，总结起来该模块可以分为六个功能模块：磁盘I/O过滤器、数据迁移、磁盘自愈、磁盘属性管理、磁盘状态检测及终端配置接口，模块划分图如**错误!未找到引用源。**所示。



图3‑4 磁盘自愈与数据主动迁移功能模块划分图

磁盘I/O过滤器用来过滤磁盘的读写请求，它由多个子过滤器组成，当上层发来磁盘读写请求时，各个子过滤器都会被调用。磁盘I/O过滤器会提供注册接口，用来安装子过滤器。在中，磁盘自愈及数据迁移功能模块都会调用注册接口安装子过滤器。数据迁移功能模块子过滤器截获磁盘I/O，当其请求的磁盘正在进行数据迁移时，则可能需要将请求重定位到数据迁移的目标磁盘。

## 功能模块设计

磁盘自愈与数据主动迁移模块可以划分为磁盘I/O过滤器、磁盘属性管理、磁盘自愈、会调用磁盘自愈功能模块，当磁盘健康状况不满足标准时，则会调用数据迁移功能模块，因此，磁盘I/O过滤器功能模块在磁盘自愈与数据主动迁移模块中非常重要。

### I/O过滤器

读写过滤器是磁盘自愈与数据主动迁移模块的驱动器，同时也是整个数据流动的入口，该功能模块的设计对于整个模块的可移植性非常关键。在设计中使用了分层次过滤磁盘I/O的思想，该功能模块提供过滤框架及注册接口，其它功能模块可调用该功能模块提供的注册接口安装过滤器，过滤磁盘I/O。在磁盘I/O过滤器的上层，提供读写磁盘的接口以供RAID系统其它模块调用，而在过滤器的最底层，则会调用操作系统提供的磁盘读写接口，真正读写磁盘。

### 属性管理

磁盘属性管理是整个磁盘自愈与数据迁移模块的核心功能模块，它管理磁盘的健康状况信息，并提供接口供磁盘I/O过滤器模块调用以报告对磁盘读写的操作结果。磁盘属性管理模块会维护磁盘健康状态信息，当信息发生改变时，会将相关信息存储到磁盘特定位置，以防止磁盘健康状况属性的丢失。状态信息反馈包括磁盘健康状况、磁盘自愈工作情况以及数据迁移进度等。图3‑5描述了磁盘属性管理功能模块的划分图。



图3‑5 磁盘属性管理功能模块划分

### 放置策略

在我们实现的原型系统中，定义了几种Region类型，分别具有不同的配置和特征（表3‑1）。其中，Type\_7相当于一个RAM Disk，当然具有比磁盘低的多的响应延迟，可以用来放置那些生存时间非常短（不超过几秒种）的对象，通常是应用程序运行过程中产生的临时文件和同步锁文件。另外三种类型分别对应了三种最常用的RAID级别，并具有不同的数据块大小，支持一定范围的应用负载。（所有表必须插入题注，注意在表格的上方，编号形式为X-X，2级编号即可，不要3级编号）

表3‑1 原型系统实现的几种类型的Region配置

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Type 0 | Type 1 | Type 5 | Type 7 |
| RAID级别 | 0 | 10 | 5 | N/A |
| 磁盘个数 | 8 | 8 | 8 | N/A |
| 分条单元大小 | 32KB | 16KB | 32KB | N/A |
| 数据块大小 | 512KB | 16KB | 64KB | 1KB |
| 数据块个数 | 8K | 8K | 8K | 8K |

在上一章我们曾经定义了一个“用户指定负载特征”扩展属性页（Page\_8000h）。该属性主要用于OSD的创建对象（CREATE）命令。利用其属性#1，用户可以直接指定该对象放置在什么类型的Region，这个属性主要是为了留给我们自己做测试使用，因为一般用户不需要了解OSD内部对象放置策略的细节。

## 本章小结

本章详细描述了RAID系统中磁盘自愈与数据主动迁移模块的设计，对其主要功能模块进行了详细的设计说明。并就各功能模块设计时需要考虑的问题及解决方案进行了讨论。

# XXX系统实现

我们将设计实现一个原型系统。该原型系统在传统的（SBC）磁盘阵列控制器的基础上，根据SCSI OSD标准实现了新的访问接口，从而改造为一款OSD磁盘阵列。支持OSD协议的同时，我们还将改变RAID的数据组织，设计新的磁盘布局，使之拥有在负载属性控制之下根据数据放置策略优化数据分布的能力。

## 过滤器实现

### 过滤器注册数据结构

*typedef struct \_DEVICE\_T{*

*ADD\_DEVICE\_FUN AddDeviceFun; //过滤器增加设备通知函数*

*DELETE\_DEVICE\_FUN DeleteDeviceFun; //删除设备*

*DELETE\_DEVICE\_FUN PrevDeleteDeviceFun; //预删除设备*

*DELETE\_DEVICE\_FUN CanDeleteDeviceFun; //设备是否可删除*

*struct \_list\_t{ //各层过滤器连接*

*void \*next; //该层过滤器的下层过滤器*

*void \*prev; //该层过滤器的上层过滤器*

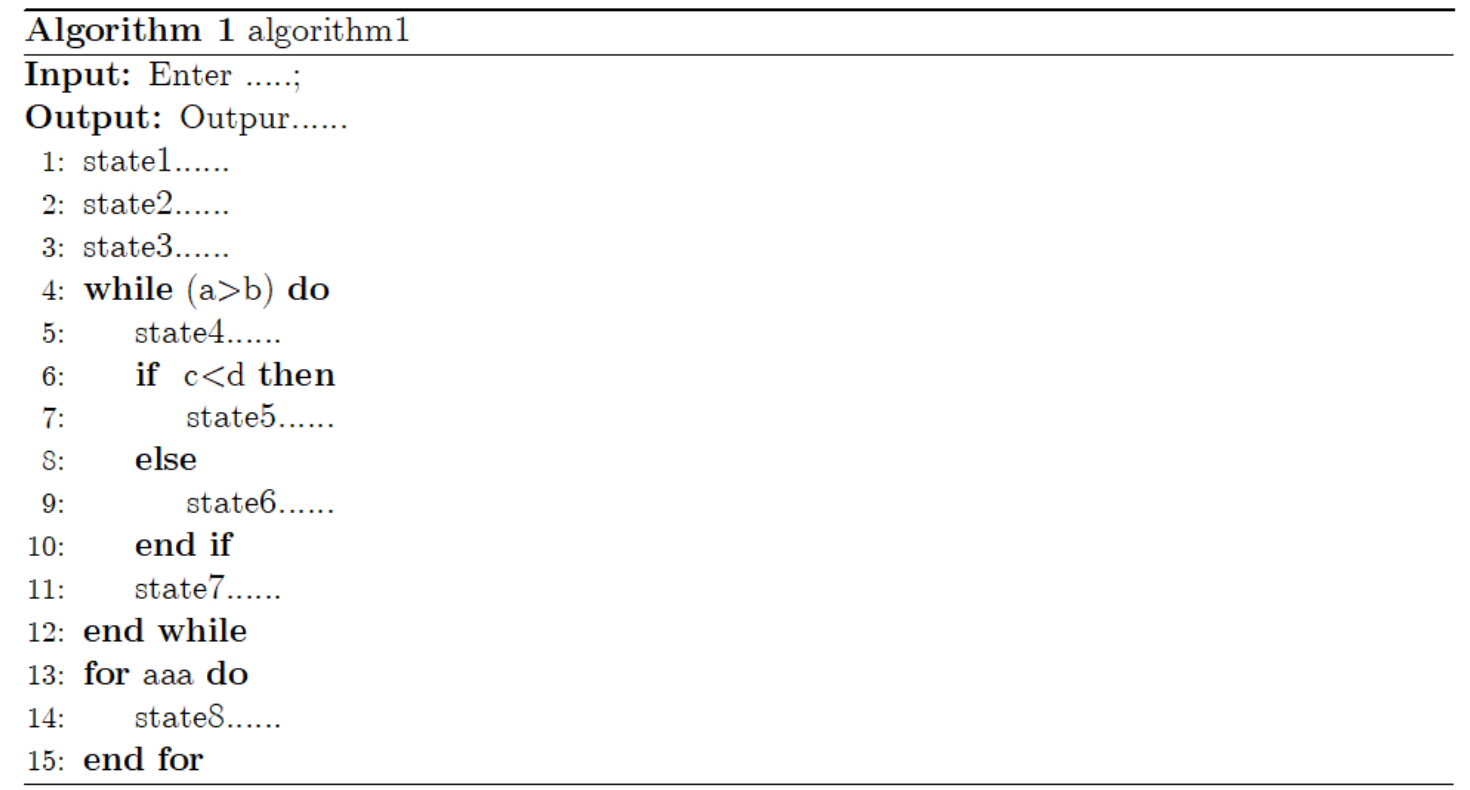
*}list\_entry;*

*char MyData[1]; //该层过滤器私有数据*

*}DEVICE\_T;*

(论文中出现代码时请遵循以上代码格式，灰色底纹，斜体字，注意论文中不能出现过多的代码，只能出现一些关键数据结构。系统实现过程尽可能的用程序流程图表示。)数据结构中比较重要的字段是IRP\_PROCESS\_FUN FunArray，它是过滤读写请求的接口集合，是一组函数指针，指向该功能模块提供的过滤读写请求的接口，这些接口在读写请求到来时会被调用。其它字段主要用于注册设备管理使用，例如ADD\_DEVICE\_FUN AddDeviceFun，各功能模块注册的过滤器不仅过滤读写请求，而且需要得到磁盘删除增加方面的通知(这一部分为了磁盘管理而使用)。

**论文中的关键算法可以使用流程图也可以使用伪代码给出，伪代码格式如下，建议采用Aurora插件在word和wps中自动生成，相关安装使用方法请自行百度。**



### 过滤器工作流程

加入磁盘I/O过滤器以后，DH模块提供给RAID系统使用的读写磁盘接口将RAID系统发送的磁盘I/O请求导入顶层过滤器，即g\_pTopDevice，调用顶层过滤器提供的处理读写I/O请求的接口，这些接口对请求进行处理后会调用它下层的过滤器将请求继续向下传递，这样每层过滤器就都能够过滤到读写请求。到达过滤器链表中的最底层就直接调用SCSI中间层提供的读写磁盘接口。当读写I/O请求返回时，会从最底层向最上层传递，这样各层既能够过滤请求，也能够分析该请求处理的结果。

## 属性管理模块实现

磁盘属性管理包括磁盘健康状况统计、数据迁移信息的管理、磁盘自愈信息管理及与配置终端接口功能模块交互等功能。磁盘健康状况统计是提供相关接口供其它功能模块调用，以更新磁盘健康状况信息。数据迁移信息的管理，是指对数据迁移中的源磁盘、目标磁盘、迁移进度及迁移状态位图等信息进行管理(对这些信息的详细说明在数据迁移功能模块实现细节中)。磁盘自愈信息管理，是对保留扇区的管理以及映射表信息的管理。

### 属性管理数据结构

*typedef struct \_DH\_DISK\_INFO\_T*

*{ int uMagic;*

*DH\_TWINS\_DISK\_T stTwinsDiskInfo; //数据迁移管理信息*

*DH\_DISK\_CONFIG\_T stDiskConfigInfo; //用户配置信息*

*DH\_DISK\_HEALTH\_INFO\_T stDiskHealthInfo; //磁盘健康相关信息*

*DH\_SECTOR\_INFO\_T stSectorInfo; //磁盘预留扇区相关信息*

*MEM\_POOL\_ROOT\_T stSectorMapTable;//坏扇区预留扇区映射表内存*

*PDH\_SECTOR\_MAP\_T pstHashLink[HASH\_TAB\_LEN]; //替换映射表*

*}DH\_DISK\_INFO\_T;*

磁盘属性管理数据结构包括uMagic(标识，由于属性信息需要存储到磁盘，因此需要标识该数据项是否有效)、数据迁移管理信息(迁移源磁盘、目标磁盘、迁移进度及迁移状态位图)、磁盘健康相关信息、预留区管理及映射表相关信息。磁盘健康相关信息的更新目前主要依靠I/O出错次数以及磁盘状态检测模块通过分析磁盘SMART信息得出的预测结果来决定。

### 属性管理模块工作原理

磁盘属性管理功能模块负责资源管理，提供一系列接口供外部模块调用。该模块在系统启动时，从磁盘相应的位置将磁盘属性读出来，并初始化好替换映射表等信息。在磁盘属性改变时，会将磁盘属性保存到磁盘中。

## 数据迁移模块实现

### 数据迁移相关数据结构

在数据迁移时，使用上述数据结构描述迁移源磁盘与目标磁盘信息，uMagic用来标识是否数据迁移信息有效，uSourceDiskSerial、uDestDiskSerial分别是源磁盘与目标磁盘的序列号，uSynPercent代表迁移进度，pBitMap是迁移状态位图，标明哪些数据块迁移完成。

### 数据迁移请求处理

数据迁移需要从源磁盘读出一块数据，然后将这块数据写到目标磁盘中，需要考虑到迁移写操作与正常写操作的同步问题。为保证请求处理的同步，DH模块对将所有请求的处理都放在一个线程中，其它模块需要读写磁盘时，将请求挂到请求处理队列中。给出了代理线程处理数据迁移请求及其它类型请求大体流程。



图4‑1 代理线程处理请求及响应

对于数据迁移请求的处理，代理线程DH\_MainThread线将源磁盘数据读出来，然后再将数据写入到目标磁盘中。在这两步处理中，也有响应加入队列的过程，这部分流程详细执行如下：

（1）使用异步的方式读源磁盘，构造SCSI请求并注册回调函数，将请求发给源磁盘；

（2）下层处理完成，调用回调函数通知，回调函数将响应挂回队列；

（3）DH\_MainThread进入下次循环，取出读源磁盘响应；

### 数据迁移速度调整

降低数据迁移速度是通过在后台迁移线程中增加对迁移队列深度的控制以及增加延迟来实现，该部分控制流程如**错误!未找到引用源。**所示。



图4‑2 迁移速度控制流程

（标准的流程图一定要有开始和结束当然有些程序没有结束，请注意流程图的画法，Yes/No分支应该清晰的标注在菱形分支附近，而不是很远，此图中少了一个N）首先根据检测到的I/O负载，判断磁盘阵列当前I/O负载，如果I/O负载过重(这种情况下通过单一的队列深度控制来调整速度已经无法满足要求)，则每迁移一部分数据就睡眠一段时间(这里设置为50MS)，然后根据I/O负载调整迁移请求队列深度。迁移请求队列是后台迁移线程初始化的用来保存数据迁移请求的队列，通过控制其深度，能够控制后台迁移线程序每次发给I/O代理线程的数据迁移请求数目，如果深度较小，那么每次发给I/O代理线程的请求数就较小，数据迁移速度就会降低，否则会提高数据迁移速度。

## 本章小结

本章描述了磁盘自愈与数据主动迁移模块的具体实现，然后对各功能模块的具体实现进行了详细分析，最后介绍了该模块的动态运行流程。

# 性能测试与分析

## 测试环境

我们首先对RAID软件进行了评估。测试平台配置为：CPU Intel Xeon 1-8GHz，512MB内存，LSI Logic 21320-R (1030) ULTRA320 SCSI HBA，连接4块Seagate Cheetah Ultra320磁盘（MODEL ST373307LC），一块Agilent 5221A FC HBA（2GB），与主机（启动器）的Qlogic 2310F FC卡通过光纤连接。为评价磁盘阵列控制器最终性能，我们利用Iometer软件对阵列控制器进行了详细的性能评测，具体测试环境见表5‑1：

表5‑1 光纤通道启动器与阵列控制器配置

|  |  |  |
| --- | --- | --- |
| 配 置 | 启动器 | 阵列控制器 |
| CPU | Xeon 2.0 | Intel 80321 |
| Memory | 512MB | 1GB |
| Fiber Channel HBA | Qlogic 23102G | Agilent DX2 |
| OS | Windows 2003 | ARM Linux |

## 功能测试

## 系统界面

不是必须

## 性能测试

其中阵列控制器挂接6个Maxtor STM3320820AS磁盘，利用6个磁盘分别构建RAID0、RAID10、RAID5三种不同的阵列级别，分条大小为128K字节，缓存大小为512MByte，LUN大小为500GByte，阵列控制器采取写回策略。



图5‑1 磁盘阵列控制器RAID5顺序读写IOPS性能

**错误!未找到引用源。**为6个磁盘构建RAID5时阵列控制器顺序读写IOPS(I/O Per Second)性能示意图，从图中可以看出，IOPS随着数据块大小变大而变小，当数据块大小为512Byte时，阵列控制器处理读请求的性能可以达到每秒15000个，写性能由于包含冗余校验信息计算，IOPS略低，约为12000，此指标高于同类美国阵列nStor 4502F，其写IOPS最大为7000左右。



图5‑2 磁盘阵列控制器RAID5读写性能

为6个磁盘构建RAID5时阵列控制器数据传输率示意图，从图中也可以看出，随着主机端I/O请求数据块大小逐渐增大，阵列控制器数据传输率逐渐增加，当数据块大小达到128Kbyte时，系统性能最佳，读性能可以达到185MByte/s，写性能约为167MByte/s。**（有坐标轴的图必须给出单位，坐标轴意义，所有测试图应该有详细分析）**

从总体性能评测结果看，本文最终设计的FC-SATA磁盘阵列控制器基本发挥了Intel IOP处理器的最佳性能，同比性能优于同类进口磁盘阵列，基本达到了设计要求，如需要更高的数据传输率，必须同时提升主机通道以及IOP本身的性能，才能有实质性的提高。

## 本章小结

本章给出了控制器完整的软硬件设计方案，提出了主机板与通道板分离的架构，可以灵活构建全系列阵列控制器产品线；提出了磁盘阵列软件分层异步体系结构，可以有效提高I/O请求处理的并行度，减少I/O处理同步等待时间；最后提出了一种针对阵列层次性结构的细粒度测试方法，可单独测试评价各模块性能，为系统性能精确评测，以及故障诊断提供决策依据，最后给出了阵列控制器性能综合评测。

# 总结与展望

基于对象的存储是为了克服当前基于块的存储存在的诸多难题，在存储接口和结构层次的重要发展。基于对象的存储设备可以具有更多智能性，可以根据应用负载选择优化的存储策略。实现属性控制的存储策略，关键需要解决两个方面的问题：第一，属性表达什么信息，怎样影响存储策略？第二，属性怎样从用户应用传递给存储设备？围绕这两个问题，作了如下几点研究和工作：

1. 提出了对象属性控制存储策略的模型。OSD协议作为SCSI的扩展集，可以在TCP/IP网络和iSCSI协议之上来传输，基于对象的存储设备接收OSD命令并进行处理。
2. 结合对应用负载特征的分析，提出根据对象属性描述的负载特征动态选择RAID级别和分条单元大小的数据放置策略。I/O请求较小的对象放在RAID-10中，同时分条单元尺寸较小；反之则放在RAID-5中。
3. 建立了属性的传递机制，一种是应用程序接口静态传递；另一种是动态机制，即根据属性当中包含的文件系统信息预测文件的分类，并根据分类做出进一步的存储策略选择。
4. 论文总结必须用这样的条目的形式给出，总结是对全文的总结，不是对毕业设计的心得，心得部分不要写。

本文为更好地认识基于对象的存储以及OSD的实现进行了有益的探索。该领域还有很多研究工作有待完成，包括：

1. 以OSD为节点构建大规模（PB级）存储系统；
2. 属性控制的caching/prefetching策略；
3. 实现Device-aware（表达设备能力，按需分配资源）；
4. 设备的学习能力，以文件系统和历史访问信息等为提示进行预测。

随着存储需求的爆炸性增长，下一代互联网络必须有新的存储结构来应对构建和管理更大规模存储系统的挑战。基于对象的存储是最有希望的突破点之一。相信存储技术的发展一定能够满足人们不断增长的存储需求。

# 毕业设计撰写要求

## 图的格式



图7‑1 图注必须在图的正下方

正文中所有插图要求图面整洁，布局合理，线条粗细均匀，大小适中。插图必须是矢量图，不能是从他处直接拷贝位图，所有曲线、图表、线路图、流程图、程序框图、示意图必须按国家规定标准或工程要求采用visio等工具绘图后复制到word。插图在论文中采用的是正文样式，所以会自动缩进，请调整标尺取消缩进，让插图严格居中，图表中的文字大小必须小于正文字号。

### 图的题注

所有插图均应有图号和图名（office中称题注，图注），图注出现在插图的正下方，小四黑体居中，在本模板中可以直接使用样式库中的题注样式即可规范格式，图号和图名后面不加标点符号，也不得加参考文献引用。

图号按章编编号，如第2章的图为图2-1、图2-2、…，第3章的图为图3-1、图3-2、…等，第一个编号是章号，第二个编号是本章的图的序号，图名是插图的名称，扼要概括图的内容，字数不宜太多。

图的题注可在引用菜单栏通过插入题注的方式插入**（最简单的方法是复制其他图的图注然后更新域即可）**，引用时采用交叉引用引用，注意题注，交叉引用是word很重要的概念，是office排版的高级技巧，撰写毕业设计论文时应研究学习一下此功能，如果插入题注没有对应的图题注，只有figure等，可以自己创建，具体流程如**错误!未找到引用源。**，**错误!未找到引用源。**所示，需要注意的是，由于office不是为中文排版设计，图编号前面会增加一个空格，建议手工删除。

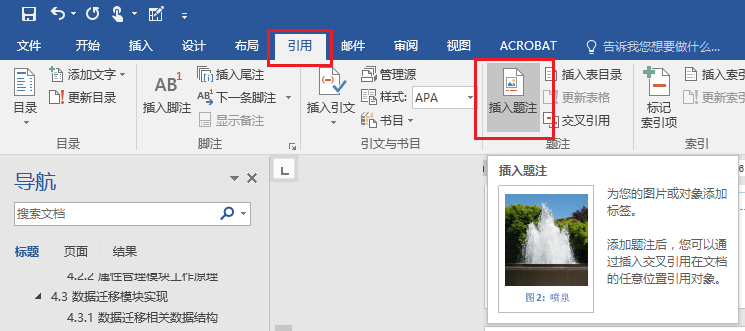


图 7‑2 插入图的题注

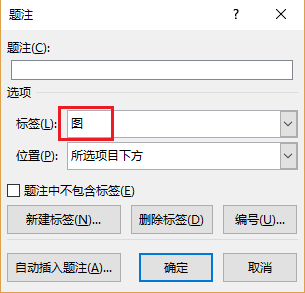


图 7‑3 插入题注对话框

### 图的引用

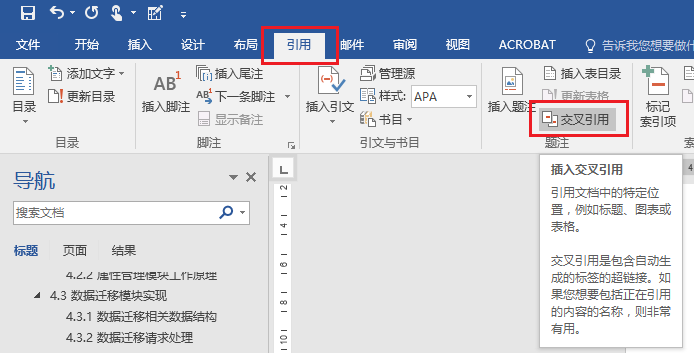


图7‑4 交叉引用

所有插图均应在正文中予以引用。引用某插图时，一般写为“…如图2-1”或“图3-2是…”。正文中的插图均须安排在文中第一次引用到该图的正文下面，一般要求先见文，后见插图，且图一般不跨页绘制。

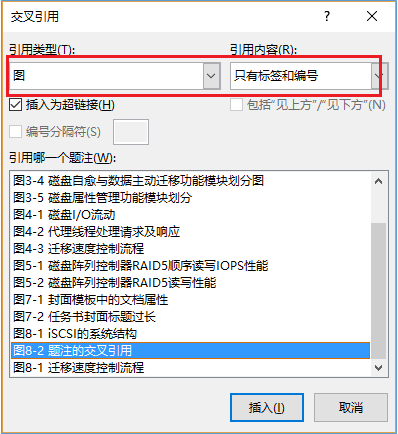


图7‑5 插入交叉引用对话框

手工输入图引用的方式非常不方便，如果图编号发生更改要全文修改编号，Office中提供交叉引用的功能进行题注的自动引用，方便编号自动更新，如**错误!未找到引用源。**所示，点击交叉引用后得到**错误!未找到引用源。**所示对话框，选择合适的图编号即可，注意引用内容选择只有标签和编号。

### 错误的插图格式

**错误!未找到引用源。**所示图的排版中插入时未取消左侧的格式缩进，右侧超出了页边距，专业术语称之为“出血”现象，请一定要仔细核对插图以及图注的缩进问题，插图和图注应该严格居中，插图不得超出页边距。另外文章红色字体过大，一般也是有问题，建议所有图标中的文字字体均小于正文字体。

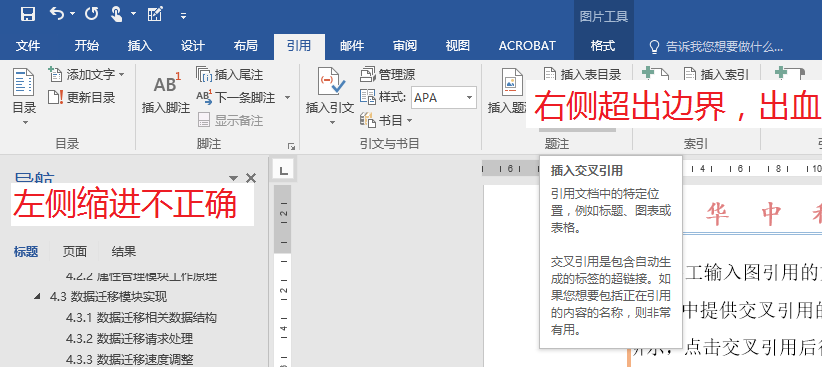


图7‑6 缩进错误，出血错误实例

不得出现下面这样大段的空行或留白现象，每章最后一节除外。

不得出现下面这样大段的空行或留白现象，每章最后一节除外。

## 表的格式要求

表的格式要求基本和图一致，注意图表中的文字均需要小于正文字体大小，与图注位置相反，表的题注必须在表的正上方。

表7‑1 表头必须在表的正上方

|  | Type 0 | Type 1 | Type 5 | Type 7 |
| --- | --- | --- | --- | --- |
| RAID级别 | 0 | 10 | 5 | N/A |
| 磁盘个数 | 8 | 8 | 8 | N/A |
| 数据块大小 | 512KB | 16KB | 64KB | 1KB |
| 数据块个数 | 8K | 8K | 8K | 8K |

表格通常采用三线表形式，三线表通常只有3条线，即顶线、底线和栏目线（见图，注意：表两侧没有竖线）。其中顶线和底线为粗线，栏目线为细线。当然，三线表并不一定只有3条线，必要时可加辅助线，但无论加多少条辅助线，仍称做三线表。

所有表均应有表号和表名（office中称题注，表注），表注出现在表的正上方，小四黑体居中。表号按章编，如第2章的表为表2-1、表2-2、…，第3章的表为表3-1、表3-2、…等。表名是表格的名称，扼要概括表的内容，字数不宜太多。

表头包括栏头、行头，与表身一起构成表格的主体。表中的竖称为栏，横格称为行。表身的内容，一般包括：数据、文字、公式和表图等。表内的数据对应位要对齐。少数表有表注，表注写在表下面且字号应比表号、表名的字小一号。

所有表格均应在正文中予以引用，具体引用方式和图的引用方式类似，如**错误!未找到引用源。**所示。引用某表格时，一般写为“…见表1-2”或“表3-1是…”。表格应尽量靠近正文的叙述，一般应先见文，后见表，表不跨节。表格允许转页。表格转页部分可以不写表号和表名，但要重复书写表头，具体设置如**错误!未找到引用源。**所示。

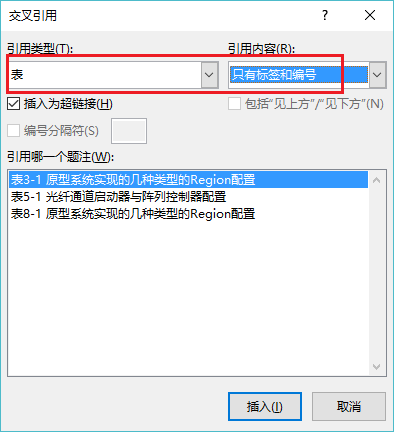


图7‑7 插入表注对话框

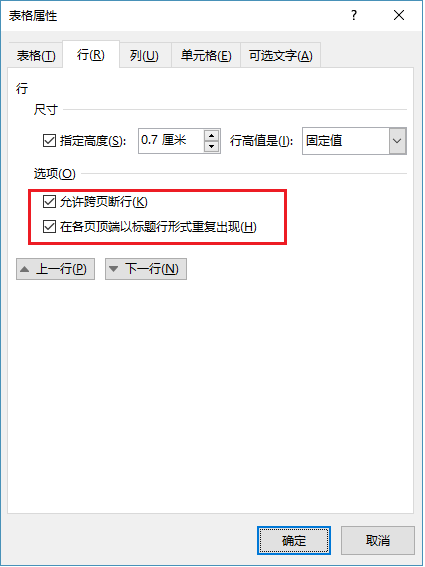


图7‑8 表格跨行设置

## 公式的格式要求

常见公式排版格式如下：

(7‑1)

公式必须进行标注，公式中的变量必须用斜体，标注要居右，公式居中，要做到这一点可以在标号前面加空格进行调整。

(7‑2)

公式一般另行居中写，公式末不加标点。若公式前有文字，如例、解、证、假定等，文字顶格写，公式仍居中写。一行写不下时，公式允许转行。公式转行需处理得当，做到既意义正确，又使版面美观匀称。

公式要有编号，公式编号按章编，如第2章的公式为(2-1)、(2-2)、…，第3章的公式为(3-1)、(3-2)、…等。公式编号写在公式右侧行末顶边线，并加圆括号。模板中公式编号已经按照图题注的方式给出，可以直接复制然后更新域自动排序。。

公式一般应在正文中予以引用，引用时以公式编号指示公式。正文中常有公式中表示量的符号说明，采用“式中”二字作为标志。一般可写成接排形式，如“式中，A指……；B指……”。

## 流程图的格式要求



图7‑9 迁移速度控制流程

（标准的流程图一定要有开始和结束当然有些程序没有结束，请注意流程图的画法，Yes/No分支应该清晰的标注在菱形分支附近，而不是很远。

## 常见格式问题

未定义标签、各章之间未分页、标题格式不正确、章节标题不合适、缩进不规范、行距不统一、不必要的留白和空行、图表中字体明显大于正文、图表比例失调，有出血现象、表注不规范、表格排版欠美观

致 谢

论文完成之际，首先要感谢我的导师郭德纲教授。他站在学科发展的前沿，从论文的选题，研究工作逐步深入，到论文的撰写，都给我以细致的指导和建设性的意见，使我得以圆满而顺利地完成……。XXX严谨的治学态度、诲人不倦的师德和一丝不苟的工作作风将会给我留下不可磨灭的记忆……。

在…… 四年里，得到了许多老师和同学的大力帮助和支持，在此表示深深的谢意。感谢XXX等老师对我的关心和帮助。在课题研究和项目开发过程中，我与很多同学一起度过了一段段难忘的时光，值得怀念。我们大家共同创造的良好的学术氛围，将给我以永远而美好的回忆。

……

最后，我要深深地感谢我的父母，他们给予我无尽的关怀和无止境的爱，….

参考文献

1. Palkar S, Lan C, Han S, et al. E2: a framework for NFV applications[C]. Symposium on Operating Systems Principles. ACM, 2015:121-136.
2. Bremler-Barr A, Harchol Y, Hay D. OpenBox: A Software-Defined Framework for Developing, Deploying, and Managing Network Functions[C]. Conference on ACM SIGCOMM 2016 Conference. ACM, 2016:511-524.
3. Sekar V, Egi N, Ratnasamy S, et al. Design and implementation of a consolidated middlebox architecture[C]. Usenix Conference on Networked Systems Design and Implementation. 2011:24-24.
4. Anderson J W, Braud R, Kapoor R, et al. xOMB: extensible open middleboxes with commodity servers[C]. Eighth ACM/IEEE Symposium on Architectures for NETWORKING and Communications Systems. ACM, 2012:49-60.
5. A. Gember, R. Grandl, A. Anand, T. Benson, and A. Akella. Stratos: Virtual Middleboxes as First-class Entities[R]. Technical report, UW-Madison 2012.
6. Zhang W, Liu G, Zhang W, et al. OpenNetVM: A Platform for High Performance Network Service Chains[C]. The Workshop on Hot Topics in Middleboxes and Network Function Virtualization. ACM, 2016:26-31.
7. Gember-Jacobson A, Viswanathan R, Prakash C, et al. OpenNF: enabling innovation in network function control[C]. ACM Conference on SIGCOMM. ACM, 2015:163-174.
8. Rajagopalan S, Dan W, Jamjoom H, et al. Split/merge: system support for elastic execution in virtual middleboxes[C]. Usenix Conference on Networked Systems Design and Implementation. 2013:227-240.
9. J. Khalid, A. Gember-Jacobson, R. Michael, A. Abhashkumar, and A. Akella. Paving the Way for NFV: Simplifying Middlebox Modifications Using StateAlyzr[C]. In Proc. of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI’16), 2016.
10. Ballani H, Costa P, Gkantsidis C, et al. Enabling End-Host Network Functions[J]. ACM SIGCOMM Computer Communication Review, 2015, 45(4):493-507.
11. Paxson V. Bro: a system for detecting network intruders in real-time[C]. Conference on Usenix Security Symposium. USENIX Association, 1998:3-3.
12. Yoan N. Iptables[M]. Miss Press, 2013.
13. Sherry J, Gao P X, Basu S, et al. Rollback-Recovery for Middleboxes[C]. ACM Conference on Special Interest Group on Data Communication. ACM, 2015:227-240.
14. Chrome G. HTTP persistent connection[J]. 2015.
15. Surhone L M, Tennoe M T, Henssonow S F, et al. Ffmpeg[M]. 2010.
16. Housley R, Hoffman P. Internet X.509 Public Key Infrastructure Operational Protocols: FTP and HTTP[J]. Ietf Rfc Sri Network Information, 1999, 11(3):82--89.
17. Maheshwari A, Sharma A, Ramamritham K, et al. TranSquid :Transcoding and Caching Proxy for Heterogenous E-Commerce Environments[C]. International Workshop on Research Issues in Data Engineering: Engineering E-Commerce/e-Business Systems. IEEE Computer Society, 2002:50.
18. J. Khalid, A. Gember-Jacobson, R. Michael, A. Abhashkumar, and A. Akella. Paving the Way for NFV: Simplifying Middlebox Modifications Using StateAlyzr[C]. In Proc. of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI’16), 2016.
19. NFV Paper [EB/OL].https://portal.etsi.org/NFV/NFV White Paper2.pdf,2017-05-05.
20. Hwang J, Ramakrishnan K K, Wood T. NetVM: high performance and flexible networking using virtualization on commodity platforms[C]. Usenix Conference on Networked Systems Design and Implementation. USENIX Association, 2014:445-458.
21. S. Han, K. Jang, A. Panda, S. Palkar, D. Han, and S. Ratnasamy. SoftNIC: A Software NIC to Augment Hardware[R]. Technical report, EECS Department, University of California, Berkeley, 2015..
22. Martins J, Ahmed M, Raiciu C, et al. ClickOS and the art of network function virtualization[C]. Usenix Conference on Networked Systems Design and Implementation. USENIX Association, 2014:459-473.
23. Panda A, Han S, Jang K, et al. NetBricks: taking the V out of NFV[C]. Usenix Conference on Operating Systems Design and Implementation. USENIX Association, 2016:203-216.
24. A. Gember, R. Grandl, A. Anand, T. Benson, and A. Akella. Stratos: Virtual Middleboxes as First-class Entities[R]. Technical report, UW-Madison 2012.
25. Palkar S, Lan C, Han S, et al. E2: a framework for NFV applications[C]. Symposium on Operating Systems Principles. ACM, 2015:121-136.
26. Rajagopalan S, Dan W, Jamjoom H. Pico replication:a high availability framework for middleboxes[C]. Symposium on Cloud Computing. 2013:1-15.
27. Sherry J, Gao P X, Basu S, et al. Rollback-Recovery for Middleboxes[C]. ACM Conference on Special Interest Group on Data Communication. ACM, 2015:227-240.
28. Qazi Z A, Tu C C, Chiang L, et al. SIMPLE-fying middlebox policy enforcement using SDN[J]. Computer Communication Review, 2013, 43(4):27-38.
29. Zhang W, Hwang J, Rajagopalan S, et al. Flurries: Countless Fine-Grained NFs for Flexible Per-Flow Customization[C]. International on Conference on Emerging NETWORKING Experiments and Technologies. ACM, 2016:3-17.
30. Bremler-Barr A, Harchol Y, Hay D. OpenBox: A Software-Defined Framework for Developing, Deploying, and Managing Network Functions[C]. Conference on ACM SIGCOMM 2016 Conference. ACM, 2016:511-524

附录：大学期间发表或提交的论文

1. A. Gember, R. Grandl, A. Anand, T. Benson, and A. Akella. Stratos: Virtual Middleboxes as First-class Entities[R]. Technical report, UW-Madison 2012.
2. Palkar S, Lan C, Han S, et al. E2: a framework for NFV applications[C]. Symposium on Operating Systems Principles. ACM, 2015:121-136.

1. (http://www.intel.com/design/IIO/) [↑](#footnote-ref-1)