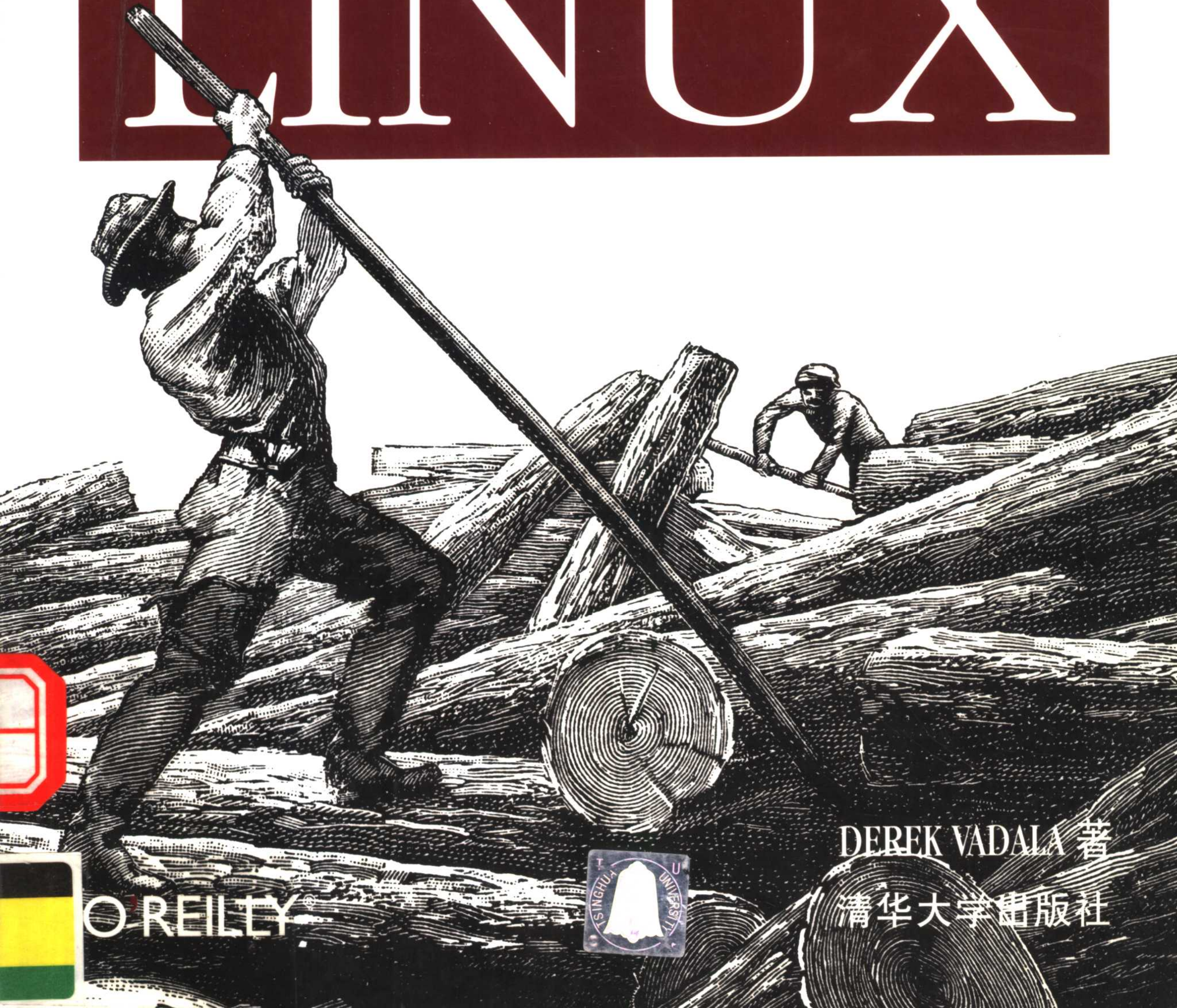
LINUX RAID 管理(影印版)

Managing RAID on LINUX



DEREK VADALA 著

O'REILLY®

清华大学出版社

图书在版编目（CIP）数据

LINUX RAID 管理: /（美）达瓦拉（Vadala, D.）著. — 影印版. — 北京：清华大学出版社，2003.7

书名原文：Managing RAID on LINUX

ISBN 7-302-06578-0

Ⅰ.L… Ⅱ.达… Ⅲ.Linux 操作系统－英文 Ⅳ.TP316.89

中国版本图书馆 CIP 数据核字（2003）第 029720 号

北京市版权局著作权合同登记
图字：01-2003-1929 号

书　　名/	LINUX RAID 管理（影印版）
书　　号/	ISBN 7-302-06578-0/TP · 4929
责任编辑/	常晓波
封面设计/	Emma Colby，张健
出版发行/	清华大学出版社（www.tup.tsinghua.edu.cn）
地　　址/	北京清华大学学研大厦（邮政编码 100084）
经　　销/	各地新华书店
印　　刷/	北京艺辉印刷有限公司
开　　本/	787 毫米 × 980 毫米　16 开本　16.5 印张
版　　次/	2003 年 7 月第一版　2003 年 7 月第一次印刷
印　　数/	0001-3000 册
定　　价/	29.00 元（册）

O'Reilly & Associates 公司介绍

O'Reilly & Associates 公司是世界上在 UNIX、X、Internet 和其他开放系统图书领域具有领导地位的出版公司，同时是联机出版的先锋。

从最畅销的*The Whole Internet User's Guide & Catalog*（被纽约公共图书馆评为20世纪最重要的50本书之一）到GNN（最早的Internet门户和商业网站），再到WebSite（第一个桌面 PC 的 Web 服务器软件），O'Reilly & Associates 一直处于 Internet 发展的最前沿。

许多书店的反馈表明，O'Reilly & Associates 是最稳定的计算机图书出版商——每一本书都一版再版。与大多数计算机图书出版商相比，O'Reilly & Associates 公司具有深厚的计算机专业背景，这使得O'Reilly & Associates形成了一个非常不同于其他出版商的出版方针。O'Reilly & Associates 所有的编辑人员以前都是程序员，或者是顶尖级的技术专家。O'Reilly & Associates 还有许多固定的作者群体——他们本身是相关领域的技术专家、咨询专家，而现在编写著作，O'Reilly & Associates 依靠他们及时地推出图书。因为O'Reilly & Associates 紧密地与计算机业界联系着，所以O'Reilly & Associates 知道市场上真正需要什么图书。

出版说明

计算机网络与通信技术的成熟与广泛应用，以及Internet与Web的迅速发展，为人类的工业生产、商业活动和日常生活都带来了巨大的影响。网络与通信技术在我国的很多领域也已经广泛应用，并且取得了巨大的效益。然而，该领域的技术创新的速度之快也是有目共睹的。为了帮助国内技术人员和网络管理人员在第一时间掌握国外最新的技术，清华大学出版社引进了美国O'Reilly & Associates公司的一批、在计算机网络理论和Open Source方面代表前沿技术或者在某专项领域内享有盛名的著作，以飨读者。本套丛书采用影印版的形式，力求与国外图书“同步”出版，“原汁原味”地展现给读者各种权威技术理论和技术术语，适合于相关行业的高级技术人员、科研机构研究人员和高校教师阅读。

本批图书包括以下几种：

- 《802.11 安全手册（影印版）》
- 《构建 Internet 防火墙（影印版）》
- 《Java 技术手册（影印版）》
- 《Open Sources（影印版）》
- 《WWW 信息体系结构（影印版）》
- 《LINUX RAID 管理（影印版）》
- 《Peer-to-Peer（影印版）》
- 《Java 实例技术手册（影印版）》
- 《Free As In Freedom（影印版）》
- 《Unix 操作系统（影印版）》

Preface

Linux has come a long way in the last decade. No longer relegated to the world of hobbyists and developers, Linux is ubiquitous and is quickly taking hold of enterprise and high-performance computing. Established corporations such as IBM, Hewlett-Packard, and Sun Microsystems have embraced Linux. Linux is now used to produce blockbuster motion pictures, create real-time models of worldwide weather patterns, and aid in scientific and medical research. Linux is even used on the International Space Station.

Linux has accomplished this because of a vast, and seemingly tireless, network of developers, documenters, and evangelists who share the common mantra that software should be reliable, efficient, and secure as well as free. The hard work of these individuals has propelled Linux into the mainstream. Their focus on technologies that allow Linux to compete with traditional operating systems certainly accounts for a large part of the success of Linux.

This book focuses on using one of those technologies: RAID, also known as a Redundant Array of Inexpensive Disks. As you will find out, RAID allows individuals and organizations to get more out of their hardware by increasing the performance and reliability of their data. RAID is but one component of what makes Linux a competitive platform.

Overview of the Book

Here is a brief overview of the contents of this book.

Chapter 1, *Introduction*, provides a quick overview of RAID on Linux, including its evolution and future direction. The chapter briefly outlines the RAID levels and identifies which are available under Linux through hardware or software.

Chapter 2, *Planning and Architecture*, helps you determine what type of RAID is best suited for your needs. The chapter focuses on the differences between hardware and software RAIDs and discusses which is the best choice, depending on your budget and long- and short-term goals. Also included is a discussion of PC hardware relevant to building a RAID system: disk protocols, buses, hard drives, I/O channels, cable types and lengths, and cases.

If you decide on a software RAID, then Chapter 3, *Getting Started: Building a Software RAID*, outlines the necessary steps in getting your first array online.

Chapter 4, *Software RAID Reference*, contains all the command-line references for the RAID utilities available under Linux. It also covers the RAID kernel parameters and commands related to array and disk management.

Chapter 5, *Hardware RAID*, covers RAID controllers for Linux. Chapter 5 also covers some widely available disk controllers and discusses driver availability, support, and online array management.

Chapter 6, *Filesystems*, offers a roundup of the journaling filesystems available for Linux, including ext3, IBM's JFS, ReiserFS, and Silicon Graphics's XFS. The chapter covers installation and also offers some performance tuning tips.

Chapter 7, *Performance, Tuning, and Maintenance*, covers a range of topics that include monitoring RAID devices, tuning hard disks, and booting from software RAID.

Appendix A, *Additional Resources*, lists online resources, mailing lists, and additional reading.

Appendix B, *Hardware RAID Controller Vendors*, offers information about RAID vendors.

A Note About Architecture

In the interest of appealing to the widest audience, this book covers i386-based systems. Software RAID does work under other architectures, such as SPARC, and I encourage you to use them. Support for hardware RAID controllers varies between architectures, so it's best to contact vendors and confirm hardware compatibility before making any purchases.

Kernels

Using RAID on Linux involves reconfiguring and modifying the Linux kernel. In general, I prefer to use monolithic kernels instead of modules, whenever possible. While kernel modules are quite useful for home desktop systems and notebooks, they aren't the best choice for servers and production systems. The choice between the two types of kernel is ultimately up to the user. Many users prefer modules to statically compiled kernel subsystems.

In order to maintain consistency, I had to settle on specific kernels that are used in the examples found throughout this book. It's inevitable that between the time of this writing and the release of the book, newer kernels will become available. This should not pose any problem for users working with newer kernels. This book uses kernels 2.4.18, 2.2.20, and 2.0.39, and focuses specifically on the 2.4 kernel.

LILO

Throughout this book, I focus on LILO when discussing boot loaders. I know that there are many other options available (GRUB, for example), but LILO has worked reliably with Linux's RAID capabilities, and some of the newer choices are not quite compatible yet.

Prompts

There are a number of command output listings throughout this book. The commands in these sections start with a prompt (either $ or #) that indicates whether the command should be executed by a normal user or whether it should be run as *root*.

```
$ less /etc/raidtab
# vi /etc/raidtab
```

For example, in the preceding code, the $ prompt indicates that the first command can be run as a normal user. By default, any user can view, but not modify, the file */etc/raidtab*. To edit that file, however, you need *root* access (as the # prompt denotes).

Conventions Used in This Book

The following typographical conventions are used in this book.

Italic
: Used for file and directory names, programs, commands, command-line options, hostnames, usernames, machine names, email addresses, pathnames, URLs, and new terms.

`Constant width`
: Used for variables, keywords, values, options, and IDs. Also used in examples to show the contents of files or the output from commands.

`Constant width italic`
: Used for text that the user is to replace with an actual value.

These icons signify a tip, suggestion, or general note.

These icons indicate a warning or caution.

Comments and Questions

Please address comments and questions concerning this book to the publisher:

O'Reilly & Associates, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
(800) 998-9938 (in the U.S. or Canada)
(707) 829-0515 (international/local)
(707) 829-0104 (fax)

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

O'Reilly has a web site for this book, where they'll list examples, errata, and any plans for future editions. The site also includes a link to a forum where you can discuss the book with the author and other readers. You can access this site at:

http://www.oreilly.com/catalog/mraidlinux/

For more information about books, conferences, Resource Centers, and the O'Reilly Network, see the O'Reilly web site at:

http://www.oreilly.com

Acknowledgments

Many people helped with the writing of this book, but the greatest credit is owed to Andy Oram, my editor. It was his early interest in my original proposal that started this project, and his suggestions, criticism, and raw editorial work turned this text from a draft into an O'Reilly book. I'm also indebted to many people at O'Reilly, for all their hard work on the numerous tasks involved in producing a book.

Neil Brown, Nick Moffitt, Jakob Oestergaard, and Levy Vargas reviewed the final draft for technical errors and provided me with essential feedback. Their insight and expertise helped make this book stronger. Many others helped review various bits of

material along the way, including Joel Becker, Martin Bene, Danny Cox, Jim Ford, Corin Hartland-Swann, Dan Jones, Eyal Lebedinsky, Greg Lehey, Ingo Molnar, and Benjamin Turner.

Thanks to all the filesystem developers who offered me feedback on Chapter 6: Stephen C. Tweedie, Seth Mos, Steve Lord, Steve Best, Theodore Ts'o, Vladimir V. Saveliev, and Hans Reiser. My appreciation also goes out to all the vendors who provided me with software, equipment, and comments: Thomas Bayens, Chin-Tien Chu, and Thomas Hall at IBM; Angelina Lu and Deanna Bonds at Adaptec; Craig Lyons and Daron Keith at Promise; James Evans at LSI Logic; Pete Kisich, Kathleen Paulus, and Adam Radford at 3ware; Joey Lai at Highpoint Technologies; Mathilde Kraskovetz at Mandrake; and Harshit Mehta at SuSE.

Thanks to my family and friends, who provided support and countless favors while I was writing this book, especially Dallas Wisehaupt, Philippe Stephan, Stephen Fisher, Trevor Noonan, Carolyn Keddy, Erynne Simpson, David Perry, Benjamin Richards, Matthew Williams, Peter Pacheco, Eric Bronnimann, Al Lenderink, Ben Feltz, and Erich Bechtel.

I owe special thanks to Craig Newmark, Jim Buckmaster, Jeff Green, and the entire staff of Craigslist.org for graciously providing me with office space and Internet access during my many excursions to San Francisco. Their hospitality directly resulted in the writing of Chapter 2. And finally, thanks especially to Eric Scheide, who encouraged me to write the original proposal for this book, gave me my first job as a Unix system administrator, and didn't argue as I slowly retired Ultrix and Solaris machines in favor of Linux.

Table of Contents

CHAPTER 1
Introduction

Every system administrator sooner or later realizes that the most elusive foe in sustaining reliable system performance is bandwidth. On one hand, network connectivity provides a crucial connection to the outside world through which your servers deliver data to users. This type of bandwidth, and its associated issues, is well documented and well studied by virtually all system and network administrators. It is at the forefront of modern computing, and the topic most often addressed by both nontechnical managers and the mainstream media. A multitude of software and documentation has been written to address network and bandwidth issues. Most administrators, however, don't realize that similar bandwidth problems exist at the bus level in each system you manage. Unfortunately, this internal data transfer bottleneck is more sparsely documented than its network counterpart. Because of its second stage coverage, many administrators, users, and managers are left with often perplexing performance issues.

Although we tend to think of computers as entirely electronic, they still rely on moving parts. Hard drives, for example, contain plates and mechanical arms that are subject to the constraints of the physical world we inhabit. Introducing moving parts into a digital computer creates an inherent bottleneck. So even though disk transfer speeds have risen steadily in the past two decades, disks are still an inherently slow component in modern computer systems. A high-performance hard disk might be able to achieve a throughput of around 30 MB per second. But that rate is still more than a dozen times slower than the speed of a typical motherboard—and the motherboard isn't even the fastest part of the computer.

There is a solution to this I/O gap that does not include redefining the laws of physics. Systems can alleviate it by distributing the controllers' and buses' loads across multiple, identical parts. The trick is doing it in a way that can let the computer deal seamlessly with the complex arrangement of data as if it were one straightforward disk. In essence, by increasing the number of moving parts, we can decrease the bottleneck. *RAID (Redundant Array of Independent Disks)* technology attempts to reconcile this gap by implementing this practical, yet simple, method for swift, invisible data access.

Simply put, RAID is a method by which many independent disks attached to a computer can be made, from the perspective of users and applications, to appear as a single disk. This arrangement has several implications.

- Performance can be dramatically improved because the bottleneck of using a single disk for all I/O is spread across more than one disk.
- Larger storage capacities can be achieved, since you are using multiple disks instead of a single disk.
- Specific disks can be used to transparently store data that can then be used to survive a disk failure.

RAID allows systems to perform traditionally slow tasks in parallel, increasing performance. It also hides the complexities of mapping data across multiple hard disks by adding a layer of indirection between users and hardware.

RAID can be achieved in one of two ways. Software RAID uses the computer's CPU to carry out RAID operations. Hardware RAID uses specialized processors, on disk controllers, to manage the disks. The resulting disk set, colloquially called an array, can provide various improvements in performance and reliability, depending on its implementation.

The term RAID was coined at Berkeley in 1988 by David A. Patterson, Garth A. Gibson, and Randy H. Katz in their paper, "A Case for Redundant Arrays of Inexpensive Disks (RAID)." This and subsequent articles on RAID have come to be called the "Berkeley Papers." People started to change the "I" in RAID from "inexpensive" to "independent" when they realized, first, that disks were getting so cheap that anyone could afford whatever they needed, and second, that RAID was solving important problems faced by many computing sites, whether or not cost was an issue. Today, the disk storage playing field has leveled. Large disks have become affordable for both small companies and consumers. Giant magnetic spindles have been all but eliminated, making even the largest-drives (in terms of capacity) usable on the desktop. Therefore the evolution of the acronym reflects the definition of RAID today: several independent drives operating in unison. However, the two meanings of the acronym are often used interchangeably.

RAID began as a response to the gap between I/O and processing power. Patterson, Gibson, and Katz saw that while there would continue to be exponential growth in CPU speed and memory capacity, disk performance was achieving only linear increases and would continue to take this growth curve for the foreseeable future. The Berkeley Papers sought to attack the I/O problem by implementing systems that no longer relied on a *Single Large Expensive Disk (SLED)*, but rather, concatenated many smaller disks that could be accessed by operating systems and applications as a single disk.

This approach helps to solve many different problems facing many different organizations. For example, some organizations might need to deal with data such as newsgroup postings, which are of relatively low importance, but require an extremely large amount of storage. These organizations will realize that a single hard drive is grossly inadequate for their storage needs and that manually organizing data is a futile effort. Other companies might work with small amounts of vitally important data, in a situation in which downtime or data loss would be catastrophic to their business. RAID, because of its robust and varying implementations, can scale to meet the needs of both these types of organizations, and many others.

RAID Terminology

One of the most confusing parts of system administration is its terminology. Misnomers often obscure simple topics, making it hard to search for documentation and even harder to locate relevant software. This has unfortunately been the case with RAID on Linux, but Linux isn't specifically to blame. Since RAID began as an open specification that was quickly adopted and made proprietary by a multitude of value-added resellers and storage manufacturers, it fell victim to mismarketing. For example, arrays are often referred to as metadevices, logical volumes, or volume groups. All of these terms mean the same thing: a group of drives that behave as one—that is, a RAID or an array. In the following section, we will introduce various terms used to describe RAID.

RAID has the ability to survive disk failures and increase overall disk performance. The RAID levels described in the following section each provide a different combination of performance and reliability. The levels that yield the most impressive performance often sacrifice the ability to survive disk failures and vice versa.

Redundancy

Redundancy is a feature that allows an array to survive a disk failure. Not all RAID levels support this feature. In fact, although the term RAID is used to describe certain types of non-redundant arrays, these arrays are not, in fact, RAID because they do not support any data redundancy.

Despite its redundant capabilities, RAID should never be used as a replacement for reliable backups. RAID does not protect your data in the event of a fire, natural disaster, or user error.

Mirroring

Two basic forms of redundancy appear throughout the RAID specification. The first is accomplished with a process called *disk mirroring*, shown in Figure 1-1. Mirroring replicates data onto every disk in the array. Each member disk contains the same data and has an equal role in the array. In the event of a disk failure, data can be read from the remaining disks.

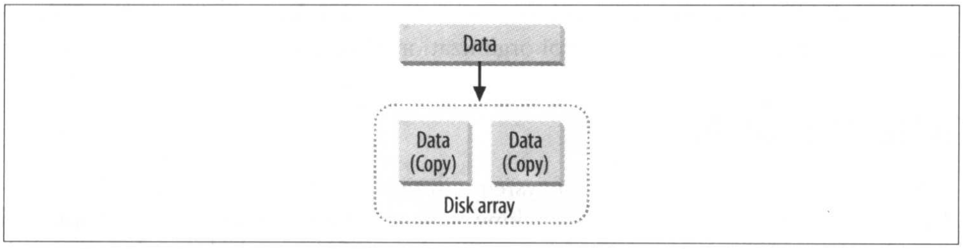


Figure 1-1. Disk mirroring writes a copy of all data to each disk.

Improved read performance is a by-product of disk mirroring. When the array is operating normally, meaning that no disks have failed, data can be read in parallel from each disk in the mirror. The result is that reads can yield a linear performance based on the number of disks in the array. A two-disk mirror could yield read speeds up to two times that of a single disk. However, in practice, you probably won't see a read performance increase that's quite this dramatic. That's because many other factors, including filesystem performance and data distribution, also affect throughput. But you can still expect read performance that's better than that of a single disk.

Unfortunately, mirroring also means that data must be written twice—once to each disk in the array. The result is slightly slower write performance, compared to that of a single disk or nonmirroring array.

Parity

Parity algorithms are the other method of redundancy. When data is written to an array, recovery information is written onto a separate disk, as shown in Figure 1-2. If a drive fails, the original data can be reconstructed from the parity information and the remaining data. You can find more information on how parity redundancy works in Chapter 2.

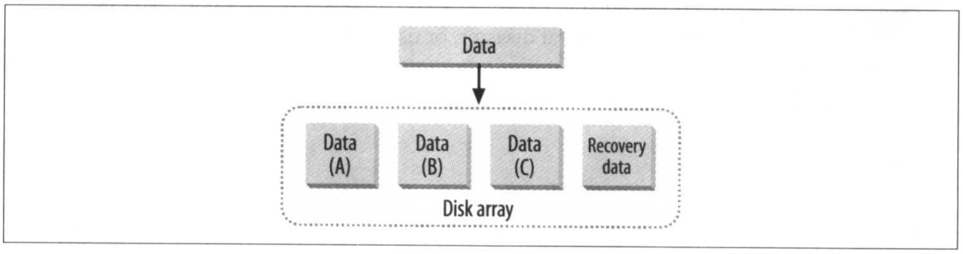


Figure 1-2. Parity redundancy is accomplished by storing recovery data on specified drives.

Degraded

Degraded describes an array that supports redundancy, but has one or more failed disks. The array is still operational, but its reliability and, in some cases, its performance, is diminished. When an array is in *degraded mode*, an additional disk failure usually indicates data loss, although certain types of arrays can withstand multiple disk failures.

Reconstruction, resynchronization, and recovery

When a failed disk from a degraded array is replaced, a recovery process begins. The terms *reconstruction*, *resynchronization*, *recovery*, and *rebuild* are often used interchangeably to describe this recovery process. During recovery, data is either copied verbatim to the new disk (if mirroring was used) or reconstructed using the parity information provided by the remaining disks (if parity was used). The recovery process usually puts an additional strain on system resources. Recovery can be automated by both hardware and software, provided that enough hardware (disks) is available to repair an array without user intervention.

Whenever a new redundant array is created, an initial recovery process is performed. This process ensures that all disks are synchronized. It is part of normal RAID operations and does not indicate any hardware or software errors.

Striping

Striping is a method by which data is spread across multiple disks (see Figure 1-3). A fixed amount of data is written to each disk. The first disk in the array is not reused until an equal amount of data is written to each of the other disks in the array. This results in improved read and write performance, because data is written to more than one drive at a time. Some arrays that store data in stripes also support redundancy through disk parity. RAID-0 defines a striped array without redundancy, resulting in extremely fast read and write performance, but no method for surviving a disk failure. Not all types of arrays support striping.

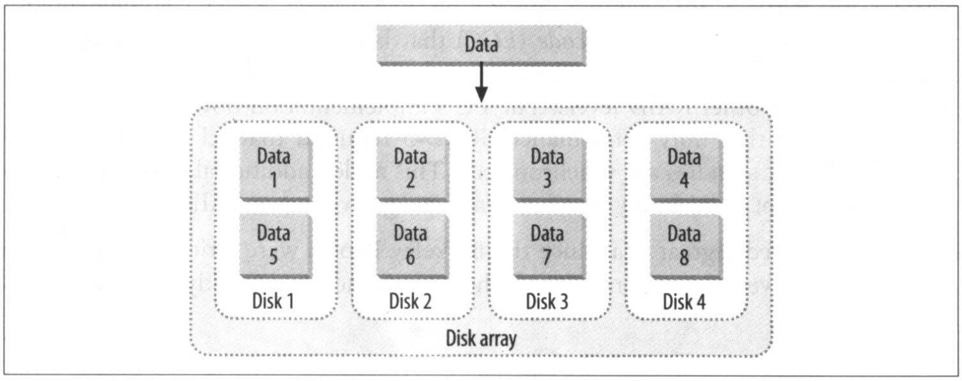


Figure 1-3. Striping improves performance by spreading data across all available disks.

Stripe-size versus chunk-size

The *stripe-size* of an array defines the amount of data written to a group of parallel disk blocks. Assume you have an array of four disks with a stripe size of 64 KB (a common default). In this case, 16 KB worth of data is written to each disk (see Figure 1-4), for a total of 64 KB per stripe. An array's *chunk-size* defines the smallest amount of data per write operation that should be written to each individual disk. That means a striping array made up of four disks, with a chunk-size of 64 KB, has a stripe-size of 256 KB, because a minimum of 64 KB is written to each component disk. Depending on the specific RAID implementation, users may be asked to set a stripe-size or a chunk-size. For example, most hardware RAID controllers use a stripe-size, while the Linux kernel uses a chunk-size.

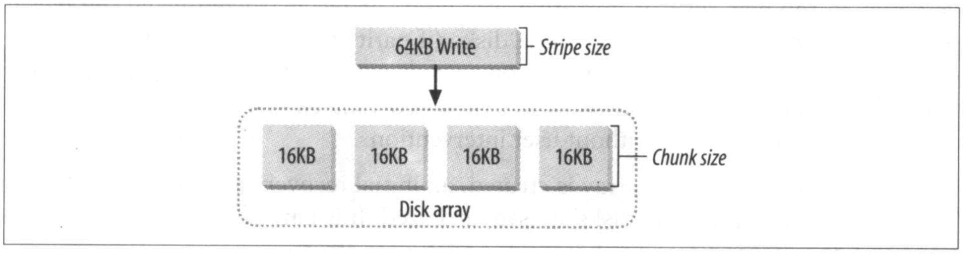


Figure 1-4. Stripe-size defines the size of write operations.

The RAID Levels: An Overview

Patterson, Gibson, and Katz realized that different types of systems would inevitably have different performance and redundancy requirements. The Berkeley Papers provided specifications for five levels of RAID, offering various compromises between performance and data redundancy. After the publication of the Berkeley Papers, however, the computer industry quickly realized that some of the original levels failed to provide a good balance between cost and performance, and therefore weren't really worth using.

RAID-2 and RAID-3, for example, quickly became useless. RAID-2 implemented a read/write level *error correction code* (ECC) that later became a standard firmware feature on hard drives. This development left RAID-2 without any advantage in redundancy over other RAID levels. The ECC implementation now required unnecessary overhead that hurt performance. RAID-3 required that all disks operate in lockstep (all disk spindles are synchronized). This added additional design considerations and did not provide any significant advantage over other RAID levels.

RAID has changed a great deal since the Berkeley Papers were written. While some of the original levels are no longer used, the storage industry quickly made additions