The following paper was originally presented at the
Seventh System Administration Conference (LISA '93)
Monterey, California, November, 1993

Where Did All The Bytes Go?

Dinah McNutt - Tivoli Systems
Michael Pearlman - Rice University

For more information about USENIX Association contact:

1. Phone: 510 528-8649
2. FAX: 510 548-5738
3. Email: office@usenix.org
4. WWW URL: http://www.usenix.org

Where Did All The Bytes Go?

Dinah McNutt – Tivoli Systems
Michael Pearlman – Rice University

ABSTRACT

Configuring, installing, re-configuring, and re-installing disk drives can be time consuming. Understanding the physical disk and how UNIX file systems are laid out on disk drives can not only help an administrator troubleshoot problems, but can allow him/her to maximize the amount of disk space available. Standard formatting utilities tend to configure disks with similar geometry identically. By using different geometry parameters or third party formatting programs, you can get the maximum amount of disk space from your drives.

This paper is a tutorial on understanding SCSI disk drive geometry (and how it differs from more traditional drives) and the steps required to get a disk drive to the point where it is usable (e.g., with a UNIX file system on it.) It includes practical tips such as questions to ask vendors so you know how much usable space you will have when you purchase a disk drive. It also include a case study where we show how you can squeeze additional usable disk space from a drive. This paper is targeted at novice system administrators, but experienced administrators who want to learn more about disk drives will hopefully learn something from reading this paper. The examples we use will be based on a BSD system, but the concepts apply to other types of UNIX systems as well.

Background and Motivation

Disk drives are probably one of the biggest headaches administrators will ever run across. Our experience has been that the number of disk drives increases the support effort significantly. One of the reasons is the amount of time involved in installation and reconfiguration of disk drives. Figuring out how to configure the drives and backing up and restoring the files on the existing drives can take a lot of time.

Standard hardware interfaces (de facto as well as official) have helped create a market where users can pick and choose different combinations of disk drives and controllers. The Small Computer System Interface (SCSI) is the most common type of bus found on workstations today. Vendors typically use a proprietary, high-speed bus to connect CPUs with memory and SCSI to connect I/O devices and peripherals. The SCSI specification is published and vendors have used this specification to manufacture hundreds of SCSI devices. This allows customers to select the device that best meets their needs and the computer vendors can focus on their processor and operating-system technologies.

Many computer vendors supply default configuration information for configuring disks drives. However, these defaults are not optimized for each disk and by taking the time to perform the calculations yourself, you can maximize the amount of disk space available after configuring the disk.

UNIX Disk Hierarchy

UNIX presents the user with hierarchical view of a disk. At the bottom level is the physical disk as seen by the device driver. At this level, a SCSI disk appears as an array of sectors and all commands to the disk must be SCSI commands. The next level is the raw disk where the disk appears as an array of sectors but can be accessed via read, write and ioctl system calls. The raw disk is used when high speed is required without the overhead of a file system structure (e.g., backup software.) The top level is the UNIX file system which presents the user with the tree structured directory system with which we are all familiar. At this level, the disk interface appears as an array of units called blocks. The file system is accessed as blocks and subdivisions of blocks called fragments. Each block is a multiple of the disk size and can only be divided into 2, 4 or 8 pieces. The file system block and fragment size are defined at file system creation time. Typical values are 8192 byte block size paired with a 1024 byte fragment size or 4096 byte block size paired with a 512 byte fragment size.

Each level of the hierarchy uses some of the available disk space to implement its own level of abstraction. Therefore, the usable capacity of a given disk depends on the level of the hierarchy at which it is being used: you have the largest capacity at the physical level and the least at the file system level. The operating system often reserves some space to store alternate disk labels and other essential information which is not accessible above the physical disk level. The file system reserves space for super blocks, cylinder group information and for inodes.

Units of Capacity

In the early days, disks had small capacity and were terribly expensive. Vendors quoted disk storage in small units (e.g., kilobytes) and there was

little confusion. Today disks are large and the cost has fallen dramatically. All other things being equal, the disk that you should purchase is the one that gives the best price per unit of storage. At some point in time, a vendor came up with the clever idea of redefining the unit of capacity to be smaller to make his/her company's disks appear more cost effective. Other companies followed suit and all disk sizes were quoted in terms of the smaller unit. Today we have two different numbers associated with each of the terms megabyte and gigabyte which has led to the anomaly that one megabyte of memory does not fit in one megabyte of disk. For the purposes of this paper we will use the following terms:

Term	Size in Bytes
1 megabyte (Mbyte)	2^{20} or 1,048,576
1 disk megabyte (DMbyte)	1,000,000
1 gigabyte (Gbyte)	2^{30} or 1,073,741,824
1 disk gigabyte (Dgbyte)	1,000,000,000

When the actual unit is unknown, we shall use ?MByte and ?Gbyte.

Rounding

Disk capacities are rounded numbers. A 1.6 Dgigabyte disk can really be as small as a 1.55 Dgigabyte disk. The loss of 0.05 Dgigabytes may look small but it is 50 Dmegabytes. With the current increasing trends in disk sizes, this rounding will be even larger when the Dterabyte disks arrive. To protect yourself, be sure to ask for your quotes in megabytes, kilobytes or even bytes. A reputable dealer has these figures available and should quote you a number plus or minus a small amount to account for the number of defects on a drive.

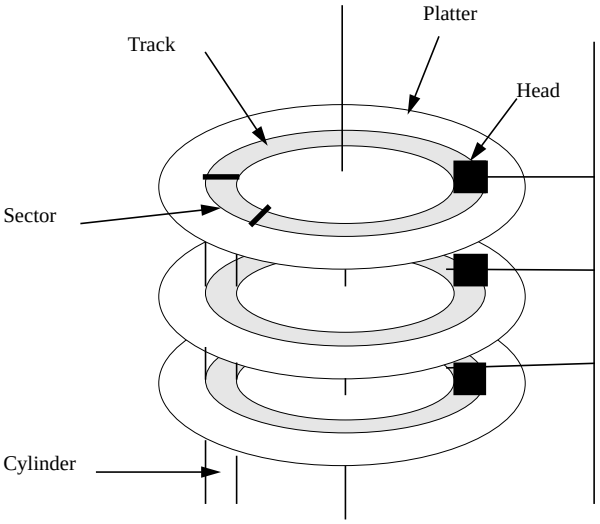

Figure 1: Disk Geometry

Disk Geometry

A disk drive is composed of one or more circular platters. Most platters can be written on both sides and the drive has as many recording heads as it does usable surfaces. The heads move radially across the platters and may be connected independently or grouped on an armature. Each radial location is called a track and each track is divided into smaller sections called sectors as show in Figure 1. A group of tracks located at the same radial position from the center of the platter is called a cylinder.

The heads float a microscopic distance off the platters. A head crash occurs when a head touches the surface of the media, removing some of the media and damaging the head. This usually results in an unusable disk drive.

Modern disks have either a fixed or variable geometry. Fixed geometry means that the number of sectors per track is constant throughout the entire drive. This is the geometry that was assumed in the design of many of the file systems found on today's machines. Variable geometry means that the number of sectors per track is not constant throughout the drive. The normal pattern for a variable geometry is that the number of sectors per track is smallest on the innermost tracks and increases as the radius of the track increases. The most common example of variable geometry is found in disks that use zone bit recording where the disk is divided into sets of contiguous cylinders, called zones, with a constant number sectors per track in each zone. The sector per track number increases as the distance of the zone from the disk center increases. This method increases the number of sectors you can have on the disk since you no longer are limited by the number of sectors that can be squeezed on the innermost cylinder.

Making the Disk Usable

The steps involved in making a disk drive usable are formatting, partitioning and labeling, and optionally creating file systems. The following sections will look at each of these steps in detail.

Formatting

Formatting is the process by which address information and timing marks are written on the disk to define each sector. The format process has two levels. The lowest level consists of

- Defining the disk geometry
- Selecting the type of defect management
- Defining sector layout parameters
- Enabling optional drive features
- Laying out the disk sectors and sector headers

Low-level Formatting

Low-level formatting is accomplished by the host computer sending the appropriate SCSI commands to the drive. The top level consists of the host writing operating system specific information to the disk. Typical information includes at a minimum a label which contains the disk geometry parameters

used to format the drive. To view the label information on a SunOS 4.1.3 system, use *dkinfo* (8[1])

```
# dkinfo sd1
1307 cylinders 9 heads 70 sectors/track
a: 32130 sectors (51 cyls)
   starting cylinder 0
b: 59850 sectors (95 cyls)
   starting cylinder 51
c: 823410 sectors (1307 cyls)
   starting cylinder 0
d: No such device or address
e: No such device or address
f: No such device or address
g: 731430 sectors (1161 cyls)
   starting cylinder 146
h: No such device or address
```

The format program gets the required information either from a static table stored in a system file or by an interactive interrogation of the user. Typical information that is required is

- The number of accessible cylinders
- The number of data heads
- The number of sectors per track
- The sector size
- The rotational speed of the disk
- Type of defect management and other related parameters

Obtain this information by reading the appropriate disk drive manual and trial and error. Formatting with the wrong choice of parameters can render a disk useless and in a state that requires it be returned to the vendor. So, we recommend obtaining the information from your vendor at time of purchase.

Most of the information required by formatting programs is not necessary for SCSI drives. SCSI disks are intelligent. The SCSI drive can be sent the desired sector size and the defect handling parameters and then can be commanded to format itself. When the format has completed, the host computer can request the drive to return its capacity in sectors as well as its head count, cylinder count and average sectors per track count which can be used for labeling and partitioning the disk. There are third party vendors marketing inexpensive disk utilities for SCSI disks which include a formatting program based upon the scheme outlined above.

Choosing the Sector Size

In this section, we shall discuss the effect of sector size on the "formatted" capacity of a disk drive. On most systems, the sector size is fixed by the operating system and the user has no freedom to choose the size without rewriting parts of the file system code. On UNIX systems, this value has traditionally been 512 bytes. In addition to the 512 bytes of data storage space, there is disk and controller overhead for storing such information as the sector ID. This overhead can occupy around 100 additional bytes which is approximately 16% of the total sector.

Therefore, by using a larger the sector size, there is less space wasted on headers and more space available for data storage. For example, the CDC Wren VII disk has an unformatted capacity of 1200 ?Mbytes[2] and with no sectors allocated for defect handling has a formatted capacity of 1050 ?Mbytes at 512 bytes per sector and 1106 ?Mbytes at 1024 bytes per sector.

Defect Management

Disk media is subject to imperfections. These imperfections may be due to the manufacturing process or to normal wear and tear. Mechanisms must be in place to ensure that these defective locations are not used to store data. The manufacturer of a SCSI disk uses sophisticated equipment to test each disk and stores on the drive a list of known defective locations. This list may grow automatically as the disk detects and optionally repairs new defects or may be manually supplemented. Defect management strategies fall in two camps:

1) Make the host computer responsible for avoiding the defective locations. The system can use the defect list as a map of the known bad sectors on the disk. When a bad sector is the next to be allocated, the device driver can switch to a free sector on the same track, skipping the bad sector.
2) Allocate space sectors during formatting which will not be visible to the device driver but which the disk can use to replace the defects.

The strategy chosen can have a dramatic effect on performance and formatted capacity. We shall discuss the following defect management strategies:

1) no management
2) host-based management
3) track-based sparing
4) cylinder-based sparing

The no management strategy formats the disk with no sectors allocated to handle defects and totally ignores the defect list on the disk. This was a common strategy many years ago. A new disk was formatted and a hard copy of the defect list was obtained using one of the following methods:

- Using the hardcopy defect list shipped with the drive
- Waiting for bad sectors to show up
- Running a program that does a read-write-read to every sector on the disk and noting the sectors that caused errors

[1]The authors have reformatted and deleted extraneous information from command output throughout this paper.

[2]It is the authors' belief that ?Mbytes figures in this paragraph are actually DMbytes.

The user would have to manually hand edit the inode and free list to indicate that these sectors were allocated to a file with a special name. These systems had modified utilities to never touch files with the magic name. Programs that accessed raw disk space were modified not to touch the bad sectors. This scheme maximized the formatted capacity but did not work if critical system blocks were defective and relied on the user never accessing the *bad* file. This scheme is rarely used today.

The host-based management strategy is to also format the disk without allocating any spare sectors but to have the device driver note the address of defective sectors. The device driver then must guarantee that the defective sectors are never accessed. This also maximizes capacity but places a lot of complexity into the device driver code.

Track-based sparing allocates a fixed number of spare sectors per track and spare tracks per disk. We'll refer to the number of spare sectors per track as *asect* and it is usually 1. We'll refer to the number of spare tracks per volume as *atrks* and it is usually twice the number of heads on the drive. If the number of defects on a particular track is less than or equal to *asect* then no seek or head switch is required by the drive to use the spare sector instead of the defective one. If there are more than *asect* defects on a particular track then this entire track is replaced by one of the spares and a seek will occur and possibly a head switch. This scheme gives the highest performance but the lowest capacity.

Cylinder-based sparing allocates *asect* spare sectors per cylinder and *atrks* spare tracks per volume which is chosen to be an integral number of cylinders (e.g., no cylinder fractions.) If there are less than or equal to *asect* defects in a given cylinder then a defective sector is replaced by one of the spares allocated in the same cylinder, which at worst results in a head switch. If the number of defects in a given cylinder exceeds *asect*, then some of the tracks will be mapped to one of the spare tracks. This will cause a seek to occur and possibly a head switch. If *asect* is less than the number of disk heads, this scheme will give a larger formatted capacity than track-based sparing with only a slight performance hit.

For SCSI disks, the best strategies to use are either track-based or cylinder-based sparing. The best way that we have found to tune the parameters in track or cylinder-based sparing is to format the drive and extract the defect list. By looking at the distribution of defects, the values of *asect* and *atrks* can be adjusted to optimal values and the disk reformatted with these values. Remember to allow some space for future defects.

Partitioning and labeling

A partition is a contiguous number of sectors that will be accessed as a raw disk device. You can have up to 8 partitions on a disk and the partitions can be defined anyway you wish. The partitions are indicated by an alphabetic character (a-h) and, traditionally, partition c is the whole disk. Figure 2 shows a disk that has been partitioned into 4 partitions: a, b, c, and g. Notices that you may use either a,b, and g or you may use c, but you may not use all 4 partitions as data written to partition c will over-write data on partitions a and b and vice versa.

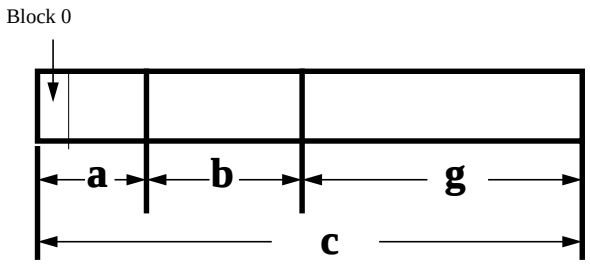

Figure 2: Overlapping Partitions

A file system is created using a disk partition. A file system partition may be as large as the whole disk or as small as a single cylinder. In all cases, the file system should start and end on a cylinder boundary as shown in Figure 3. Some versions of UNIX expect the file system to end on a cylinder boundary and you may get unexpected results if you are not careful about how you lay out your file systems.

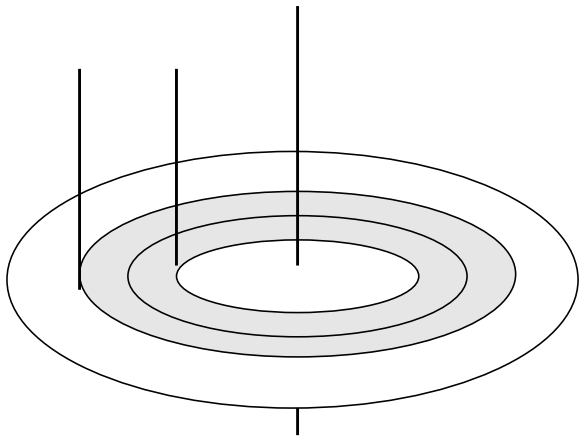

Figure 3: Filesystems and Cylinders

The Berkeley Fast File System is structured as a set of contiguous cylinder groups. For fast file system partitions it is important that the partition size be a multiple of the number of sectors in a cylinder group so that no space is wasted. Any cylinders that are left over due to this constraint can be assigned to a swap partition or to any other partition which will be accessed solely as a raw disk.

The disk label contains the information about how many partitions are on the disk and where they are located. The label is normally located in block 0 of the disk. As shown in Figure 2, block 0 is also part of partition a. It is a good idea not to make the first partition on your disk a swap partition as some version of UNIX will allow you to swap on block 0, thus overwriting your disk label. Some versions of UNIX store the label in a protected partition, so you cannot accidentally over-write the label. Check with your vendor to see how your system(s) work.

Making file systems

A file system is composed of superblocks, inodes and data blocks. The superblock stores information such as the size of the file system, pointers to the inodes, etc. Inodes are where the information about each file is contained: the size of file, last date modified, access permissions, user owner, group owner and pointers to the data blocks allocated to the file.

We will limit our discussion of making file systems to some of the issues that impact capacity. On BSD systems, the *newfs* command is used to create a file system on a raw disk partition. The options to *newfs* of interest are:

- i This options specifies the number of bytes/inode or the density of inodes in the file system. On SunOS 4.1.3 systems, the default is 8192 bytes/inode. For most file systems, you can reduce this number to 4096 bytes/inode with no problems. The exception is file systems that have many small files. By using 4096, you decrease the total number of inodes on the file system and increase the amount of space available for data storage.
- m The minimum free space threshold which is reserved from normal users. This value is usually 10%. One a 1 Gbyte system, 10 % is around 100 Mbytes which is a significant amount of space. On large disks, we recommend reducing this value to 5 %. BSD systems have a command called *tunefs* which allows you to modify the minimum free space threshold after the file system has been created.
- o File systems can be optimized to either minimize the amount of time spent allocating blocks or minimize the space fragmentation on the disk. By default, if the minimum free space threshold is less than 10%, the file system is optimized for space. If it is 10% or greater, it is optimized for time. Some implementations dynamically change the optimization based on the amount of free space in the file system.

Newfs is simply an easy-to-use front end for *mkfs* which is found on both BSD and System V systems (although the command is different on each system.)

Determining Usable Capacity

Unformatted versus formatted capacity

Disk specification sheets cite either unformatted or formatted capacity. Examining these numbers to determine how much data space you are buying requires expertise and should not be necessary. However, dealers cite these numbers and you need to know what they mean.

The unformatted capacity is the number of raw bytes on the disk and will always be more than the formatted value. The formatted value is the manufacturer's estimated at how much space will be available for user data after the disk has been formatted. This number will vary on different UNIX systems, so ask the vendor what the formatted capacity will be on your system.

Case Study - SunOS and Maxtor LX213S

Recently, we had to replace one of our Sun 207 Dmegabyte internal SCSI disk drives. This drive is listed by Maxtor as having 248 Dmegabytes unformatted capacity and 213 Dmegabytes of formatted capacity. We will look at 3 different methods of formatting the disk drive and see which method yields the maximum usable disk capacity.

The first example uses Sun's format program with the values supplied from /etc/format.dat:

```
disk_type = "SUN0207" \
    : ctlr = SCSI \
    : trks_zone = 9 : atrks = 2 \
    : asect = 4 : ncyl = 1254 \
    : acyl = 2 : pcyl = 1272 \
    : nhead = 9 : nsect = 36 \
    : rpm = 3600 : bpt = 18432
```

Using *dkinfo* to display the number of available sectors:

```
# dkinfo sd0
1254 cylinders 9 heads 36 sectors/track
c: 406296 sectors (1254 cyls)
```

We see that this disk has a real formatted capacity of 406296 * 512 = 208023552 bytes or 208 Dmegabytes. Observe that there is a 5 Dmegabyte discrepancy between our formatted capacity and Maxtor's quoted formatted capacity. Now, let's see how much usable capacity is available after making the UNIX file system.

```
# newfs /dev/rsd0c
/dev/rsd0c: 406296 sectors
            in 1254 cylinders
            of 9 tracks, 36 sectors
   208.0MB in 79 cyl groups
   (16 c/g, 2.65MB/g, 1216 i/g)

# df /dev/rsd0c
Filesystem   kbytes    used   avail
/dev/sd0c    189858      9   170863
```

```
# df -i /dev/rsd0c
Filesystem    iused   ifree
/dev/sd0c         4   96060
```

Creating the file system cost 208023552 - 189858*1024 = 13608960 or 6.5 percent of the formatted capacity. There is an additional 10 percent of the data space that is held back from normal users as expected. Now recreate the file system with approximately 50 percent less inodes:

```
# newfs -i 4096 /dev/rsd0c
/dev/rsd0c: 406296 sectors
            in 1254 cylinders
            of 9 tracks, 36 sectors
   208.0MB in 79 cyl groups
   (16 c/g, 2.65MB/g, 640 i/g)

# df /dev/rsd0c
Filesystem   kbytes    used   avail
/dev/sd0c    195546       9  175982

# df -i /dev/rsd0c
Filesystem    iused   ifree
/dev/sd0c         4   50556
```

Now file system creation cost 208023552 - 195546*1024 = 7784448 bytes or 3.7 percent of the formatted capacity. Again, we see that 10 percent of the data space was held back.

The second example uses a commercial formatting program which obtains the disk geometry from the disk.

```
# dkinfo sd0
1411 cylinders 7 heads 42 sectors/track
c: 414834 sectors (1411 cyls)
```

Notice that the disk geometry is totally different from the physical geometry of the disk which is used by Sun's format which is why the disk appears to have only 7 heads. The disk's formatted capacity increased to 414834*512 = 212395008 bytes or 212.4 Dmegabytes. In this case, there is only 0.6 Dmegabytes difference from Maxtor's formatted capacity figure. Moreover, since the SunOS reserves 2 cylinders for defect management, adding this back into our formatted capacity we get 212395008 + 2*7*42*512 = 212696064 bytes or 212.7 Dmegabytes which with rounding is equal to Maxtor's figure. Let's look at the usable data capacity. We'll create the file system using a reduced number of inodes since we have already demonstrated this will increase the amount of usable disk space.

```
# newfs -i 4096 /dev/rsd0c
/dev/rsd0c: 414834 sectors
            in 1411 cylinders
            of 7 tracks, 42 sectors
   212.4MB in 89 cyl groups
   (16 c/g, 2.41MB/g, 576 i/g)

# df /dev/rsd0c
Filesystem   kbytes    used   avail
/dev/sd0c    199567       9  179601

# df -i /dev/rsd0c
Filesystem    iused   ifree
/dev/sd0c         4   51260
```

We now see a loss of 212395008 - 199567*1024 = 8038400 or 3.8 percent of the formatted capacity.

By not using Sun's format program and the default number of inodes we gain 9942016 bytes (9.9 Dmegabytes) of data space or 4.7 percent of the formatted capacity quoted by the vendor. Both of the format programs in this example use cylinder-based defect handling but the Sun program used four spare sectors per cylinder and the third party program only three. It has been our experience that this third party program shows even more impressive gains on larger disks with more heads.

Summary

Vendors cite many different numbers as the capacity of a given disk. It is important when comparing disks to use the same metrics. We hope that this paper will help you to understand these metrics. We have given some examples of parameters that can be changed with the goal of increasing the data space of a given disk.

Some caveats are in order. Formatting is a dangerous process and has the capability of rendering a disk useless. The best choice for the partition size and file system parameters depend on the disk usage patterns. These parameters are used to determine static data structure sizes which if too small can inhibit further data storage even though space is available. Poorly chosen values have the ability to lessen disk performance when space becomes tight.

Acknowledgements

Many thanks to Tom Skerl of Applied Tensor Technology for his helpful comments and to Britton Dick of Rare Systems for his patience in supplying disk parameters.

References

Applied Tensor Technology, *SCSI Disk Utilities User's Guide*, Version 1.7, January 1993.

Control Data Corporation, *Product Specification For The Wren VII SCSI Disk Drive Model 94601*.

P. Galvin, D. McNutt, and M. Pearlman, ''Tricks Of The Trade For Systems Administrators'', *Sun User Group Proceedings*, December, 1992.

M. Loukides, *System Performance Tuning*, O'Reilly and Associates, 1991.

Maxtor Corporation, *XT-8000S Product Specification and OEM Technical Manual*.

D. McNutt, ''Where Did All The Bytes Go?'', *SunExpert Magazine*, pp. 49-53, May 1991.

D. McNutt, ''SMD Disk Drives: PartII'', *SunExpert Magazine*, pp. 44-47, June 1991.

Author Information

Michael Pearlman is the System Manager for the Department of Computational and Applied Mathematics and for the Department of Statistics at Rice University. Reach him via U. S. Mail at Department of Computational and Applied Mathematics; Rice University; POB 1892; Houston, TX 77251. Reach him electronically at canuck@rice.edu .

Dinah McNutt is a System Administration Consultant for Tivoli Systems where she works with customers of Tivoli helping them with system administration problems and customization of Tivoli's system administration software. She has been doing system administration for over 8 years and has written technical articles on the subject for SunExpert Magazine, RS/Magazine, and the X Resource Journal. Ms. McNutt currently writes the Daemons and Dragons column for UNIX Review magazine. She can be reached at dinah@tivoli.com.