by Henry Newman

ROBERT WHITMER

Exploring File Systems

As I mentioned last month we will be moving on to volume management and file systems. This is the next layer below the user application, and just as with user applications, we must understand the nuances of volume managers and file systems. These nuances are critical to scaling, and I/O scaling is the bottleneck in today's computing environments.

The goal for the next two months is to ensure you understand how to evaluate which volume manager and/or file system to use, and which options to use based on how volume managers and file systems work. This will allow you to make good decisions based on the information about application I/O performance as discussed in April (see `http://swexpert.com/CB/SE.C11.APR.01.pdf`), since all file systems and volume managers don't provide the same performance.

Understanding the volume manager and file system is an important part of the architecture process for large storage systems where large amounts of I/O are done. This must be an important part of the decision criteria on what server to buy and/or what volume manager and/or file system to buy.

File Systems Evolution

To start our journey let's go back in time to the creation of UNIX file systems. It is 1970 and K. Thompson, Ritchie, M. D. McIlroy, and J. F. Ossanna are developing the UNIX operating system–the PDP-7 UNIX file system to be specific. Structurally, the file system of PDP-7 UNIX was nearly identical to today's. It had an i-list: a linear array of i-nodes each describing a file. An i-node contained less than it does now, but the essential information was the same: the protection mode of the file, its type and size, and the list of physical blocks holding the contents. It also had directories or a special kind of file containing a sequence of names and the associated i-number. And it had special files describing devices. The device specification was not contained explicitly in the i-node, but was instead encoded in the number: specific i-numbers corresponded to specific files. (Note: Many of these details are drawn from a paper first presented at the Language Design and Programming Methodology conference in Sydney, Australia, September 1979. The conference proceedings were published as Lecture Notes in *Computer Science #79: Language Design and Programming Methodology*, Springer-Verlag, 1980. My rendition is based on a reprinted version appearing in *AT&T Bell Laboratories Technical Journal #63*, No. 6, Part 2, October 1984, pp. 1577-93. Surprisingly some of these concepts were based on a paper written as early as 1965. That even predated UNIX and was part of the Multics operating system. If you want to see some of the historical documents, take a look at *A General-Purpose File System For Secondary Storage,* by R. C. Daley, Massachusetts Institute of Technology, Cambridge, MA, and P. G. Neumann

Bell Telephone Laboratories Inc., Murray Hill, NJ.

Even more detailed discussions about file systems and allocation methods can be found in Association for Computing Machinery, Inc., *The Bell System Technical Journal #57*, No. 6, Part 2 (July-August 1978), Copyright 1974. In turn, that was a revised version of an article that appeared in *Communications of the ACM #17*, No. 7 (July 1974), pp. 365-375. That article was a revised version of a paper presented at the Fourth ACM Symposium on Operating Systems Principles, IBM, Thomas J. Watson Research Center, Yorktown Heights, NY, October 15-17, 1973.)

So you say to yourself after reviewing and digesting all this history, "Henry, is UFS structurally really about 35 years old?" My answer is "you betya!"

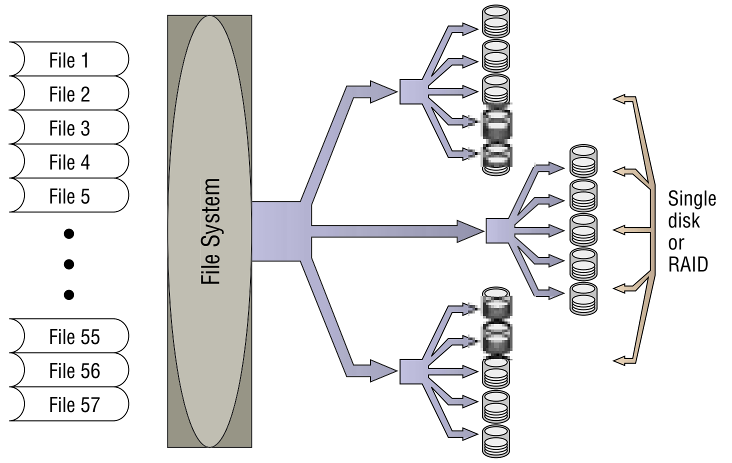
Over that time disk technology has changed as was discussed in my February column, but so have applications. I/O requirements have changed as applications have grown to scale with storage. A recent article in *The Economist* (Dec. 9, 2000) stated that more data would be created in the next three years than since the dawn of humanity.

What we need to do is determine how to analyze our needs and figure out the volume manager and file system options we have, and then use this information as part of an architecture plan. Some volume managers and file systems on some servers might be better at different I/O sizes or failover and reliability while others may be better at performance for different stream counts or asynchronous/threaded I/O.

Over the next two months, we'll explore how to understand and analyze volume managers and file systems by understanding the application and the interface file system and volume manager. We'll try to use this understanding to formulate questions for vendors.

Remember the goal is to figure out what works best (and to incrementally improve what you have) given your hardware and applications.

Figure 1. Standard Striping



Multiple I/O streams–Each stream (file) is transferred partially to multiple groups of striped drives in parallel, I/O based on allocation size.

File Systems Today

Structurally local or direct attached file systems have not evolved too far from 1965. (In a future column we will look at SAN file systems.) Most file systems build on the work done more than 35 years ago. Yes, people have added journaling for both metadata and in some cases data, but the data layout is quite similar, and some file systems now use a separate cache rather than using memory mapped I/O. Since that time much of what has been done is at best evolutionary and not very revolutionary. Volume management is a relatively newer concept compared with file systems, but still builds on the work from the 1960's. Basically we are still living in caves, but we have found fire.

The two most important functions a file system and/or volume manager needs to do is device management and space management.

If you need more space than is available in a single disk or RAID you will usually need to use a volume management tool. Some file systems manage their own data volumes so volume managers are not required and/or possibly not available. Among them are Cray's UNICOS, Sun's QFS and SAM-FS, and Fujitsu's FPFS. Most file systems currently available, however, use a volume manager if they need more than one device. These include but are not limited to Veritas file system VxFS (Sun, HP, Linux), IBM JFS, all flavors of Linux, Digital's UNIX AdvFS, NT NTFS, EMC (Crosstor) and SGI IRIX XFS.

Device Management

Standard volume management, either within the volume manager or within the file system, is done in two ways: striping or round-robin.

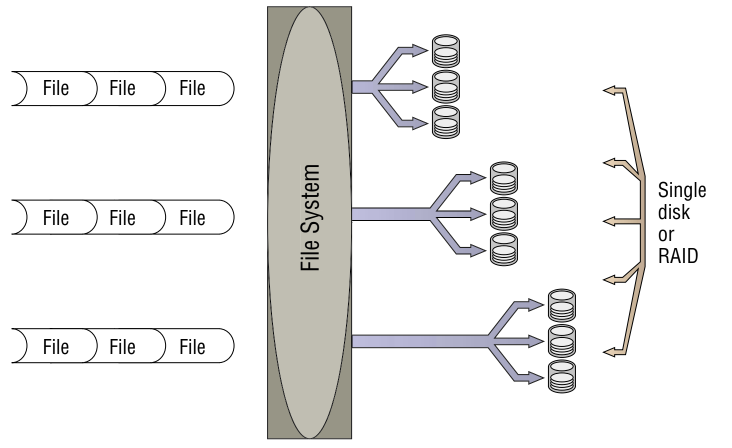
Striping is the standard method for volume managers and round-robin is the standard for file systems, which manage their own volumes.

Most volume managers stripe the data across all of the devices (either RAID or disks) based on a stripe width. The goal is to achieve parallelism on the devices. The stripe width is generally set to a value based on the number of streams of I/O and the I/O request size such that all of the devices can be in use. The goal is to keep all of the disks busy to achieve parallelism. This is very important in a number of application environments like databases. On large databases, performance depends on making sure that as many disks as possible are used so striping across all of the disks statistically distributes the load to ensure the best performance possible (see Figure 1).

For file systems that support round-robin allocation, each new open and/or directory create moves to the next device. With round-robin allocation, the goal is to reduce the number of head seeks and missed revolutions by having files be allocated more sequentially.

If each disk is writing a single stream it will likely allocate sequentially down the device and be able to read sequentially. If RAID is used, the

Figure 2. Round Robin



Multiple I/O streams–Each open (file) is transferred to a single disk or RAID.

read-ahead cache will generally have a much higher hit rate than in the striped example. This method does not work as well in general for databases because the data accesses are so random you need to use as many disks and channels as possible to get the best possible I/O performance (see Figure 2).

Tuning for Larger Requests

On some operating systems I/O requests are limited by values in the kernel and/or device drivers. On Solaris, for example, by default, the largest physical request that can be made is 128 KBs. This can be changed by a modification to `/etc/system` to add:

```
set maxphys= "value in decimal, octal or
hexadecimal"
```

To set `maxphys` 8 MB

```
set maxphys=8388608
```

or

```
set maxphys=0x800000
```

In Solaris the `sd`, `ssd` and `st` (SCSI, Fibre Channel, and tape) device drivers support an upper limit of 1 MB transfer size even though `maxphys` is set to a value larger than 1 MB.

The `sd` driver is the SCSI device driver, which is a published interface. If you buy an HBA from a vendor other than Sun, this is the driver that is used. This device driver is also used for Sun supported SCSI cards.

The `ssd` driver is the Fibre Channel device driver used by Sun-only FC HBAs. Either the Sun PCI Fibre Channel card or the SBus Fibre Channel card uses this. No other HBA vendor uses this interface as it's not a published interface.

The `st` driver is the Sun tape driver used by almost all tape manufacturers. This is an important driver because tape error recovery is non-trivial and because it also supports almost all enterprise tapes from IBM and StorageTek, to LTO, DLT, AIT and even 4MM and 8MM tapes. It is important to realize that only a few tape drives can take advantage of requests larger than 1 MB. Sony DTF-2 and Ampex drives are two examples.

You must make changes to each driver by changing the configuration files in `/kernel/drv/xxx.conf`. Adding the following per target or at the bottom of the file

```
"driver name"_max_xfer_size="value";
```

for example, changes the `sd` driver to a transfer size of 16,777,216 (16 MB):

```
sd_max_xfer_size=0x1000000;
```

In SGI IRIX, the equivalent value is `maxdmasz`. This is the largest value that can be transferred from the user space to/from the system.

By changing both `maxphys` in `/etc/system` and the driver, large requests can be made to the file system. This does not guarantee that large requests will be passed to the device driver. That depends on the volume manager and file system allocations, which is what we will discuss next month.

Understanding volume manager and file system allocations and layout will allow you to match the available hardware to the application, and if you are fortunate enough to be able to buy new hardware and software, you will be able to determine which server vendor, volume manager and file system can meet your requirements today and in the future.

Next Month

This is Part 1 in a Part 2 series. Next month I will discuss file system allocation and specific issues on how to evaluate vendor offerings.

***Henry Newman** is a 19 year veteran in high-performance systems working mostly in I/O and system tuning. He is currently employed by a consulting company.* `hnewman@cpg.com`.

Further Reading

Multics Operating System
http://www.staff.city.ac.uk/~sh392/multics/fjcc1.html
http://www.staff.city.ac.uk/~sh392/multics/fjcc4.html

Early UNIX File System and Operating System
http://cm.bell-labs.com/cm/cs/who/dmr/hist.html
http://www.dei.isep.ipp.pt/docs/unix-Part_I.html
http://www.dei.isep.ipp.pt/docs/unix-Part_II.html
http://cm.bell-labs.com/cm/cs/who/dmr/cacm.html