

Effective Use of Local Workstation Disks in an NFS Network

Paul Anderson – University of Edinburgh

ABSTRACT

This paper presents an analysis of the NFS traffic generated by a typical group of Unix workstations. The relative advantages and disadvantages of holding different categories of data on local workstation disks are then discussed, in the light of these results. Finally, a program is described that automatically reconfigures a local disk, at regular intervals, to transparently hold copies of the most frequently used programs and data. Trace-driven simulations of this program, using the NFS trace data, and experience of the system in practice, show that considerable savings can be made in server and network traffic using comparatively small amounts of local disk space.

Introduction

A typical Unix network includes one or more file servers that provide the bulk of the file store for the workstations on the network. The use of central servers is usually more cost-effective in terms of storage and easier to administer than a highly distributed filesystem. However, many individual workstations also incorporate their own, much smaller disks, that are generally used to hold frequently accessed data, improving the workstation performance and reducing the load on the servers and the network.

Special filesystems, such as the Andrew File System [1] (AFS), are capable of utilising local disk space as a filesystem cache that automatically attempts to minimize remote file accesses. However, the most common remote filesystem for Unix workstations is probably Sun's Network Filesystem [2] (NFS) which provides no such inbuilt facilities. In a "traditional" NFS network, a small local disk would typically be configured to hold swap space and some, or all, of the vendor's Unix implementation. This scheme is easy to administer, because the local disk should only require updating for changes to the system configuration or new releases of the operating system. However, much of the traffic on a typical network involves access to home directories and locally-maintained software. These data change more frequently and are much larger and hence more difficult to manage when they are distributed across local disks. Many installations attempt to store some local software and user files on workstation disks, but the choice of which data to hold is often arbitrary and measurement of the actual effectiveness is difficult.

By monitoring the NFS traffic from a group of workstations, it is possible to categorize the filesystems being accessed and predict the traffic savings that could be obtained by storing different filesystems on a local disk. This allows the effectiveness

of simple disk allocation schemes to be compared. For certain types of data (read-only program and data files), a more detailed analysis, at the individual file level, shows which files are being accessed most frequently and the savings that could be obtained by holding specific subsets of these files locally. By examining the access times of such files, it is possible for a program to regularly re-configure a local filesystem so that it always holds an optimal subset.

NFS Traffic Analysis

The Department of Computer Science network at the University of Edinburgh includes several hundred heterogeneous workstations spanning several subnets. Each subnet is self-contained in terms of workstation services, such as booting, base operating system and local software, but the automounter AMD [3] provides a global namespace that presents a uniform filesystem across all the workstations, using NFS as the remote access protocol [4, 5].

A typical selection of workstations on a single subnet was chosen as a representative sample for the purposes of traffic analysis. These consisted of 26 assorted Sun workstations running SunOS 4.1, classified as follows¹:

- personal* 23 personal workstations, 4 with their own swap disks.
- public* 1 public multi-user machine with a local disk holding the base operating system and local swap space.
- compute* 2 compute servers with their own disks, used for swapping only.

These workstations serve a computer science research laboratory, running applications such as text processing, mail, and program development. The

¹The file servers on the subnet are not included, since the traffic to the servers will be exactly that traffic generated by the workstations.

environment is similar to many academic and research installations, and, although there may be significant differences in usage patterns for teaching or commercial networks, many of the following results should still be appropriate.

The programs *rpcspy* and *nfstrace* [6] were used to collect data on all NFS transactions from the above workstations over a period of one month. These data were analyzed by several *perl* [7] scripts that divided the traffic into the following categories by examining the NFS filehandles present in the transactions:

- home* Access to user's home directories.
- local* Access to locally maintained software.
- root* Access to the workstation's *root* partition.
- usr* Access to the workstation's *usr* partition.
- swap* Access to the swap file.
- misc* Miscellaneous access to remote filesystems. This includes print spool directories, network-wide temporary directories and remote traffic to other domains.

As would be expected, the amount and distribution of traffic between the categories varies considerably between individual workstations, and from day to day. This is owing to the changing nature of the individual's work; a particular user might even be absent for a week, leaving a workstation idle. However, it is the average figures that reflect the total impact of all the workstations on the network and the servers.

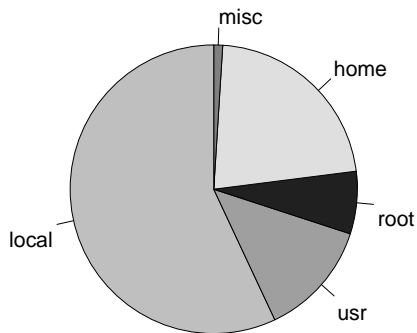


Figure 1

On the diskless workstations, swapping traffic accounted for an average 30% of the total NFS traffic during the month. For individual workstations, this varied between 10% and 40%, obviously depending on the type of work being performed by the user, and on the real memory available². It was noticeable that users with heavy compute requirements are not necessarily those that generate most swap traffic,

²Mostly 16Mb in this sample.

since they often tend to use remote compute servers for large jobs.

The public machine and the personal workstations showed a roughly similar distribution of the remaining traffic (Figure 1), although the balance between local software and home directories varied considerably; in the extreme case of the two compute servers, these two values were actually reversed. This appears to be due to compute-intensive programs accessing large data files from user's home directories, and paging off users own program binaries.

Standard Configurations for a Local Disk

A conventional allocation of space on a local workstation disk would probably include *swap*, *root* and *usr* filesystems.

In general, local swap space is almost always effective, although the actual amount of disk space required, and the amount of traffic saved on any individual workstation, will vary greatly. The above situation is probably typical, representing an average traffic saving of around 30%, with a swap space of 20-30Mb.

Local storage of the *root* filesystem is also efficient, occupying about 5Mb of disk and saving 8% of the traffic, but the total saving is small and the use of local root partitions is more difficult to manage.

In those situations, where users rely largely on the software supplied with the vendor's operating system, local storage of the entire *usr* filesystem could produce a substantial saving in network traffic; this may also provide some degree of self-sufficiency for the workstation in the event of a server failure. In many large installations, however, users require access to a much wider range of software; in the above example, most users rely on local software even for their shells and window systems, so a local copy of the *usr* would only generate a saving of about 10% for 130Mb of disk space, and would not provide any useful degree of resilience.

Public Files and Software

Network wide public programs and read-only data files (such as fonts) account for a large proportion of the network traffic. For the above sample, most of this traffic is concentrated on the separate filesystem holding locally maintained software (*local*), although for many installations, a larger proportion might be generated from the standard *usr* filesystem. However, both these filesystems have exactly the same characteristics and it would seem reasonable to hold local copies of frequently used public programs and data files, whatever their origin. Such files are not normally written to by the workstation and it is usually sufficient to update the local copies from some master copy at relatively

infrequent intervals - for example nightly, or simply on demand. This seems to be common practice and several schemes have been documented, using widely available software[8, 9, 10, 11]. In general, however, there will be far too much software available on the network for copies of everything to be held locally (our network has over 700Mb of *local* software and 130Mb in *usr*), so some subset of the available software needs to be selected. A number of schemes have been proposed[10, 12, 13, 14] for organising master copies of the software so that sensible subsets (usually corresponding to individual applications) can be extracted and selectively copied onto a local disk. Some of these, such as *Depot* [12] and *lfu* [10] include software which can replace certain subsets of files with symbolic links. This would allow a local filesystem to store actual replicas of some files, and links to remote copies of other files, thus permitting files to be transparently migrated on and off the local disk by changing links into file copies and visa-versa.

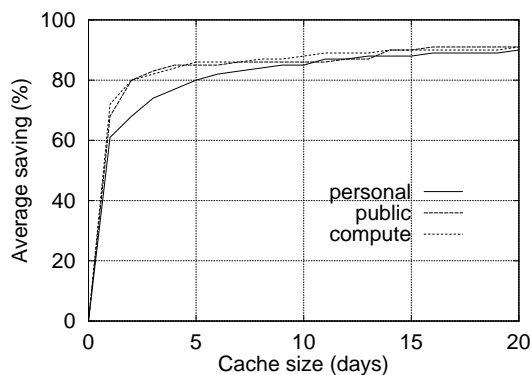


Figure 2

This type of scheme has the potential to be very effective; for our network sample, Figure 2 shows the percentage of *local* traffic³ which could be saved by holding copies of just those files accessed in the last N days. These results are surprisingly consistent across the different types of workstation and show that nearly 90% of network traffic from the *local* software occurs to files that have been accessed in the last two weeks; it seems that only a small fraction of the total available software is actually in use at any one time.

Figure 3 shows the local disk space required to hold all files accessed in the last N days, implying that 90% of the *local* traffic on a personal workstation can be saved for only 40Mb of disk space, *providing* that the appropriate files can be chosen.

³Local software has been emphasised in this discussion because it forms a much larger proportion of the traffic in our network sample, but there is no reason to expect that the results from the *usr* filesystem would be significantly different.

Unfortunately, the optimal fileset for local storage can be very different on different workstations and obviously varies with time; this is reflected in the larger quantities of disk space required by the public machine and the compute servers, which are used by many different people. Making a reasonable choice manually is very difficult; for example, out of the hundreds of fonts for the window system, each user will have a personal preference and will use only a small, but different, fraction of those available.

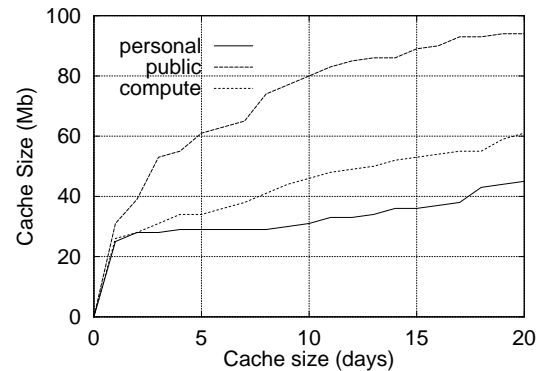


Figure 3

File Caching

The above results suggest that a file caching scheme which stored only the recently used files on a local "cache" disk, could be very effective. Other work on general-purpose file caching schemes [15] has confirmed the effectiveness of this technique, but it is unlikely that any general-purpose cache could be effectively implemented without extensive modifications to the kernel filesystem. However, this particular category of files has a number of properties which simplify the problem considerably, and many of the necessary mechanisms are already in place:

- Immediate reflection of master changes into the cache is not normally critical.
- The local copies are normally *read-only* as far as the workstation is concerned.
- A file can be moved out of the local cache, simply by replacing it with a symbolic link to a network-mounted copy of the file. Conversely, the a file can be encached by replacing the link with an actual copy of the file.
- If one of the available distribution schemes is already being used, then cache re-arrangement can conveniently occur at the same time as the local files are updated from the master.

The only remaining problem is to select the optimal set of files to be held in the cache, and for the software distribution program to re-arrange the local disk so that these files are held locally, and other files are replaced with links back to the network copies.

It should be quite possible for the selection of the optimal fileset to be performed independently of the cache update, and this is probably the cleanest solution. However, if the update program is already traversing the filesystem to identify out-of-date files, it is very easy for the same program to collect information about the previous usage of these files, and this is the approach adopted below.

A Caching Version of the *lfu* Program

The Computer Science department network uses nightly runs of the *lfu* [10] program to update copies of the *local* software onto multiple servers and onto the larger workstations. This program simply performs a selective copy of files from the master directories (mounted over NFS) onto the local disk. Those files which have been updated on the master are re-copied, and manually selected subsets of files can be blocked, or replaced with symbolic links to copies elsewhere on the network.

To investigate the possibilities of an automatic caching scheme, *lfu* was modified to:

- Collect information on the previous usage of the files and links on the local disk.
- Apply an algorithm to each file, computing a *cache score* intended to reflect the desirability of holding a local copy of the file.
- Re-arrange the local disk to hold as many files as possible with the highest cache score (replacing other files with symbolic links).

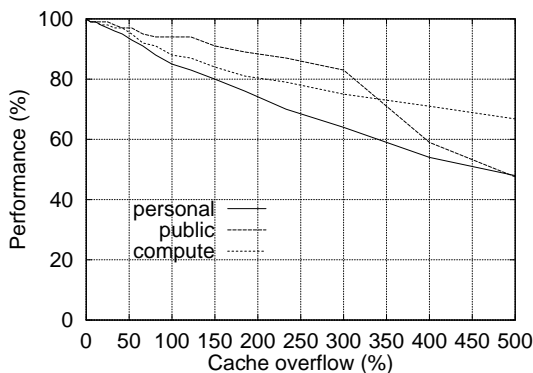


Figure 4

As each file is examined, the "last access time" (*atime*) of the file is used to record whether or not the file has been used since the last update. This allows a running history of the file usage over the last two weeks to be constructed and maintained on the disk. For an infinite-sized cache, this would be sufficient and the 90% saving indicated by the simulations could be expected simply by holding local copies of all files used in the last two weeks. In practice, the cache disk has a fixed size, and there needs to be some way of deciding which files to remove from the cache if it becomes full. Two *perl* scripts were used to perform full simulations of the

cache behaviour, on the NFS trace data, to investigate the performance of different fixed cache sizes, with different algorithms; *least-frequently-used* and *least-recently-used*. The daily inspection of the file *atime* provides only a very coarse measure of the file usage, but the following results show that this is more than adequate in practice; Figure 4 gives the simulated performance degradation, for different levels of cache overflow, using the *least-frequently-used* algorithm (100% represents the performance of an infinite cache).

The trace-driven simulations show a very similar performance for the two algorithms, indicating that the most *frequently* used files are also the most *recently* used files, and that the choice of algorithm is not critical.

The *least-frequently-used* algorithm was implemented in *lfu*, with the additional option of locking certain files into the cache, or varying the priority applied to the cache score. A number of personal workstations and a Sun 690/MP compute/multi-user machine were configured to run this caching scheme. The workstations were allocated 70Mb and the 690/MP, 120Mb. This includes the 40Mb (90Mb) indicated by Figure 3, and 30Mb of overhead for the directories and symbolic links needed to reference the 43,000 files on the master. Figure 5 shows the real performance of the cache on the 690/MP.

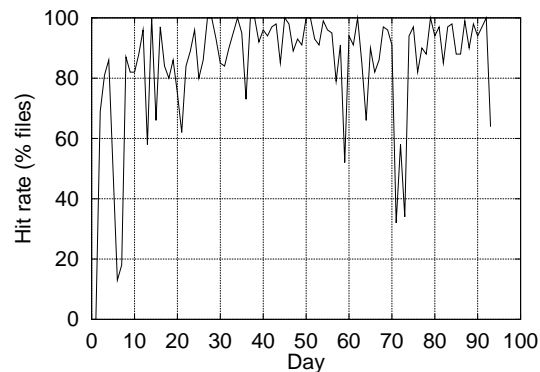


Figure 5

In this case, the data were gathered from the live *lfu* program and shows the percentage of different files, on each day, which were accessed from the cache. Although this is not directly comparable with the simulation data (which shows the percentage of blocks transferred) it provides an indication that the cache is performing as expected. Examination of the actual files being decached, shows that the sizing of the cache is also approximately correct; most files being removed from the cache having been used on no more than one day in the last two weeks.

Some Practical Issues

A nightly update of the cache generates some network traffic that is not included in the above figures; this could be a problem for sites running jobs that require significant amounts of network bandwidth during the night. However, the bulk of this traffic would already be generated by any conventional distribution scheme and is concerned with identifying files that need to be updated. Examination of typical *lfu* updates (which are not particularly efficient), using *etherfind*, [16] shows that about 1KB of network traffic is generated for each file scanned, yielding a total of about 40Mb for the 40,000 files on the sample system⁴ and the extra traffic necessary to adjust the cache, amounts to only an additional 5-10%. Careful scheduling of the nightly updates (which normally take about half an hour each) can usually avoid network congestion and interference with other background jobs, such as backup.

Holding only the most recently used files, means that the local disk provides no real resilience against server failure; certain very important files (such as files required only at boot time) may not be used regularly enough to appear in the cache. The graph in Figure 2 does indicate that a reasonable degree of resilience may not be achievable without holding copies of all possible files, since there appears to be a constant 10% of "new" files used, on average, each day. A more successful solution to this problem is to provide several alternative servers carrying complete sets of all the files (This redundancy is comparatively easy to arrange, using AMD). It is possible however, to "wire down" the contents of the cache at any point (for example, after a reboot), or to manually specify files which should be locked in the cache for resilience purposes.

The coarse-grain measure of file usage can sometimes cause the cache to be "flooded" by a single unexpected access to a large number of infrequently used files. For example, a user running the command "grep FOO *" in */usr/include* will cause the *atime* of all files in the directory to be updated, possibly raising their cache score above more regularly used files⁵. The use of a *least-frequently-used* algorithm (rather than *least-recently-used*) and a threshold of two days usage for initial entry into the cache, prevent nearly all of these problems in practice.

Several technical problems prevent the *atime* of files from always giving a reliable indication of the

file usage. In particular, the *atime* of continuously running programs is not continuously updated, and the *atime* of files which are exported over NFS is not always correct. Symbolic link *atimes* are also updated rather too easily; for example, by the command "ls -l". *lfu* uses a number of heuristics to deal with the worst of these cases and they do not appear to be a problem in practice.

Home Directories

Home directory access is responsible for a large percentage of the traffic from most workstations. If these directories are held on a central fileserver, considerable savings could be expected by moving them onto a local disk. For the above sample, Figure 6 shows the percentage of this traffic that could be saved by transferring user's home directories onto their own workstations, and the amount of disk space that would be required in each case⁶.

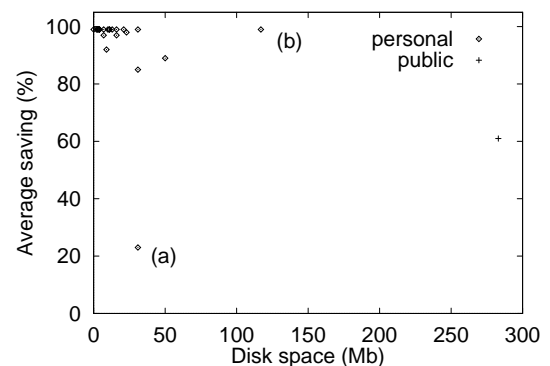


Figure 6

In most cases, virtually all of the home directory traffic can be eliminated by allocating about 50Mb of disk space to the user's home directory⁷. However, this has a number of drawbacks:

- The amount of space required is difficult to predict and may vary greatly between users. In the above example, one user (b) requires much more than 50Mb and many users require much less. Reallocation and efficient use of this space is difficult.
- The directory must still be available from elsewhere on the network which means that the workstation must export the filesystem over NFS. Integrating filesystems from large numbers of different workstations and arranging backups is more difficult to manage.
- The savings are greatly reduced in any

⁴Clearly, if a lot of files require updating because the masters have changed, then this figure will increase.

⁵This can be observed at several points in Figure 5, which displays the cache hit rate as the percentage of different files, rather than network traffic. The low hit-rate around day 71 was generated by a user browsing a large number of rarely used fonts.

⁶Users without their own workstation are transferred onto the public machine.

⁷The one workstation showing a particularly low saving in the above example (a) represents a machine that was being used by several different users for a particular package with node-locked licensing.

situation where a user regularly works on more than one machine; for example, in a pool of ten compute servers, or public machines, which are used at random, the saving will be less than 10%.

When a caching scheme is being used for public software, then it may be beneficial to encourage users to export copies of their own software along with the other public software. This is especially useful if the software is heavily used and/or frequently used by other people, when it provides the performance benefits of caching, together with resilience for other users of the program.

Some Possible Developments

Most existing distribution schemes work well when they are updating small numbers of files on a large number of machines, or a large number of files on a small number of machines. The above caching scheme is appropriate for most workstations with their own disk, however small the disk, so it makes unusual demands on the distribution system by requiring very large numbers of master files to be scanned by every workstation, even though the number of files actually transferred is usually very small. This scanning process is by far the most costly factor in the nightly update operation and some type of update program which could simultaneously broadcast the information to several clients would considerably reduce the server and network load.

The current implementation links only files, and always copies directories. This means that there may be very large directory hierarchies, that are never accessed by a particular machine, but still occupy significant amounts of space on the local disk for the directories themselves and the symbolic links that they contain. In theory, such a hierarchy could be replaced by a single symbolic link to the remote root of the hierarchy, but in practice, it is difficult (perhaps impossible) to identify individual workstation access to files in such a hierarchy and to arrange for its expansion when a decision is made to encache one of the files that it contains.

Conclusions

Clearly there will be some variation in usage patterns between sites, that will affect the optimal configuration for local workstation disks. There are also a number of trade-offs to be made; for example, the trade-off between efficiency and ease of management involved in the placement of home directories. The degree to which traffic can be saved⁸ by caching

⁸Note that this does not necessarily imply an improved performance for every workstation; network access to fast disks on a lightly loaded server can be faster than slow local disks.

a small number of well-chosen files on a personal workstation is, however, surprising and a number of conclusions would seem to be generally applicable:

- Holding *swap* and *root* partitions on a local disk is generally effective.
- Holding full copies of the *usr* filesystem may not be worthwhile in many cases.
- Caching of public files is very efficient in terms of reducing the network and server load. This does involve additional administration and nightly network bandwidth, but this is not significantly more than existing schemes which update fixed sets of files onto a local disk.
- Holding local copies of home directories is normally effective only for personal workstations, or machines that are used significantly more often by an individual user. It is likely to be difficult to match space requirements to available disk space though, and some compromise such as providing the user with both a remote, and a local directory may be necessary.

For a typical personal workstations from the survey, the following configuration would be an effective way of organising a local 150Mb disk:

- 25Mb *swap*, saving 30%
- 5Mb *root*, saving 5%
- 70Mb *local* cache, saving 90% of 41% = 36%
- 50Mb *home*, saving 15%

Thus providing a total saving of over 80% of the network and server load generated by the workstation.

Optimization of local disks on public machines and compute servers is more difficult; caching of public files is still effective, but the home directory traffic becomes a more significant proportion of the total traffic and this is more difficult to place and less effective than on a personal machine.

Acknowledgments

Thanks are due to Matt Blaze for the use of the *rpcspy* and *nfstrace* programs, and to the system staff in the Computer Science Department for their suggestions and assistance during development and testing of the *lfu* program.

Availability

The caching version of *lfu* is available for anonymous ftp from `ftp.dcs.ed.ac.uk` in the directory `pub/lfu`. The documentation file in this directory also includes the references[10, 4, 5].

References

1. Edward R Zayas, "AFS-3 Architectural Overview," in *AFS-3 Programmer's Reference Manual*, Transarc Corporation, September 1991.
2. R Sandberg, D Goldberg, S Kleiman, D Walsh,

- and B Lyon, "Design and implementation of the Sun Network File System," *Proceedings of Usenix Conference*, Summer 1985.
3. Jan-Simon Pendry, *AMD – An automounter*, Department of Computing, Imperial College, London, May 1990.
 4. Paul Anderson and Alastair Scobie, "The Local UNIX directory hierarchy," CS-TN-21, Department of Computer Science, University of Edinburgh, Edinburgh, August 1991.
 5. Paul Anderson, "Installing software on the Computer Science Department network," CS-TN-24, Department of Computer Science, University of Edinburgh, Edinburgh, August 1991.
 6. Matt Blaze, *NFS Tracing by passive network monitoring*, Department of Computer Science, Princeton University.
 7. Larry Wall, *Perl – Practical Extraction and Report Language*.
 8. Sun Microsystems, "rdist (1)," in *SunOS reference manual*, Sun Microsystems, 1990.
 9. Steven Shafter and Mary Thompson, *The SUP software upgrade protocol*, Carnegie Mellon University, September 1989.
 10. Paul Anderson, "Managing program binaries in a heterogeneous UNIX network," *Proceedings of LISA V Conference*, pp. 1-9, Usenix, 1991.
 11. Bjorn Satdeva and Paul M Moriarty, "Fdist: A domain based file distribution system for a heterogeneous environment," *Proceedings of LISA V Conference*, pp. 109-126, Usenix, 1991.
 12. Wallace Colyer and Walter Young, *Depot: A tool for managing software environments*, Carnegie Mellon University, April 1992.
 13. John Sellens, "Software maintenance in a campus environment: the Xhier approach," *Proceedings of LISA V Conference*, pp. 21-28, Usenix, 1991.
 14. Kenneth Mannheimer, Barry A Warsaw, Stephen N Clark, and Walter Rowe, "The Depot: A framework for sharing software installation across organizational and UNIX platform boundaries," *Proceedings of LISA IV Conference*, pp. 37-46, Usenix, 1990.
 15. Matt Blaze and Raphael Alonso, *Long-term caching strategies for very large distributed file systems*, Princeton University.
 16. Sun Microsystems, "etherfind (8)," in *SunOS reference manual*, Sun Microsystems, 1990.

Author Information

Paul Anderson graduated in Pure Mathematics from the University of Wales in 1977. He taught Mathematics and Computer Science at the North East Wales Institute of Higher Education until 1984 when he became system manager for the Institute,

establishing a new computer centre and software development team.

In 1988 he moved to the University of Edinburgh as Systems Development Manager with the Laboratory for the Foundations of Computer Science, where he is currently managing the laboratory network and working with other system managers to develop the computing facilities. Paul can be reached by mail at the Laboratory for the Foundations of Computer Science; Department of Computer Science; University of Edinburgh; King's Buildings; Edinburgh; EH8 3JZ; U.K. Reach him electronically at paul@dcs.ed.ac.uk.

