32

AIX

June 1998

In this issue

update

Published by

Xephon
27-35 London Road
Newbury
Berkshire RG14 1JL
England
Telephone: 01635 550955
From USA: 01144 1635 33823
E-mail: HarryLewis@compuserve.com

North American office

Xephon/QNA
1301 West Highway 407, Suite 201-405
Lewisville, TX 75067
USA
Telephone: 940 455 7050

Contributions

If you have anything original to say about AIX, or any interesting experience to recount, why not spend an hour or two putting it on paper? The article need not be very long – two or three paragraphs could be sufficient. Not only will you actively be helping the free exchange of information, which benefits all AIX users, but you will also gain professional recognition for your expertise and that of your colleagues, as well as being paid a publication fee – Xephon pays at the rate of £170 ($250) per 1000 words for original material published in AIX Update.

To find out more about contributing an article, see *Notes for contributors* on Xephon's Web site, where you can download *Notes for contributors* in either text form or as an Adobe Acrobat file.

Editor

Harold Lewis

Disclaimer

Readers are cautioned that, although the information in this journal is presented in good faith, neither Xephon nor the organizations or individuals that supplied information in this journal give any warranty or make any representations as to the accuracy of the material it contains. Neither Xephon nor the contributing organizations or individuals accept any liability of any kind howsoever arising out of the use of such material. Readers should satisfy themselves as to the correctness and relevance to their circumstances of all advice, information, code, JCL, scripts, and other contents of this journal before making any use of it.

Subscriptions and back-issues

A year's subscription to *AIX Update*, comprising twelve monthly issues, costs £175.00 in the UK; $265.00 in the USA and Canada; £181.00 in Europe; £187.00 in Australasia and Japan; and £185.50 elsewhere. In all cases the price includes postage. Individual issues, starting with the November 1995 issue, are available separately to subscribers for £15.00 ($22.50) each including postage.

AIX Update on-line

Code from *AIX Update* is available from Xephon's Web page at www.xephon.com (you'll need the user-id shown on your address label to access it).

Printed in England.

Performance management and measurement

It's my aim in this article to discuss performance measurement and management in broad terms, so that you can evaluate how to set about this task. I also intend to provide an overview of the benefits that are available from a methodical approach to performance management, and to provide practical advice on performance management for AIX systems. This article is aimed at AIX administrators who have perhaps mastered the use of a number of performance tools and now wish to bring them together into a cohesive structure for performance management.

UNIX PERFORMANCE IN THE SERVER MARKET

The past ten years has witnessed a substantial change in the way computers are used in business. During this time, IBM's dominant position in the mainframe market has been undermined by 'plug-compatible' manufacturers, while advances in microprocessor technology has made systems smaller and faster. At the same time the use of networks has increased over a thousandfold.

Microsoft's marketing wizardry has made it the main force in desktop systems and allowed it to move into the server market, where it has eroded Unix's dominance of the bottom end of the market. Other major players, like Sun Microsystems, HP, and Digital (now part of Compaq), have emerged from their traditional markets to take large slices of the corporate IT cake. The business world has been turned upside down by technology.

Every week brings a new announcement of faster processors, faster disks, and improved operating systems and databases, and each new software release requires more powerful systems on which to run. As software sophistication increases, so does its system requirements.

IBM AND THE RS/6000

The RS/6000 was sidelined by IBM for many years as a small system that probably wouldn't catch on. When IBM finally realized what was

happening they had almost missed the boat, and it took a considerable investment to bring the RS/6000 in line with the very successful HP-9000. IBM, with their might, enormous user base, and successful marketing techniques, managed to stay in the game and even to get ahead of it.

IBM now has tens of thousands of RS/6000s installed in the UK alone, and well over half a million worldwide. This is an impressive achievement, even for IBM. Since the release of the first Symmetric Processing (SP) system, IBM has notched up sales averaging 1,300 per annum worldwide, putting the number of SP frames at around 4,000 (they cost on average $250,000 each).

IBM's revenue for worldwide sales of the RS/6000 is over $25 billion. With 2,500 technical specialists working on AIX, five manufacturing plants churning them out, and sales operations in 120 countries, this demonstrates a considerable commitment to this platform.

Although losing out to Microsoft at the desktop and workstation level, the RS/6000 is holding its own at workgroup level, with growth of around 11% to 12% per year, and is surging ahead at the enterprise server level with growth of over 40% a year.

In October 1997 IBM launched the RS/6000 Enterprise Server Model S70. This advanced server supports full 64-bit symmetric multiprocessing (SMP) with up to 12 processors and many in-built reliability features. It's designed to run all of a medium sized company's business or act as a 'super server' to a number of smaller network servers. S70 prices start at $125,000, which is relatively inexpensive for a system that's more powerful than mainframes were just ten years ago.

As the machines increase in speed and capacity, so too does the need for systems and performance management. AIX 4.3, which is required to run the S70 in 64-bit mode, provides many performance improvements, especially when running 64-bit applications, but is still only an operating system. AIX won't analyse the workload and redistribute it for better throughput. It won't monitor the disk and the cache. It won't measure application resource usage, and it provides no estimates of future capacity requirements.

PERFORMANCE MEASUREMENT

A major issue in most distributed systems is how to collect performance data about applications running on network servers. How often should data be collected? How detailed should it be? Where should it be stored? How will the process of collecting data affect the performance of both servers and the network?

In this respect AIX is arguably the best of the various versions of Unix. Its sophisticated graphical interface, which emerged in its present form in AIX 3.2, has been built on considerably and has largely eliminated the need for the command level interface. This, in turn, has altered the skill level of Unix system administrators, who may not be as familiar with commands used to obtain performance information, such as **sar**, **iostat**, **vmstat**, and **netstat**, as their predecessors. Administrators want to have easy, graphical interfaces to their performance information. They require recent and historical information for performance management and they don't want the process of collecting performance data to cause them further management problems.

System adminstrators need performance tools to enable them to monitor and manage their systems that don't incur a significant overhead and that require minimum system management.

Throughout this article I provide examples using Landmark System's PerformanceWorks. As a product manager for PerformanceWorks, I'm clearly not a disinterested observer. Thus, readers should be aware that other performance management systems are available for Unix systems in general and AIX systems in particular, and they should evaluate the various products available with reference to the needs of their particular systems.

DATA COLLECTION IN THE DISTRIBUTED ENVIRONMENT

The best way to collect data from a network server is using a local agent running on the server. The alternative is to collect data across the network by polling, which is slow and network-intensive. Polling can also fail to identify problems in time for action to be taken. Note that, while the agent doesn't impose a performance penalty on the network,

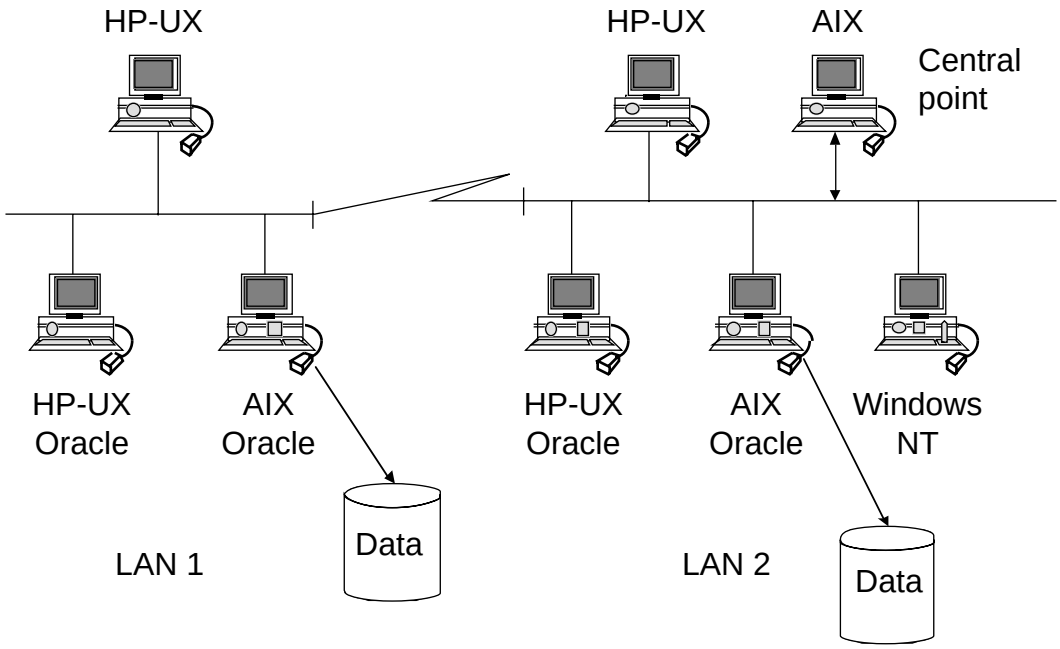

Figure 1: Local data collection

it imposes an overhead on the server, so it has to be written to be as small and efficient as possible. Figure 1 shows a typical network architecture with distributed agents and stores.

There are several requirements of an agent:

- It has to have a mechanism to access the required data.
- It has to be able to monitor that data frequently.
- It must be able to test the data against pre-defined thresholds.
- It must be able to send alerts when thresholds are broken (greater than, less than, matching pattern, etc).
- It must be able to collect data at predefined intervals and relay it to a central component that stores it.

This is true of all information about any part of the operating system or database.

Within the operating system there are a number of areas that must be monitored:

- *CPU*
  This is the primary resource at every server and workstation. CPU consumption has to be measured at both the system and process level. An overview of kernel and user CPU consumption provides a quick view of total processor activity, while more detailed statistics about processor queue length, the number of systems calls, and the number of page faults can help to evaluate the CPU load. Showing processor usage for each process allows administrators to identify heavy CPU users, while a breakdown by processor in an SP machine highlights potential load imbalance (Figure 2 – the screenshot is from PerformanceWorks).

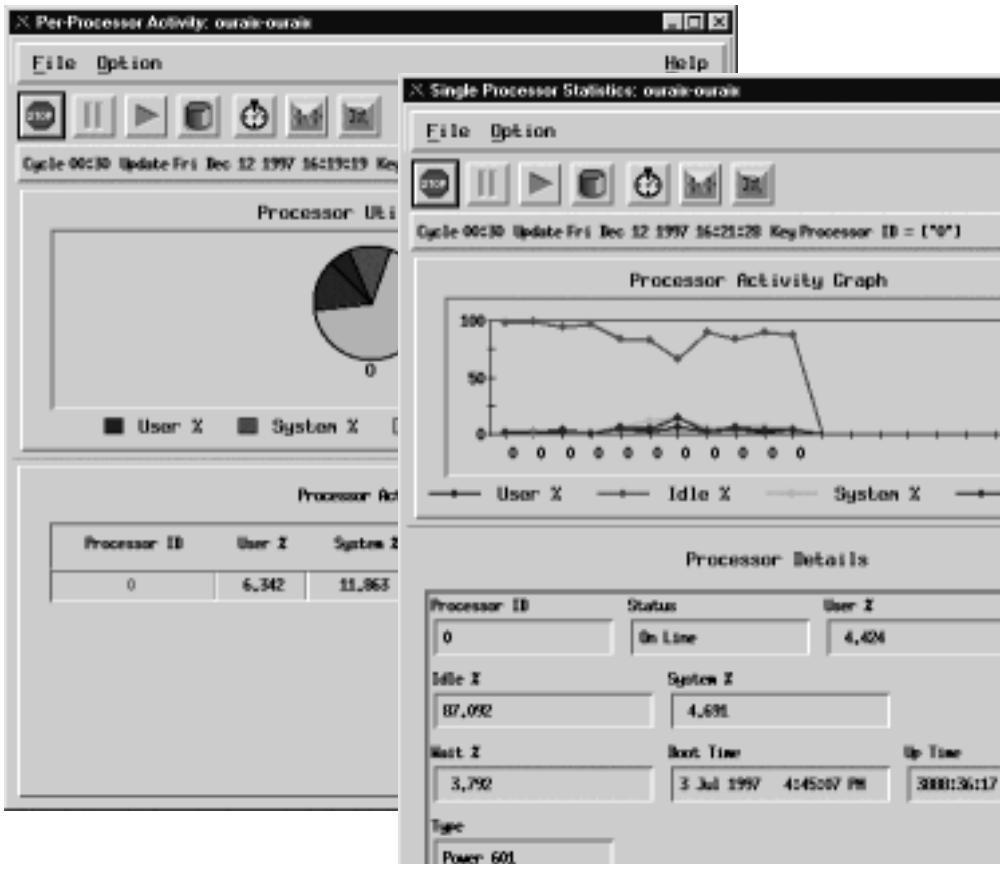

Figure 2: Graphical and tabular information with drill-down

- *Memory*
  As before, it's necessary to start with a high level overview of system and user memory consumption to highlight potential problems. This information should be used alongside paging rates to highlight problems, such as processes using excessive memory, 'memory leaks', lack of paging space, and poor use of memory. This requires the ability to view memory usage by process. Increasing the available memory can result in dramatic performance improvements in AIX, as long as memory is responsible for bottlenecks, something that can only be ascertained by examining current and historical usage. (See Figure 3.)

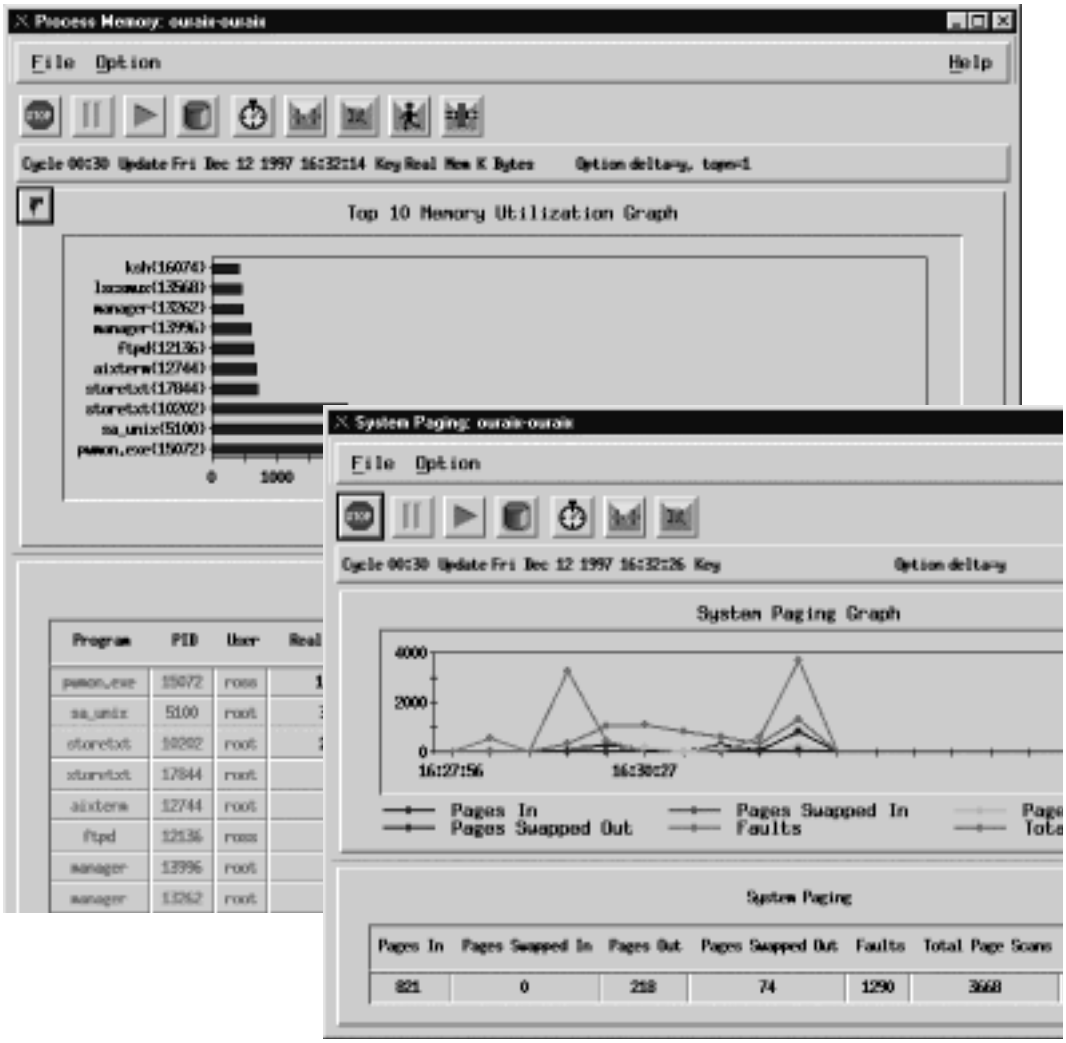

Figure 3: Monitoring processes for memory use and paging

- *Cache*
  Cacheing is an efficient mechanism for reducing I/O to disk. An analysis of the cache hit ratio provides information on how well the cache is being used – a low value requires further investigation. The size of the cache may need adjusting to suit the type of I/O load. Sequential I/O should yield a high cache hit ratio. However, the presence of many pinned records uses up available cache memory quickly as do 'lazy writes'. Also, low priority applications and processes that use cache take longer to fill the application buffer (they may be paged out), thus tying up memory. Each of these areas need to be looked at in order to obtain the best use of the system's cache.

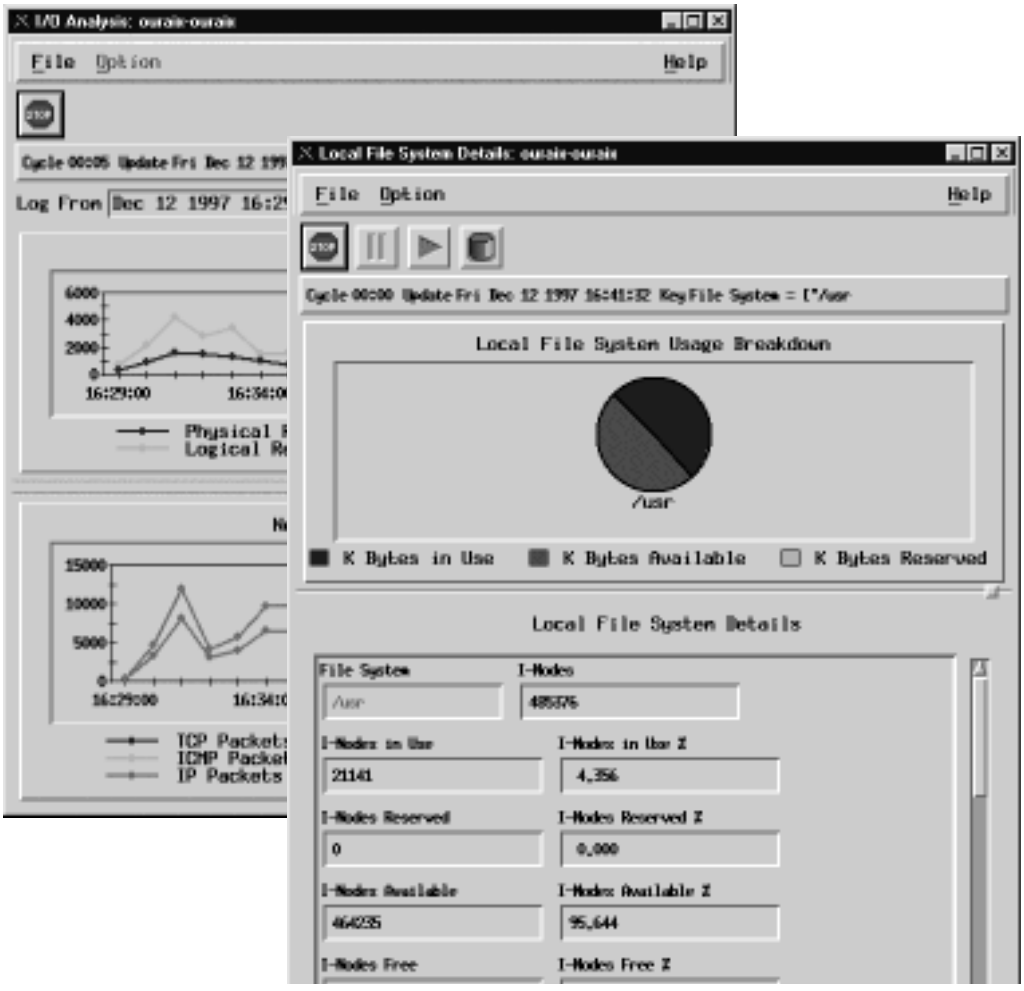

Figure 4: Analysing disk and cache I/O

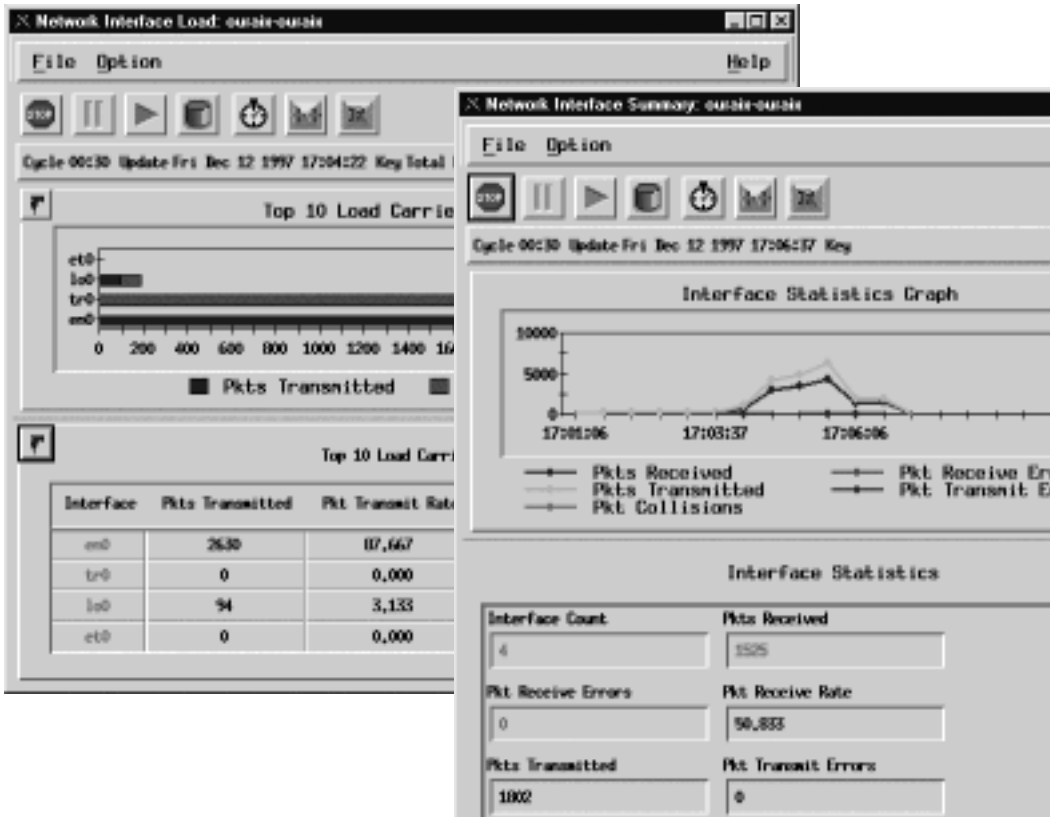

Figure 5: Network I/O at the server level

- *Disk I/O*
  I/O to disk is often the slowest function performed by a computer. Even with the latest fast SCSI drives, each I/O can still take as long as 10,000 processor instructions. It's essential, therefore, to monitor disk I/O. Both programs and the operating system can request I/O. Paging and swapping cause I/O. File access causes I/O. Loading programs causes I/O. Monitoring which applications perform I/O to particular volumes allows you to reduce the time spent by the system waiting for I/O. The information you require to assess the I/O includes the amount of data transferred, the number of physical I/Os, the number of activities queued, the time taken for each I/O, and the related cacheing information. Figure 4 on page 9 shows a display of disk and cache I/O. Historical data allows you to know when I/O problems began and to assess their impact on various applications and users.

- *Network I/O*
  Communication between systems is fundamental to many mission-critical applications. At each server or workstation it is possible to obtain details of all packets arriving at and leaving from the machine. These may be expressed in terms of datagrams, frames, segments, or packets, depending on the protocol or interface in use. It's also necessary to look at the fragmentation of IP packets when using TCP/IP and at the success or failure of any connections made. All of these measurements may be made at the server/ workstation level. (See Figure 5 opposite.)

DATABASES

Many well-known databases don't use standard filesystem access. They have their own I/O mechanisms and therefore bypass operating system I/O counters. Databases are 'structured', meaning that a request can cause one or many physical I/Os. Data held in tables is accessed either sequentially or through an index, depending on how the programmer has coded the query using SQL. Logs or journals are kept to enable recovery and backout.

A database is usually a complete system within the operating system. It may suffer from memory, CPU, paging, cache, and I/O problems,

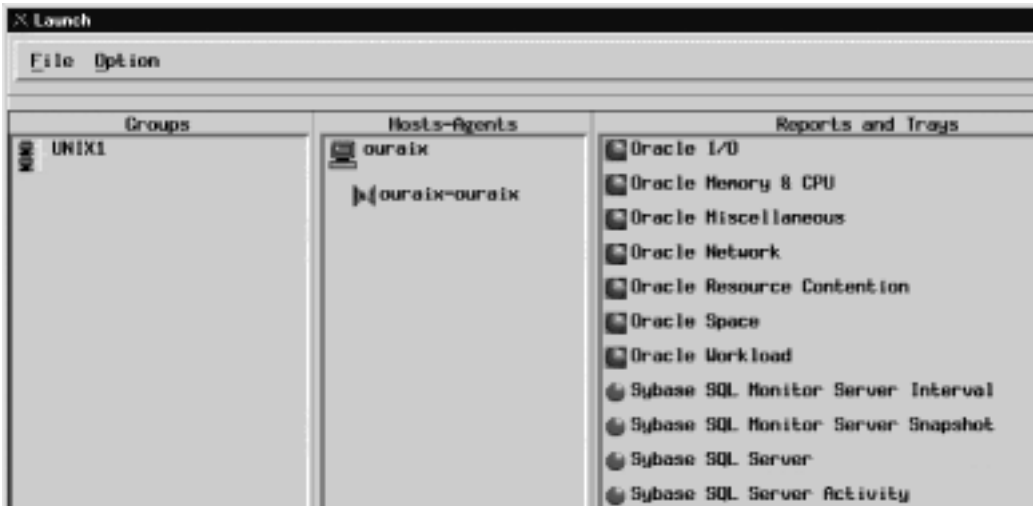

Figure 6: Reporting on applications

just as the operating system does. Databases are further complicated by having to allow many users to access the same tables, sometimes causing locking problems and delays to the user transactions.

Each database system varies in its mechanism for storing and retrieving data, and therefore requires a different agent to access and store performance data. The agent is responsible for compiling information about incoming SQL calls and their effect on performance. This means collecting detailed statistics about tasks, tables, indexes, data spaces, errors, locks encountered, processes invoked, and actual I/Os being performed, in addition to other machine resources being used.

On top of all this, information about network access to the database can usually be recorded. This information is different from that gathered at the operating system level and relates only to network I/O performed specifically in relation to the database. Figure 6 on page 11 shows some 'off the shelf' reports for Oracle, Sybase, and Microsoft SQL Server provided by PerformanceWorks.

THE NETWORK

As mentioned previously, it is possible to obtain a significant amount of information about the size, number, and rate of network packets sent and received by the server. Timings for network packets is a little more difficult to obtain, as user information may not be identifiable from the packet information. There are several ways of monitoring network activity, including 'sniffer' products that monitor the actual network cable and record the amount of data passing. Other products record specific transactions by their TCP/IP header and monitor each send/receive pair. These products record timings, destinations, and the total number of message pairs required to make up a complete transaction.

As more client/server applications are deployed and more use is made of networked applications, so network monitoring is becoming an essential part of enterprise performance management. Most companies have e-mail systems and many are implementing intranets in their internal networks. Use of these facilities demands additional bandwidth, leaving less for other applications.

Collecting information about network use is difficult as the overhead associated with examining every network packet makes such analysis impractical. Analysing a specific transaction provides a better solution, though it may fail to highlight instances when network traffic is approaching its limit. A combination of products is probably required to cover all aspects of the network monitoring that are required.

One product worth considering here is Optimal Expert from actualIT, which analyses a transaction route across the network and provides a modelling facility to identify ways to speed up throughput.

END-TO-END RESPONSE TIME

In order to set up and monitor end-user Service Level Agreements (SLAs), it's necessary first of all to define what measurements of service level are important to end-users. Are system availability, application availability, network access, and response time being provided to a guaranteed standard? Each of these will probably comprise part of the agreement – after all, providing a user with 100% system availability without ensuring that a critical application is running is a little pointless.

End-to-end response time has lately become a measure of service that users tend to require of SLAs. On mainframe systems it has always been possible to measure this (or at least to make a reasonable 'guesstimate'), though this has not been possible in distributed environments.

There are several mechanisms now being developed by hardware and software vendors to address this problem. A consortium of HP, Tivoli, Landmark, and others has developed a mechanism called Application Response Measurement (ARM) that provides end-to-end response timing for applications. This requires some input from the user, in terms of either an application call to be monitored or a definition of what constitutes the transaction. Hopefully ARM will allow administrators to identify transactions that cause users undue delay and to focus tuning effort on aspects of performance that provide the most benefit to users.

Performance tuning is a never-ending process. Just when you think the system is running well, something changes – a change to a program, a new application, a new user, more customers, etc – and the whole process begins afresh. Every change, however small, has an effect on the operation of servers and networks. Collecting all of the information discussed above enables these effects to be identified quickly and reacted to, thus keeping the performance at a constant level (or, at least, within the service level agreement).

Performance data can also be used to perform intensive tuning exercises, where the objective is to reduce response time and improve user productivity. The result of such an exercise can be dramatic and have a positive effect on the whole enterprise, even when only part of it is directly affected. This is because the reduced resource requirements of the 'tuned' application allow other applications to use more, thus improving their throughput and response.

As no two systems are the same, it is meaningless to quote performance targets for which to aim, such as '80% CPU' and '60% cache hit ratio'. Administrators usually look for 'rules of thumb' to guide them when tuning systems, and usually feel let down when none are forthcoming. AIX, surprisingly, runs far better when heavily loaded than with only a light load. There are upper and lower boundaries for this, perhaps between 70% and 95% CPU load. 100% CPU load is even better, but leaves little room for the normal peaks and troughs of the average day's work.

The following suggestions are for general performance tuning:

1 Analyse disk usage by process.

   – Split the most heavily used files and databases across multiple physical disks.

   – Put the system's pagefile(s) on less heavily used disks.

   – Use raw data partitions for databases where possible – their I/O routines are generally more efficient than standard AIX routines.

   – Buy more disks as spreading the load improves performance.

- Buy more disk controllers (again, spread the load).
- Use faster disks and faster channels (use SCSI II where possible).

2 Analyse the paging rates. High paging (over 50 pages/sec) could be caused by a single large application as well as by many running simultaneously.

- Adjust the mix of applications to compensate.
- Check processes for memory leaks by monitoring them over a period of time.
- Remove non-essential processes from the mix.
- Increase the pagefile size. It should be at least three times the real memory size, though much more than four times provides only marginal benefit.
- Add more memory.

3 Analyse total CPU usage. If it is at 100% regularly:

- Analyse operating system CPU usage – it may have a memory problem.
- Check the historical performance figures – did CPU utilization increase suddenly, in which case did the increase coincide with an application change, or was it gradual?
- Analyse CPU usage by process in case one process is hogging the CPU.
- Consider offloading some of your workload.
- Upgrade the CPU.

4 Analyse network traffic. Note that traffic for different network protocols is usually collected separately. If traffic is high:

- High volume traffic for local applications is not a problem as long as the network is capable of handling it, though remote applications should keep volume to a minimum.

- If many users use a common application and run it on the server and not locally, then distribute the application code.
- If your network has routers or bridges, use TCP/IP in preference to any other protocol.
- Implement switching technology for better throughput and balancing.
- Consider upgrading the network itself to Fast Ethernet, ATM, FDDI, etc.
- Ensure the backbone in a large distributed network is as fast as economically possible.

5 Use hourly or daily summary information on I/O, paging, CPU, and disk space usage to plan hardware upgrades. A minimum of three month's worth of historical information is required for this.

It's worth pointing out that most of the above tips are based on evening out usage so that no single resource causes a bottleneck.

It is also important when tuning a system to monitor processes that comprise each application and to understand the way the application works. Planning can then be carried out to accommodate new users or increased transaction volumes. Planned growth is always better than 'fire fighting', and having a proper performance management strategy along with good performance tools makes life considerably easier.

SUMMARY

While IBM and other manufacturers continue to increase the speed, reliability, processing power, and performance of their systems, each new system costs that little bit more. The fact that organizations continue to pay for upgrades proves that performance is an issue. With good management, performance can be managed and demand anticipated.

Ross McCarroll
Product Manager

AIX filesystems

A filesystem is a hierarchical tree structure comprising files and directories. This type of structure actually resembles an inverted tree, with the trunk at the top and its branches at the bottom (at least, as it's normally graphically portrayed). The file tree uses directories to organize data and programs into groups, thus allowing the management of several directories and files at the same time.

A filesystem is a hierarchical structure with a single root. The structure includes the base filesystems and any created filesystems. If a filesystem is to be accessible, then it must be mounted on a directory mount point. Multiple filesystems may be mounted on a directory structure that presents a single filesystem image.

AIX supports multiple filesystem types, including:

- Journaled File System (JFS)

  This is the basic filesystem type. It supports the entire set of filesystem commands.

- Networked File System (NFS)

  A filesystem type that permits files on remote systems to be accessed as though they are on the local machine.

- CD-ROM File System (CDRFS)

  A filesystem type that allows the contents of a CD-ROM to be accessed through normal filesystem interfaces (open, read, and write).

Both local and remote filesystems are accessed using the **mount** command. This makes the filesystem available for read and write access from one's system. Mounting or unmounting a filesystem usually requires system group membership. Filesystems can be mounted automatically, as long as they're defined in the */etc/filesystems* file. Unless a user or process is accessing it, local and remote filesystems can be unmounted with the **unmount** command.

JOURNALED FILE SYSTEM (JFS)

AIX's basic filesystem is the Journaled File System (JFS). This filesystem uses database journaling techniques to maintain structural consistency. This prevents damage to the filesystem when the system is halted abnormally – an abnormal halt in an AIX system results in loss of information in RAM, but does not damage the filesystem. You can imagine the looks of horror I get from users of other types of Unix when I nonchalantly turn off my AIX system – this can cause untold damage to other types of Unix, requiring expert use of **fsck** (filesystem check) and **fsdb** (filesystem debugger) to restore the system.

Every journaled filesystem resides on a separate logical volume. The operating system mounts journaled filesystems during initialization using information in the */etc/filesystems* file. This multiple filesystem configuration is useful for system management functions, such as **backup**, **restore**, and **dd**, as it isolates part of the file tree to enable one to work on it.

There are three types of journaled filesystem:

- *Standard*
  This is the standard AIX Version 3 filesystem, with a maximum filesystem size of 2 GB.
- *Compressed*
  A compressed JFS has its data automatically compressed using LZ compression before being written to disk. Data is automatically expanded when read from disk.
- *Large file enabled*
  This filesystem has a maximum size of 128 GB. This type of JFS has been available since AIX Version 4.2.

NETWORKED FILE SYSTEM (NFS)

The Network File System (NFS) is a distributed filesystem that allows users to access files and directories located on remote computers and use them as if they were local. This allows users to issue operating system commands to create, remove, read, write, and set file attributes on remote files and directories.

CD-ROM FILE SYSTEM (CDRFS)

The CD-ROM File System (CDRFS) allows users to access the contents of a CD-ROM using normal filesystem interfaces, such as open, read, and close. It's a read-only local filesystem that uses the AIX Logical File System (LFS) layer to support the following volume and file structure formats:

- *The ISO 9660:1988(E) standard*
  The CDRFS supports ISO 9660 level 3 interchange and level 1 implementation.
- *The High Sierra Group specification*
  This predates the ISO 9660 standard and provides backward compatibility with earlier CD-ROMs.
- *The Rock Ridge Group Protocol*
  This specifies extensions to the ISO 9660 standard that are fully compliant with ISO 9660 and provide full POSIX filesystem semantics based on the System Use Sharing Protocol (SUSP) and the Rock Ridge Interchange Protocol (RRIP), thus enabling users to mount and access CD-ROMs in the same way as any other Unix filesystem.
- *The CD-ROM eXtended Architecture File Format*
  The CD-ROM eXtended Architecture (XA) format specifies extensions to the ISO 9660 that are used in CD-ROM-based multimedia applications, such as PhotoCD. AIX supports only the Mode 2 Form 1 sector format.

PRACTICE

Enough theory. What implications do these various formats have when it comes to creating and mounting filesystems? The standard Unix command to make a filesystem is the appropriately named **mkfs** (make filesystem). This command can be used to make only journaled filesystems. The **crfs** command (create filesystem), on the other hand, creates filesystems and stores information about them in the */etc/filesystems* file.

- Creating a standard JFS:

```
mkfs logical_volume
```

or

```
crfs -v jfs -p rw -d logical_volume -m mount_point -A yes
     -t no -a frag=4096 -a nbpi=4096 -a ag=8
```

- Creating a compressed JFS:

```
mkfs -ocompress=LZ,ag=8,frag=512,nbpi=512 logical_volume
```

or

```
crfs -v jfs -a compress=LZ -p rw -d logical_volume -m mount_point
    -A yes -t no -a frag=512 -a nbpi=512 -a ag=8
```

- Creating a large file-enabled JFS:

```
mkfs -obf=true,ag=64,nbpi=4096 logical_volume
```

or

```
crfs -v jfs -a bf=true -p rw -d logical_volume -m mount_point
    -A yes -t no -a -a nbpi=4096 -a ag=64
```

- Creating an NFS:

```
/usr/sbin/mknfsmnt -f mount_point -d remoteDirectory -h
     remoteHost -n -N -A -t rw -w bg -H -Y -Z -X
```

- Creating a CDRFS:

```
crfs -v cdrfs -p ro -d cd0 -m mount_point -A yes
```

The parameters **-ocompress=LZ**, **-a compress=LZ**, **-obf=true**, and **-a bf=true** may be included depending on whether **mkfs** or **crfs** is used to make either a compressed or large file-enabled JFS. So, what are the **ag**, **frag**, and **nbpi** parameters?

ag specifies the allocation group size in megabytes. An allocation group is a grouping of inodes and disk blocks, similar to Berkely Software Distribution (BSD) cylinder groups. The default **ag** value is eight. This option applies only to AIX Version 4.2 or later.

frag specifies the JFS fragment size in bytes. A filesystem fragment is the smallest unit of disk storage that can be allocated to a file. The default fragment size is 4,096 bytes.

nbpi specifies the number of bytes per inode (‘nbpi’). The nbpi is the

ratio of filesystem size in bytes to the total number of inodes. The default nbpi value is 4,096 bytes. The values 32,768, 65,536, and 131,072 are available only to AIX Version 4.2 or later.

If these additional parameters seem unnecessary, then consider a filesystem on a news server. Most articles are less than 1,024 bytes in size, which means that three quarters of the filesystem is wasted. This comes from the fact that the fragment size (4,096 bytes) minus size of an average file (1,024 byes) leaves 3,072 wasted bytes per fragment.

Some limitations:

- Only JFS filesystems created with the default **ag**, **bf**, **compress**, **frag**, and **nbpi** values and a size of less than 2 gigabytes are recognized by an AIX Version 3.2 system. Furthermore, filesystems created with an **ag** value greater than eight are not recognized by AIX Version 4.1.
- The **ag**, **bf**, **compress**, **frag**, and **nbpi** attributes are set at filesystem creation and cannot be changed after the filesystem is created. The **size** attribute defines the minimum filesystem size and cannot be decreased once the filesystem is created (a workaround is presented in *Decreasing filesystem size* in *AIX Update* Issue 31, page 3).
- The root filesystem (*/*) cannot be compressed.

As briefly mentioned in *AIX large file support* in the April 1998 issue of *AIX Update*, the command **dumpfs** (dump filesystem) provides information on the filesystem. The command:

```
dumpfs /dev/hd4 | pg
```

Produces the following output:

```
/dev/hd4:
magic                 0x65872143  cpu type             0x0
file system type      0           file system version  1
file system size      32768       fragment size        4096
block size            4096        allocation grp size  2048 (frags)
inodes/allocation grp 4096        compress             0
file system name      /           volume name          root
log device            0xa0003     log serial number    0x12
file system state     1           read only            0
last change           Fri Apr 30 10:55:17 DFT 1998
```

The command also produces some other information that experts can use to debug a filesystem.

CONCLUSION

Don't be overwhelmed by all the parameters available when creating a filesystem – just use **smit** to obtain default parameters. It's to its credit that AIX supports as many options and alternatives as it does when it comes to storing information in filesystems.

Werner Klauser
Klauser Informatik (Switzerland)

RISC System Cluster Technology

This articles examines the RS/6000's RISC System Cluster Technology, discussing its features and benefits.

INTRODUCTION

They always say (whoever 'they' are is a question in its own right!) that you shouldn't start a discussion with an apology. However, before too many people get upset, let's just say that the contents of this article, some of the opinions expressed in it, and my interpretation of them may not necessarily be agreed upon by all readers. I would also like to point out that the opinions expressed here are not in any way those of IBM, but are mine alone (though they may, in places, coincide with those of IBM, if an organization can be said to have an opinion).

Discussions of computing in general and the issue of increased availability and clustering in particular can quickly become unnecessarily complex, so I've assumed familiarity with common IBM products to try and keep the discussion as simple as possible. Anything which is not (at least in my opinion) common knowledge I will try to highlight as I go along. There has to be a limit, however, at

which explanation for explanation's sake offers no value, so if you get lost along the way stick with it through the more complex stuff and it'll get back to reality sooner or later (I hope!).

The basic premise of this article is that we are on the threshold of a new technology that brings with it a new set of benefits. Before you groan, put your head in your hands, and complain that you've just got to grips with the last lot of stuff IBM foisted on you, let me say that RISC System Cluster technology offers the first real opportunity for lights-out computing in a Unix environment. In my opinion this is neither hype nor 'vapourware' – it's a product you can see and touch. There's some work to be done in implementing this technology, but the benefits are there to be had.

RISC SYSTEM CLUSTER TECHNOLOGY

HACMP (High Availability Cluster Multi Processing) for AIX is the best selling and most widely installed Unix high availability and clustering product around. The product is also highly rated by industry consultants, such as DH Brown. So, if the product is so good, why change it? The answer is that there are things that it either doesn't do or could do better.

Number one on this list is scalability. HACMP supports only up to eight nodes per cluster. This is more than most – if not all – of its competitors offer, but still falls short of the scaleability required by some organizations, even if it meets the current needs of most users. Note the 'current' – IBM predicts demand for larger and larger clusters for delivering cost-effective processing power. Larger clusters means more scaleability. The current limit is a historic rather than technical limitation, so we need to remove it in order to move forward. It arises from the mechanism used by the HACMP Cluster Manager for 'heartbeating'. The bi-directional heartbeat mechanism requires each node to check its two neighbours for life, and it's one factor that limits scaleability. Another limitation arises from broadcasting that occurs when a failure is detected. As cluster size increases, more network bandwidth is required by this. While network bandwidth is relatively cheap these days, it's still possible to improve the current mechanism. So, at some point we need to change the heartbeat mechanism.

Secondly, most industry studies that have investigated the causes of outages and downtime have found software and user error to be much more likely causes than hardware failure. The traditional HACMP product is primarily designed to monitor hardware. It has no more than a primitive ability to use AIX functions to monitor software. While it is possible to enhance HACMP and AIX to provide more comprehensive monitoring mechanisms, this demands a sound knowledge of AIX and good programming skills. These skills are both in short supply and expensive, so few organizations have invested in putting software monitoring and management through HACMP into practice. There are a few organizations with whom I have worked that have implemented this, and their clusters really are something to behold. However, if more organizations are to follow their lead, then it makes sense for the vendor to offer a series of well-defined and easily extensible mechanisms to allow the monitoring of software and the automation of basic user functions.

A final reason for a new clustering technology is that IBM has a plethora of clustering products that really ought to be unified. In the AIX arena there are two major management products for clusters: HACMP and PSSP (the system management product for RS/6000 SPs). In addition, there are workload management tools, such as Loadleveler and Network Dispatcher/Interactive Session Support, and other utilities, such as NetTape and RVSD to name but two. Each of these offers its own features and benefits, in addition to which people often prefer one function in one environment and a different function in another, as long as functions are tailored to their own environments. IBM would rather offer you a unified product, however, as testing every product in every environment is impractical, expensive, and makes it difficult to guarantee support. Additionally, a customer buying a $50,000 system usually needs only part of its functionality, and may not be able to afford the infrastructure required by components designed for $5,000,000 systems.

However, what if the underlying infrastructure allowed anything to work anywhere? This would provide benefits to both customers, who'd be able to use the product they need, and vendors, who'd be freed from the tyranny of supporting multiple products and environments.

Let's work through the list of components in a roundabout fashion. HACMP doesn't have a heartbeat mechanism that scales, so where can we find a scalable heartbeat mechanism? The product that provides this is Scalable POWERparallel (to give it its full name) or SP. Actually it's PSSP that provides the heartbeat mechanism, not the SP hardware. SP scales to very large systems – the 'largest' systems in the world (depending on your definition of 'large') are SP systems comprising hundreds of nodes. Hence the SP heartbeat mechanism is proven to scale well. The heartbeat component in PSSP is called Topology Services, and has been present in PSSP releases for some time.

Topology Services (or 'HATS' – High Availability Topology Services) provides a unidirectional heartbeat mechanism. Nodes are installed in a 'heartbeat ring' so that each node checks on its downstream neighbour. This reduces the overhead of the mechanism compared to its bi-directional equivalent in HACMP. Also included are functions, such as source routing and reliable messaging (the ability to guarantee delivery of a message to a cluster node provided there is at least one network path to it), that are provided by Topology Services to create a solid foundation on which to build a clustering architecture. The downside of unidirectional heartbeat rings is that it generally takes longer to discover failures than using bi-directional rings. However, the added efficiency of HATS means that the difference is small for small clusters ('small' still means clusters of more than eight nodes). In larger clusters the difference is greater, though this is perhaps what you'd expect of larger, more complex systems.

The layer above Topology Services is Group Services (HAGS), which provides a mechanism for processes and subsystems to coordinate activities, synchronize, and transfer data reliably. These services may be provided to a set of interrelated processes belonging to the same application or to processes from different applications. Processes that wish to use Group Services facilities subscribe to a group. Group members use voting protocols to coordinate their activities and transfer state data to each other. While this article isn't really about what Group Services is or what services it provides, it is nevertheless important to know that this component is a key part of the infrastructure.

The final component of the underlying cluster infrastructure is Event Management (HAEM). This component is the focus of this article and I'll discuss in detail both its facilities and the way in which its power may be harnessed to drive automated recovery and automated operations. Event Management includes a Resource Monitor that provides information to its local Event Management daemon. Event Management daemons pass information to one another using Group Services, thereby providing a unified Event Management environment that's both consistent and cluster-wide, and also allows events to be ordered in terms of when they occurred. An Event Management client registers its interest in one or more events using a construct called a 'predicate'. When the event occurs, and the conditions defined in the predicate are met, the Event Management client is notified.

I hope these three key components – Topology Services, Group Services, and Event Management – are familiar to users of SP systems, as they've been part of PSSP for a while. Users of generic RS/6000 systems may also be aware of them – as well as being parts of PSSP, these components are also parts of something rather uncatchily named 'RISC System Cluster Technology' (RSCT). Readers may also have read about an IBM technology called 'Phoenix'. This was far too catchy a name for IBM, and was probably also trademarked to the hilt, so Phoenix became RSCT. Note that early press releases mentioned IBM's intention to deliver Phoenix technology on platforms other than RS/6000 SP systems – RSCT is the vehicle for this. It will become the key enabling technology for both SP systems and generic RS platforms, and is the first step in the process of merging PSSP and HACMP.

Event Management provides facilities that dramatically extend the ability of products such as HACMP to provide availability services. With any computing system, results are only as good as the data that is provided – the key data suppliers to Event Management are its resource monitors. There are five or six resource monitors that are supplied with the system to address the main monitoring requirements of most systems. In addition, an API is available that makes it relatively simple to create new resource monitors. Resource monitors either monitor resources themselves or rely on other sources of data, such as the Performance Toolbox (PTX) or the AIX error log. Rather

than looking at the whole spectrum of availability options, it is probably best to concentrate on one or two key areas where the use of Event Management either provides new functionality, such as a simple mechanism for software monitoring, or allows a task to be completed more simply than before, such as reacting to a new error log entry. Throughout this discussion, it's important to remember that, as Event Management provides a single view of events throughout the cluster, an Event Management client reacting to an event doesn't have to be physically on the node at which the event occurs.

Resource monitors supply information about the resources on a system. A resource is an entity that provides a service either to users or the system. For each attribute of a resource, the resource monitor defines something called a 'resource variable'. Thus a disk resource might have a series of attributes, such as percentage of space used, percentage of time busy, etc, in which we might be interested. As it's likely that there are many resources of a particular resource type in the system, a mechanism is needed to differentiate between them. The resource monitor does this by providing different instances of the resource variable. So, if a cluster node has three disks, the disk resource monitor would report three instances of the disk resource variable for that node.

Reporting by the resource monitor to the Event Management subsystem occurs at predefined times. This may be periodically, at configurable intervals, or every time the measured attribute changes.

The Event Management client acts on information about the condition of system resources. A condition in which an EM client is interested is called a predicate; an EM client indicates that it wishes to be notified when the state of a specific resource instance matches a certain condition by registering a predicate for this particular resource variable instance. The predicate is thus a condition that a resource variable must meet in order to generate an event. More strictly, it is a relational expression involving an instance of a resource variable and other elements, such as constants and the value of the variable instance from the previous observation. This predicate is applied to each instance of the resource variable as it is observed. If the predicate is true, an event is generated. Just to add to the fun (some might say complexity), more

than one predicate may be applied to any instance of the same resource variable at the same observation.

A simple predicate might be *X < 10*, where *X* is a resource variable that represents the percentage of free space in a file system. This predicate generates an event whenever the file system's free space is observed to be less than 10%.

A predicate has to take one of the following forms:

```
expression rel_op constant

expression rel_op expression

predicate log_op predicate

unary_op predicate

unary_op expression
```

where:

```
expression::          var_name

                      var_name arith_op constant

                      var_name arith_op expression

var_name::            X

                      X@var_name_mod

var_name_mod::        one of P, R, PR, sbs_sn, Psbs_sn

unary_op::            !

rel_op::              one of ==  !=  <  >  <=  >=

log_op::              one of &&  ||

arith_op::            one of *  /  %  +  -

sbs_sn::              a structured field serial number
```

Operators have the same meaning and precedence, and parentheses are used in the same way for grouping, as in C. Predicates use the letter 'X' to represent the resource variable name. The variable name may be repeated in the predicate and may be in any one of its modified forms.

Here are some examples of valid predicate definitions:

```
X == 0
```

The value of the resource variable instance is equal to zero.

```
X < 20 || X > 80
```

The value of the resource variable instance is less than 20 or greater than 80.

```
!(X < 20 || X > 80)
```

The value of the resource variable instance is neither less than 20 nor greater than 80.

```
X@R > X@PR
```

The current raw value of the variable instance is greater than the previous observed raw value.

```
X >= X@P + 5
```

The current value of the variable instance is greater than or equal to the value of the variable instance from its previous observation plus five.

Having covered the concept of a predicate, we now need to apply it to resource variables.

As stated before, a resource variable is associated with an attribute of a system resource. It has a name, a value type, a data type, an instance vector, and (optionally) a location.

The name of a resource variable is a string that consists of a resource name followed by a resource attribute (using periods as separators). There is a convention for naming resource variables in a hierarchical fashion starting with the vendor that supplies the resource. Some example resource variable names are:

```
IBM.PSSP.aixos.CPU.%user

IBM.PSSP.aixos.Mem.Kmem.inuse

IBM.PSSP.aixos.PagSp.%totalfree

IBM.PSSP.aixos.Disk.busy
```

As most resources in the system are not unique, the various instances of a type of resource need unique identifiers to be referenceable. This

is provided by multiple instances of the resource variable. This means that there needs to be a method of uniquely identifying each resource and its associated variables. This is provided by an instance vector. Each resource in the system is associated with one, and only one, instance vector. An instance vector is a list of elements, where an element is a name-value pair that allows unique identification.

The form of an instance vector is defined by the form of the resource variable with which it is associated. The name of a vector element must be unique within a given variable's instance vector. These names may be used in other instance vectors and may or may not have the same form. In other words, the name space for instance vector element names is local to each defined resource variable. Wildcards may be used when referring to instance vector elements when the event that is being monitored may occur at more than one location. The set of values in the instance vector uniquely identify the resource in the system. By extension, they also uniquely identify the resource variable in the system.

Below are examples of PSSP resource variable names and instance vectors.

```
Resource variable name          Instance vector

IBM.PSSP.aixos.CPU.user         NodeNum=5

IBM.PSSP.aixos.CPU.user         NodeNum=3

IBM.PSSP.aixos.Disk.busy        NodeNum=*;Name=hdisk0

IBM.PSSP.aixos.Disk.busy        NodeNum=5;Name=hdisk1

IBM.PSSP.aixos.FS.%totused      NodeNum=5;VG=rootvg;LV=hd4
```

IMPLICATIONS AND EXAMPLES FOR AVAILABILITY

HACMP is not normally responsible for monitoring processes. However, Event Management allows it to take on this role with the aid of a resource monitor that monitors processes. This provides two Event Management resource variables:

```
IBM.PSSP.Prog.pcount

IBM.PSSP.Prog.xpcount
```

These two variables provide similar functions, so care should be taken to ensure the correct one is used. *IBM.PSSP.Prog.pcount* is associated with all processes that are running a program, regardless of how the program was called. *IBM.PSSP.Prog.xpcount*, on the other hand, is associated only with those processes that are running a specified program as a result of calling an *exec()* routine. Typically, a process runs a program by calling an *exec()* routine, specifying the program. It's possible, however, that a process is running a program after inheriting it from its parent process, and not from calling an *exec()* routine. Some daemons do this. For example, the **ps** output below shows that only the process whose PID is *16706* called an *exec()* routine to run the **biod** program. All other processes inherited **biod** from their parent process – the process with PID *16706*.

```
# ps -ef | grep biod
root 15942 16706   0   Oct 12      -  0:16 /usr/sbin/biod 6
root 16706  2344   0   Oct 12      -  0:15 /usr/sbin/biod 6
root 16972 16706   0   Oct 12      -  0:15 /usr/sbin/biod 6
root 17224 16706   0   Oct 12      -  0:15 /usr/sbin/biod 6
root 17486 16706   0   Oct 12      -  0:15 /usr/sbin/biod 6
root 17744 16706   0   Oct 12      -  0:15 /usr/sbin/biod 6
```

Use *IBM.PSSP.Prog.pcount* to monitor processes that inherit programs, and *IBM.PSSP.Prog.xpcount* to monitor processes that explicitly call an *exec()* routine to run programs.

Using *IBM.PSSP.Prog.pcount* to monitor certain classes of program, such as those that create child processes to run other programs, may generate unwanted events. Consider **inetd** as an example. **inetd**'s function is to spawn other daemons. When a service is requested of a daemon it controls, **inetd** calls *fork()* to create a child process. The child process starts out running the **inetd** program, but soon calls an *exec()* routine to run the appropriate daemon, such as **telnetd**. Therefore, for a brief period, more than one process is running the **inetd** program. If *IBM.PSSP.Prog.pcount* is used to monitor **inetd**, events might be generated showing these child processes running **inetd** during this period. To avoid these 'false' events, it would be better to use *IBM.PSSP.Prog.xpcount*.

As this example demonstrates, you need some knowledge of how a program operates in order to decide the best way to monitor it. For most cases, it is probably appropriate to use the *.xpcount* resource variable to monitor programs. However, if the program to be monitored is inherited by long-running processes that don't call an *exec()* routine, *.pcount* might be appropriate.

Both *IBM.PSSP.Prog.pcount* and *IBM.PSSP.Prog.xpcount* have instance vectors that specify the program name, the user name, and the node number. Wildcards cannot be used to specify the *ProgName* and *UserName* instance vector elements, but can be used to specify *NodeNum*.

If you wish to monitor the **inetd** process on node 5, the instance vector to use is:

```
ProgName=inetd;UserName=root;NodeNum=5
```

A change in the value of either *IBM.PSSP.Prog.xpcount* or *IBM.PSSP.Prog.pcount* indicates that the number processes running the program has changed. To determine whether the number has increased or decreased, compare the current value of the process count with the previous value. This comparison should be in the predicate, generating different events if the number increases or decreases.

Suppose the event you're monitoring is a change in the number of processes running a program. The predicate for this event is:

```
X@0 != X@1
```

This doesn't tell you whether new processes have started or existing processes have stopped or died, only that the number of processes has changed. If you wish to monitor for a reduction in the number of processes (resulting from processes stopping or dying), the predicate is:

```
X@0 < X@1
```

This is used to monitor a set of processes with the same name, such as the **biod** daemon example above. If you're monitoring process death (when no processes run the program), the predicate tests that the number of processes running the program equals zero:

```
X@0 == 0
```

This can also be used to monitor for the existence of a single important process.

Both *IBM.PSSP.Prog.pcount* and *IBM.PSSP.Prog.xpcount* are designed to monitor programs that have long lifetimes. If they are used to monitor programs that run for only a few seconds, they may fail to detect them.

The choice of recovery action to handle the death of a process event has to be made with care. While it may seem that the obvious course of action is to restart the process, this can lead to further problems. Consider the following examples:

- *A resource is or becomes unavailable*

  In many situations, when a process fails it's not because of a fault in the process itself but as a result of a problem elsewhere in the system. For instance, suppose a process logs activity to a file. When the filesystem that contains the log file fills up, the process is unable to log, and so abends. In this situation, restarting the process has no effect whatsoever, as the process simply dies again. Only by rectifying the underlying cause of the process's death can the process be restarted successfully.

- *Controlled stop of a process versus process death*

  When a process is stopped in a controlled fashion it often performs clean-up activities. If a process dies, no clean-up occurs, and so an attempt to restart the process fails. Consider an application that runs as a single process. When the application runs, it stores its process ID in the file */tmp/app.PID*. This file exists to prevent multiple instances of the process from running on the system at the same time. Should the application's process fail, an attempt to restart it will also fail as the file */tmp/app.PID* has not been deleted. This fools the start-up routine into thinking that an instance of the application is already running on the system. The only way to restart the application is by first removing the remains of the previous instance of the application.

- *Failure of one of a number of multiple interrelated processes*

  Many applications consist of multiple interrelated processes.

Such processes often communicate by some proprietary mechanism, such as using an area of shared memory. In such instances, the uncoordinated restart of a single process from the group may lead to additional problems. For instance, the newly restarted process may simply be unable to communicate with the ones from the original invocation of the application.

When running an application that consists of multiple processes, you should always restart the entire application if one of its constituent processes fails, rather than restarting just the failed process (unless you're sure that processes that comprise the application may be restarted independently).

- *Controlled process death*

  It is difficult for Event Management to differentiate between the real failure of a process and a controlled stop, particularly if the process is stopped legitimately by a qualified user. This requires the recovery process to differentiate between the two. It's annoying for users to kill a process only to see it restart a few moments later.

This example shows how an action is taken in response to an event about a software or (more correctly) process failure. This function was not easily and reliably available before, and has been added through RSCT. A range of other functions have also been added that, while not actually new, are much simplified or extended.

Consider, for example, the way the system normally reacts to an event that writes an entry to the AIX error log. Such events are mainly of interest as a result of the fact that they're used to monitor devices that are not monitored by HACMP itself. To monitor the AIX error log you create an error notification object. When an error is logged by a device driver, the error logging process compares the error with the list of objects in which you're interested. If a match is found, an event is generated. Anyone who's ever tried to use error notification 'in anger' knows how much 'fun' it can be! RSCT makes this whole process much easier.

Just as in the case of processes, there is a resource monitor that monitors the error log. The Event Management resource variable used for monitoring the AIX error log is:

```
IBM.PSSP.pm.errlog
```

IBM.PSSP.pm.errlog contains information from an entry written to the AIX error log. The sole instance vector for the resource variable is *NodeNum*. This specifies the number of the node on which the error log that you wish to monitor resides. As in many other situations, the *NodeNum* instance vector element may be 'wildcarded'. The resource variable's value is a structured byte-string that contains strings of data from the AIX error notification daemon.

If we wish to monitor the AIX error log on node 1 of the cluster, the instance vector is:

```
NodeNum=1
```

A change in the value of an instance of *IBM.PSSP.pm.Errlog* indicates that a new entry has been written to the error log(s) being monitored. To monitor changes to the error log (which indicate new log entries) use the predicate:

```
X@0 != X@P0
```

In other words, you're interested in instances when the value of SBS field serial number 0 (the sequence number of the error) is different from its previous value. Each entry in the AIX error log has a sequence number; these are assigned in increasing order.

Creating a condition with a predicate like this tells you that an error has occurred and has been logged. It doesn't tell you anything about the importance of the error. To monitor the error log for a specific error or set of errors, use a predicate similar to the one below.

```
X@0!=X@P0 && X@1=="0x476b351d"
```

In the above predicate, *X@1* refers to SBS field serial number 1. This is the error id associated with the error. In this case, *476B351D* is the error id for a *TAPE_ERR2* error, which is a permanent failure of a tape drive attached to the system. The above predicate could also have been written in the form:

```
X@0!=X@P0 && X@8=="TAPE_ERR2"
```

Here *X@8* is the field in the SBS that contains the error label. To identify the error label or error id for a particular error, use the

command **errpt -t** (the error id for a failure is the first field and the error label is the second field in the command's output).

The action used to handle an error log entry event depends on such factors as how critical the resource is within the system, and whether recovery is possible (in which case you've got to decide where recovery should take place). There are various generic actions that may be taken, including:

- Restart the failed resource locally.
- Failover the resource to another local resource.
- Failover the resource to another cluster node.

Hopefully this example illustrates how RSCT dramatically improves the availability of the system. Availability is not just about failure handling, however – you also need good system management to ensure that the system remains available.

IMPLICATIONS AND EXAMPLES FOR SYSTEMS MANAGEMENT

AIX provides many facilities, such as the Logical Volume Manager and its dynamically pageable kernel, to reduce downtime arising from system management. The function that a component such as HACMP adds to this is the ability to maintain a service to end users while a cluster node is down for maintenance, such as applying system fixes or upgrading software and hardware. This improves system availability. In effect this is like a controlled failure – a node goes down and its workload is automatically moved to a surviving node so users can continue working. When the maintenance process is complete, the node can be reintegrated into the cluster, and its workload restored to its original location. This process may be repeated on other cluster nodes to allow maintenance of all systems.

It's important to differentiate between the service provided by a node, which can be provided by other cluster nodes, and that provided by the cluster as a whole. Using HACMP, all high-availability services provided by individual nodes may be moved to other nodes.

There are a number of areas still that need to be addressed in order to

improve availability and reduce administration. As computer systems get larger and more complex they become increasingly hard to manage. Clustering adds to this problem. Ten independent systems in a network are significantly easier to manage than ten systems in a cluster because of interdependencies between clustered systems. Consequently, tools are needed to assist cluster management.

Another consideration is that mission-critical systems should have as little reliance on external resources as possible. As much of the necessary functionality should be on the local system as possible. This allows swifter action to be taken in response to a failure, and also increases the likelihood of corrective action working.

Consider the case of a component failure in a computer system and its recovery. In the vast majority of cases, the system is significantly faster than operators in detecting the failure of a component. The time it takes for the system to detect a failure depends on how often the component is used, sampled, or monitored for correct operation. The time it takes for an operator to detect the same failure depends on a complex set of factors, such as whether he or she is present when a physical manifestation of the problem arises (a message on a console, a warning light on a device, smoke billowing out of a component, etc) and the tools that are available to assist in isolating an error, and is also dependent on the skill of the operator in question. Even the most skilled operator is potentially several orders of magnitude slower than the system in detecting a failure. Until a problem is detected and the failing component identified, no recovery action can take place.

When it comes to recovery from failure, the system again has an advantage in terms of speed. This may not, however, be as important as its advantage in being able to take correct action. My experience is that skilled operators know what action to take and promptly start the correct recovery procedure. But what if the operator is new to the job or is unfamiliar with this particular system? What if this recovery procedure hasn't been invoked for several months or even years? What if there isn't a recovery procedure for this failure that has been defined within the organization? In the worst case, an incorrect recovery action could be invoked that actually causes more trouble than the original problem! Whatever the situation, operators usually

take longer to recover from the failure. This is time when the system is probably unavailable to run its production workload.

RSCT provides facilities to allow system conditions to be handled automatically before they require maintenance. Administrators can monitor trends in the state of various system components as they change over time or against a pre-defined set of thresholds using Event Management and an appropriate set of instance vectors and predicates. This allows potential problems to be rectified before they become critical and result in failure. The result is a perceptible increase in availability as the recovery of non-critical problems is usually quicker than the recovery of critical problems and failures. While the system still needs to be taken off-line to perform certain maintenance tasks, using RSCT reduces planned downtime.

HACMP and RSCT can be regarded as providing 'system intelligence'. Once configured, the facilities they provide mimic those of a skilled operator who's always on duty and never makes mistakes. This allows the system to react swiftly to conditions as they arise, to select the correct course of action to handle a particular event, and to run this action in the most suitable way to maintain the availability of the system. If the action itself fails, RSCT can even determine a secondary approach to handle the condition and use that instead.

To illustrate the power of Event Management in automating system operations, consider the following two examples of the automation of common tasks. One of the most common and easily avoidable causes of outages on Unix and other systems is a failure to perform periodic housekeeping, resulting in filesystems becoming full. While many sites implement **cron** jobs to manage filesystems, if **cron** runs at night and the critical filesystem becomes full during the day, you have a problem on your hands. However, you can use Event Management to automate disk space management. In an AIX environment the main critical filesystems are likely to be */*, */var*, and */tmp*. Filesystem space can be monitored using the resource variable:

```
IBM.PSSP.aixos.FS.%totused
```

The typical instance vectors for monitoring filesystems across a cluster are:

```
NodeNum=*;LV=hd4;VG=rootvg

NodeNum=*;LV=hd9var;VG=rootvg

NodeNum=*;LV=hd3;VG=rootvg
```

A typical predicate for the ‘approaching full’ condition is:

```
X>90
```

When this evaluates to true, an event is generated and an automatic action can be taken to free up filesystem space.

Another common problem with heavily loaded and badly configured systems is paging space running out. As AIX uses virtual memory, it relies on correctly functioning paging space(s). If paging space runs low, processes may fail. If paging space runs out, the system may fail. The system monitors the number of free paging space blocks and can detect a shortage. When the number of free blocks falls below a threshold known as the ‘paging space warning level’, the system informs all processes (excepts *kprocs*) of the condition by sending a *SIGDANGER* signal. If the shortage continues and falls below a second threshold known as the ‘paging space kill level’, the system sends a *SIGKILL* signal to alert processes that don’t have a signal handler for *SIGDANGER* (the default action is to ignore *SIGDANGER*). The system continues sending *SIGKILL* signals until the number of free paging space blocks is above the paging space kill level. If you don’t wish to kill processes in this fashion because they don’t handle SIGDANGER appropriately, then you need a warning mechanism. This may be implemented using the following Event Management resource variable:

```
IBM.PSSP.aixos.PagSp.totalfree
```

By default, *SIGDANGER* is first sent when the number of free paging space pages falls below 512, though this may be changed using **vmtune**. The predicate for the resource variable thus needs to be triggered before this limit is reached, for example using:

```
X<600
```

While proper capacity planning should ensure that this event does not occur, additional paging space could be automatically added or other action taken to restrict the heaviest users of page space if it does.

CONCLUSION

I hope these simple examples show you something of the power of RSCT and its ability to enhance availability and automate system management. While IBM has been at the forefront of clustering technology, it hasn't always backed this up with products for managing clusters. Given the current focus on clustering that's resulted from Microsoft's Wolfpack (which has still to live up to its promises), the industry seems to have woken up to the reality of clustering. Perhaps now RSCT will get the attention it deserves.

John Easton
IBM (UK)

SMP's SystemGuard and Fast IPL

I consider myself a Unix professional with good AIX skills. Nevertheless there are still many things that I haven't come across and that still trip me up – no matter how much you know about a system of the complexity of AIX, there are always things left to learn. Here's something that I learned about the hard way and would like to share with you.

While working at an AIX site recently, I was more than a little surprised at having to wait almost ten minutes for a model 7012 G to boot. After using **info** to investigate why this might be the case and finding nothing special, I asked around and looked at the *Operators Guide*. Here's what I discovered about the lengthy boot process (or, as IBM calls it, IPL or Initial Program Load) when using RS/6000s with symmetric multiprocessors.

SYSTEMGUARD

IBM's Symmetric MultiProcessor (SMP) products are designed to be used as servers in commercial environments. The availability of these servers and their resources is very important. Given that such servers

often operate unattended or in remote locations, remote diagnostics and services are key components.

IBM's family of SMP servers all have a service processor, called the SystemGuard, that continually monitors the system's hardware and operating system. If, for example, a CPU were to fail, the system would detect this, reboot itself automatically, and run without the failed CPU.

The main features of the SystemGuard are:

- Initialization of process flow management.
- Local and remote system control (power-on/off, diagnostics, reconfiguration, maintenance).
- Console mirroring to make remote actions visible and controllable by the customer.
- Run-time surveillance.

POWER-ON TESTS (PON)

PON is run by SystemGuard whenever the system is powered on. PON comprises two types of test:

- Tests of the processor, cache, memory, and related hardware. These test, understandably, cannot be turned off.
- Tests of other system resources. These other tests can be turned off by setting the 'Fast IPL' flag.

HOW TO SET FAST IPL

If the Fast IPL flag is enabled, SystemGuard skips the Power-On Self Test (POST). By default, the Fast IPL flag is disabled. Furthermore, if you enable it remains enabled for one reboot and is then automatically disabled by the system.

There are three ways to enable Fast IPL: through the Stand-By Menu in Stand-By mode, through the Maintenance Menu, or by using AIX commands.

Setting Fast IPL from the Stand-By menu

1 Set the System Key to Service.

2 Enter the Stand-By menu with the command **sbb.**

3 Enter ‘1’ to select Set Flags.

4 Enter ‘6’ to set the Fast IPL flag.

Setting Fast IPL from the Maintenance menu

1 Enter the Maintenance menu.

2 Enter ‘8’ to select the SET PARAMETERS menu.

3 Enter ‘4’ from the SET PARAMETERS menu to select the MISCELLANEOUS PARAMETERS menu.

4 Enter ‘3’ to enable the Fast IPL flag.

Setting Fast IPL from an AIX command prompt

1 Log into AIX as a root user.

2 Type **mpcfg -df** to find the index of the Fast IPL flag and see its current value.

3 The index number should be 11 and the current value should be zero (by default).

4 Type **mpcfg -cf 11 1** to change the status of the Fast IPL flag to enabled. ‘c’ is for ‘change’, ‘f’ is for ‘flags ‘11’ is the index, and ‘1’ the value itself’.

Fast IPL is now enabled and will last one reboot. Reboot the system with **reboot** or **nicer**, or shutdown with **shutdown -Fr**. You’ll find the time it takes to reboot the system is reduced from more then ten minutes to only one or two.

Werner Klauser
Klauser Informatik (Switzerland)

Contributing to *AIX Update*

AIX Update is primarily written by practising AIX specialists in user organizations – not journalists, or consultants, or marketing people. In our view, the information and advice provided by such people – people like you and your colleagues – are far more valuable to their fellow professionals than the alternatives available from other sources.

We don't expect you to write an original article from scratch – just send us listings or specifications of any relevant programs, utilities, scripts, user modifications, or other code that might be of use to other installations, with a short explanation of why it was developed and what it does. And most IS departments produce a great many internal technical reports and other documents, many of which can easily be adapted for publication. Xephon's editorial staff will transform even the most informal listing or document into a polished article fit for publication. So you don't have to spend any time on reformatting or rewriting your contribution – just send it as it is and we'll do the rest.

Contributors aren't just helping their fellow professionals – they also receive a significant material reward themselves. We pay good rates for the articles we publish: $250 (£170) per 1000 words if we get copyright, and $140 (£90) per 100 lines of code.

A copy of *Notes for contributors* can be downloaded from Xephon's Web site at *www.xephon.com*. Articles for submission to *AIX Update* can be sent to the editor, Harry Lewis, at *HarryLewis@compuserve.com.*

AIX news

IBM has announced AIX Version 4.3.1, offering greater scalability, extended support for standards, and conformance to the Open Group's Unix98 specification. It also includes the first full release of Web-based System Manager and eNetwork LDAP Directory Version 1.1.1.

The Web-based System Manager is for managing AIX systems from anywhere, and includes management tools for devices, print queues, communications, file systems, logical volumes, back-ups, users and groups, processes, and subsystems.

The eNetwork LDAP directory is based on LDAP Version 2 plus some extensions for Version 3, using DB2 as the directory data storage facility.

System performance has been improved with streamlined JFS locking, Java performance is claimed to have improved by up to 40% using new versions of the Java toolkit (1.1.4) and the JIT compiler, there's also improved DES and CDMF encryption and decryption algorithms, and an optional fork interface primarily for use by applications such as Web servers.

AIX 4.3.1 now supports a maximum of 32,767 open files per process, one million open files per system, 32,767 threads per process, and 2GB shared or mapped files.

The product is shipping now and there won't be any changes to current pricing.

IBM also announced Version 3.2 of its eNetwork Firewall for AIX, which combines application proxies, 'Socks' circuit gateway, and filtering architectures that work together to provide more flexibility than a single firewall architecture product.

It includes Socks v5 security protocol standard, which supports User Datagram Protocol (UDP) applications such as net.meeting, RealAudio, and RealVideo. It also provides real-time performance statistics by both application and user-id, and outbound authentication. Some of the facilities come via Aventail.

Also new is Security Dynamics ACE/Server software and two SecurID tokens for two-factor user authentication. There's also support for eight translated languages, including French, Japanese, Korean, and Chinese.

Out now, it costs US$2,499 for a 25-user licence.

For further details contact your local IBM representative.

* * *

Bull has begun shipping an enhanced Escala family of AIX clusters, including its first 64-bit model, and Web-based tools for simplified administration and management of both production and OLTP-intensive applications. Prices start at US$30,000.