92

AIX update

June 2003

In this issue

AIX Update

Published by

Xephon
27-35 London Road
Newbury
Berkshire RG14 1JL
England
Telephone: 01635 38342
From USA: 01144 1635 38342
E-mail: trevore@xephon.com

North American office

Xephon
PO Box 350100
Westminster, CO 80035-0100
USA
Telephone: 303 410 9344

Subscriptions and back-issues

A year's subscription to *AIX Update*, comprising twelve monthly issues, costs £180.00 in the UK; $275.00 in the USA and Canada; £186.00 in Europe; £192.00 in Australasia and Japan; and £190.50 elsewhere. In all cases the price includes postage. Individual issues, starting with the November 1999 issue, are available separately to subscribers for £16.00 ($24.00) each including postage.

AIX Update on-line

Code from *AIX Update*, and complete issues in Acrobat PDF format, can be downloaded from our Web site at http://www.xephon.com/aix; you will need to supply a word from the printed issue.

Editors

Trevor Eddolls

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.

Contributions

When Xephon is given copyright, articles published in *AIX Update* are paid for at the rate of £170 ($260) per 1000 words and £100 ($160) per 100 lines of code for the first 200 lines of original material. The remaining code is paid for at the rate of £50 ($80) per 100 lines. In addition, there is a flat fee of £30 ($50) per article. To find out more about contributing an article, without any obligation, please download a copy of our *Notes for Contributors* from www.xephon.com/nfc.

Printed in England.

Capacity Upgrade on Demand facilities planning and implementation

Capacity Upgrade on Demand (CUoD) enables selected IBM pSeries systems additional operation and configuration flexibility. Available for a fee, CUoD allows customers to install additional processor boards populated by active as well as inactive processors. Inactive processors can be activated on demand to meet increasing workload demands. With dynamic LPARs, this operation can be performed without interrupting the normal operation of the system.

CUOD PROGRAM OVERVIEW

IBM currently offers Capacity Upgrade on Demand for p660-6M1, p670, and p690 systems. Availability of this program for Model p650 and Model p655 is expected to be announced soon.

This article will focus on the unique features of Models p670 and p690 CUoD features.

The design is based on the use of a CUoD capacity card that has a smart chip to provide the required high level of asset protection for the CUoD function. This capacity card will be placed in a slot in the primary RIO book (FC 6404). This CUoD capacity card tracks activated CUoD resources. The new design of Capacity Upgrade on Demand for IBM pSeries is based on the IBM iSeries design. Hence, it differs from the former method of implementation of the CUoD on previous pSeries models such as the pSeries 660 Model 6M1 and the pSeries 680.

For Models p670 and p690, the program enables installation of MCMs (8-board processors) with four active and four inactive processors. Processors can be activated in pairs to expand the server's performance. Model p660-6M1 can be equipped with an additional 4-way processor card with 0 or 2 pre-activated processors.

Feature code	Machine Type-Model	Server	Description
7412	7040-671	P670	Two processor activation for CUoD processor feature #7402 (8-way 1.1 GHz POWER4 with 128MB L3 cache, 4-way active).
7413	7040-681	P690	Two processor activation for CUoD processor feature #7403 (8-way 1.1 GHz POWER4, 4-way active).
7415	7040-681	p690	Two processor activation for CUoD processor feature #7405 (8-way 1.3 GHz POWER4 Turbo, 4-way active).

Figure 1: Available features

Models p670 and p690 CUoD processor activation is based on an activation code, which has to be purchased as an upgrade (MES feature) like any other hardware upgrade.

Available feature codes are summarized in Figure 1.

In addition to placing an order for the appropriate CUoD features, the customer must submit to IBM the VPD (Virtual Product Data) – a collection of system and profile information describing the hardware configuration of the server to be upgraded. IBM will perform the verification of order details and will generate an activation code and a password used to activate the ordered features. Each activation code is unique to a specific system.

The rules below must be followed when the evaluation of alternative system configurations is performed:

- The first MCM in a p670 or p690 system implementing CUoD for processors must always have eight fully-active processors.
- The second, third, and fourth MCMs in the p690 or the second MCM for p670 may be implemented with CUoD for processors if desired.

- Each CUoD MCM must have a minimum of four of the eight processors active.
- Ordering the appropriate activation feature number activates additional processors on the CUoD MCMs. If more than two processors are to be activated at the same time the activation code should be ordered in multiples.

According to these rules it is possible to configure the Model p670 with the following number of processors: 8, 12, 14, and 16. Model p690 can be configured with 8, 12, 14, 16, 20, 22, 24, 28, 30, and 32 processors.

PROGRAM REQUIREMENTS

The following are required to support CUoD for processors:

- AIX 5L V5.1 (with AIX 51-003 or higher Recommended Maintenance Package and PTFs U483632 and U483633) or later. These PTFs resolve the APAR IY36013: “ADD SUPPORT FOR CAPACITY UPGRADE ON DEMAND TO AIX 5.1”. They demand installation of AIX 5.1F ML03 (5100-03). These PTFs update the AIX 5.1F configuration methods to correctly flag the status of unlicensed CUoD processors and allow AIX 5.1 to generate the VPD information needed to activate CUoD processors. A system running AIX 5.1 (5100-03) with these PTFs will be able to activate CUoD processors without requiring any AIX 5.2 partitions.
- Hardware Management Console for pSeries with Version 1.0 Release 3 or later software.
- p690 (7040-681) system microcode (machine code) at the 10/2002 update or later.
- AIX 5L V5.2 or later is recommended because it provides enhanced CUoD function.

CUoD on servers configured with AIX 5.1 is subject to the following limitations:

- In SMP mode, activation of the standby processors requires

a system IPL to enable additional processors.

- In AIX 5.1 LPAR mode, activation of the standby processors requires a partition IPL to enable additional processors.
- In a mixed AIX 5.1 and 5.2 LPAR mode, activation of the standby processors for any partition running AIX 5.1 requires a partition IPL. AIX 5.2 partitions can dynamically allocate additional processors.
- Dynamic processor sparing is limited to systems and/or partitions that are running AIX 5.2. Systems running only AIX 5.1 provide processor sparing for a failed processor after a system IPL.

ACTIVATION PROCESS FOR CAPACITY UPGRADE ON DEMAND

After the customer has determined the need to activate a certain number of standby processors, they must place an order for a proper number of CUoD activation features with their IBM sales representative. Then Vital Product Data (VPD) from the server to be upgraded has to be sent to IBM in order to generate the activation code.

The submission of the VPD can be performed in one of two possible fashions:

- For systems operating under the Electronic Service Agent control you can submit the VPD automatically.
- For other systems, the information has to be collected manually and entered into the Web-based VPD Profile Entry Tool.

Electronic Service Agent-based VPD entry

Run Service Agent:

1 In the Navigation area, click the Service Applications icon.

2 In the Contents area, double-click the Service Agent icon.

3 In the Service Applications window, click Service Agent registration/customization.

4 Type the Service Agent password. The default password, which is case sensitive, is:

```
password
```

5 Check whether Service Agent has collected VPD properly from the systems. You can view the status in the GUI (**Network/hmcSystem/CECs/cecmtms**).

6 If VPD is displayed:

- in the Navigation Window, click **SAS-Connections**
- click **Send VPD**
- select the HMC that is monitoring the system and click **Send VPD**.

7 If VPD is not present, ensure that inventory scout can be executed successfully.

8 Ensure VPD collection is not disabled in Service Agent (**Network_hmcSystem/LinuxHArdwareService/VPD Enabled & VPD Interval**).

9 If VPD is enabled, change the VPD interval to a different value. This action will force Service Agent to collect VPD again.

10 It may take about 5 or 10 minutes to collect VPD from all the partitions.

11 Do steps 5 and 6 to send VPD to IBM.

Web-based VPD entry

Follow these steps to collect the VPD from the server.

From the HMC follow this path:

1 Select **Server and Partition_Server Management**.

2 Select the managed system in the right side of the HMC interface.

3 Right-click on the managed system.

4 Select **Capacity Upgrade on Demand**.

5 Select **Processor**.

6 Select **Processor Capacity Settings**.

The following dialog box displays the current processor capacity settings.

7 Click the **Display Order Information** button:

- System type (four ASCII characters)
- System serial number (eight ASCII characters: pp-sssss).
- CUoD capacity card CCIN (four ASCII characters).
- CUoD capacity card serial number (10 ASCII characters: pp-sssssss).
- CUoD capacity card unique ID (16 ASCII characters).
- CUoD resource identifier (four ASCII characters).
- Activated CUoD function: CUoD increments in use, resources currently paid for (four ASCII characters).
- CUoD sequence number (four hexadecimal characters).
- CUoD activation code entry check (1 byte hex check sum, two ASCII characters—based on items: 1, 2, 3, 4, 5, 6, 7, and 8).

You will have to submit this information at http://www-1.ibm.com/servers/eserver/pseries/cuod/vpd_form.html.

Retrieval of the CUoD activation code

The CUoD activation code specific to your system can be retrieved from the Web, normally within one business day of a

confirmed order to IBM, along with the submission of the VPD, or you can wait to receive it by mail.

To retrieve the CUoD activation code on the Web, follow these steps:

1 Go to the Web site http://www.ibm.com/servers/eserver/pseries/cuod/index.html.

2 Follow the path to the *How to Order Standby Processors and activation codes*.

3 Select the link to the *View activation codes by Machine Serial Number*.

   Or go directly to http://www-912.ibm.com/pod/pod.

4 On the Web page, enter the machine type and serial number of your server.

5 Print and/or record the POD activation code displayed on the Web page.

Activation of standby processors by CUoD activation code entry

Use the system HMC to enter the activation code by performing the following steps:

1 In the *Contents area*, select the managed system.

   Select **Capacity Upgrade on Demand**.

2 Select Processor.

3 Select **Processor Activate**.

   The processor activation code dialog box will appear.

4 Enter the activation code and click OK.

   If you entered a valid activation code, the *CUoD Activation Success* window opens.

5 Click **OK**.

Following successful processor activation, you can assign the activated processors to a partition and start using the new processor capacity.

If you are using dynamic logical partitioning (DLPAR), you need not reboot the system to use the processors. If you are not using dynamic reconfiguration, you must reboot the managed system before the newly activated processors can be used.

CAPACITY UPGRADE ON DEMAND PROCESS HINTS AND TIPS

Dynamic processor sparing

In environments with CUoD, dynamic processor sparing allows inactive processors to act as ‘dynamic spares’. An inactive processor is transparently activated if a failing processor reaches a predetermined error threshold, thus helping to maintain performance and improve system availability. Starting with AIX 5L V5.2, this capability is offered on pSeries servers with CUoD to help minimize the impact to server performance caused by a failed processor. Dynamic processor sparing will happen automatically when using DLPAR and the failing processor is detected prior to failure. If not detected prior to failure or not using DLPAR, a reboot of the system will activate an alternate processor from the inactive spares. In this fashion, the user can re-establish the required performance levels without waiting for parts to arrive on-site.

Dynamic processor sparing does not require the purchase of an activation code; it requires only that the system have inactive CUoD processors available.

Prohibition of unauthorized movement of processor features

Any unauthorized (non-IBM upgrades) movement of CUoD processor hardware will disable the capability to use standby processors.

Any physical removal of processor hardware may make the

server itself unusable (all start-up and standby processors should be available in the server to IPL).

Software licence fees

Activating a processor does not change the software tier for the pSeries 690 or the pSeries 670.

Software providers that base their software licensing fees on the number of processors on a server should be contacted prior to performing the CUoD features activation. The customer should complete all financial arrangements and receive updated licences that support the total number of activated processors.

When a POD Activation Feature is ordered, additional charges for IBM software licensing are initiated for software products currently installed on the server that are licensed by processor.

CUoD error messages

The system firmware will detect some error conditions concerning CUoD. The HMC will indicate several errors (as listed below) to the operator in case of CUoD failure situations.

- 1 – processor capacity activation failed.
- 3 – processor capacity activation code is not valid.
- 5 – processor capacity activation code is not valid, next failed attempt will require a re-boot.
- 7 – processor capacity activation code is not valid; a re-boot is required to make another attempt.
- 9 – activation code entered incorrectly, key check error. This is most likely a keying error and the user would be instructed to please re-enter the activation code.
- 10 – CUoD capacity card detected from another system or has corrupted data. Contact service provider to replace the card.

- 12 – request not allowed: CUoD capacity card is failing or has corrupted data. Contact your service provider to replace the card.
- 13 – request not allowed: more processors were selected than are available for immediate activation.

Enhancement to the lsvpd command

The **lsvpd** command lists the VPD data of the server. This command has been modified in AIX 5L Version 5.2 to obtain the processor and memory CUoD capacity card information system parameter from the firmware. The **lsvpd** command appends the system-wide keyword string, which is N5 for the processor and N6 for the memory CUoD capacity card information, and displays it along with the other VPD data that is being currently displayed. There is no error checking on the format or contents of the card's VPD data.

REFERENCES

1 *IBM eserver pSeries 670 and 690 Planning Guide for Capacity Upgrade on Demand*, White Paper. IBM Corporation.

2 *AIX 5L Differences Guide Version 5.2* Edition, SG24-5765-02, IBM Corporation.

3 *IBM eserver pSeries 670 and pSeries 690 System Handbook*, SG24-7040-01, IBM Corporation.

Alex Polyak
System Engineer
APS (Israel)

Saving error messages to a file

In order to collect and save error messages to a file so that you

can examine them later, use the following procedure. First of all create the file */tmp/<filename>*. Then edit, using vi or whatever, the file */etc/syslog.conf*. Find the debug line and make it active. Change */dev/console* to */tmp/<filename>*.

Enter:

```
ps -ef | grep syslogd
```

which displays the PID for syslogd. Then kill the current process for syslogd by:

```
kill -9 <pid> ()
```

Lastly, restart the syslog daemon by:

```
/etc/syslogd ()
```

Christopher Chapman
AIX Specialist (USA)

Using CVS for version control: an introduction

WHAT IS CVS?

CVS is the Concurrent Versioning System. It allows centralized management of many versions of files – both text and binary in nature. It is not so much for back-up (although this is a possible use) as for maintaining integrity of files when many developers work on them almost simultaneously.

The canonical users are developers (for storing their source-code), and system-administrators (for storing system configuration files such as Web server and DNS configs). It is particularly useful where one has to make a change on one server, test it, commit the change to a central system, and propagate that change to other servers.

ADVANTAGES AND DISADVANTAGES

While there are several alternatives, CVS has the longest history

and greatest popularity of any version-control system, particularly amongst the Free Software/Open Source movement.

Its storage of binary files is inefficient, and its detection thereof is not totally foolproof.

Renaming/moving of files and directories is not a native operation.

The command line syntax can be a little hard to understand, at least initially. There are, however, several graphical applications operating as front-ends to CVS.

CVS can be configured to operate both across a locally accessible filesystem or over the network using either its own 'pserver' protocol or a command-execution command such as **rsh** or **ssh**.

SETTING UP AND USING CVS

CVS requires a subdirectory to be set aside for use as a repository. For a single-user system with just a few files whose versions need tracking, *~/cvs/* is a perfectly adequate system. For a vast multi-user system like AIX with many developers hacking at source-code, *~cvs* could be set to */home/cvs/* and mounted on a separate partition.

Either way, the first thing that must be done is initialize the repository. For example:

```
zsh, hay  8:55PM tim/ % cvs -d /home/tim/cvs init
zsh, hay  8:55PM tim/ % find cvs
cvs
cvs/CVSROOT
cvs/CVSROOT/Emptydir
cvs/CVSROOT/loginfo
cvs/CVSROOT/loginfo,v
[...snip...]
cvs/CVSROOT/rcsinfo
cvs/CVSROOT/modules
cvs/CVSROOT/modules,v
cvs/CVSROOT/config
cvs/CVSROOT/config,v
cvs/CVSROOT/history
```

This shows the initialization of */home/tim/cvs* as a new directory

for the cvs repository, and a selection of the files it creates for its own administrative purposes.

Here is a good place at which to introduce Rule 0: you do not edit *,v files. In fact, you do not edit anything within this repository, because you can, and therefore should, use the **cvs** command itself for the purpose. All files are edited in a working-directory, checked out from the repository.

Adding a module

Adding a module should be done by checking-out a copy of the CVSROOT directory above, thus:

```
zsh, hay  8:55PM tim/ % mkdir temp
zsh, hay  9:Ø6PM tim/ % cd temp
zsh, hay  9:Ø6PM temp/ % cvs -d ~/cvs co CVSROOT
cvs checkout: Updating CVSROOT
U CVSROOT/checkoutlist
U CVSROOT/commitinfo
U CVSROOT/config
U CVSROOT/cvswrappers
U CVSROOT/editinfo
U CVSROOT/loginfo
U CVSROOT/modules
U CVSROOT/notify
U CVSROOT/rcsinfo
U CVSROOT/taginfo
U CVSROOT/verifymsg
```

Then the module name and a subdirectory in which to store it are appended to the CVSROOT/modules file, and the file committed back into the repository with an appropriate comment:

```
zsh, hay  9:Ø6PM temp/ % mkdir ~/cvs/mylocalconfigs
zsh, hay  9:Ø7PM temp/ % echo mylocalconfigs mylocalconfigs >> CVSROOT/
modules
zsh, hay  9:Ø8PM temp/ % cd CVSROOT
zsh, hay  9:Ø8PM CVSROOT/ % cvs commit -m "Added module
'mylocalconfigs'" modules
```

For checking in modules:

```
/home/tim/cvs/CVSROOT/modules,v  <--  modules
new revision: 1.2; previous revision: 1.1
done
```

```
cvs commit: Rebuilding administrative file database
zsh, hay  9:Ø8PM CVSROOT/ %
```

At this point, the directory designated as a CVS repository has been initialized, and space for storing a new module assigned.

The temporary *CVSROOT/* checkout directory may safely be removed at this point.

EXAMPLES OF USAGE

This section introduces the CVS command line for various simple and commonplace activities.

Overview

The general form of the command is:

```
cvs [options] subcommand [options] [files]
```

Checkout

The first task, having created a new module for storing our files in CVS, is to check out a working directory from that module:

```
zsh, hay  9:23PM tim/ % mkdir src
zsh, hay  9:23PM tim/ % cd src
zsh, hay  9:23PM src/ % cvs -d ~/cvs/ co mylocalconfigs
cvs checkout: Updating mylocalconfigs

zsh, hay  9:23PM src/ % cd mylocalconfigs
zsh, hay  9:23PM mylocalconfigs/ % ls
CVS/
zsh, hay  9:23PM mylocalconfigs/
```

There are currently no files in this module, but the working directory has the CVS/ subdirectory showing it to be under CVS's control. Two of these administrative files are of interest:

```
zsh, hay  9:23PM mylocalconfigs/ % cat CVS/Root
/home/tim/cvs/
zsh, hay  9:25PM mylocalconfigs/ % cat CVS/Repository
mylocalconfigs
```

These can be used later to indicate what module and CVSROOT

are in effect when exploring a random working directory.

To explain the CVS command above, **-d** is the path to the CVSROOT in question; in this case, it is obviously on a local filesystem, but alternative syntaxes can be employed:

```
:pserver:cvs@oss.sgi.com:/cvs
```

denotes use of a publically accessible server, using CVS's native pserver protocol for access.

```
tim@myserver.org.uk:/home/tim/cvs/local
```

denotes use of whatever the CVS_RSH environment variable is set to (rsh or ssh for preference) to access a directory on myserver as a CVSROOT.

co is an abbreviation for the **checkout** command.

Adding and committing files to a module

Adding files to CVS is accomplished with the **add** subcommand, thus:

```
zsh, hay 10:19PM mylocalconfigs/ % cp /etc/hosts .
zsh, hay 10:19PM mylocalconfigs/ % cvs add hosts
cvs add: scheduling file `hosts' for addition
cvs add: use 'cvs commit' to add this file permanently
zsh, hay 10:19PM mylocalconfigs/ % cvs commit -m "Added hosts file"
cvs commit: Examining .
RCS file: /home/tim/cvs/mylocalconfigs/hosts,v
done
Checking in hosts;
/home/tim/cvs/mylocalconfigs/hosts,v  <--  hosts
initial revision: 1.1
done
zsh, hay 10:19PM mylocalconfigs/ %
```

This also introduces the important **commit** subcommand, which puts your local changes into the upstream repository, and its **-m** sub-option, which supplies a comment for the change on the command line – without this, $EDITOR would be spawned for you to enter the comment.

Now is a convenient point at which to introduce Rule 1: enforced by CVS, you must run **cvs update** before committing your

changes. This ensures that the integrity of the repository is maintained, that changes which you've made are not conflicting with anyone else's. In the event of a conflict, the offending files will be flagged with error messages and lines beginning with 'C', and the areas of conflict within them flagged by plaintext arrow lines for human resolution.

Opinions and practices vary when submitting diffs or committing changes to CVS; some people like each file to be committed with a separate comment, others prefer to commit and comment atomic actions, such as 'added support for foo-feature to all files', as the case may be.

Adding files in bulk: import

While it is quite possible to add all files and subdirectories by hand with the **cvs add** subcommand, it obviously does not scale well beyond a handful of files or directories.

The solution is to use the cvs **import** command, thus:

```
zsh, hay 11:27PM Pictures/ % ls
CVS/      ChangeLog car_pics/ hosts
zsh, hay 11:27PM Pictures/ % cvs -d ~/cvs/ import car_pics car_pics
"CarImport"
zsh, hay 11:28PM Pictures/ % ls ~/cvs
CVSROOT/  car_pics/ mylocalconfigs/
```

This shows the subdirectory *car_pics* being imported as a new module into *~/cvs/*.

Obtaining status information

Taking this example a little further, suppose you share system administration of this server with a few other people, also allowed and encouraged to use CVS. After a period of time, you need to check that all files in the configs directory have been kept up-to-date. The **status** subcommand can help here:

```
zsh, hay 10:44PM mylocalconfigs/ % cvs status
cvs status: Examining .
===================================================================
File: hosts   Status: Up-to-date
```

```
   Working revision:   1.1  Mon Jan 13 22:19:26 2003
   Repository revision: 1.1  /home/tim/cvs/mylocalconfigs/hosts,v
   Sticky Tag:     (none)
   Sticky Date:        (none)
   Sticky Options: (none)

zsh, hay 10:44PM mylocalconfigs/ %
```

The status of ‘Up-to-date’ means that repository and local checked-out working copies are in-sync. If you modify the file, it becomes ‘locally modified’, thus:

```
zsh, hay 10:45PM mylocalconfigs/ % cvs status
cvs status: Examining .
===================================================================
File: hosts   Status: Locally Modified

   Working revision:   1.1  Mon Jan 13 22:45:42 2003
   Repository revision: 1.1  /home/tim/cvs/mylocalconfigs/hosts,v
```

and if you run a **cvs diff**, it will show you what’s been changed:

```
zsh, hay 10:45PM mylocalconfigs/ % cvs diff
cvs diff: Diffing .
Index: hosts
===================================================================
RCS file: /home/tim/cvs/mylocalconfigs/hosts,v
retrieving revision 1.1
diff -r1.1 hosts
4,5d3
< # 10.0.0.7       mydomain.com host.mydomain.com
<
zsh, hay 10:46PM mylocalconfigs/ %
```

This change can then be committed back up to the repository, in turn:

```
zsh, hay 10:48PM mylocalconfigs/ % cvs commit -m "Removed a line"
cvs commit: Examining .
Checking in hosts;
/home/tim/cvs/mylocalconfigs/hosts,v  <--  hosts
new revision: 1.2; previous revision: 1.1
done
```

It might be advisable to pay attention to the changes in revision number at this point; we shall return to it later.

Suppose, later, someone else has amended the file. In this case, the status subcommand would show ‘needs patch’, and you would use **cvs update** to bring changes from the repository to the working directory.

In this way, for example, you could store all your Web server config files in a central repository, and make the live directories’ checkouts of the appropriate modules, such that changes can be made and tested on one server, then only correct functional changes committed to CVS for posterity and application to the other servers.

CVS command global options

There are a few CVS options that should be noted. These come immediately after the **cvs** command itself, before the subcommand is given.

-q means an operation is quiet. This is particularly useful when updating files in a large repository; it will default to listing every subdirectory as it updates them in turn. With **-q** in force, it will list only files where it has something to say.

-n means ‘dummy run’; it will produce all the status and output for the command, but not actually perform it.

While it is possible to use **cvs status** to examine everything in a directory, it can be more expedient to combine the above two options as a quick succinct status check on the current working directory, at one line per file rather than a whole paragraph:

```
zsh, hay 10:57PM scripts/ % cvs -nq update |& tee ../tim-cvs.txt
? mkdep
? split-include
? lxdialog/lxdialog
P tkcond.c
M tkparse.c
```

This shows that a patch needs to be applied to tkcond.c (it is out of date compared with the repository) and that tkparse.c has local modifications pending (it is more up-to-date than the repository).

-zN (for a small value of N such as 3 to 5) means to compress the link using zlib compression, like **gzip**, to the repository server. Note that this adds a significant load on the server so small values are essential.

These options and more can be placed in *~/.cvsrc* so that they apply as defaults to all future commands, unless other behaviour is requested:

```
#`cvs' denotes the cvs command itself, not a subcommand
cvs -z3
#this makes diff take the `-u' option
diff -u
#this makes update create directories and purge files by default
update -dP
```

Removing files

Removing files from CVS's control is a little tricky for reasons that will become apparent later. However, the general syntax is:

```
zsh, hay 11:Ø2PM mylocalconfigs/ % cvs rm hosts
cvs remove: file `hosts' still in working directory
cvs remove: 1 file exists; remove it first
zsh, hay 11:Ø3PM mylocalconfigs/ % rm hosts
zsh, hay 11:Ø3PM mylocalconfigs/ % cvs rm hosts
cvs remove: scheduling `hosts' for removal
cvs remove: use 'cvs commit' to remove this file permanently
zsh, hay 11:Ø3PM mylocalconfigs/ % cvs commit -m "Removed hosts file "
hosts
Removing hosts;
/home/tim/cvs/mylocalconfigs/hosts,v  <--  hosts
new revision: delete; previous revision: 1.2
done
```

If you use the **-Rf** options to the subcommand **rm**, then the **cvs rm** will affect a local **rm** (unlink) from the directory as well, meaning you either have to remember the filename(s) – hard, if you specified a blob for the filelist! – or commit everything in the current directory for it to take effect.

There is no specific syntax to cater for renaming a file or a directory. You must copy the file/directory to its new name, remove it from its old and add the new, with a comment like 'renamed directory' in order to preserve any history about it.

Rolling back unwanted commits

Reverting to previous versions is a very tricky subject; there are at least three possible meanings pivoting around the revision number of a file at any particular instant. Suppose you wish to revoke the (rather trivial) deletion of a line in the host's file, earlier: the options are either to undo the deletion of the line in CVS while preserving the current revision number; to roll the repository back to a state in which 1.2 had not yet happened, therefore this file was at revision 1.1; or to pull revision 1.1 out of CVS and re-add it as revision 1.3.

The latter is by far the easiest option because the other two require advanced hacking on the repository itself, which is bad practice, while this one preserves the complete honest history of everything that happened (good or bad!).

To achieve this reversion, we invoke a tag on the file, thus:

```
zsh, hay 11:18PM mylocalconfigs/ % cvs update -r 1.1 hosts
U hosts
zsh, hay 11:18PM mylocalconfigs/ % cvs status
cvs status: Examining .
===================================================================
File: hosts   Status: Up-to-date

Working revision:  1.1  Mon Jan 13 23:18:05 2003
Repository revision:   1.1  /home/tim/cvs/mylocalconfigs/hosts,v
Sticky Tag:        1.1
Sticky Date:       (none)
Sticky Options:    (none)
```

At this point, the file is back at revision 1.1, and we can take a copy for posterity, update back to the latest of everything, replace the latest with the copy of the old, and re-commit it:

```
zsh, hay 11:18PM mylocalconfigs/ % cp hosts hosts.old
zsh, hay 11:20PM mylocalconfigs/ % cvs update -dPA hosts
U hosts
zsh, hay 11:20PM mylocalconfigs/ % mv hosts.old hosts
zsh, hay 11:20PM mylocalconfigs/ % cvs commit -m "Replaced with old
version"
cvs commit: Examining .
Checking in hosts;
/home/tim/cvs/mylocalconfigs/hosts,v  <--  hosts
new revision: 1.3; previous revision: 1.2
done
```

Some explanation is required – this is the topic of the next section. At this stage, the reader is encouraged to take a practical exercise, create a repository of their own, and commit any configuration files pertinent to their environment into it, if they have not done so already.

BRANCHING AND TAGGING: DEVIATIONS FROM THE HEAD

The CVS has facilities for maintaining separate branches of a file. For example, the author uses CVS for maintaining a *~/etc/* directory with various configuration files (emacs, Xemacs, zsh, ssh, procmailrc, etc) containing several years' worth of edits. However, the procmailrc used at home is not the same as the one used at work.

Similarly, in software development, you normally wish to freeze certain versions of the source files to say 'this is Release 1.0', or whatever. And, subsequently, development continues apace for the next release (whenever that may be) but certain bug-fixes from the further developments should trickle down into Release 1.0 (new features, by definition, should not appear as updates in previous releases).

The system CVS uses for this is called branching. The author designates one branch for each different checkout of the *~/etc/* directory – one for work, a couple for home, another for an external server. At any time, there is one central branch called the head, which is normally used for development work with results being merged into other branches.

It is typical for projects controlled in this way to follow a similar procedure: development occurs at all times freely in the head, with features coming and going as required. Then, when things are generally stable and a feature-freeze is enforced, a branch for the forthcoming release is created. Once this happens, the developers submit patches for bug-fixes (never for new features) to be committed onto that branch (by running **cvs commit** in a working directory checked-out onto that branch). The separation between development introducing new features and the back-

porting of bug-fixes and security problems to older releases is widely accepted as a good thing.

A branch can be created by using the **cvs tag -b BRANCHNAME** command. Notably, branch-names are normally in uppercase, with underscore, dash, and numbers allowed in addition to alphabetic characters.

A set of files or complete working directory may be moved onto a particular branch, or checked out afresh onto a specific branch, with either:

```
cvs update -dP -r BRANCH-NAME
```

or:

```
cvs co -r BRANCH-NAME
```

as required. From this point on, such files will show a ‘sticky tag’ in their status information:

```
zsh, hay 11:27PM etc/ % cvs status procmailrc
===================================================================
File: procmailrc        Status: Up-to-date

Working revision:    2.88.4.23       Sun Jan 12 19:27:25 2003
Repository revision: 2.88.4.23       /home/tim/cvs/configs/procmailrc,v
Sticky Tag:          server (branch: 2.88.4)
Sticky Date:         (none)
Sticky Options:      (none)
```

There is still the head, on which development occurs; performing an update with the **-dPA** option will revert a tagged working directory or file-list to the head. Once a file (or working directory) is ‘on’ a specific branch, further changes are applied in the context of that branch alone.

This is what makes file deletion tricky: a file can exist happily in the head, but not in a specific checkout of a branch; the choice of branch determines whether or not you see the file at all. For example, on workstations, the author needs a *~/etc/powershellrc* file; on my external server, it is irrelevant. Conversely, on the external server, there are several simple game configuration files that are surplus to requirements at work. A few branches in one

central CVS repository on the external server suffices to store these different views on the state of a home-directory.

You can display the differences between any two versions by time, by revision number, or by branch. Thus, to see the first few words that have changed in the database used by an anti-spam filtering system since the start of October, one could type:

```
zsh, hay 11:37PM etc/ % cvs diff -D "2002-10-01" ifile_datahead -20
Index: ifile_data
===================================================================
RCS file: /home/tim/cvs/local/configs/ifile_data,v
retrieving revision 2.22
retrieving revision 2.57
diff -u -r2.22 -r2.57
--- ifile_data  29 Sep 2002 23:00:05 -0000      2.22
+++ ifile_data  3 Nov 2002 00:00:04 -0000       2.57
@@ -1,5408 +1,8598 @@
good spam
-455017 651695
-1917 1474
-suitable 2 0:1
-gogo 11 1:24
-cdt 2 0:3
```

(It helps that this file is copied and committed to CVS as an automated nightly **cron** job.)

Obtaining historical data

As one might hope, it is possible to obtain both a log and a history of what events have occurred in a given repository, thus:

```
zsh, hay 11:33PM mylocalconfigs/ % cvs log
RCS file: /home/tim/cvs/mylocalconfigs/hosts,v
Working file: hosts
head: 1.6
branch:
locks: strict
access list:
symbolic names:
keyword substitution: kv
total revisions: 6;    selected revisions: 6
description:
----------------------------
revision 1.6
date: 2003/01/13 23:20:38;  author: tim;  state: Exp;  lines: +2 -0
```

```
Replaced with old version
----------------------------
revision 1.5
date: 2003/01/13 23:17:28;  author: tim;  state: Exp;  lines: +0 -2
Removed a line
[...snip...]
revision 1.1
date: 2003/01/13 22:19:42;  author: tim;  state: Exp;
Added hosts file
=============================================================================
```

This shows a blow-by-blow account of everything that has happened to all the files under the current directory.

The output from the **cvs history** command is geared towards generating reports of various kinds of activity:

```
zsh, hay 11:38PM mylocalconfigs/ % cvs history -c -a -D "last week"
M 2003-01-13 21:08 +0000 tim 1.2 modules CVSROOT        == ~/temp/
CVSROOT
A 2003-01-13 22:19 +0000 tim 1.1 hosts   mylocalconfigs == ~/src/
mylocalconfigs
M 2003-01-13 22:48 +0000 tim 1.2 hosts   mylocalconfigs == ~/src/
mylocalconfigs
R 2003-01-13 23:03 +0000 tim 1.3 hosts   mylocalconfigs == ~/src/
mylocalconfigs
A 2003-01-13 23:17 +0000 tim 1.4 hosts   mylocalconfigs == ~/src/
mylocalconfigs
M 2003-01-13 23:17 +0000 tim 1.5 hosts   mylocalconfigs == ~/src/
mylocalconfigs
M 2003-01-13 23:20 +0000 tim 1.6 hosts   mylocalconfigs == ~/src/
mylocalconfigs
```

ChangeLog

A useful file that should accompany all CVS working-directories, generated from the **cvs history** reports, is a ChangeLog. There are utilities (mostly Perl scripts) to format the output properly, and it should be noted that the command **C-x v a** (**vc-update-change-log**) from within emacs in CVS mode will update or generate a complete ChangeLog for you.

As a typical example, the module used in the above examples has the following ChangeLog:

```
2003-01-13  tim haynes
```

```
    * hosts: Replaced with old version

    * hosts: Removed a line

       * hosts: Re-added file

       * hosts: Removed hosts file :(

       * hosts: Removed a lineq

       * hosts: Added hosts file
```

Remote Access Methods

Thus far, we have seen only local filesystem operation of CVS in action. A few words need to be said concerning the alternatives.

The pserver method requires an initial command on the client before a checkout can commence, namely **cvs login**. This site, username, and password are then stored in *~/.cvspass*. For example:

```
$ cvs -d :pserver:cvs@cvs.xemacs.org:/pack/xemacscvs login
(Logging in to cvs@sunsite.dk)
CVS password: cvs

$ cvs -z3 -d :pserver:cvs@cvs.xemacs.org:/pack/xemacscvs checkout -d
xemacs-21.5 xemacs

$ cat ~/.cvspass
/1 :pserver:cvs@cvs.xemacs.org:24Ø1/pack/xemacscvs [censored]
```

CONFIGURING AND BUILDING CVS FROM SOURCE

Installing CVS on AIX is a relatively simple process because, in its latest version (1.11.5.1), it is shipped as a regular gzipped tarball, the sources being set up by a regular autoconf system.

A typical run of the installer could look like this:

```
bash$ gzip -9cd < ~/cvs-1.11.5.tar.gz | tar xvpf -
x cvs-1.11.5
x cvs-1.11.5/contrib
x cvs-1.11.5/contrib/README, 5Ø29 bytes, 1Ø media blocks.
x cvs-1.11.5/contrib/ChangeLog, 2Ø5Ø1 bytes, 41 media blocks.
x cvs-1.11.5/contrib/Makefile.am, 3222 bytes, 7 media blocks.
```

```
x cvs-1.11.5/contrib/Makefile.in, 11728 bytes, 23 media blocks.
x cvs-1.11.5/contrib/clmerge.in, 3337 bytes, 7 media blocks.

[...snip...]
x cvs-1.11.5/vms/vms.h, 1023 bytes, 2 media blocks.
x cvs-1.11.5/vms/vmsmunch.c, 12406 bytes, 25 media blocks.
x cvs-1.11.5/vms/vmsmunch.h, 1039 bytes, 3 media blocks.
x cvs-1.11.5/vms/vmsmunch_private.h, 5415 bytes, 11 media blocks.
x cvs-1.11.5/vms/waitpid.c, 1931 bytes, 4 media blocks.

bash$ cd cvs-1.11.5/
bash$ ls
AUTHORS         Makefile.am    build.com      depcomp         os2
BUGS            Makefile.in    compile        diff            src
COPYING         NEWS           config.h.in    doc             stamp-h1.in
COPYING.LIB     PROJECTS       configure      emx             tools
[...snip...]
MINOR-BUGS      aclocal.m4     cvsnt.mak      noautomake.sh
```

Now we check the configuration script:

```
bash$ ./configure --help
Usage: configure [options] [host]
Options: [defaults in brackets after descriptions]
Configuration:
  --cache-file=FILE       cache test results in FILE
[...snip... common options]
--x-libraries=DIR       X library files are in DIR
--enable and --with options recognized:
--disable-dependency-tracking Speeds up one-time builds
--enable-dependency-tracking  Do not reject slow dependency extractors
--with-krb4=value       set default $(KRB4) from value
--with-gssapi=value     GSSAPI directory
--enable-encryption     enable encryption support
--enable-client         include code for running as a remote client
(default)
--disable-client        don't include remote client code
--enable-server         include code for running as a server (default)
--disable-server        don't include server code
```

Having seen what options are available, we run the configuration for real and provisionally compile it:

```
bash$ ./configure --prefix=/usr/local/stow/cvs --without-x   --enable-
client --enable-server --enable-encryption && nice make
creating cache ./config.cache
checking for a BSD compatible install... /usr/local/bin/ginstall -c
checking whether build environment is sane... yes
checking for gawk... no
```

```
checking for mawk... no
[...snip...]
creating tools/Makefile
creating vms/Makefile
creating windows-NT/Makefile
creating windows-NT/SCC/Makefile
creating zlib/Makefile
creating config.h
creating src/options.h
make  all-recursive
Making all in lib
[...snip...]
Target "all" is up to date.
Target "all-am" is up to date.
```

Finally, we become root and install it into */usr/local/stow/cvs/*:

```
bash$ su
Password:
bash# make install
        set fnord  -n; amf=$2;  dot_seen=no;  target=`echo install-
recursive
 | sed s/-recursive//`; list='lib zlib diff src man doc contrib tools
 windows-NT os2 emx vms'; for subdir in $list; do echo "Making $target
in
 $subdir"; if test "$subdir" = "."; then dot_seen=yes;
 local_target="$target-am"; else local_target="$target"; fi; (cd
 $subdir && make $local_target) || case "$amf" in *=*) exit 1;; *k*)
 fail=yes;; *) exit 1;; esac; done; if test "$dot_seen" = "no"; then
 make "$target-am" || exit 1; fi; test -z "$fail"
Target "install" is up to date.
bash# exit
```

And behold, CVS is ready to roll.

A word on security

There are a few considerations when it comes to securing a CVS installation.

Firstly, there are permissions on local filesystems to worry about. It is advisable to use one user group for people who are permitted to commit to the repository; this can, of course, be refined to a per-module basis. The permissions should be set such that that group has write access. Also, the administrator should own the CVSROOT/ directory to avoid anyone else adding unwanted modules, etc.

Secondly, there are networking concerns. If the contents of a module are sensitive, ssh-based access should be mandated. This is still similar to the local file access system above from the point of authentication onwards.

Thirdly, providing a public pserver server is a large responsibility. Any network traffic is plaintext; the password is trivially mangled to deter only the least interested sniffers. It should be installed with as little running as root as possible, with no valid CVS username being also valid for any other access system such as shell.

Fourthly, there is the potential to configure commands to be run on certain cvs events – particularly useful is that commits can be e-mailed to a list of administrators. Beware the user that these things run as! Never let a script run with variable user-input data as root.

Lastly, a small thing to worry about: CVS can be configured without the **-enable-rootcommit** option; this will disallow root from committing files. However, it is trivial *in extremis* to bypass this. If a commit attempt fails with this error, run with your LOGNAME environment variable set to your real userid instead:

```
zsh/scr, hay 12:33AM apache/ # cvs commit
cvs [commit aborted]: 'root' is not allowed to commit files
zsh: exit 1     cvs commit
zsh/scr, hay 12:34AM apache/ # LOGNAME=tim cvs commit
...
```

FURTHER READING

The best document for learning about CVS online is the Cederqvist, to be found at http://www.cvshome.org/.

A useful collection of CVS error messages is to be found at http://www-es.fernuni-hagen.de/cgi-bin/info2html?(cvs)Error%20messages.

Tim Haynes
Open Source and Free Software Consultant (UK)

Modelling the performance of DB2 UDB-enabled applications running on AIX

BACKGROUND

In today's corporate environment, legacy applications are on the move. Monolithic IT processes and attitudes are on the wane, and flexibility, agility, and speed are in the ascendancy.

What does that mean? Big iron systems are out, and modular systems are in. Business applications are being designed with the end user, not the systems analyst, in mind. A new wave a application developers understand the intricacies of Unix and Windows workstations and Java Class libraries, but have little knowledge of, or desire to understand, legacy technologies and programming languages.

As this trend continues, IT professionals are going to have to adapt, or find alternative ways of making a living. So assuming that we really want to keep our jobs (and some semblance of job satisfaction), and justify our salaries (somehow our partners seem to like the lifestyle to which they have become accustomed), we have to grow our knowledge base. New platforms and technologies mean learning new skills.

The experience of the past decade has made companies cautious with regard to application migration. Tales, both rumoured and factual, of platform migration projects that have gone horribly wrong are myriad. No CIO or head of IS wants to be the next victim of a 'make it or break it' career decision going the wrong way.

What do legacy architects have that cannot be duplicated by three years of a good college computer science programme? Experience! How can we best utilize the experience we have gained through years of systems and applications programming? By recognizing that new technologies are still influenced by the

performance factors we have learned to respect while working with legacy systems.

CONCEPT

It's simple to say, 'move this application off the mainframe and implement it on a Unix system'. From a business perspective, it might even make sense. Companies have spent years acquiring client/server hardware and software. It can be debated whether these systems are truly less expensive, when considering the total cost of ownership, but the fact is that the new generation of programmers and systems administrators arriving in the workforce are more familiar with client/server systems than with older legacy hardware and the applications that they support. If you look through the graduating classes of the college and university programs in your local area, you will probably find more Java and Web than COBOL and CICS programmers. So, whether we like it or not, client/server systems are here to stay and applications will continue to move to distributed platforms as the top corporate decision makers try to consume all the available resources they have already acquired.

When legacy applications migrate from their 'mainframe' roots out into the new enterprise server environment, more than just a set of application program code must move with it. Consider the following example.

If we were to concentrate only on moving the front end or the application program, we would have to ignore a great deal of the workload associated with the business functionality served by that application.

When a legacy application and its front end move onto new platforms, local instances of some supporting subsystems must be installed as well. However, the chances are that some links will remain with the legacy data sources as well.

Dynamically linking the data existing in the legacy environment with the distributed environment warrants a discussion all on its

own. In this article, we will concentrate on the issues that arise once the application and the supporting subsystems are installed in the distributed environment. Typically, we can assume that if an existing legacy system copes with the volume of transactions necessary to support the business, just moving the calls for data retrieval to another platform will not significantly affect the ability of those legacy systems to continue to act as a data server.

A project of this complexity is not going to be undertaken blindly. Plans will be made to test the feasibility of migrating the chosen application to a particular platform. Usually, this means moving the application to a test bed environment and then performing a series of functional and performance tests to ensure that the final migration will be successful.

PROJECT SPECIFICATIONS

In the business case this article is based on, the legacy application being migrated was designed to support a community of 53,000 users. The estimated application activity was as follows:

- The number of users active on the system during an average day would be between 42,000 and 43,000.
- The number of users active at peak load times would be about 45,000.
- The number of concurrent active users would be about 6,700.
- The production database would consist of about 1.275TB of client data.

Each user would be working on multiple activities, each of which consisted of approximately 0.5MB of client data residing in the active database.

Activities would take two paths, one requiring up to 15 days to resolve and one requiring up to five days to resolve. Ninety per cent of the average user workload is made up of the activities that

take five days to resolve.

The average lifespan of a client task would be around 91 days before ageing would take the data out of the client/server repository and migrate that data back to the legacy servers or discard it as being irrelevant.

The full production environment would consist of an AIX server farm with four gateway servers, two application servers, and two database servers. The database servers were configured with four DASD adapters each attached to an IBM SSA DASD farm, running RAID 10 protocol in order to provide redundancy and mirroring of the data.

Service Level Agreements were specified for the migrated application environment as requirements for successful completion. The Agreement stated that:

1 Average transaction response time will be 3 seconds.

2 95% of transactions must respond within 4 seconds.

3 99% of transactions must respond within 5 seconds.

TEST ENVIRONMENT

A pilot project was started in order to perform a functional test of the migrated application and provide a baseline performance benchmark. Based on the results of the benchmark, decisions would be made as to whether the new implementation could comply with the service levels outlined above.

The evaluation environment was implemented on an RS6000 7026B80 with two processors running AIX 5L V5.1.0 with 4GB of memory. DB2 UDB Enterprise Edition V7.2 was selected as the database engine, with 2 x 36.4GB local hard drives and 14 x 18GB external IBM SSA disk drives.

The test environment was designed to accommodate up to 4% of the production database, with process stubs in place to simulate the responses from the legacy environment calls the application workload would generate. The testing architects

decided that, with a sample of production data, simulated user workloads could be run against that data at increasing arrival rates. Once a baseline measurement was in place the expected performance could be extrapolated against the business requirements.

CONSIDER THE PERFORMANCE FUNDAMENTALS

In order for the capacity planning process to work, the data that is being fed into the process has to be valid. As with any statistical process, if you put garbage in, you will get 'interesting' results coming out.

In order for the baseline measurements to be accurate, the test environment must provide an accurate reflection of the production environment. In production, a team of DBAs and DB2 systems programmers make sure that DB2 UDB has the resources that it requires to support the user community and their application workload. However, in the test environment, DB2 was to be installed and run 'out of the box' without any customization. If this was the case, then the performance metrics would be skewed to begin with, and would not allow realistic extrapolation of larger workloads.

If DB2 were being installed on OS/390, care would be taken to ensure that adequate resources existed at least in the following areas:

- DASD
- Memory
- CPU
- Network.

Regardless of platform, relational database systems are remarkably consistent in their resource requirements. So, not surprisingly, these are some of the same issues that should be examined in a UDB installation on AIX. We can even go so far as to create some basic formulas for certain resource categories

that will allow the calculation of the size of the instance that can be supported, to ensure that benchmark measurements will produce results that can then be scaled.

DASD

A great deal of time and effort is spent determining the size of the raw data that a given DB2 application requires. However, the raw data is only the basic measurement of DASD required to support application processing. Data space is required to support the other structures required by DB2 processing such as indices and DB2 work space. Experience has shown that the amount of DASD required to back the raw data is dependent on the type of processing the system is designed to support:

- OLTP (OnLine Transaction Processing) applications typically require a DASD pool three times the size of the raw data.
- DSSs (Decision Support Systems) typically require a DASD pool four times the size of the raw data.
- Data Warehouse applications typically require a DASD pool five times the size of the raw data.

The application outlined above was OLTP by design, so the supporting environment needed to ensure that three times the amount of DASD was available to support the test database. With the specification that 4% of the production system data was to be extracted as the test data, we can calculate:

1.275 TB of user data * .04 = 51 GB of raw data.

Therefore, a simple check of the test environment DASD space shows:

14 * 18GB disk drives in the IBM SSA = 252 GB of DASD

So, the testing environment could actually support a slightly larger sample database if DASD space was the only criterion measured. However, accessing DASD is not just a function of platter size. The following key issues will also affect DB2 UDB database I/O processing:

- What type of DASD server is attached?
- How many paths are available to get data to and from the DASD farm?
- Where are the DB2 objects located in the SSA loop?
- How much cache is available for I/O mitigation purposes?
- What RAID technology level is being implemented?

There are other issues that may affect performance, but let's examine these core elements for their effect.

What type of DASD server is attached?

In the case outlined above, IBM SSA DASD has been installed on both the production and the test environments. In some cases, where client server machines are installed along with mainframe servers, IBM ESS DASD may also be an option.

Obviously, there are a number of other DASD manufacturers apart from IBM, and each one will have its own particular list of positive and negative features. From this point on, we will be considering the performance implications of the IBM SSA DASD in the test environment.

How many adapters are available?

Each path from an adapter to the string of DASD devices in a SSA server environment is a loop. Data can be both read and written at the same time, with the available bandwidth for either reading or writing being 50% of the total bandwidth. Because of this limitation, the location of the UDB logs is a critical issue. Since each data manipulation transaction requires logging, the amount of data being written to the path where the logs reside is likely to be very high. Updates by application programs to their underlying data may be severely impacted without the ability to separate the DB2 logs from the DB2 data objects themselves.

Where are the database objects?

Because the IBM SSA DASD devices are connected together in

a chain, data packets from those disks that are farthest away are passed to each adjacent disk until they reach the adapter itself. This means that large data transfers from devices located farther down the loop will heavily influence the activity of the disks closest to the front of the adapter loop. Fetch-critical application tables need to be placed on the devices that are farthest away from the adapter, while heavily prefetched tables need to be adjacent to the adapter.

How much cache is available for I/O mitigation purposes?

Once you have determined that DASD Fast Write to cache is enabled for the SSA, the major benefit to application performance is going to depend on the amount of cache available for I/O mitigation. No matter which platform DB2 runs on, the fewer physical I/Os that are executed, the faster the database will respond.

What RAID technology is being implemented?

Levels are:

- RAID 0 – data striping; has no data redundancy capabilities.
- RAID 1 – data mirroring; absolute data integrity.
- RAID 2 and 3 – even data distribution; data integrity ensured by parity.
- RAID 4 – RAID 3, but block instead of bit or multi-byte data transfer.
- RAID 5 – even data distribution; parity information more effectively processed.
- RAID 10 – combined RAID 0 and RAID 1 provides both data striping and data mirroring with the performance benefit of RAID 0.

Depending on whether you need to have performance (RAID 0 or RAID 10) or absolute data integrity (RAID 1 –5), application response times will vary significantly, based on what you choose

to implement. Remember, any I/O that does not have to occur will improve application performance. Regardless of which RAID level you choose, make sure that you have the appropriate sized cache for I/O mitigation.

Memory

The memory requirements of a system supporting DB2 will generally be broken into components:

- DB2 buffer cache (bufferpools).
- Operating system memory (tool, connection, and subsystem memory).

Formulae and methods abound for calculating the amount of memory that should be allocated to DB2 bufferpools in support of application processing. Methods vary depending on both the size of the underlying databases, and the type of data access.

When allocating memory for the buffer cache, the other memory requirements for DB2 need to be considered. DB2 is an application, with multiple components, that will run in the AIX environment. The fact that DB2 accesses data and manages its buffer areas can often obscure the point that, as a program, DB2 needs application memory to run.

Over-allocating DB2 buffer caches can result in DB2 being impacted just at the time that application activity is highest. This will cause a significant slowdown because AIX has to manage its virtual memory in support of the database and its applications.

When calculating the amount of memory that DB2 'the program' needs, the number of concurrent connections from the outside to DB2 should be considered. The following numbers can be used, as a 'rule of thumb' in determining how much memory DB2 connections will require:

- 10 connections – 80MB
- 25 connections – 96MB
- 50 connections – 186MB.

When you make your calculations for a specific number of connections, take the values listed above and add the MB values based on the number of connections required. For example, for 115 users, add 50 connections twice (186MB * 2) together with 25 connections (96MB) for a total of 468MB (supporting 125 users).

If you are using DB2 Administration Tools on your DB2 instance, you need to allocate 30MB of memory for the tools and their collection buffers.

The actual memory required for DB2 and its associated address spaces can be estimated at the time of installation. As the DB2 workload grows, systems administrators need to review the memory management in the AIX environment. When swapping starts to occur on a regular basis, reconsideration of either the amount of physical memory or its allocation will be required.

CPU

With the increase in processing power of client/server hardware, CPU utilization is becoming less of an issue than it has been. Obviously, the operating system will determine the most effective level of CPU usage for a given installation. For our purposes, we will be discussing levels that are effective for AIX 5L V5.1.0.

Given that the environment is going to be used as a database server, the expected CPU consumption will depend on the type of application workload that is to be supported. The following estimates can be used as a baseline for workload type and CPU percentage busy:

- OLTP – 70–80%
- Web – 50%
- DSS – 80–90%
- Batch – 90–00%.

Let’s consider each of the workload types.

OnLine Transaction Processing (OLTP)

With a recommended starting point of 70–80% CPU busy, the OLTP workload is characterized by a large number of users generating relatively short duration transactions. Typically, these users are clustered into office or other contained environments and are limited to those who work for an enterprise. Normally, a given level of transactions will be active at all times. However, users being who and what they are, there will be surges of incoming transactions. Usually, surges can be counted on to occur just before lunch, and again just before the end of the working day. Other periods of increased activity may be present in a normal environment as well.

By reserving 20-30% of the CPU, these transitory surges can be accommodated without significant variance in the end-user response time.

Web

With a recommended starting point of 50% CPU busy, the Web workload is characterized by a potentially huge number of users generating transactions of varying types and impact to the system. Because users are able to access applications from any location, they may or may not be limited to enterprise employees. Since the user community is varied, and can be extensive, the arrival rate of transactions will be more difficult to fix to a specific level.

By leaving half of the CPU idle during ‘steady state’, arriving workloads that generate large result sets, or large bursts of incoming transactions, can be accommodated without significant reductions in application performance.

Decision Support Systems (DSS)

With a recommended starting point of 80-90% CPU busy, the DSS workload is characterized by a small number of users generating transactions with a specific execution profile. Typically, these users are clustered into one or more departments and the application they are using is one that requires more elaborate

user think time. While transactions may be resource-intensive, the fact that their arrival rate is lower and steadier means that there will be fewer anomalous interactions that can cause response time problems.

Batch

With a recommended starting point of 90-100% CPU busy, the batch workload is characterized by a static number of jobs, exercising the database server intensively. Typically, batch jobs will be planned so as not to compete for the underlying data. So, the idea is to start the workload and then finish it as quickly as possible. During the batch cycle, all available resources should be dedicated to completing the workload, so the machine can be put back into service for normal use.

Of course, the descriptions above are for a perfect world, where we can dedicate database environments to specific types of processing. With the cost of Unix servers decreasing constantly, more and more often this may be the case.

If you are mixing the workloads on a server environment, obviously you will need to take the figures for each type of application and then adjust your target CPU utilization figures to match the predominant workload either by volume of work or by importance of the business application that workload represents.

Network

No matter how well tuned your database environment is, if you can't get end user requests into the server and results back to them, your system is not performing. The network is the path that the end user traverses to access the data they need.

Make sure that you understand your network layout. Know how many devices exist between the user community, the application servers, and the database server environment. Remember, each hop in your network represents another pinch point where bottlenecks can occur. Networks support user requests, but they must also handle server responses to those requests. If you

overrun your bandwidth with returning data, your end users will be the first to suffer the impact.

If you are working within an intranet, the layout of the network can normally be determined and controlled so that there are no dynamic changes to the network path for a given group of users. Of course, redundancy should be built into an intranet's design, so that there is no single point of failure. Normally, environmental designs have two or more communication paths, and require routers to switch traffic between those paths during normal operations. If a path becomes inoperative, the router will continue sending traffic down the remaining routes until the route can be re-established.

From a testing standpoint, it is worthwhile to run at least a subset of your performance tests against the failed route scenario, to determine the actual throughput in the case of a partial network outage. Part of the service level specification should state response time requirements during component failure, whether that be network, system or database interruptions.

When you are monitoring network traffic, the amount of data being sent with each request should be considered. Consider the following example.

In the application specified above, the normal result set for user requests ranges in size from 100 bytes to greater than 5MB. However, the distribution of user transactions being processed shows that over 90% of the incoming and outgoing traffic will be less than 950 bytes in length.

Suppose the TCP/IP frame data packet size within the intranet has been set to 2KB. This means that, for over 90% of the inbound and outbound data traffic, 50% or more of the data frame will be empty. When you start to calculate bandwidth of the network, you will have to cut your estimates in half to account for the 'white space' that will be constantly moving with the end user data.

Adjusting TCP/IP frame data packet size is one consideration, but you may want to talk with project architects about the

possibility of batching requests programmatically or through the use of middleware technology. Always make sure you know how much actual data is being passed, and how much network bandwidth is being wasted.

SOME FINAL THOUGHTS…

It has been said, “What you can’t measure, you can’t manage”. I would venture to say that that is an understatement. When you are working through the component pieces of a multi-tiered application, it is essential that you be able to collect the necessary performance metrics for each step, and also be able to correlate that information after collection. During the process of collecting DB2 UDB performance metrics, we found that, because the application architects chose Tuxedo and the Tuxedo XA protocol to access DB2 UDB, unit of work id information that normally would be available in the SQLDA using various DB2 monitoring or tracing products was not available for collection. While the Tuxedo XA protocol was implemented to allow two-phase commit processing to be controlled by the Tuxedo client, rather than handling processing commit logic within the application as it passed from server to server, the protocol handler removes the SQLDA segment from the DB2 session information it passes back to the application. The result, in our case, was that it became impossible to account, on a transaction-by-transaction basis, for the impact an individual SQL statement, or, indeed, a particular application program, would have within DB2.

With this said, tuning of the environment was still possible, but only at the server level, and not at the application level. Capacity planning was made much more difficult, given that workloads could not be measured exactly, so scaling issues remained irresolvable.

One final hint for a DB2 UDB implementation on AIX: if DB2 were being installed on a ‘mainframe’ platform, care would be taken to ensure that the dispatching priority of the subsystem components was set either manually or by WorkLoad Manager. However, DB2 is often installed within AIX, and the default priority of the installed processes is never changed.

If you use the AIX command:

```
ps –efl | grep db2
```

you will be able to see both the PRI (priority) value of each process and in the next column the NI (Nice) value. By default, the NI value is 20; however, this value can range between the values of –20 and 39.

Both the process NI and PRI values are negatively correlated, meaning that, the higher either value is set, the lower the process's importance is.

As in any other type of system, you will want to make sure that the DB2 processes are ranked below the network and any middleware processes, so that when result sets are created they can be delivered quickly to the end users. However, the DB2 processes themselves should be above the rest of the processing that will occur on a dedicated database server.

IN CONCLUSION

Using the knowledge we have acquired while working with legacy systems, we can implement capacity planning methods to help determine the scalability of converted systems. Just because the hardware and subsystems are different it doesn't mean that we have to disregard all those lessons that we learned working with older enterprise technology.

Aaron Cain
Consultant
The Performance Edge Limited (UK)

Articles for *AIX Update* can be sent to Trevor Eddolls at trevore@xephon.com. A copy of our *Notes for Contributors*, which explains the terms and conditions under which we publish articles, is at www.xephon.com/nfc.

Heroix has begun shipping Version 2.0 of its Heroix eQ Management Suite for multi-platform monitoring and management, which is used to associate the performance of applications, databases, servers, and infrastructure with the specific business activities they support.

New in the release is the next generation of Solution Studio, which lets sites extend and customize monitoring and management of applications and underlying infrastructure, without, it's claimed, writing any code. The new version of Solution Studio, via wizards and templates, allows business-focused monitoring and management to be set up with a few clicks.

Additional features include an extended Web Management Console, which makes management features accessible from any platform, as well as a new Web Reporting portal, an enhanced Web Event Monitor, trend-based alerts, and support for Veritas NetBackup and new versions of Apache, Oracle, Windows Server, and CIM.

The extended Web Management Console makes GUI-driven management functions accessible from all eQ supported platforms. The new Web Reporting portal consolidates reports and graphs from multiple monitored resources and makes them available on-line. Admin staff choose the frequency and timing of automatic updates and specify who may view the information.

It runs on AIX, HP-UX, Solaris, Linux, Windows Server (including NT/2000/2003), and OpenVMS and covers packaged and custom applications, database systems, messaging platforms, Web servers, operating systems, and IT infrastructure components.

For further information contact:
Heroix, 120 Wells Avenue, Newton, MA 02459, USA.
Tel: (617) 527 1550.
URL: http://www.heroix.com/products/detail_eQ.htm.

* * *

Legato is shipping its DiskXtender 2000 Version 5.4 and DiskXtender Unix/Linux Version 2.6 data migration tools, which move inactive or fixed content data to more cost-effective storage, while maintaining user access.

The software migrates inactive data to nearline storage, such as low-cost ATA disk devices, specialized network storage devices, SAN-attached arrays, tape, or optical storage.

Version 5.4 of DiskXtender 2000 delivers integrated support for migrating fixed-content data to the EMC Centera network storage system, including the ability to define retention policy for regulatory compliance and cacheing to speed storage and retrieval performance.

DiskXtender Unix/Linux Version 2.6 delivers integrated support for the EMC Centera and now supports AIX.

For further information contact:
Legato, 2350 West El Camino Real, Mountain View, CA 94040, USA.
Tel: (650) 210 7000.
URL: http://www.legato.com/products.