Managing a Penguin Farm on the VM Prairie

Share 104, Anaheim CA

Session 9212

March 2005

Gordon W. Wolfe, Ph.D.

The Boeing Company

gordon.w.wolfe@boeing.com

Disclaimer

- Use the contents of this presentation at your own risk. Neither the author nor the Boeing Company accept any liability, implicitly or explicitly, direct or consequentially, for use, abuse, misuse, lack of use, misunderstandings, mistakes, omissions, mis-information for anything in or not in, related to or not related to, pertaining or not pertaining to this document, or anything else that a lawyer can think of or not think of.

Introduction

- VM originally created to run many operating systems on one real machine
- Perfect for running Linux guests. VM acts as “hypervisor’ or “Netbios” for Linux.
- Won’t go into reasons for using VM or Linux here.
- Assume you are familiar with both VM and Linux.
- Presentation will be specific to z/VM 5.1.0 and SuSE Linux SLES8/SLES9 as practiced at Boeing, with extensions by implication to other operating systems releases.
- Object here will be to discuss how to run MANY linuxes as VM guests and have a productive system
- I will sometimes note availability of some commercial software products. I am not aware of them all. Sorry if I missed anyone. Nothing in this presentation is an endorsement by Boeing of any product or Company.

The Problem

- VM can support hundreds (or thousands) of virtual servers. We estimate about 350 productive Linux servers per z800 IFL engine.
- When you get that many guest operating systems, how do you
  - Keep everything consistent?
  - Let them talk to each other and clients?
  - Handle updates?
  - Use VM's resources most efficiently?
- You need a Plan!

Implementation at Boeing

- One z800 with two IFL Engines and two s390 engines
- Six z/VM 5.1 LPARs - One for Linux
- OSA Express
- SuSE SLES7 on 8 guests
- SuSE SLES8 on 36 guests
- SuSE SLES9 (64-bit) being installed

Consistency!

- Multiple Linux guests become impossible to manage if they are all different.
- STANDARDIZE, STANDARDIZE, STANDARDIZE!
  - Stick to one distribution, one release of Linux
  - Try as much as possible to make every Linux guest work like every other Linux Guest.
  - Keep similar files in the same places

Have Written Policies

- Have Formal Service Level Agreements
  - Times of Operation/time of maintenance
  - Guaranteed levels of performance
  - Software levels
  - General agreement, not one for each server.
  - On-line server request form
  - Help desk
  - Announcement bulletins

Have Written processes to follow

- For cloning a new server
- For fixing a damaged boot disk
- For upgrading software
- Adding disks, using LVM, etc.

Don’t Give Your Customers Root!

- You will know what and how software is installed
- Customer can’t modify the kernel
- Customer can’t modify security arrangements
- Give customer “su” if privileged commands are needed - on a command-by-command basis.
- Let your Linux support personnel handle all changes

Keep Policies and forms accessible on the web

- Written policy on how to obtain a server. Maybe even form for requesting a server.
- Lots of how-to documents for users:
  - Setting up KDE
  - Keeping Linux secure
  - Running Samba
  - Running Apache
  - Etc.
- Can use Linux Apache for this purpose!

Common 191 disk

- Owned by clone server
- PROFILE EXEC
  - Choose boot from DASD (default) or Reader
  - If boot from DASD, SWAPGEN EXEC for V-DISK
  - <server> EXEC to couple VCTCA's or set up Guest Lan
- Contains files needed to boot from reader
  - INITRD
  - <server> PARMFILE
  - IMAGE
- See Appendix 1 in handouts

Memory and Swap

- Linux swap to VM V-DISK in real/expanded storage
  - Means VM does real paging - more efficient!
  - Set up in PROFILE EXEC on common 191 disk
  - Use SWAPGEN EXEC from sinenomine.com
  - V-DISK defined in directory entry
- Keep Linux Memory to a minimum (otherwise Linux fills up with buffers)
  - Let it use its own swap
  - Only need 64MB for Apache, Samba server
  - Need 256 MB for Oracle 9i - minimum!
  - Need 750MB for WebSphere Commerce Suite!
- Booting from the reader requires at least 128MB for SLES8

Patch and CD server

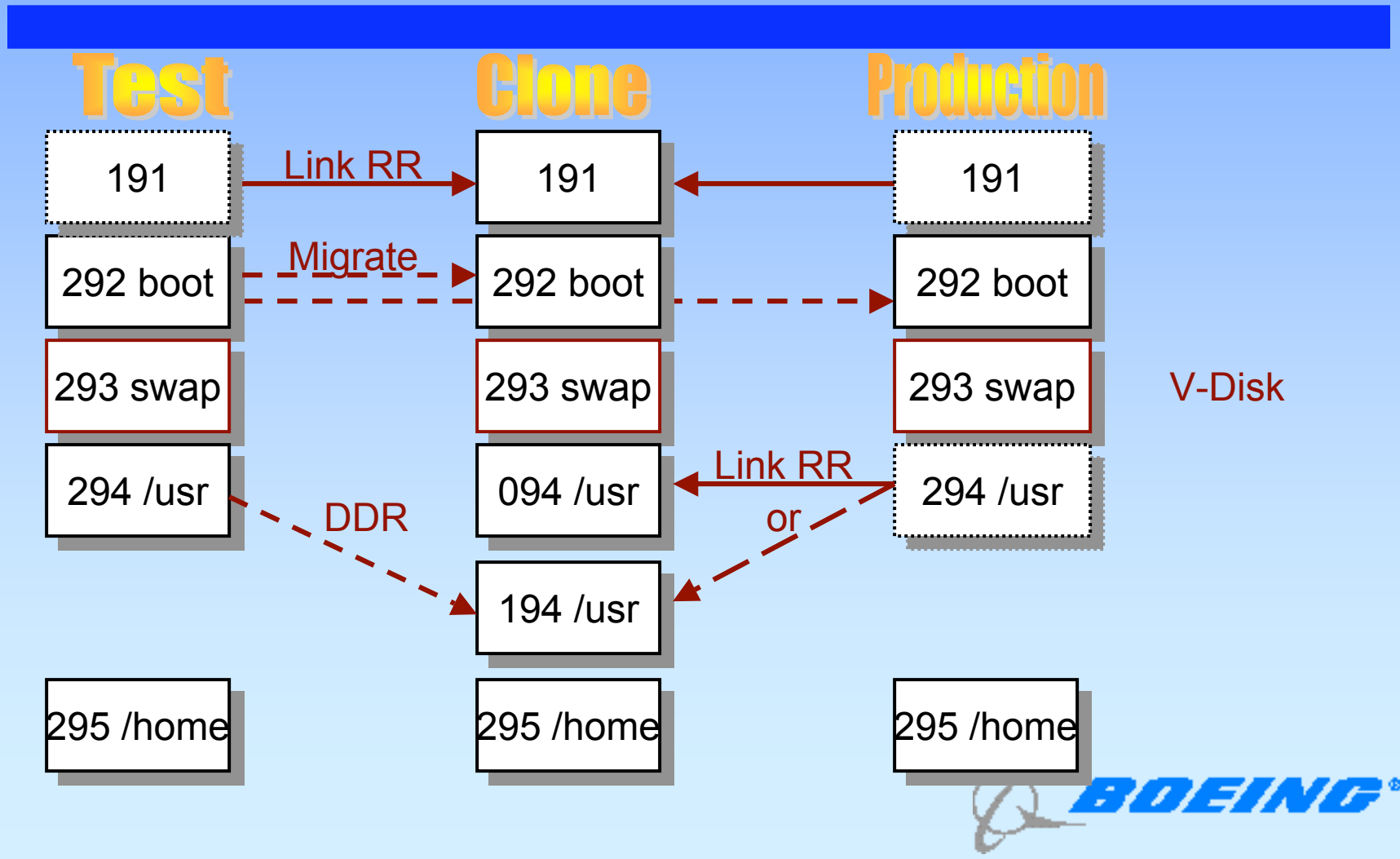
- NFS Server accessible r/o by any Linux administrator
  - Use mount -t nfs -o patchsvr:/share /mountpoint from server1
- /cd - to do installs of new software - copy install CDs to this directory
  - **/cd/SLES8**
    - **../CD1**
    - **../CD2**
- /cd/sles8/patches - to install patches (Samba share /patches)
  - **…/patches/SP3/CD1**
  - **…/parthces/SP3/CD2**

BOEING®

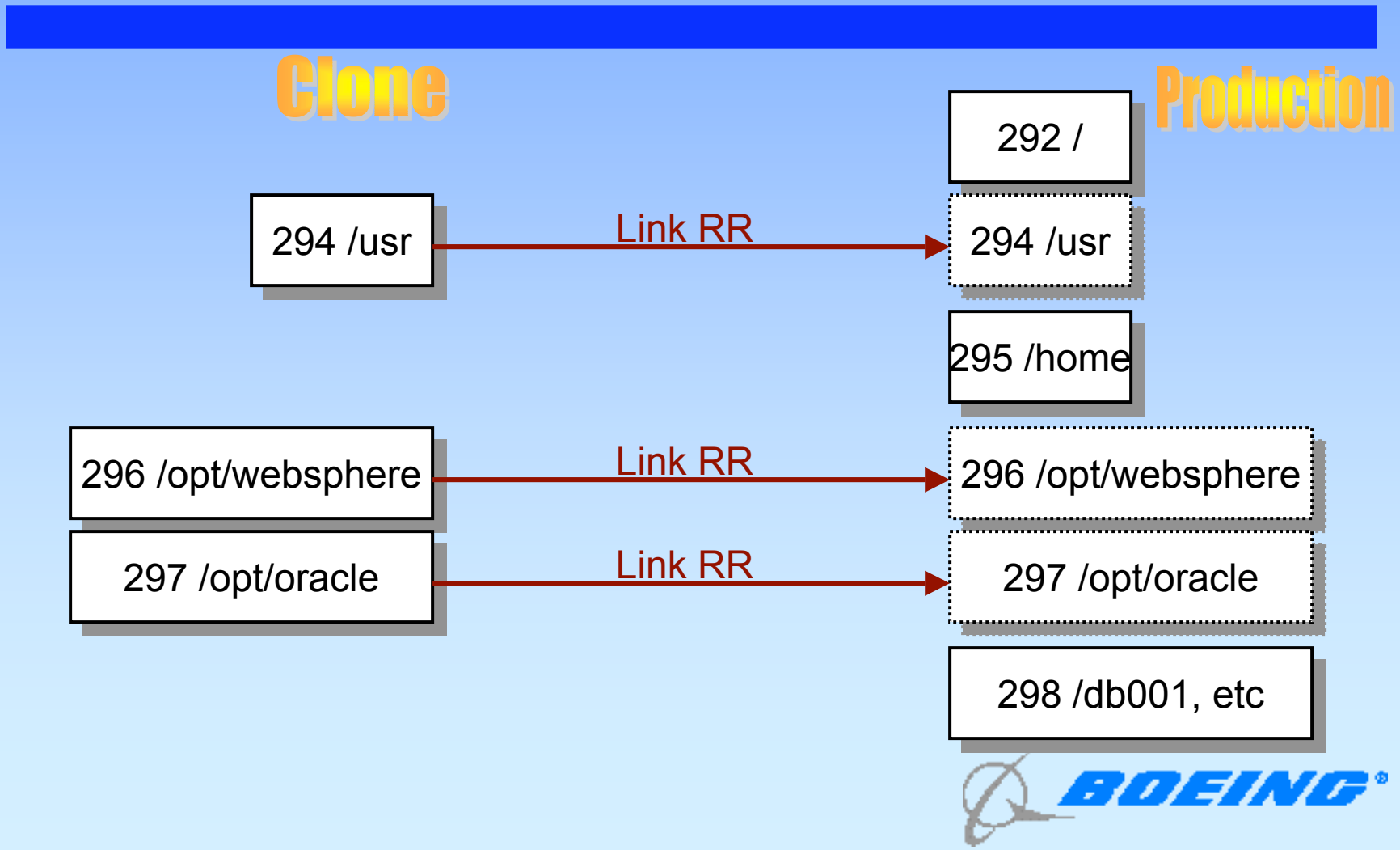
Method One: Shared read-only /Usr disks

- Advantages:
  - Reduces usage of disk space
  - All updates to /usr done in one place
- Disadvantages:
  - Have to have multiple disks for each release new/old
  - Have to separate out upgrade files on /usr and elsewhere
  - Rather labor intensive for upgrades.
  - Rpm database not necessarily kept in sync

Test, Clone and Production



Different Filesystems for Different Applications



Shared /usr and other Filesystems

- Disk(s) owned by clone server
- 094 for production , 194 for new/updated self-link 194 as 294 RW
- Clone machine is shut down virtually all the time, except when updating files.
- Link clone 094 disk as 294 RR in directory of server
- /etc/fstab should mount /usr “ro”
- Parm line in /etc/zipl.conf should read dasd=0293,0292,294(ro),295-2AF root=/dev/dasdb1
- Keep extra DASD devices in parm in case you need to add one later.
- When clone machine done updating 194, do
  - mount /usr -o ro,remount
  - sync;sync
  - Shutdown
- Same for other R/O filesystems

Routing Updates and Changes

- Use scp and ssh to route new files around. (RPM -ql <package> gives list of all files contained in <package.rpm>)
- Have to set up each server to allow no-password ssh/scp using public keys from a userid on your test machine.
- Sample scripts in appendix 4 for automated method.
- Shut down server and swap LINK LXCLONE 094 294 RR for LINK LXCLONE 194 294 RR
- Reboot and hope it comes up.

Read/Write on a Read-Only Directory

- R/O user means some functions have to be moved to a R/W disk, e.g.
  - Apache Webserver
  - Move /usr/local/httpd to /home/httpd or other R/W location
  - Update location in /etc/httpd/httpd.conf

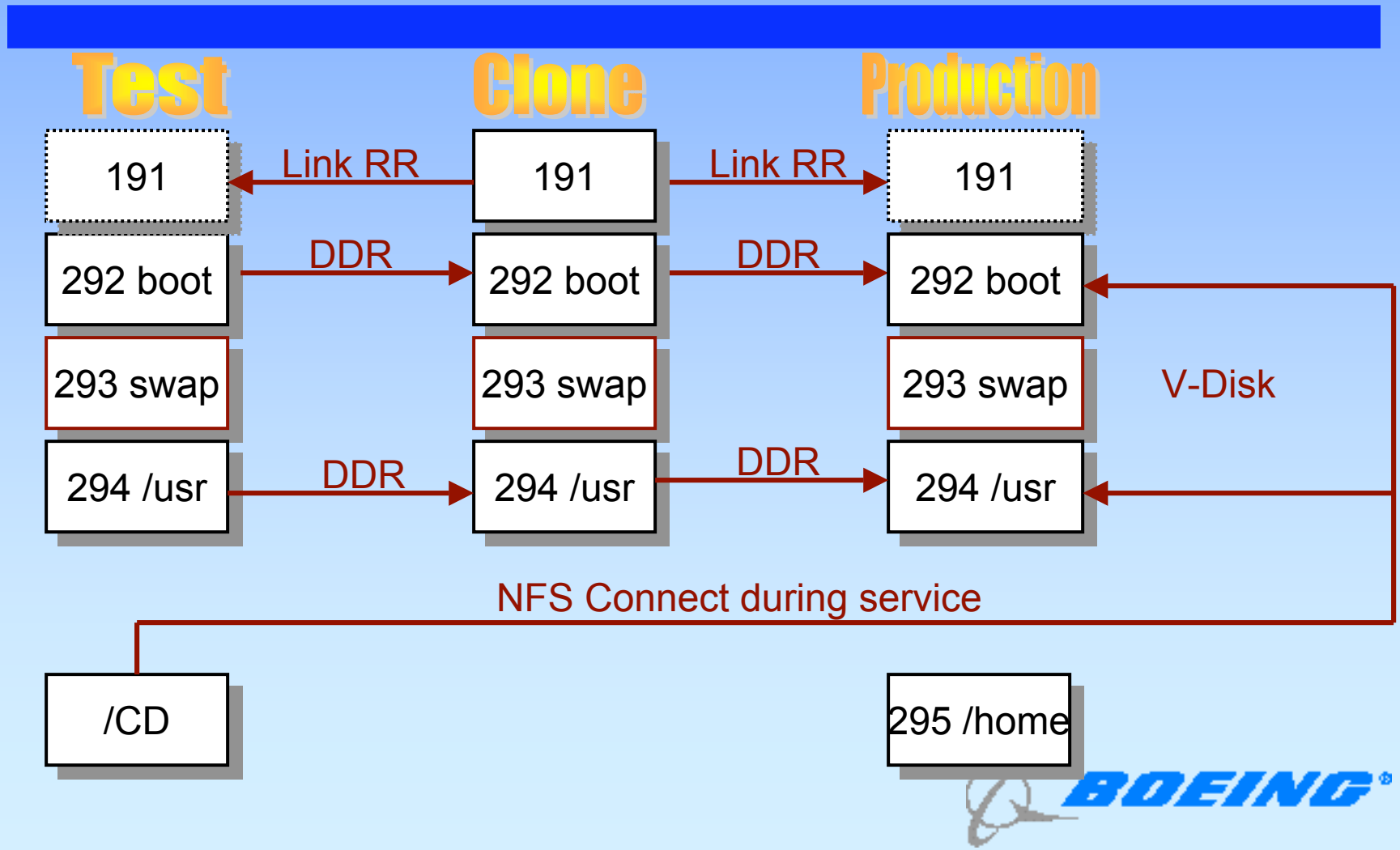
Additional Files in a Read-Only Directory

- Create a new subdirectory on a R/W disk
  - Mkdir /home/mystuff
- Copy files from R/O directory (/usr/mystuff) to it
  - Cd /usr/mystuff
  - tar cf - . | tar xpf - -C /home/mystuff
  - Add or change anything you want to here
- Mount R/W subdirectory over R/O subdirectory cd /home
  - mount -o rw --bind /home/mystuff /usr/mystuff

Method 2: Every user gets nonshared Read-Write disks

- Advantages
  - Upgrades are made easy - just install rpm's from central nfs server.
  - Rpm database in sync
  - Less labor intensive
- Disadvantages:
  - /usr requires a full-pack 3390-3 for each linux. (disk is cheap)
  - Can instead dedicate half a 3390-9 for all system areas.

Test, Clone and Production



Upgrades

- Have to set up each server to allow no-password ssh/scp using public keys from a userid on your test machine.
- Ssh command to mount central nfs repositiory of rpm's
- Ssh commands to install rpms
- Umount nfs
- Can place all this in a single script or use automation in appendix.

Cloning Penguins: Part 1

- Create exec to do couples (ctc or guest lan), parm file on 191
- Set up TCPIP to talk to server
- Create the directory entry
  - Link RR to 191, /usr, other product disks
  - Add in V-disk entry for swapping
  - Add in disk for /home
- DDR the boot (and /usr?) disk from the clone server

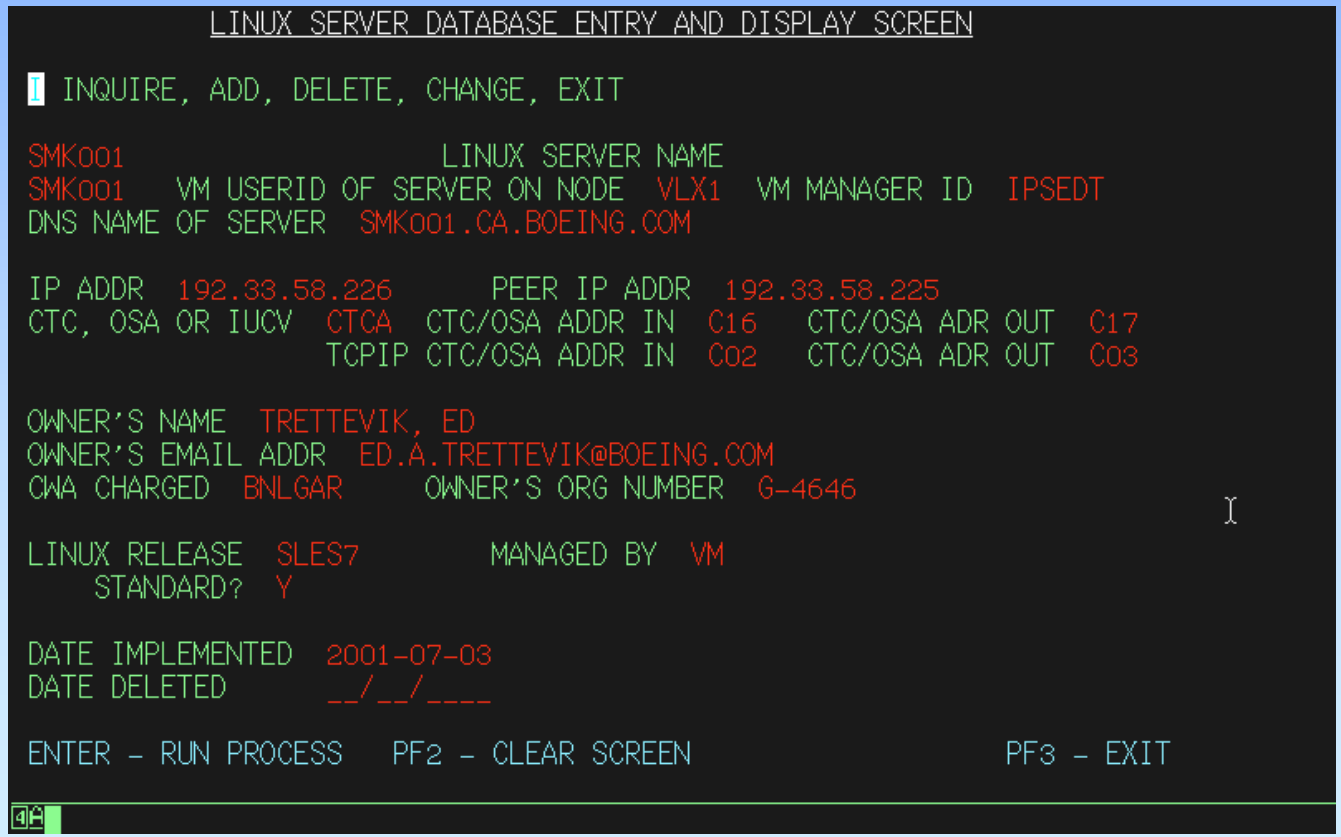
Cloning Penguins: Part 2

- Boot the new server from the reader
  - Mount /dev/dasdb /mnt
  - Chroot /mnt
  - Mount /dev/dasdc /usr
  - dasdfmt /dev/dasdd (and fdasd, mke2fs)
  - Mount /dev/dasdd1 /home
  - Update files to make new server (SuSE)
    - /etc/rc.config
    - /etc/httpd/httpd.conf
    - /etc/route.conf
    - /etc/hosts
    - /etc/fstab
    - /etc/smb.conf
    - Run /sbin/SuSEconfig
    - Run zipl
  - Exit, umount everything and shut down

Cloning Penguins: Part 3

- Boot the new server from DASD
- Set up authorities to start and stop server
- Place DNS name in DNS server
- Whole thing takes about 2-3 hours. Can be automated with CMS execs and shell scripts preinstalled on the clone server.
- Be Sure to keep a database of servers!
  - DNS, userid, IP address, owner-name, comm method, usage type

Database of Servers



Cloning Software Available (not an endorsement by Boeing)

- VMLINMAN
  http://www.glasshousesystems.com/home.html
- STK SNAPVANTAGE
  http://www.storagetek.com/prodserv/products/software/svan/
- LINUXCARE LEVANTA http://www.linuxcare.com
- ADUVA at http://www.aduva.com
- Unsupported demo software from Tung-Sing Chong at IBM Endicott
  - Code at http://www.vm.ibm.com/devpages/chongts/
  - Instructions at http://www.vm.ibm.com/devpages/chongts/tscdemo.html

Starting Servers

- Create a server start/stop userid
- Allow SMSG commands via WAKEUP
- Have list of authorized users
- Exec to do actual AUTOLOG of server
- PROFILE EXEC in Clone machine defaults to “boot from disk” if no virtual console attached.
- User/owners of servers require second VM id to start system

Stopping Servers (part 1)

- Put VMPOFF=LOGOFF into parm file /etc/zipl.conf to log off Linux userid when linux O/S quits
- Use journaled filesystems (ext3, reiserfs, jfs) in case something breaks and you can’t shut down cleanly

Stopping Servers. Part 2

- With z/VM 4.3's SERVC facility and Linux 2.4.7 or later
  - Linux can be patched to shut down automatically at CP SHUTDOWN or CP SIGNAL SHUTDOWN
  - Put ca:12345:ctrlaltdel:/sbin/shutdown -t1 -h now in /etc/inittab - One process runs in Linux all the time.
  - Included with SLES8, SLES9

Linux and Guest Lans

- Guest Lans are faster and easier than CTC or IUCV, but require z/VM 4.3.0 for full function, and SLES8.
  - Up to 700 mb/sec
  - Let SYSTEM own the guest lan
  - Define lans in TCPIP's PROFILE TCPIP
    - DEVICE HIPR3 HIPERS 1F00 PORTNAME LINUXLAN AUTORESTART
    - LINK QDIO3 QDIOIP HIPR3
  - Define the guest lans in AUTOLOG1
    - DEFINE LAN LINUXLAN MAXCONN INF OWNERID SYSTEM TYPE HIPER
  - Be sure Linux is set up to use guest lans
    - /etc/sysconfig/network/ifcfg-hsi0 /etc/sysconfig/network/routes
    - /etc/modules.conf /etc/chandev.conf
  - In server startup exec, issue
    - CP COUPLE A001 TO SYSTEM LINUXLAN
  - **VSWITCH (z/VM 4.4) even easier!**

Tuning Linux to work with VM

- On-Demand Timer: echo "0" >/proc/sys/kernel/hz_timer in SLES8, SLES9
- V-DISK for swap, minimum virtual storage (Use the SWAPGEN EXEC. See Appendix 2) In SLES8, use dasd_diag_mod.
- MINIOPT CACHE RECORDMDC for real DIAG minidisks. CMS FORMAT/RESERVE + mke2fs. No dasdfmt or fdasd! 13X throughput for 50% increase in I/O CPU.
- Use “dasdfmt -d cdl” or reserve first cylinder of pack for VM disk label:
  - MDISK 294 3390 1 3338 V163E1 MR LINUX USR DASDC
- Use INCLUDE files and ACIGROUPS in the directory to better manage
- Stagger file-level backups of Linux servers.

BOEING®

Updates and Changes

- Load patches, CD's onto a Linux Patch server
  - Easier to use than CD or Windows server
  - Makes it available to those Linux owners who do their own updates
- Update test server first
- When working, move to clone server, including 194 disk
- Route around to production servers

“Outsource” Linux maintenance to someone else

- Unix gurus
- NT server people
- End user as last resort - use at least for creating own userids and groups
- Set up formal Service Level Agreements to follow.

Backing up Linux with VM

- VM:Backup will do physical (track) backups of Linux, but must restore an entire filesystem disk set to recover one file. E.g. all of /usr.
- Problem is compounded if using LVM for multiple-volume filesystem. Have to restore ALL volumes of logical volume to restore one file!
- Same for DDR or any other VM backup system. No file-level backups.

Backing up Linux with TSM

- TSM (Server on VM or z/OS) works well for file-level backups
  - VM server release 3, z/OS at release 5.
  - but uses lots of network capacity, especially first time!
  - Don't use software compress in client machine. CPU hog!
  - Can do full or incremental, keep multiple generations.
  - Stage to DASD first, then move to tape to minimize number of tape drives used.

Other ways to back up Linux at file level

- CA-Brightstor.
- Veritas NetBackup
- Other products on near horizon.
- If tape available to linux, can use amanda.

Roll your own Backups

- Give Linux server capability to use BFS, OpenExtensions in directory, BFS server
- Use Linux NFS, VMNFS server to mount BFS
- Use Tar to back up to NFS-mounted BFS
- Use VM:Backup to back up BFS
- Can do with 4 shell scripts and 3 files, but very labor intensive, requires root privilege, and uses LOTS of disk space! (Contact me if you want the scripts)
- With Neale's cpint package and tape support in 2.4.7 (Both included in SuSE SLES8) can go direct from Linux direct to tape with "tar" or "dump", but tape scheduling is a nightmare. Cpint can be used to SM VMTAPE MOUNT (no response to Linux) and CP DETACH

Accounting

- Most Unix-type systems do not do job accounting very well.
- Hooks and packages available but require extensive kernel mods
- Future kernel may have job accounting in it.
- Under VM it's simple! Use a separate server for each account! If you have to share data among servers, use NFS!
- VM:Account will create (with exits) and collect charge records for CPU, DASD, BFS, tape mounts, and so on.
- VM:Account will also do ad-hoc reports on usage. Cumbersome to set up, but well worth it in the end. Beware! Users will want you to run reports for them!

Close

- VM can be used as “hypervisor” for many linux guests
- The more guests you have, the more work maintaining them is.
- Many members of Linux-VM community have come up with some ideas for managing many servers.
- Commercial software solutions are still on the horizon.
- See Mark Post’s LinuxVM page at http://linuxvm.org
- Join the Linux-390 Listserver! (address on LinuxVM page above)

Appendix 1 - Starting up Linux

```
/*-------------------------------------------------------------*/
/* Common Profile Exec for Linux Server Machine                */
/* By Gordon Wolfe, VM Technical Services            05/22/00*/
/*-------------------------------------------------------------*/
address command

'SET RDYMSG SMSG'
'CP SET ACNT OFF'
'CP SET RUN ON'
'CP SET RELPAGE OFF'
'CP SET EMSG ON'
'CP LIMIT CLEAR CP'

'CP SET PF11 IMMED FILEL'
'CP SET PF12 RETRIEVE'
'CP TERMINAL LINESIZE 255'
'CP TERMINAL CHARDEL OFF'
'CP SET EMSG ON'

/* Set up for this particular linux server machine                  */
'ESTATE' userid() 'EXEC A'
if rc <> 0 then exit rc
'EXEC' userid()

/* Determine if we start up Linux now.                              */
startflag = 'N'
iplflag   = 'D'

/* Are we running disconnected? if so, start linux.                 */
'PIPE CP QUERY' userid() '| var usrline'
parse value usrline with . . term .
if term = 'DSC' then startflag = 'Y'
```

Appendix 1 - Continued

```
/* Not disconnected? ask to start up.                                  */
/* Also find out where to start up from.  Reader IPL or DASD ipl.      */
else do
  say 'Do you want to start up LINUX now? (Y/N)'
  pull ans .
  if left(ans,1) = "Y" then startflag = 'Y'
  if startflag = 'Y' then do
     say 'Do you want to IPL from DASD or from the reader? (D/R)'
     say 'The default is to IPL from DASD.'
     pull ans .
     select
       when left(ans,1) = 'R' then iplflag = 'R'
       when left(ans,1) = 'D' then iplflag = 'D'
       when ans         = ''  then iplflag = 'D'
       otherwise do
         say ans 'is an invalid choice.'
         exit 8
       end
     end
  end
end

if startflag = 'Y' then do
  if iplflag = 'R' then queue 'EXEC SLES7IPL'
  if iplflag = 'D' then do
    queue '1'
    queue 'LXSWAP'
    'FORMAT 293 E ( BLK 4096'          /* format/reserve V-disk      */
    if rc <> 0 then exit rc            /* for swap space             */
    queue '1'
    'RESERVE LINUX SWAP E6'
    if rc <> 0 then exit rc
    queue 'EXEC IPLDASD'
  end
end
exit
```

Appendix 1 - Continued

```
/* Sample LINUX002 EXEC for Linux userid LINUX002  */
address command
'CP COUPLE C16 TO TCPIP C20'
'CP COUPLE C17 TO TCPIP C21'
  or
'CP COUPLE A001 TO SYSTEM LCMEXT'

(Parmfile file Linux userid LINUX002)
ramdisk_size=32768 root=/dev/ram0 ro ctc=0,0xC16,0xC17,ctc0

/* IPLDASD EXEC */
ADDRESS COMMAND
TRACE o
'CP CLOSE RDR'
'CP PURGE RDR ALL'
'CP DETACH 190'
'CP DETACH 19E'
'CP IPL 292 CLEAR'

/* SLES7IPL EXEC                                          */
/* exec to IPL Linux from the reader and run from a ramdisk */
/* by Gordon Wolfe, VM Technical Services         08/17/01  */
address command
trace o

/* Do we have enough virtual storage to do this?          */
'MAKEBUF'
buf1 = rc
'EXECIO 1 CP ( STRING Q V STOR'
pull . . stor .
'DROPBUF' buf1
stor = STRIP(stor,'L','0')
stor = STRIP(stor,'T','M')
if stor < 64 then do
  say 'Virtual storage must be 64M or greater to IPL from reader'
  say 'Perform CP DEF STOR 64M and IPL CMS.'
  exit 8
end
```

Appendix 1 - Continued

```
/* Do we have the files we need?                                  */
'ESTATE' userid() 'PARM *'
if rc <> 0 then do
  say 'File' userid() 'PARM * not found.'
  exit 28
end

/* Find the filemode for the files we need                        */
'PIPE CMS LISTF SLES7 IMAGE * |',
  'take 1 |',
  'var imageloc'
parse value imageloc with . . fm .
fm = left(fm,1)

/* All looks okay, proceed.                                       */
'CP CLOSE RDR'
'CP PURGE RDR ALL'
'CP SPOOL PUN * R'
'PUNCH SLES7 IMAGE' fm '( NOH'
'PUNCH' userid() 'PARM' fm '( NOH'
'PUNCH SLES7 INITRD' fm '( NOH'
'CP CHANGE RDR ALL KEEP NOHOLD'
'CP IPL 00C CLEAR'
```

Appendix 2 - Modified File in Linux for swapping to V-Disk

SuSE Linux 2.2.16
/sbin/init.d/boot
At about line 153, a change was made to change swap from that listed in fstab to use the FB-512 V-disk at /dev/dasda1. This device should have been CMS FORMAT/RESERVEd by CMS before IPLing linux. Then at boot time a mkswap is done on /dev/dasda1 and swapon is used to start swapping on this device. The lines in /sbin/init.d/boot should read:

```
#cho "Activating swap-devices in /etc/fstab..."
#wapon -a &> /dev/null
echo "Creating swap file signature"
mkswap /dev/dasda1
echo "Activating swap partition"
swapon /dev/dasda1
```

SuSE SLES7 2.4.7
This process has been moved to /etc/init.d/boot and is done at line 200.

Appendix 3 - Shutting Down Linux from VM

```
/* EXEC to send shutdown commands to a Linux guest                    */
/* Assumes root password is same as Linux VM userid password.         */
/* Also assumes server has a parmline containing vmpoff=LOGOFF        */
/* Userid running this exec must be a VMSECURE administrator          */
/* By Gordon Wolfe, VM Technical Services                  03/09/2001*/

address command

arg server .
'PIPE CP Q SECUSER' server '| drop 2 | var bkupsecuser'
if rc <> 0 then do
  say server 'is an unknown Linux server.'
  exit 8
end
parse var bkupsecuser with oldsecuser .
if oldsecuser = 'not' | oldsecuser = 'NOT' then oldsecuser = 'OFF'


line = 'shutdown -h now'

/* Set the secondary userid to ourself.                               */
'CP SET SECUSER' server userid()

/* Get the console logged on as root                                  */
call linuxpwd server
if result <> 0 then exit result

/* And send the shutdown command                                      */
'CP SEND' server line

/* Clean up and quit.                                                 */
done:
'CP SET SECUSER' server oldsecuser
exit
```

Appendix 3 - Continued

```
linuxpwd: procedure
address command

arg linuxmach .
if linuxmach = '' | linuxmach = 'LINUXMACH' then do
  say 'No Linux machine specified'
  return 8
end

/* Get the password for the server        in the proper case.        */
rootname = 'root'
call getpass linuxmach
if result = 28 then do
  say 'no password on file for' linuxmach
  return 8
end
else pw = lowercas(result)


do i=1 to 3
  'CP SEND' linuxmach rootname
  'CP SLEEP 1 SEC'
  'CP SEND' linuxmach pw
  'CP SLEEP 1 SEC'
end
return 0



getpass: procedure
/* procedure to query VMSECURE for the password of the server  */
arg finduser .
```

Appendix 3 - Continued

```
'ERASE LINUX TEMP A'
'MAKEBUF'
buf1 = rc
queue 'SSAVE LINUX TEMP A'
queue 'QQUIT'
'VMSECURE EDIT' finduser '( NOPROF'
if rc <> 0 then do
  say finduser 'not found in VMSECURE'
  'ERASE LINUX TEMP A'
  return 28
end
'DROPBUF' buf1
'PIPE < LINUX TEMP A |',
  'locate /USER / |',
  'take 1 |',
  'specs word 3 1 |',
  'var pw'
'ERASE LINUX TEMP A'
return pw


/* Lowercas                                                        */
/* translates input argument to lowercase                          */
/* By Gordon Wolfe, Vm Technical Services                06/23/98 */
lowercas: procedure
arg inp
out = translate(inp,'abcdefghijklmnopqrstuvwxyz','ABCDEFGHIJKLMNOPQRSTUVWXYZ','.
return out
```

Appendix 4 - Routing Updates to Servers From a Central Maintenance Server

File /root/updates/hosts - Place the names of the hosts to which updates will be routed in this file.

```
#Nickname in hosts file or DNS name    VM userid
#----------------------------------    ---------
clone                                  LINUX000
test                                   LINUX001
patch                                  LINUX002
webserver                              LINUX003
```

File /root/updates/files - Place the fully-qualified filenames of files that will be routed to the above hosts:

```
/etc/hosts
/etc/profile.local
product-20020505.rpm
```

File /root/updates/precommands - Place commands that you want the above hosts to execute before moving any files

```
mkdir /root/test
```

File /root/update/postcommands - Place commands that you want the above hosts to execute after moving files:

```
id
rpm -Uvh product-20020505.rpm
```

Then run /root/updates/update to send all files to all hosts then execute all commands on all hosts:

```
rm runupdate
rexx updates.rex $1
chmod 770 runupdate
/root/updates/runupdate
```

Appendix 4 - Continued

This calls the rexx program updates.rex, which is

```
/* updates.rex                                                    */
/* An exec to create a shell script to take a list of files       */
/* from the file ./files and send them to a group of linux servers*/
/* listed in a file named ./hosts                                 */
/* Then execute a number of commands taken from a file ./commands */
/* Called from shell script "update"                              */
/* Assumes:                                                       */
/* file names in ./files are fully qualified path names           */
/* host names in ./hosts are resolvable through /etc/hosts        */
/* the shell script created will be run from userid root          */
/* Userid root on the receiving host has ssh configured to allow  */
/* file copies and commands without a password from root on server */

trace off
signal off error

parse arg onehost .
if onehost <> '' then say "Processing for" onehost "only."

/* First, get names of files into stem variable                   */
jfiles = 0
oldq = queued()
"cat files >FIFO"
do until queued() = oldq
  parse pull file
  if left(file,1) = "#" then iterate
  jfiles = jfiles + 1
  parse value file with fname.jfiles fnewname.jfiles .
  if fnewname.jfiles = '' then fnewname.jfiles = fname.jfiles
end
```

Appendix 4 - Continued

```
/* Next, get commands to execute into stem variable                  */
j1cmds  = 0
oldq = queued()
"cat commands >FIFO"
do until queued() = oldq
  parse pull file
  if left(file,1) = "#" then iterate
  j1cmds  = j1cmds  + 1
  parse value file with precommand.j1cmds
end
 j2cmds  = 0
oldq = queued()
"cat commands >FIFO"
do until queued() = oldq
  parse pull file
  if left(file,1) = "#" then iterate
  j2cmds  = j2cmds  + 1
  parse value file with postcommand.j2cmds
end


/* Next get list of hosts to send files to                           */
oldq = queued()
nhosts=0
"cat hosts >FIFO"
do until queued() = oldq
  parse pull line
  if left(line,1) = "#" then iterate
  nhosts=nhosts+1
  parse value line with hostname.nhosts vmuserid.nhosts .
  if onehost = hostname.nhosts then khost=nhosts
end
if onehost <> '' then do
  vmuserid.1 = vmuserid.khost
  nhosts = 1
  hostname.1 = onehost
end
```

Appendix 4 - Continued

```
/* for those hosts that are logged on, build a script                */
/* Is the host logged on?                                            */
do j=1 to nhosts
  oldq = queued()
  "hcp q" vmuserid.j ">FIFO"
  do until queued() = oldq             /* get just last line         */
    parse pull line
  end
/*    Host is not logged on.  Ignore it for now.                      */
  if strip(line) <> 'Ready;' then say '+++' hostname.j 'not running.'
  else do
/*    Host is indeed running.                                         */
 /*    Build the shell script to execute the precommands             */
    "echo -e echo -e Executing precommands on" host ">> runupdate"
    do i=1 to j1cmds
      newline = ”ssh"
      newline = newline “root@”host precommand.i
      "echo -e" newline ">> runupdate"
    end                                /* do i=1 to j1cmds           */
/*    Build the shell script to actually send the files               */
    host = hostname.j
    "echo -e echo -e                          "  ">> runupdate"
    "echo -e echo -e Sending files to" host ">> runupdate"
    do i=1 to jfiles
      newline = "rsync -pogt -r -e ssh -l" fname.i
      newline = newline host":"fnewname.i
      "echo -e" newline ">> runupdate"
    end
/*    Build the shell script to execute the postcommands              */
    "echo -e echo -e Executing postcommands on" host ">> runupdate"
    do i=1 to j2cmds
      newline = ”ssh"
      newline = newline “root@”host postcommand.i
      "echo -e" newline ">> runupdate"
    end                                /* do i=1 to j2cmds           */
  end                                  /* else do                    */
end                                    /* do j=1 to nhosts           */
exit
```