# Tools, Tips and Tricks for Managing Cray XT Systems

*Kurt Carlson*
*Arctic Region Supercomputing Center (ARSC)*
*University of Alaska Fairbanks (UAF)*
*909 Koyukuk Drive, Suite 105*
*Fairbanks, AK 99775-6020 USA*
*1-907-450-8640*
kcarlson@arsc.edu
http://www.arsc.edu/

**ABSTRACT:**

Managing large complex systems requires processes beyond what is taught in vendor training classes. Many sites must manage multiple systems from different vendors. This paper covers a collection of techniques to enhance the usability, reliability and security of Cray XT systems. A broad range of activities, from complex tasks like security, integrity and environmental checks of the Cray Linux Environment, to relatively simple things like making 'rpm -qa' available to users will be discussed. Some techniques will be XT specific, such as monitoring L0/L1 environment, but others will be generic, such as security tools adapted from other systems and re-spun as necessary for the XT Cray Linux Environment.

**KEYWORDS:**

Cray XT, System Administration, Security, Reliability, Usability

## Introduction (top -- intro -- index -- general -- install -- CNL -- ongoing -- conclusion)

Managing a Cray XT system, like any other complex production system, can be reduced to simple principles:

- Understanding what you have (baseline system)
- Knowing when something changes (problem identification)
- Managing changes

Although these are simple concepts, there are many details to resolve in execution. This paper will discuss how this is accomplished at the Arctic Region Supercomputing Center (ARSC) located at the University of Alaska Fairbanks (UAF). There really are no *Best Practices* since sites have different requirements and different oversight, but there are a number of *Good Practices* to work from. This paper will discuss practices which have evolved over 15 years at ARSC in managing multiple platforms:

- Cray YMP, T3E, J90, SV1, SX6 (NEC), X1, XD1, XT5
- Irix workstations and servers
- Solaris workstations and servers
- AIX Clusters
- Linux workstations, servers, and clusters

These practices come from combined ARSC and UAF staff expertise spanning four decades and aggregate centuries of experience in high performance, enterprise, and departmental computing and drawing on multiple conferences and user groups and peer relationships with many other sites. We borrow good ideas when we find them and share them when we can. The evolution of tools and practices does mean that not all were designed well up-front. In describing what and how at ARSC, a translation of how tasks can be done better in general and for a particular site's need must also occur.

### Concepts for Managing Any System

- Do not change anything in system space directly.
- Maintain a repository with history to recreate local customizations.
- Log actions.
- Avoid working alone and communicate what you are doing.
- Avoid operating directly as root: interruptions make mistakes too easy and logging is difficult.
- Establish appropriate auditing processes.
- Automate monitoring and review processes as much as possible.
- Re-use tools and practices from other systems wherever reasonable. Common practices allows others to fill-in.
- Continually improve processes. If something breaks once, it is likely to break again. Improve detection and avoidance.
- If you do something more than once you are likely to have to do it again: document and automate.

### Included with this Paper (for CUG)

- **`admpkg.tgz`** - tar of various referenced tools
- **`CrayFiles.tgz`** - tar of various referenced examples
- **`doc/`** - supplemental documentation (Index to ARSC XT5)

All the above *extras* were written at the University of Alaska or ARSC. They are provided *as is*, scripts and documentation in particular follow

ARSC naming and conventions and will require modification for use elsewhere.

### Acronyms / Definitions

- ARSC - Arctic Region Supercomputing Center
- UAF - University of Alaska Fairbanks
- HPCMP - DoD High Performance Computing Modernization Program
- DoD - U.S. Department of Defense
- DSRC - DoD Supercomputing Resource Center
- "our peers" (HPCMP DSRCs with Cray XTs) - NAVY, ERDC, ARL, and in 2010 AFRL
- TDS - Test and Devlopment System
- PDU - Power Distribution Unit
- CLE - Cray Linux Environment
- SNL - Service Node Linux
- CNL - Compute Node Linux
- SMW - System Management Workstation
- CMS - Cray Management System (*mazama*)
- NHC - Node Health Check
- HSM - Hierarchical Storage Manager
- ACL - Access Control List
- NIC - Network Interface Card

ARSC is a department within UAF with primary funding from "our sponsors", the HPCMP . ARSC supports high performance computing research in science and engineering with emphasis on high latitudes and the arctic serving both HPCMP and UAF.

---

## Index (top -- intro -- index -- general -- install -- CNL -- ongoing -- conclusion)


### General

a. Make 'rpm -qa' available to users on login nodes
b. Make CNL more Linux-like
c. Make tracejob work from login nodes
d. Job tools: xtjobs, search_alps, qmap

1. Establish practices for managing system changes
2. Develop reusable tools
3. Manage/log root access
4. Eliminate passwords where possible
5. Baseline configuration, other assets, and peers
6. Responses for bad computer support advice

### Install

1. Understand boot disk layout
2. Resolve uid|gid collisions (cluster, external NFS)
3. Mount most filesystems nosuid,nodev
4. Reduce exports (no global, ro where appropriate)
5. Reduce memory filesystems (default 1/2 memory)
6. Audit/secure system access points
7. umask management
8. Eliminate unnecessary services: xinetd, chkconfig
9. Eliminate unnecessary services: rpm -e
10. Comments on non-Cray ssh and sudo
11. sdb and boot node on non-cluster networks
12. Avoid ipforwarding
13. Enable process accounting (and roll files)
14. Raise maximum pid
15. Establish external system trust relationships
16. Audit files Cray wants preserved with upgrades
17. esLogin Inet configuration
18. Customize startup and shutdown auto scripts
19. Shutdown & Startup procedures beyond auto
20. Emergency power off procedure

### CNL

1. Allow drop_caches for users (benchmarks)
2. Use read-only rootfs and o-w /tmp, and /var
3. Secure ssh: root authorized_keys, shadow
4. Mount lustre nosuid,nodev
5. Establish core_pattern
6. Access to external license server
7. Dump procedures and dump archival
8. Home and /usr/local filesystem access

### Ongoing

1. Audit/reduce suid binaries
2. Audit/reduce other+write files
3. Audit/resolve unowned files
4. Identify dangling symlinks
5. Eliminate other+write in suid filesystems
6. Clean-up old/unusable Cray modules and rpms
7. Audit orphaned process
8. Session life limits
9. Establish process limits
10. Audit open ports (lsof, nessus)

11. Audit locally created directories
12. Audit all system file changes
13. Audit system state: chkconfig
14. Audit system state: sysctl
15. Audit system state: iptables
16. Audit system state: mtab, lsmod, nework, ppid=1
17. Audit modules state / history
18. Audit rpm state / history
19. Audit filesystem content
20. Audit system state: disk storage

21. Purge /tmp and lustre scratch
22. Health check and event paging, internal
23. Health check and event paging, external
24. Roll and protect system logs
25. Archive external logs
26. Backup disk storage configuration (LSI)
27. Check L1 and L0 temperature, voltage, health
28. Check node response
29. Usage allocation
30. Audit CNL state (user perspective)

31. Audit for unsafe user .files
32. Audit for excessive disk use
33. Synchronize passwords where required
34. Audit passwd, shadow, group files
35. Audit .k5login (or authorized_keys) content
36. Establish system backups
37. Audit cron usage
38. Run external security scans
39. Ongoing system regression tests (benchmarks)
40. Stay aware of what peers are doing

9. Audit and manage raw image
10. Compute node health checks (NHC)

---

## Tasks - General (top -- intro -- index -- general -- install -- CNL -- ongoing -- conclusion)

Since this *is* a CUG paper we will start with some Cray specific general tips which directly benefit users before we get completely lost in system administration.

### a. General: Make 'rpm -qa' available to users on login nodes

**Technique:** symlink
**Alternative:** unhappy users, eslogin

Advanced Linux users want to know what level of various packages are on systems they use. The Cray XT environment almost immediately causes a reaction when these users cannot issue a simple `'rpm -qa'`. The following work around permits this:

```
boot001: sudo xtopview -m "expose rpm query to service nodes"
default/:\w # mv /var/lib/rpm /var.rpm
default/:\w # ln -s ../../var.rpm /var/lib/rpm
default/:\w # exit
boot001: xtprocadmin | grep service | grep -v 0x0 | awk '{print $3}' >~/SNL
boot001: export WCOLL=~/SNL # list of service nodes
boot001: sudo pdsh \
  "mv /var/lib/rpm /var/lib/rpm.org; ln -s /var.rpm /var/lib/rpm"

login1: rpm -qa | head -3
insserv-1.04.0-20.13
libusb-0.1.12-9.2
libacl-2.2.41-0.7
```

The information is in `/rr/var/lib/rpm` which is overmounted and invisible on service nodes. The only requirement is to move the useless version on the service nodes then point to the shared version. Use the relative symlink `"../../var.rpm"` because SMW upgrades accessing shared root require it.

### b. General: Make CNL more Linux-like

**Technique:** LD_LIBRARY_PATH; clone bin, lib, et.al.
**Tool**: `pkg/cnl.tgz`
**Alternative:** unhappy users, dvs+dsl

Users get annoyed by the required use of static binaries and the inability to run scripts or common commands on compute nodes. While command/script use should be minimized, exercising a `gzip` of results in the context of a compute node may make more sense than overburdening under-powered login nodes. By cloning Linux binaries and libraries into a Lustre directory and use of LD_LIBRARY_PATH, any user can make CNL more complete even if a site chooses not to. Consider this the poor-man's DSL/DVS (no servers required).

```
login1: cat /usr/local/cnl/source.ksh
#!/usr/local/cnl/bin/ksh
PATH=/usr/local/cnl/bin:/usr/local/cnl/usr/bin:/bin:/usr/bin:/usr/local/bin
LD_LIBRARY_PATH=/usr/local/cnl/lib64:/usr/local/cnl/usr/lib64:/usr/local/cnl/lib64/ast
export LD_LIBRARY_PATH PATH
set -o vi

login1: cat cnl.ksh
#!/bin/ksh
echo; df -h /usr/local
echo; uals -zZ  --newer 1d
echo; uname -rn

login1: qsub -I
qsub: waiting for job 15464.sdb to start
qsub: job 15464.sdb ready
login1: cd $PBS_O_WORKDIR
login1: aprun -b -n 1 /bin/ksh -c ". /usr/local/cnl/source.ksh; ./cnl.ksh"

Filesystem            Size  Used Avail Use% Mounted on
```

```
7@ptl:/smallfs        1.1T   25G 1018G   3% /lustre/small

- 0750  1561     206        77 100406.0145 cnl.ksh

nid00022 2.6.16.60-0.39_1.0102.4784.2.2.48B-cnl
Application 66047 resources: utime 0, stime 0
```

In the context of this paper, this technique will be used both for audit CNL state and node health checks (NHC).

The included `pkg/cnl.tar` at ARSC is installed as:

```
boot: sudo ssh nid00031 # log on compute node
Enter password:
# ognip initramfs - Access must be specifically authorized by ARSC
#
# ualscnl -dLMyf /usr/local /usr/local/cnl
D 0755 /usr/local -> /import/home/usrlocal
D 0755 /usr/local/cnl -> /lustre/small/cnl
# exit
Connection to nid00031 closed.
```

See `pkg/cnl/doc/README.ARSC` for how this was constructed. There *are* better ways, but this was hacked quickly as a proof-of-concept and happened to be good enough to live on. At ERDC, NAVY, and NICS this is installed in my personal directory.

### c. General: Make tracejob work from login nodes

This is a trick we borrowed from one of our peers, I do not recall if it was ERDC or NAVY. With `/ufs` globally mounted for SNL, placing the PBS files there and symlinking allows `tracejob` to work from login nodes which is quite useful:

```
boot: push -m pingo1,sdb cmd -eqf uals /var/spool/PBS
nid00003:l 0777  root  crayman   8 090225.1830 /var/spool/PBS -> /ufs/PBS
nid00032:l 0777  root  crayman  15 090225.1822 /var/spool/PBS -> /ufs/PBS/pingo1
```

### d. General: Job tools: xtjobs, search_alps, qmap

**Tool**: `pkg/xts/bin/xtjobs,search_alps` , `pkg/pbstools/bin/qmap`

Probably most sites already wrote tools like the following, but just in case:

- Identify resources used by running jobs:

```
pingo1: xtjobs -h

Usage: xtjobs [-options] # reparse apstat -rvv and qstat -a

Options:
   -h         # this usage information
   -g nodes   # nodes greater or =,  value: 0
   -l nodes   # nodes less or =,     value: 0
   -R minutes # time remaining <=,   value:
   +R minutes # time remaining >=,   value:
   -d days    # older than days,   current: 0
   +d days    # newer than days,   current: 0
   -n nid     # specific nids,     current:
   -u user    # specific users,    current:
   -a apid    # specific apids,    current:
   -r resid   # specific resids,   current:
   -j jobid   # specific pbs jobs, current:
   -o jobname # specific jobnames, current:
   -q queue   # specific queues,   current:
   -Q         # do not display headers
   -s         # save files
   -z file    # redisplay
   -f fields  # apid,job,userid,nodes,pes,queue,remain

Comma separate above and dash separate nid ranges.
```

```
Available fields are:
  resid,apid,job,userid,nodes,pes,memory,ppn,depth,queue,name,
  walltime,elapsed,remain,date,list
Use '-f all' or '-f ""' for all fields.
Use '-' to exclude a field, e.g., '-f all,-list'.
Use '+' to add fields to defaults, e.g., '-f +list,apid'.

pingo1: xtjobs +R 1440 # jobs with 24+ hours remaining...
#    ApId job-pbs Userid  Nodes   PEs Queue    Remain
#    ---- ------- ------  -----   --- -----    ------
   577285  191149 pharmer     1     8 standard  37:57
   579767  191403 wrench      1     1 standard  65:18
   579769  191404 wrench      1     1 standard  65:18
   579770  191405 wrench      1     1 standard  65:19
#                 Totals      4    11 4 jobs

pingo1: xtjobs -g 64 # jobs using 64+ nodes...
#    ApId job-pbs Userid  Nodes   PEs Queue    Remain
#    ---- ------- ------  -----   --- -----    ------
   580386  191466 jangle     64   512 challeng  03:28
   581533  191566 pontoon    64   512 standard  00:53
   581817  190819 debrites   64   512 backgrou  07:43
   581828  191616 lexars     64   512 standard  11:47
#                 Totals    256  2048 4 jobs

pingo1: xtjobs -n 514 -f all # job using nid00514
#ResId   ApId job-pbs Userid Nodes PEs memory ppn dpt Queue   JobName WallTm Elapse Remain     Date.hhmmss nid-list
#-----   ---- ------- ------ ----- --- ------ --- --- -----   ------- ------ ------ ------     ----.------ --------
  2688 581499  191590 mortons   32 256 4000MB   0   1 weather wrfrun.p 11:15  02:32  08:43 20100416.114327 512-543
```

- Identify what jobs ran on a node previously:

```
sdb:search_alps -h
search_alps [-h] [-j id | -a id | -r id] [-l] [-f "format_string"] \
  [-cabinets cabs | -n nids] apsched_log

     -h, --help   show this help dialog
     -n nodes(s), --nids node(s)
     -c nodes(s), --cabinets node(s)
     -j id,       --jobid id
     -a id,       --apid id
     -r id,       --resid id
     -l,          --lookup looks for uid in /etc/passwd
     -s,          --summary add a summary of core count and node count
     -f , --format
                        %f = file
                        %t = start time
                        %e = end time (attempts to get date from filename)
                        %a = apid
                        %j = pbs_jobid
                        %r = resid
                        %p = pagg
                        %u = uid
                        %c = command
                        %n = nids
                        %m = matching nid name

Alps Logs:
   sdb: /var/log/alps/apsched????

Additional Information:
   Application: search_alps
   Version: 1.0.2

sdb: search_alps -n nid00590 -l /var/log/alps/apsched0415 -f "%t %e %j %c\n"
08:34:52 04/15 08:41:28 191280 testSAmodel_INT_T_solve_trip_norotor
```

```
09:01:01 04/15 17:23:49 191265 wrf.exe
17:25:59 04/15 17:28:39 191389 main.x
17:28:42 04/15 23:35:11 191390 cam
```

- What is running on the system?
  Note, `xtnodestat` provides similar capabilities but `qmap` is a common tool for all our systems.

```
pingo1: qmap -m # omit the node map (too long)
qmap:  Fri Apr 16 14:23:43 2010

Nonrunning Jobs
Id Job Id  User      Queue        Proc Stat   Waiting  Wall Time  Job Name
1  190820  debrites  background    512  H  3+09:04:31   08:00:00  b2
2  190821  debrites  background    512  H  3+09:04:28   08:00:00  b2
3  191617  lexars    standard_lg   512  Q    00:21:27   11:59:00  rungap1024piui

Running Jobs
Id Job Id  User      Queue         Proc Nodes  Remaining  Wall Time      Job Name
A  190819  debrites  background     512    64   07:34:07   08:00:00            b2
B  191149  pharmer   standard_sm      8     1 1+13:44:21 3+12:00:00          wind
C  191403  wrench    standard_sm      1     1 2+17:05:38 3+12:00:00        serial
D  191404  wrench    standard_sm      1     1 2+17:06:18 3+12:00:00        serial
E  191405  wrench    standard_sm      1     1 2+17:06:35 3+12:00:00        serial
F  191466  jangle    challenge_lg   512    64   03:18:18   16:00:00    open_20mil
G  191509  earwax    standard_lg     64     8   01:57:05   10:00:00 myproject.pbs
H  191513  earwax    standard_lg     64     8   02:02:26   10:00:00 myproject.pbs
I  191566  pontoon   standard_lg    512    64   00:43:12   03:00:00           w21
J  191577  mikilisi  challenge_sm    14     2   20:12:16   24:00:00        rbands
K  191578  weperg    standard_lg    224    28   14:44:11   16:00:00    HCO2--HCOO
L  191580  sing      background     128    16   04:35:05   08:00:00      r268.bat
M  191590  mortons   weather        256    32   08:35:43   11:15:00     wrfrun.pbs
N  191596  sing      background      64     8   06:48:24   08:00:00      r377.bat
O  191602  destires  background      64     8   06:49:00   08:00:00     pscript662
P  191607  sing      background      64     8   06:47:51   08:00:00      r378.bat
Q  191608  destires  background      64     8   07:16:36   08:00:00     pscript661
R  191611  grustol   standard_lg     32     4   03:20:43   04:00:00      ar_uwbase
S  191613  grustol   standard_lg     32     4   03:31:08   04:00:00      ar_uwemis
T  191616  lexars    standard_lg    512    64   11:37:52   11:59:00 rungap1024piu
U  191618  jhuatuci  standard_sm     16     2   15:39:31   15:59:00         mix-2
V  191619  cvijanov  background      32     4   06:56:58   07:10:00 q_sthrh09_ctr
W  191620  grustol   standard_lg     32     4   03:20:03   03:30:00   lfcst_gps25
X  191622  grustol   standard_lg     32     1   03:50:39   04:00:00      ar_gps15
```

System Administration:

1. **General: Establish practices for managing system changes**

   **Documentation**: ConfigFiles -- push -- chk_sanity -- PingoFiles -- PingoMachines
   **Tool**: `pkg/push` , `pkg/chk` , `pkg/edconfig` , `pkg/uak/src/mkbko,ckbko`
   **Alternative:** various

   A primary concept for managing systems is to ensure any customization to system configuration files is retained over upgrades, reinstalls, and is consistent across nodes. To accomplish this, ARSC adopted a practice of never editing files directly in system space. Modifications are made in the `"ConfigFiles"` directory, which for our Cray is named `/var/local/CrayFiles`. Files are mapped by path. For example, the `fstab` for the boot node is:
   `/var/local/CrayFiles/etc/fstab/fstab.boot001`.
   Reference the ConfigFiles documentation for additional information.

   The managed or audited files are identified by the presence of a `backout` directory. By use of the `edconfig`, mkbko, and ckbko tools history is retained for these files. The auto-built CrayFiles.list identifies the managed files. The managed nodes are identified in machines.cray which for our Cray is all the service nodes, the SMW, and the eslogin nodes. The push command can execute commands on all or subsets of nodes, copy or back-copy files in general, or push update `CrayFiles` onto nodes. For CLE, push required a modification to use `xtspec` for service nodes. The chk_sanity script runs nightly and on demand after any system maintenance to validate the `CrayFiles` are correct on all nodes.

   ARSC has used this process for over a decade. Currently a single management system manages `LinuxFiles` for a 80 workstations and servers and `SunFiles` for 50 Solaris servers. Each cluster or platform has its own `"ConfigFiles"`.

   Throughout this paper there are references to `CrayFiles/...` examples which are included under `CrayFiles.tgz`.

2. **General: Develop reusable tools**

**Documentation**: uals -- Software
**Tool**: `pkg/uals` , `pkg/uak`

As well as common practices for managing systems, reusable tools help in managing a variety of platforms. We tend to do this in general, but two which even pre-date existence of ARSC at the Univesity of Alaska merit mention.

The uals package started with my initial UNIX involvement when I discovered `'ls -l'` used varying syntax which makes parsing by fields awkward. Originally based on OSF1 source, it was reworked in 1997 from the GNU fileutils-3.12 `ls` source so it could be made freely available. Since I had the tool, anything I needed for identifying files was added over the years. The current version has the following extensions:

```
login1: uals --help | grep '^+'
+ Indicates University of Alaska (uals) addition|modification of GNU
+ -e, --lc_time=FORMAT    time format, defaults to '%y%m%d.%H:%M'
+ -E, --time=mtime|modify show and sort by modify|mtime|write time
+     --full-time         list full date and time: yyyymmdd.hhmmss
+ -g                      include group in display (default)
+ -h, --help | --?        display this help and exit
+     --device            display device (hex) owning file
+     --input             read file names from stdin
+     --list              list all directories in file path
+ -M, --o_mode            display permissions as octal (default)
+     --x_mode            display permissions 32bit: xfffff.o7777`
+     --alpha-uid-gid     default, can use with -n
+     --both-uid-gid      both numeric and alpha UIDs and GIDs
+ -o                      include owner in display (default)
+ -O, --old-time          use 'old' ls time format (default yymmdd.hhmm)
+ -P, --a_mode            display permissions as alpha
+     --mark              print ? for punctuation (not .-_)
+     --mount | --xdev    but do not walk directories on other disks
+ -s, --bytes             print byte size of each file
+     --sum               sum -r each file
+ -T                      display *only* totals
+ -V, --version           output version information and exit
+ -W, --tabsize=COLS      assume tab stops at each COLS instead of 8
+ -y, --fields            fields to display: +zHDiMtmlogKksbrEduacfZT*A
+ -Y, --blocks            show size in blocks not bytes
+     --blksize=size[kmg] show size in blocks calculating blocks
+ -z                      do not display directory header line
+ -Z                      do not display totals
+ -*, --verbose           all available file information
+     --size=SIZE         files with size greater than SIZE
+     --s_gt=SIZE         files with size greater than SIZE
+     --s_lt=SIZE         files with size less than SIZE
+     --type=t            files of  specified type(s)
+     --t_ne=t            files not specified type(s)
+     --newer=TIME|file   files newer than time|file
+     --new_m=TIME|file   files newer than time|file
+     --new_c=TIME|file   files change time newer than time|file
+     --new_u=TIME|file   files access time newer than time|file
+     --stale=TIME|file   files|links|dirs candidates for purging
+     --older=TIME|file   files older than time|file
+     --old_m=TIME|file   files older than time|file
+     --old_c=TIME|file   files change time older than time|file
+     --old_u=TIME|file   files access time older than time|file
+     --name=PATTERN      filenames matching PATTERN
+     --n_ne=PATTERN      filenames not matching PATTERN
+     --path=PATTERN      path/files matching PATTERN
+     --p_ne=PATTERN      path/files not matching PATTERN
+     --d_eq=PATTERN      walk directories matching PATTERN
+     --d_ne=PATTERN      walk directories not matching PATTERN
+     --link_gt           files with more than specifed links
+     --inum              files with specified inode
+     --group=group       files belonging to the group
+     --g_ne=group        files not belonging to the group
```

```
+     --g_gt=gid          files with higher gid
+     --g_lt=gid          files with lower gid
+     --user=userid       files belonging to the userid
+     --u_ne=userid       files not belonging to the userid
+     --u_gt=uid          files with higher uid
+     --u_lt=uid          files with lower uid
+     --nogroup           files belonging to unknown group
+     --nouser            files belonging to unknown userid
+     --mode=mask         files matching mode mask
+     --m_ne=mask         files not matching mode mask
+     --acl               files with an acl
+     --or                or (vs. default and) access filters
+     --show_filter       display filters
```

Long ago ran out of letters in the alphabet for all the options. The uals command is imbedded in many of our maintenance scripts and there are multiple references in this paper. The find-like capabilities are especially useful to allow NOPASSWD `sudo` access to `uals` which is not appropriate for `'find -exec'`.

The `uak` package (see Software) is a collection of tools which pre-dates my UNIX experience needing a common parser for Vax/VMS and IBM/MVS utilities. The tools are somewhat disparate, they were collected in one package because maintaining many packages is more overhead than I could generally tolerate. The oldest UNIX tools are uaklogin which merges `ps` and `utmp` and uakpacct which reads `pacct` files with a number of selection and reporting options (also mentioned CUG Summit 2000 paper "Reporting Unicos and Unicos/mk System Utilization"). Most of these are written in C, a few are scripts. Several are mentioned in this paper: uakpacct, ckbko, mkbko, rpmchk, and modchk.

Above tools have been available via ftp://ftp.alaska.edu/pub/sois/ for over a decade. They are also included in this paper in `admpkg.tgz`. ARSC installs tools under `/usr/local.adm/bin,sbin,man,etc` with a symlink `'/usr/local/adm -> /usr/local.adm'`. A number of the scripts included in `admpkg.tgz` do code these PATHs since we generally do not write scripts to assume PATH.

### 3. General: Manage/log root access

**Technique:** sudo
**Example**: `CrayFiles/usr/software/adm/etc/sudoers`

ARSC presently manages 50+ Solaris servers, 80+ Linux workstations and servers, a 400+ node Linux cluster, a 44 node Scyld Cluster, and the Cray XT systems (production and TDS). With common tools and practices System Administrators can fill in across platforms. This does mean more than a dozen people need root authority. Having more than one person operating as root can get out of hand quickly. Having dozens of windows open on multiple platforms could easily lead to accidents if root shells were left open. The general rule at ARSC is avoid operating as root **[*]**, use `sudo` instead. Besides being safer, `sudo` logs actions. On systems which have root passwords for emergency purposes, they are locked in the safe and not used. The majority of our cron entries do not run as root. We use less privileged UIDs with appropriate commands `sudo`-authorized. Besides system administrators, user support staff also need a collection of commands with elevated authority which can be configured in `/etc/sudoers`. ARSC builds sudo from source (discussed later) and for security sets the binary `'chmod 4110'` and `'chgrp sudoers'`. The `/etc/sudoers` file can become complex, see our `CrayFiles` as an example.

On some platforms we build multiple versions of `sudo`

- sudo.krb: kerberos+securid authentication
- sudo.ace: securid authentication
- sudo.pwd: password authentication

The `sudo.krb` is preferred, but the `sudo.pwd` exists for systems (like `boot001`) which have no network connectivity to the `kdc` and on systems which may need `sudo` with network disabled.

**[*]** - there are a few circumstances where operating directly as root is appropriate, such as during CLE upgrades. Our preferred manner of doing this is:

```
user@smw: script Rxxxxx_activity.log # use script to log activities...
user@smw: sudo su - root
root@smw: # do whatever is necessary...
root@smw: exit # as soon as reasonable, attach log to change/problem ticket...
```

### 4. General: Eliminate passwords where possible

**Technique:** krb5, securid
**Alternative:** various

Many sites consider static passwords insufficiently secure and use token based authentication in some form. Our sponsors require Kerberos+SecurID or a PKI card, reader, and access code for authentication. Most static passwords have been eliminated from our systems. As noted above in the sudo discussion, root passwords are locked in a safe where they are still retained for emergency use. For our XT5 login nodes, there are no static passwords. For System Administrators requiring elevated `sudo.pwd` authority, static passwords are retained on boot and SMW nodes.

5. **General: Baseline configuration, other assets, and peers**

   ARSC operates multiple systems, presently three independent clusters as well as many other servers and a large collection of workstations. We are also one of six HPCMP DoD Supercomputing Resource Centers (DSRC). Much of this paper focuses on tools we utilize center wide and not specifically on our XTs. Equally important is that our users move between systems and between centers. It is extremely important for their sanity, and our survival, that common practices are used both on our systems and our peer sites. The HPCMP addresses this with a Baseline Configuration Team (BCT) which tries to define commonality for all compute resources. At times this can only define the lowest common denominator as some clusters have remarkably different use characteristics, but is still important. The BCT is supplemented with various Communities of Practice (such as CrayCoP and a Workload Management CoP) which share expertise on specific systems or capabilities. Undoubtedly, Teragrid and the European DEISA must also have some oversight and coordination entities, and we all try to share information and practices via CUG, *right?*

6. **General: Responses for bad computer support advice**

   **Tool**: `pkg/uak/src/sik.c`
   **Documentation**: sik

   This is an irrelevant task to test if anybody is reading. The Vendor Response Kit (aka, Shakespearean Insult Kit), can be used when responding to vendor support organizations which give very bad advice. It is irrelevant, of course, because Cray would *never* do that. :)

```
login1: sik 5
Thou gleeking sheep-biting puttock.
Thou qualling spur-galled lewdster.
Thou artless rough-hewn hugger-mugger.
Thou mangled dismal-dreaming lout.
Thou surly elf-skinned flax-wench.
```

---

## Tasks - Install (top -- intro -- index -- general -- install -- CNL -- ongoing -- conclusion)

The following tasks are predominantly one-time only, either during installation or later.

1. **Install: Understand boot disk layout**

   **Documentation**: xt_fdisk
   **Example**: `CrayFiles/etc/sysset.conf`

   Since the SMW, the boot node, and the sdb share the boot raid, it is *critical* to understand which is mounting what. If two nodes ever mount the same filesystem there will be corruption. Been there... we once forgot to shutdown boot001 prior to an upgrade on the TDS (*thank goodness for backups*). The Cray manuals have a good table of the layout, but you still need to make additional filesystems for `xthotbackup` and configuring `/etc/sysset.conf`. Note in xt_fdisk there is unused space at the end of each partition. The Cray partitioning procedure builds filesystems base 1000 instead of 1024 leaving unused space.

2. **Install: Resolve uid|gid collisions (cluster, external NFS)**

   **cron**: sysmon@dispatch chk_passwd (daily) (manual initially)

   The SMW and boot node (SNL) are built by separate parts of Cray which unfortunately pay no attention to synchronizing UIDs and GIDs. This is a low level risk, but really should be resolved if any NFS is to be used between the SMW and SNL. In our case, the `CrayFiles` are stored on the SMW so they are available with the XT down and exported to the boot node for actual system maintenance. In addition to syncing within the cluster, there may be a need for some site synchronization if NFS without uid mapping is used. The process of syncing is manual and more easily accomplished when a system is new:

   - identify the conflicts via find on each filesystem
   - carefully chown|chgrp each, one `uid` | `gid` at a time
   - update `/etc/passwd` and `/etc/group`
   - reboot and validate
   - run unowned file check

3. **Install: Mount most filesystems nosuid,nodev**

**Example**: `CrayFiles/etc/fstab`
**Example**: `CrayFiles/opt/xt-boot/default/etc/boot.xt`

Mounting non-system filesystems nosuid,nodev reduces the risk of root kits, especially on XT systems where system filesystems are read-only to users. In CLE there are several places to address this:

- In `/etc/fstab` for regular filesystems.
- In `/opt/xt-boot/default/etc/boot.xt` for `/var` and `/tmp`.
  *When customizing `boot.xt` establish a means, like `CrayFiles`, to ensure any Cray upgrades to the file do not regress your changes and to ensure you incorporate any Cray changes.*
- CNL use read-only rootfs and o-w /tmp, and /var
- CNL mount lustre nossuid,nodev

Examples:

```
smw: cat /var/local/CrayFiles/etc/fstab/fstab.login
ufs:/ufs                       /ufs              nfs rw,nodev,nosuid,soft,bg         0 0
boot:/export/localadm          /import/localadm  nfs ro,nodev,nosuid,soft,bg         0 0
sqfs-s:/export/seawolf/u1      /import/archive/u1 nfs rw,nodev,nosuid,soft,bg,noauto  0 0
sqfs-s:/export/seawolf/u2      /import/archive/u2 nfs rw,nodev,nosuid,soft,bg,noauto  0 0
sqfs-s:/export/seawolf/projects /import/projects  nfs rw,nodev,nosuid,soft,bg,noauto  0 0
adm1nfs:/export/support        /import/support   nfs rw,nodev,nosuid,soft,bg,noauto  0 0

smw: cd $V/CrayFiles/opt/xt-boot/default/etc/boot.xt
smw: ckbko -2 -diff boot.xt.template | grep '^<'
< # History
< # -------
< # 2009-08-13 kac reduce size of /var/[tmp|lock|run], /tmp, and /dev (R29196)
< # 2009-06-11 jvw added nodev,nosuid to /tmp, & /var/[tmp|lock|run] R27875
<                       mount -o nodev,nosuid,size=512m -t tmpfs none /var/lock
<                       mount -o nodev,nosuid,size=512m -t tmpfs none /var/run
<                       mount -o nodev,nosuid,size=512m -t tmpfs none /var/tmp
<       mount -o nodev,nosuid,size=512m -n -t tmpfs tmpfs /tmp
<       rc_status -v -r
<       echo -n "Re-mounting /dev (nosuid,size=512m)"
<       mount -o remount,nosuid,size=512m /dev
```

4. **Install: Reduce exports (no global, ro where appropriate)**

**Example**: `CrayFiles/etc/exports`

Cray XT systems are delivered with "unsafe" NFS exports. Easy to fix, routine system setup:

```
smw: cd $V/CrayFiles/etc/exports; ls
backout exports.boot001 exports.nid00003 exports.piman exports.template

smw: grep -v ^# exports.boot001
/rr   192.168.0.0/255.255.0.0(ro,no_root_squash,sync)
/snv  192.168.0.0/255.255.0.0(rw,no_root_squash,async,no_subtree_check)
/export/localadm 192.168.0.0/255.255.0.0(ro,no_root_squash,async)
/export/boothome 192.168.0.0/255.255.0.0(rw,no_root_squash,async) \
  10.3.1.1(rw,no_root_squash,sync)

smw: grep -v ^# exports.nid00003
/ufs 192.168.0.0/255.255.0.0(rw,no_root_squash) \
  pingob-s(rw,no_root_squash,async) pingoc-s(rw,no_root_squash,async)
```

In addition, `nfsserver` is enabled across SNL by default and should be disabled as an unnecssary service:

```
boot: push -s cmd -eqf chkconfig nfsserver
          ogman:nfsserver  on
         boot001:nfsserver  on
       nid00003:nfsserver  on
       nid00004:nfsserver  on
       nid00007:nfsserver  on
       nid00008:nfsserver  on
       nid00011:nfsserver  on
```

```
boot: sudo xthowspec -r /rr/current /etc/init.d/rc3.d/S15nfsserver
node:0:/etc/init.d/rc3.d/S15nfsserver:default
node:3:/etc/init.d/rc3.d/S15nfsserver:node    # <-- Note !!!
node:4:/etc/init.d/rc3.d/S15nfsserver:default
node:7:/etc/init.d/rc3.d/S15nfsserver:default
node:8:/etc/init.d/rc3.d/S15nfsserver:default
node:11:/etc/init.d/rc3.d/S15nfsserver:default
class:ost:/etc/init.d/rc3.d/S15nfsserver:default
class:service:/etc/init.d/rc3.d/S15nfsserver:default
class:login:/etc/init.d/rc3.d/S15nfsserver:default
default::/etc/init.d/rc3.d/S15nfsserver:default

boot: sudo xtopview -m "chkconfig nfserver off" -e "chkconfig nfs off"
***File /etc/init.d/.depend.boot was MODIFIED
***File /etc/init.d/.depend.start was MODIFIED
***File /etc/init.d/.depend.stop was MODIFIED

boot: push -s cmd -eqf chkconfig nfsserver
          ogman:nfsserver  on
         boot001:nfsserver  on
        nid00003:nfsserver  3
        nid00004:nfsserver  off
        nid00007:nfsserver  off
        nid00008:nfsserver  off
        nid00011:nfsserver  off
```

Unfortunately, `nfsserver` does get re-enabled with every CLE upgrade.

### 5. Install: Reduce memory filesystems (default 1/2 memory)

**Example**: `CrayFiles/opt/xt-boot/default/etc/boot.xt`

Linux, not just CLE, will default memory filesystems as half of system memory. That is a bad idea on shared systems allowing a careless user to consume memory by dropping large files in `/tmp` or over-using `/dev/shm`. A number of utilities use $TMPDIR if defined, so pointing TMPDIR to `/workdir/$USER` is a good idea. In addition, the memory filesystems can be remounted smaller:

```
smw: cd $V/CrayFiles/opt/xt-boot/default/etc/boot.xt
smw: grep size= boot.xt.template
                        mount -o nodev,nosuid,size=512m -t tmpfs none /var/lock
                        mount -o nodev,nosuid,size=512m -t tmpfs none /var/run
                        mount -o nodev,nosuid,size=512m -t tmpfs none /var/tmp
        mount -o nodev,nosuid,size=512m -n -t tmpfs tmpfs /tmp
        echo -n "Re-mounting /dev (nosuid,size=512m)"
        mount -o remount,nosuid,size=512m /dev
```

If CNL o-w /tmp is set you can skip doing this on compute nodes.

### 6. Install: Audit/secure system access points

**Technique:** tcpwrappers
**Example**: `CrayFiles/etc/hosts.allow,hosts.deny`
**Alternative:** iptables, network ACLs

Open services and ports should be appropriately protected. Using tcpwrappers, the strongest approach is to implement a general deny rule after specifically allowing required access, for example:

```
smw: cd $V/CrayFiles/etc/hosts.allow; ls
backout               hosts.allow.esl    hosts.allow.login
hosts.allow.nid00003  hosts.allow.piman  hosts.allow.template

smw: grep -v ^# hosts.allow.login
kshd klogind sshd ftpd telnetd  : KNOWN UNKNOWN ALL
sshd-adm : 199.165.84.   199.165.85.  199.165.86.     172.16.
sshd-xt  : 192.168.      172.16.1.237 172.16.1.234
in.rshd  : 192.168.0.1   172.16.1.237 172.16.1.234    172.16.1.130
```

```
smw: cd $V/CrayFiles/etc/hosts.deny; ls
backout  hosts.deny.template

smw: grep -v ^# hosts.deny.template
ALL : KNOWN UNKNOWN ALL
```

Note, the allow rules do vary by node type. The 'KNOWN UNKNOWN ALL' is not really redundant as it allows host names to be logged. In the above **rshd** has been enabled within the cluster and to storage servers. Within the cluster this provides emergency access if `sshd` ever breaks.

For management nodes (at least), network ACLs and `iptables` should also be used to secure entry.

### 7. Install: umask management

**Technique:** policy, profile

Linux, by default, is distributed in a permissive manner with a default `umask 022` permitting other+read of files. The policy of our sponsors is 037 and ARSC policy is 077. Not everybody agrees with those policies, but the premise is users can change it in their login profiles and by default files should not be public. Where this is implemented varies by UNIX derivative or Linux distribution, for CLE (SLES):

```
login1: sudo grep 'umask 077' /etc/* # managed via CrayFiles
/etc/csh.cshrc:umask 077        # ARSC set
/etc/csh.login:umask 077        # ARSC s/022/077/
/etc/csh.login.local:umask 077
/etc/profile:umask 077  # ARSC
/etc/profile.local:umask 077
```

The `umask` should also be set for `sudo` either with configure option `'--with-umask=077'` (we build our own) or `sudoers` specifying `'Defaults umask=077'`.

For system administrators, this can present some issues as there are numerous poorly written install procedures which rely on the permissive umask. Cray has `xtopview` establish `'umask 022'`, but when installing RPMs on boot or smw you may need to:

```
sudo ksh -c "umask 022; rpm -i application.rpm"
```

### 8. Install: Eliminate unnecessary services: xinetd, chkconfig

**Example**: `CrayFiles/etc/xinetd.d/*`
**Example**: `CrayFiles/var/local/logs/CF/chkconfig.txt`

For `xinetd` turn off unnecessary services:

```
login1: cd /etc/xinetd.d; grep disable *
chargen:     disable = yes
chargen-udp: disable = yes
cvs:         disable = yes
daytime:     disable = yes
daytime-udp: disable = yes
echo:        disable = yes
echo-udp:    disable = yes
eklogin:     disable = no
fam:         disable = yes
ftp:         disable = no
krsh:        disable = no
netstat:     disable = yes
rexec:       disable = yes
rlogin:      disable = yes
rsh:         disable = no
rsync:       disable = yes
servers:     disable = yes
services:    disable = yes
systat:      disable = yes
telnet:      disable = no
time:        disable = yes
time-udp:    disable = yes
```

This will vary by node type. The allowed services should be appropriately protected via tcpwrappers and the state of `xinetd` be audited with chkconfig. Making a pass through `'/sbin/chkconfig --list'` for other unnecessary services is also a good idea.

Our sponsors also require `/etc/xinetd.conf` and `/etc/xinetd.d/*` be mode `o-rX`.

### 9. Install: Eliminate unnecessary services: rpm -e

**Technique:** manual, rpmreduction
**Example**: `CrayFiles/var/local/CF/rpm.txt`

Cray, unfortunately like many vendors, allows Linux to install a large collection of unnecessary RPMs. The reason is that one person's unnecessary may not match another, but this results in many things which generally are inappropriate. Those with `suid` binaries or other+write directories are targets for removal as is anything which is never going to work under CLE. Cray made a good pass at removing really inappropriate stuff with the `rpmreduction` added with CLE 2.2. I suspect they did this as much for their benefit as ours because it cut down the number of SLES security notices they needed to follow and reduced the volume of field notices... an excellent idea. Under CLE 2.0, AWE made an aggressive cut at RPM reduction and shared that information to help us identify what to cull under 2.1 and 2.2.

A quick scan of our Change Management records show we removed at least:

```
inn gup samba* qpopper i4l* capii4linux wwwoffle apache2* ircd squid* mgetty sax2
SDL sox suck talk tango thineramik tn5250 php5 radius mozilla-ns* quagga ppp
wvdial minicom
```

Some of these were subsequently removed by CLE 2.2 `rpmreduction`. The list above is likely not complete, see `CrayFiles/var/local/logs/CF/rpm.txt` and audit rpm state for our current state. After `rpmreduction` we had to reinstall `ghostscript` and `curl-devel` due to user requirements.

We also remove `'bind bind-chrootenv bind-devel'` as unnecessary services. However, CLE upgrades consistently put them back for some reason. This will redefine a socket in `/etc/sysconfig/syslog` which then needs to be commented out, the syslog-ng config file needs to be rebuilt, and `syslog-ng` restarted. To remove `bind` is the right thing to do, but if you are not overly paranoid you might ignore it.

We also remove `sudo` and `ssh` for custom builds...

### 10. Install: Comments on non-Cray ssh and sudo

Our sponsors require, for good reasons, custom builds of `sudo` and `ssh`.

For `sudo`, besides the requirements of our authentication method (kerberos+securid), this allows use of the configure option `'--with-secure-path=${SECP}'` which is a *good idea*. Cray also does a custom rpm for sudo, probably because the security feature of `sudo` which prohibits `/etc/sudoers` being a symlink. Unfortunately, CLE upgrades reinstall `sudo` (like `bind`). We build to a private location so they could co-exist, but it is an unnecessary `suid` binary. In the spirit of protecting `suid` binaries we also `'chmod 4110'` and `'chgrp sudoers'` the binary for authorized staff since `sudo` is also subject to possible exploits.

For `ssh` this is more complicated. Our peer HPCMP/DoD sites each handle this differently. CLE operations rely on `ssh` to boot the system. For this reason, one site chose (under a DoD waiver) to only replace `ssh` under shared root for the login nodes. ARSC replaced it on SMW, boot, and SNL. Cray upgrades do not put it back (which is good). However, it does require carefully building the imbedded PATH as Cray boot procedures do not use explicit binary path specification. Specifically, we could not boot the cluster after one `ssh` upgrade.

We run three instances of `sshd` for user login, for site-wide automation, and for cluster operations:

1. **`sshd`** for user login:
   ```
   Port  22
   ListenAddress 199.165.85.217
   ListenAddress ::
   ```
2. **`sshd-adm`** for site-wide automation:
   ```
   Port  30
   ListenAddress 199.165.85.217
   ListenAddress 172.16.1.238
   AllowUsers backup@admin1.arsc.edu sysmon@admin1.arsc.edu ...
   ```
3. **`sshd-xt`** for cluster operations:
   ```
   Port  22
   ListenAddress 192.168.0.4
   ListenAddress 172.16.1.238
   AllowUsers  *@boot001 *@nid00003 *@ogman-s.arsc.edu *@ogboot-s.arsc.edu ...
   ```

The same binary (symlink) is used for all three. The complete `sshd_config*` files are included in `CrayFiles/usr/software/pkg/ssh/*/etc/sshd_config*`.

### 11. Install: sdb and boot node on non-cluster networks

This task really covers what is gained and risked by placing management nodes (SMW, boot, sdb) on non-cluster networks.

The sdb typically ships without a NIC. However, PBS needs an ethernet port for flexlm license configuration. NICs are cheap, why not? Besides PBS, a NIC allows the sdb to be a gateway to external license servers (if on a public network) which can be useful. Also, ARSC has a private storage network so the 2nd port allows access to the back-end HSM which is useful for archiving `/ufs` (PBS) and syslog.

The boot node has an unused port on its NIC. We connect this to the private storage network which allows both archive of `/snv` and complete backup of boot and shared root. Regular backups, besides just `xthotbackup` prior to upgrades, are an *Extremely Good Idea*.

For the SMW, one simply has to connect it to some network so administrators can reach it. We connect to both the storage network (archival and backup) and an internal network.

The benefits are clear, what are the risks? All management systems should be protected by network ACLs. However, we run nessus scans within that protection as required by our sponsors. We found nessus scans were able to hang PBS `altair_lm` and kill `apsched` blocking jobs. Before we disabled mazama we discovered nessus put `mzlogmanagerd` into an infinite loop debilitating the SMW. Open ports are discussed later, but as Matt Ezell with NICS pointed out via Xtreme (subsequent Cray FN5653), the `erd` ports 7002,7004 have inadequate protection and are a security risk which argues for iptables protection even on the private cluster networks.

### 12. Install: Avoid ipforwarding

**Technique:** disable mazama
**Tool**: `pkg/xts/bin/no_mazama.ksh`

Enabling ipforward on any system has ramifications. The enabled node effectively becomes a gateway and unless restricted via iptables or some other means may permit inappropriate actions. The CLE 2.2 implementation of mazama (also known as CMS) enables ipforward on the boot node so the SMW and sdb could communicate. While the communication is a good idea, for us the ipforward is unnecessary with the NIC in our sdb. Since there were no restrictions, the ipforward also allows any cluster node to communicate through the boot node... not such a good idea. Cray is revising CMS for 3.1 to listen on a port through the boot node. However, as noted with `erd` above and later for open ports, this may not be the end of the story.

Under CLE 2.2 we could see no benefit to mazama (CMS). We disabled it. Unfortunately, every CLE|SMW upgrade feels it should re-enable mazama. We finally got tired of this and hacked out the `no_mazama.ksh` script.

Cray is enhancing CMS significantly with 3.1, the decision to disable may merit review.

### 13. Install: Enable process accounting (and roll files)

**Technique:** configuration, logrotate, uakpacct
**Example**: `CrayFiles/etc/init.d/acct`
**Example**: `CrayFiles/etc/logrotate.d/acct`
**Tool**: `pkg/uak/src/uakpacct.c`
**Documentation**: uakpacct

I have been surprised by the number of systems which either do not enable process accounting or accept the UNIX default to delete `pacct` files after the rudimentary nightly reporting. The `logrotate.d/pacct` with several Linux distributions disable `pacct` or lose records as they break the link to the inode prematurely or fail to 'accton' to the new inode. Real accounting comes from reducing the PBS logs, what `pacct` files provide is a record of what happened which may not make logs: when a daemon died, what signal or code a process exited with, what resources a process consumed (was it a memory or cpu hog?), whether a process used suid, and with current implementations pid and ppid. The records are far from perfect: limited command names and inability to retain transient ppid relationships, but the records can be a primary source for problem determination especially for security events.

Because many under value `pacct`, analysis tools been sparse. The uakpacct program started early in my UNIX involvement with OSF1 in mid-90s and has gone through every derivative I have managed since. A couple examples:

```
scyld: uakpacct -r hum,cpu,mem -mem 1000000 -u csauser \
  -f /var/account/pacct-20100330 # OoM on 24G systems
#End_Date/Time Start_hh:mm:ss :Userid  :Command :Flg:Exit:   CPU : Avg.Mem:
#-------------       -------- :------  :------- :---:----:   --- : -------:
03/29 11.12.09       11.08.17 :csauser :perl    :020:  -9: 708.0s:  25.08G:

boot: uakpacct -pid 35397 -c ferret -long -f 32.pacct-20091107
```

```
#   Date Time  :Userid :Group :tty(hex):Command :Flg:Exit:UserCPU:SYS_CPU:Elapsed: Avg.Mem:
#   ---- ----  :------ :----- :--------:------- :---:----:-------:-------:-------: -------:
20091106@104926:#2950  :#4491 :0000880e:ferret  :000:   0: 782.1s: 251.9s:2292.2s:   6.06G:
20091106@112854:#2950  :#4491 :0000880e:ferret  :020:  -9: 742.4s: 246.9s:3069.6s:   6.06G:
20091106@133621:#2950  :#4491 :0000880e:ferret  :000:   0: 709.1s: 246.6s:1132.2s:   6.06G:
20091106@135557:#2950  :#4491 :0000880e:ferret  :000:   0: 757.1s: 251.3s:1455.3s:   6.06G:

iceberg1: uakpacct -c tod -f /var/adm/Spacct1.0521
#End_Date/Time Start_hh:mm:ss  Userid   Command  Flg Exit    CPU
#-------------       --------  ------   -------  --- ----    ---
05/20 08:55:57 05/18 13:18:21  root     tod      021   -9   10.9s
```

First two examples are memory abuse, the second one shows only one of four "events" actually got the OoM sigkill (9). The third example identifies when and why (sigkill) a daemon died.

On average I only need to look at `pacct` records once or twice a month, but they are invaluable when needed. The files should be rolled daily and archived securely. If one is ever seriously attacked, expect `pacct` files to be tampered with or destroyed.

### 14. Install: Raise maximum pid

**Example**: `CrayFiles/etc/sysctl.conf`

The default max pid is unreasonably low and can easilty be raised:

```
smw: cd $V/CrayFiles/etc/sysctl.conf
smw: ckbko -6 -sdiff *template

#2009-02-04 kac set pid_max=65535, default 32768    <
kernel.pid_max = 65535                              <
```

For a Scyld cluster we implemented recently, we set `'pid_max = 524288'` due to the shared process space of the compute nodes.

This is one of several tricks we borrowed from Nick Cardo at NERSC when he graciously visited ARSC, *in winter*, to provide insights as we were implementing our XT.

### 15. Install: Establish external system trust relationships

**Technique:** sshd-adm, authorized_keys

As shown previously in non-Cray ssh, we run an `sshd-adm` listening on Port 30 using PubkeyAuthentication and AllowUsers restrictions for off cluster automation (`cron`) of system monitoring, log archival, backups, and user|group maintenance activities. The tcpwrappers rules also limit what uses this access method.

### 16. Install: Audit files Cray wants preserved with upgrades

**Example**: `CrayFiles/opt/cray/etc/*.ini`

Upgrading to CLE 2.2 (SMW 4.0) we discovered the install process clobbers the `/opt/cray/etc/*.ini` files. For us the solution was obvious, add them all to `CrayFiles` so we have a history. The primary issue in the upgrade was with `sedc_srv.ini` which had modifications required for a field notice which the update regressed.

### 17. Install: esLogin lnet configuration

**Documentation**: ARSCesLoginLustre

The esLogin nodes were delivered to ARSC configured as NFS. We previously published a Comparison of NFS vs. LNET which demonstrated performance made LNET a better choice for ARSC. Configuring LNET was a series of trial and errors. Jim Williams at ARSC and John Metzner ARSC/Cray did an outstanding job figuring this out. ARSC provided details to Cray for other customers.

### 18. Install: Customize startup and shutdown auto scripts

**Example**: `CrayFiles/opt/cray/etc/Pingo.start,Pingo.stop`

Probably every site customizes the auto scripts already. Ours are provided merely as an example. We do intentionally name them `Pingo.stop` and `Ognip.stop` not `auto.*` to ensure an administrator intending to boot the TDS (Test and Development System) does not forget which window they are in and inadvertently shut down the production system. *Let's not do that!*.

### 19. Install: Shutdown & Startup procedures beyond auto

**Documentation**: CommonShutdown -- CommonStartUp

**Tool**: `pkg/xts/bin/xt_mounts` , `pkg/downtime/Restrict,downtime` , `pkg/PbsSt/pbs_start_downtime`

There really is only so much which can be done with the `auto.*` scripts. System maintenance involves extra steps to schedule, quiesce (PBS dedicated_time), warn users, restrict usage for testing, cleanly un-mount filesystems, and validate configuration on restart. The ARSC Shutdown and StartUp documentation is provided as a model. The associated scripts are provided as examples as well. These models worked well when we implemented a Scyld cluster in March.

### 20. Install: Emergency power off procedure

**Tool**: `pkg/xts/sbin/epo_shutdown_xt`

The PDUs which power the system have battery backup, but the computer room air conditioning (CRAC) does not. While our power supply is very stable, a power outage greater than 10-15 minutes with the CRAC down runs the risk of the equipment overheating and shutting itself down. This is a situation we prefer to avoid. In addition, ARSC shares the facility with University of Alaska administrative computing systems which have priority access to the battery backup for an extended outage. There is a requirement that the computer room operator be able to quickly and simply emergency power off ARSC systems and we strongly prefer they not just flip the PDU breakers. With well defined Shutdown procedures, completely scripted shutdown is possible. The `epo_shutdown_xt` script executes each step with a reasonable timeout to ensure the system will go down. The script is executed via a locked switch the operators can push which then activates the shutdown via the sshd-adm connection.

---

## Tasks - Compute Node Linux (CNL) (top -- intro -- index -- general -- install -- CNL -- ongoing -- conclusion)

The following tasks relate to primarily Compute Node Linux.

### 1. CNL: Allow drop_caches for users (benchmarks)

**Technique:** policy, drop_caches, and init
**Tool**: `pkg/uak/src/drop_caches.c`
**Example**: `CrayFiles/opt/xt-images/templates/default/init`
**Example**: `CrayFiles/opt/xt-images/templates/default/sbin/drop_caches`

In Linux root can use the following technique to free cached memory:

```
login2: free # show free memory
             total       used       free  shared buffers   cached
Mem:       8108228    5092992    3015236       0       0  3883016
-/+ buffers/cache:    1209976    6898252
Swap:            0          0          0
login2: sudo sync # flush filesystem buffers
login2: sudo ksh -c "/bin/echo 3 >/proc/sys/vm/drop_caches"
login2: free
             total       used       free  shared  buffers   cached
Mem:       8108228     711848    7396380       0        0    28164
-/+ buffers/cache:     683684    7424544
Swap:            0          0          0
```

Under CLE 2.1 doing this on compute nodes would have been desirable to help diagnose memory leaks. We added this to nodehealth checks (NHC) in CLE 2.2, but have not experienced the memory leaks we suspected in 2.1. Another reason to drop caches is to clean a node so subsequent jobs do not waste their allocation flushing buffers left behind by predecessors and to avoid start-up jitter. Also, benchmarkers do prefer consistent state of nodes.

Under 2.1 we added a suid binary to CNL for benchmarkers. With our NHC in CLE 2.2 it is likely no longer required. To retain the suid does require a `chmod` in `default/init` (shown in o-w /tmp, and /var). Dropping the caches is useful for benchmarking, we used it extensively for IO testing LNET vs. NFS for esLogin nodes. Since we prefer to reduce suid we use `sudo` for non-CNL use of the `drop_caches` binary which includes the `sync` and can report additional information from `/proc/meminfo`:

```
login1: sudo /usr/local.adm/bin/drop_caches -info 3
#       Before:
#     MemTotal:     7.733 GB
#      MemFree:     6.297 GB
#       Cached:     0.871 GB
#       Active:     0.588 GB
#     Inactive:     0.295 GB
#        Dirty:     0.000 GB
#       Mapped:     0.027 GB
#         Slab:     0.477 GB
```

```
#         After:
#      MemTotal:      7.733 GB
#       MemFree:      7.593 GB
#         Cached:     0.027 GB
#         Active:     0.038 GB
#      Inactive:      0.001 GB
#         Dirty:      0.000 GB
#         Mapped:     0.027 GB
#           Slab:     0.024 GB
MemFree:     6.297     7.593
```

Above demonstrates freeing 1.3GB of cached memory.

2. **CNL: Use read-only rootfs and o-w /tmp, and /var**

   **Example**: `CrayFiles/opt/xt-images/templates/default/init`
   **Example**: `CrayFiles/opt/xt-images/master/default/init`

   The read-only CNL rootfs became the default in CLE 2.2. It could also be enabled in CLE 2.1. This option helps ensure compute nodes do not get corrupted by user jobs. Incorporating this in 2.2 had the side effect of mounting `/var` and `/tmp` 1777 which is not needed. Only root needs to write there. It is not desirable for users to leave cruft stealing memory from future jobs.

   To reset `/var` and `/tmp` as 755:

```
smw: cd $V/CrayFiles/opt/xt-images/templates/default/init
smw: grep ARSC init.piman
#2009-06-23 kac ARSC R28366 suid drop_caches
#2009-06-16 kac ARSC R28302 /bin/chmod go-w /var /tmp with default cnosrorootfs:=true
/bin/chmod 0755 /var # 2009-06-16 kac ARSC R28302
/bin/chmod 0755 /tmp # 2009-06-16 kac ARSC R28302
/bin/chmod 4755 /sbin/drop_caches # 2009-06-23 kac ARSC R28366
```

   Note, in modifying `/opt/xt-images/templates/default/init` you must also audit the contents of the `master/default/init` file and incorporate any changes Cray may make in upgrades. Another option would be to add a local init script for CNL to avoid having to audit Cray changes to `init`.

3. **CNL: Secure ssh: root authorized_keys, shadow**

   Secure compute node access to keyed access from `root@boot001` only:

```
smw: cd $V/CrayFiles/opt/xt-images/templates/default
smw: cat etc/passwd/passwd.piman
root:x:0:0:root-pingo-cnl:/root:/bin/sh
smw: sudo cat etc/shadow/shadow.piman
root:!:14137::::::
smw: uals -Z root/.ssh
d 0750  root     crayman  4096 081011.0927 authorized_keys
smw: sudo ssh nid00031 ualscnl -Z /etc/passwd /etc/shadow
- 0644 0        0          40 040914.1720 /etc/passwd
- 0600 0        0          19 040914.1720 /etc/shadow
```

   CLE 2.1 did not include a shadow file, but it could be added. CLE 2.2 may have included one. Void the password to allow only key access. We see no need for direct ssh access to compute nodes except from boot node (very restricted access).

4. **CNL: Mount lustre nosuid,nodev**

   **Example**: `CrayFiles/opt/xt-images/templates/default/etc/fstab`

   Just do it (reference mount most filesystems nosuid):

```
smw: cd $V/CrayFiles/opt/xt-images/templates/default/etc/fstab
smw: cat fstab.piman
19@ptl:/largefs /lustre/large   lustre  rw,nosuid,nodev,flock   0 0
16@ptl:/smallfs /lustre/small   lustre  rw,nosuid,nodev,flock   1 0
```

5. **CNL: Establish core_pattern**

**Example**: `CrayFiles/opt/xt-images/templates/default/etc/init.d/core_pattern`

Establish a meaningful pattern for compute node core files, following is host.executable.pid (see: `man core`):

```
smw: cd $V/CrayFiles/opt/xt-images/templates/default/etc/init.d/core_pattern
smw: grep -v ^# core_pattern.piman
echo core.%h.%e.%p > /proc/sys/kernel/core_pattern
```

Users need something to help segregate files.

### 6. CNL: Access to external license server

**Technique:** RSIP
**Example**: `CrayFiles/etc/rsipd.conf`
**Example**: `CrayFiles/opt/xt-images/templates/default/etc/krsip.conf`
**Alternative:** other?

Our sponsors use a consolidated license server hosted on an external system. For compute nodes to utilize licensed programs, the compute nodes need to be able to contact the server. The solution offered by Cray is to configure a service node (large systems may need multiple) with RSIP enabled to bridge the compute nodes to the external world. By default, RSIP really does open the compute nodes to the world, for example:

```
boot001: sudo ssh nid00031 # log into a compute node (for example)...
# ognip initramfs - Access must be specifically authorized by ARSC
#
# . /usr/local/cnl/source.ksh # use more complete CNL Linux
# uname -nr
nid00031 2.6.16.60-0.39_1.0102.4784.2.2.48B-cnl
# ping 199.165.84.229 # external mail server
199.165.84.229 is alive!
# /u1/uaf/kctester/telnet 199.165.84.229 25 # user copy of binary
Trying 199.165.84.229...
Connected to 199.165.84.229.
Escape character is '^]'.
220 Sendmail ESMTP 20090828.ARSC; Sun, 11 Apr 2010 09:35:02 -0800 (AKDT)
quit
221 2.0.0 arsc.edu closing connection
Connection closed by foreign host.
# exit
Connection to nid00031 closed.
```

Note, while the above was initiated via ssh as `root@boot001`, there is *nothing* to prevent a user from crafting an `aprun` program or script to do almost anything. One can restrict ports within `rsipd.conf` and can extend iptables or network ACLs on the RSIP server to restrict capabilities if desired.

### 7. CNL: Dump procedures and dump archival

**Documentation**: xtcompute -- Diag_ldump
**Tool**: `pkg/xts/bin/craydump_archive.ksh` , `pkg/xts/sbin/xt_chmod.ksh`

The documentation links above are local procedures for dealing with a failed compute node. The "Creating ldump-support-files" cited in Diag_ldump is likely no longer required for CLE 2.2. The referenced `craydump_archive.ksh` moves the dumps to our storage server. The `xt_chmod.ksh` script corrects other+write files generated by `xtdumpsys`.

### 8. CNL: Home and /usr/local filesystem access

**Technique:** /lustre/small
**Alternative:** dvs, es*

Our system, originally CLE 2.1, was proposed with $HOME resident on `/ufs` not visible on compute nodes. Based both on the available size and the inconvenience to users that was considered unacceptable. Like a number of other sites, we established a smaller lustre filesystem which houses $HOME and `/usr/local` which has binaries and libraries suitable for CNL use. One of our peer sites does similarly, one does not. Some sites implement global filesystems and external lustre is emerging as a good option. Mileage varies by site and user needs.

### 9. CNL: Audit and manage raw image

**Documentation**: xtraw
**Tool**: `pkg/xts/bin/xtraw-mk` , `pkg/xts/bin/xtraw-sdiff`

The process to generate a new raw image for CNL is a bit involved and can be confusing. There are complexities in syncing contents of `/opt/xt-images/master/default` and `/opt/xt-images/templates/default` and when issues forced us to customize `parameters-cnl` and `parameters-snl`, it became apparent that scripting the process to reduce errors and validate the contents of a new raw image before booting were desirable.

We ended up with the following scripts:

```
smw: xtraw-mk -?

Usage: xtraw-mk -options # generate new XT boot image (/rawX)

Options:
  -help          # usage information
  -d             # debug (just show commands)
  -r /rawX       # raw partition,   current:
  -o /rawX       # current /rawX,   current: /raw0
  -c compare     # compare override, for -o:
  -u directory   # unpack location, current: /var/local/output/xtraw
  -x version     # XT version,      current: 2.2.48B
  -s source      # source files,    current: ognip-2.2.48B
  -t [-]target   # target files,    current: ognip-2.2.48B-new
  -e step        # execution step,  current: 0
     old     |0    unpack old /raw via 'xtbootimage -e'
     make    |1    cpio src->target xt-images directory
     clone   |2    source->target via xtclone
     package|3     package CNL via xtpackage
              3b   package SNL via xtpackage -s
     image   |4    generate raw via xtbootimg
     new     |5    unpack new /raw, compare w/[old|compare]
  =e step        # execute only step

Note, this script must be run via sudo unless -d(ebug).

The '-compare' may be either /rawX or directory name in:
  /var/local/output/xtraw
Note, the xtraw directories are 600mb each and should
be periodically cleaned out.
```

With an accompanying script to compare contents of existing extracts:

```
smw: xtraw-sdiff 20100304.raw0/old  20100304.raw0/new

# sdiff -s 20100304.raw0/old.img.ls 20100304.raw0/new.img.ls # ? y

0644  root  root  09165 CNL0/initramfs.gz    | 0644  root  root  29771 CNL0/initramfs.gz
0644  root  root  43124 CNL0/size-initramfs  | 0644  root  root  43107 CNL0/size-initramfs
0644  root  root  11616 SNL0/initramfs.gz    | 0644  root  root  05452 SNL0/initramfs.gz
0644  root  root  59482 SNL0/size-initramfs  | 0644  root  root  34892 SNL0/size-initramfs

# sdiff -s 20100304.raw0/old.snl.ls 20100304.raw0/new.snl.ls # ? y

0644  root  root  35175 etc/localtime       | 0644  root  root  22072 etc/localtime

# sdiff -s 20100304.raw0/old.cnl.ls 20100304.raw0/new.cnl.ls # ? y

0755  crayadm crayadm     0 var             | 0755  root    root        0 var
0644  root    root    35175 etc/localtime    | 0644  root    root    22072 etc/localtime
0755  root    crayman 62128 sbin/drop_caches | 0755  root    crayman 60526 sbin/drop_caches
0755  crayadm crayadm     0 var/lib         | 0755  root    root        0 var/lib
                                            > 0700  postfix root        0 var/lib/postfix
```

The above is showing mode, ownership, and sum of files with the objective to confirm that desired changes in the image have occurred before attempting a boot.

## 10. CNL: Compute node health checks (NHC)

**Example**: `CrayFiles/etc/sysconfig/nodehealth`

**Tool**: `pkg/xts/sbin/cnl_nhc`

We have implemented the CLE 2.2 nodehealth feature:

```
smw: cd $V/CrayFiles/etc/sysconfig/nodehealth
smw: egrep -v '^#|^$' nodehealth.template
runtests: always
connecttime: 60
iterationlimit: 1
suspectenable: y
suspectbegin: 180
suspectend: 7200
recheckfreq: 900
Application: Admindown 240 300
Alps: Admindown 30 60
Filesystem: Admindown  60 300 0 0 /lustre/large
Filesystem: Admindown  60 300 0 0 /lustre/small
Site:       Admindown  30  60 0 0 /usr/local/sbin/cnl_nhc
```

The differences from the distributed are 'always', the two lustre tests, and the site test. The site `cnl_nhc` script does employ the more complete Linux CNL described previously. The actions are straight forward. With an expanded command set (including `gawk`), writing a C program for these simple actions is inappropriate.

Functionally, `cnl_nhc`:

1. gathers `/proc/meminfo`, `buddyinfo`, and `slabinfo`
2. issues drop_caches
3. rolls off `/var/logs/alps/apinit*` files (out of CNL memory)
4. rolls off any `/tmp/lnet-ptltrace*` files (out of CNL memory)
5. exits with error (admin down node) only if CNL memory free memory beneath threshold

The logging goes onto lustre, hence the filesystem checks must run first. Under CLE 2.1 we suspected memory leakage problems or memory buffer fragmentation due to sporadic problem reports from users, typically after several weeks of system uptime. This prompted gathering much of the information, but we have not had reports of the issues under CLE 2.2. We continue to gather the information. If we implement `'CNL_csa'` accounting, which deposits `pacct` files, we will clean those off the compute nodes daily.

---

## Tasks - Ongoing (top -- intro -- index -- general -- install -- CNL -- ongoing -- conclusion)

The following tasks are ongoing. The majority are `cron` entries.

### 1. Ongoing: Audit/reduce suid binaries

**Tool**: `pkg/permchk`
**Example**: `CrayFiles/usr/local.adm/etc/sguid.list`
**Documentation**: permchk
**cron**: sysmon@smw permchk (Daily)

A `suid` or `sgid` binary runs with privileges associated with the user or group, typically root. As such, any coding flaw can result in a privilege escalation. A good security practice is to eliminate unnecessary `suid` binaries. By doing this ARSC has been immune to a number of exploits over the years. With Linux (reduced set of `suid` binaries) this is less of an issue as the `suid` set is much smaller than traditional UNIX distributions, but auditing remains a good practice.

The permchk wrapper around the `process-setuid` script accomplishes the `suid` audit at ARSC using the `sguid.list` registry. The registry contains the ownership, mode, and sum of all `s[gu]id` binaries and other binaries we audit. Besides identifying new `s[gu]id` binaries, this tool also reports files with ACLs. Typically files in system space do not carry ACLs, which is good because in several ACL implementations the reported mode can conceal dangerous individual or group privileges if ACLs are present.

The script uses:

```
uals --type f -URA --or --acl --mode 7000 --fields tf --mount $FS
cat $FILELIST | uals --input -UdL --fields tmogsr
```

The first variant identifies s[gu]id or ACL files in a filesystem. The second variant generates the mode, ownership, size, and sum of a list of files, with `'--input'` using a single process for a collecting the information.

### 2. Ongoing: Audit/reduce other+write files

**Tool**: `pkg/worldwrt`
**cron**: sysmon@boot worldW_xt (Daily)

World writable (o+w) files in user space pose no system threat, but are a *bad idea* on shared access systems, especially if shared filesystems (NFS) are used. Most are the result of mistakes. ARSC policy prohibiting world write is enforced by the `worldW_xt` script which automatically sets o-w on non-exempted files and directories. The script emails user services who will reach out to assist repeat offenders.

The script utilizes:

```
uals -RzZA --t_ne=lscp --mode=002 --mount --mark $FS
```

to identify files.

### 3. Ongoing: Audit/resolve unowned files

**Tool**: `pkg/xts/bin/chk_unown`
**cron**: sysmon@smw chk_unown (Monthly)

Has any vendor ever distributed an operating system which did not include unowned files? These are more sloppy than a security risk, but in the event a newly assigned UID or GID conflicts with existing cruft this should be cleaned up. A common cause of these files are poorly written install scripts which un-tar as root (always a bad idea) and do not clean up. On XT systems, both PGI and PathScale have been known to do this.

ARSC runs the `chk_unown` script monthly, it uses:

```
uals --mount --or --nouser --nogroup -RzZA $FS
```

to scan filesystems of interest.

### 4. Ongoing: Identify dangling symlinks

**Technique:** manual
**Tip**: `uals -RzZA --mount --type l -L / >/dev/null`

Dangling symlinks can exist for legitimate reasons, but more often result from accidents or upgrades which do not clean up after themselves. They do not impose enough risk that we have automated a script, but we do manual scans on new systems and after major software upgrades. For example:

```
boot: sudo uals -RzZA --mount --type l -L / >/dev/null
...
uals: /lib/modules/2.6.16.60-0.39_1.0102.4784.2.2.48B-ss/source: No such file or directory
uals: /lib/modules/2.6.16.60-0.39_1.0102.4784.2.2.48B-ss/build: No such file or directory
uals: /lib/modules/2.6.16.60-0.39_1.0102.4787.2.2.41-ss/.../fsfilt_ldiskfs.ko: No such file ...
uals: /lib/modules/2.6.16.60-0.39_1.0102.4787.2.2.41-ss/.../ptlrpc.ko: No such file ...
...
uals: /opt/cray/x2/mazama/4.0.0/sbin/rcmzsd-client: No such file or directory
uals: /opt/cray/x2/mazama/default/sbin/rcmzsd-client: No such file or directory
uals: /opt/flexlm/default/flexlm.lic: No such file or directory
uals: /opt/gnome/lib64/libgthread.so: No such file or directory
uals: /opt/gnome/lib64/libglib.so: No such file or directory
uals: /opt/gnome/lib64/libgmodule.so: No such file or directory
uals: /opt/gnome/share/applications/defaults.list: No such file or directory
...
```

CLE upgrades do not clean up previous versions very well even after removing the older RPMs.

### 5. Ongoing: Eliminate other+write in suid filesystems

**Technique:** policy; configuration

Mounting user filesystems nosuid and eliminating other+write (1777) directories on system filesystems reduces the risk of rogue suid binaries from root-kits. CLE makes this easy as it splits `/tmp` and `/var`, default Linux installs do not. Reference audit/reduce other+write for ongoing checks and mount most filesystems nosuid,nodev and CNL mount lustre nossuid,nodev for more information.

### 6. Ongoing: Clean-up old/unusable Cray modules and rpms

**Technique:** manual

Almost immediately after upgrading to CLE 2.2 on our TDS, we had a user issue a `'module load'` of 2.1 environment because they wanted to use an environment which they knew generated correct results. We understand their logic and we appreciate the concern that

change is not always desirable when one wants to just conduct research, but loading 2.1 modules under 2.2 does not work well... as they quickly discovered. Cray neither cleans up nor provides any guidelines for cleaning up previous versions. In one sense this can be good as it provides a path to backout changes. In another sense this is not-so-good, especially with major release upgrades presenting users with unusable or inappropriate choices.

Under CLE 2.2 we have the following set of Cray modules:

```
smw: sgrep xt- $V/CrayFiles/var/local/logs/CF/modules.txt/*login | grep 2.2
xt-boot/2.2.41
xt-boot/2.2.41A
xt-boot/2.2.48B
xt-lustre-ss/2.2.41A_1.6.5
xt-lustre-ss/2.2.41_1.6.5
xt-lustre-ss/2.2.48B_1.6.5
xt-os/2.2.41
xt-os/2.2.41A
xt-os/2.2.48B
xt-pe/2.2.41
xt-pe/2.2.41A
xt-pe/2.2.48B
xt-service/2.2.41
xt-service/2.2.41A
xt-service/2.2.48B
```

This is not so bad, but we cleaned out all the 2.1 releases (including `catamount`). The process was:

- Clone /rr/20080730 to /rr/20090609
- Validate contents identical
- Remove (rpm -e) xt-*.2.1* RPMs and gcc-catmount
- Review removed files for appropriateness
- Remove old opt/module directories (rr)
- Remove old opt/modulefiles/Base-opts/2.1.*.lusrelsave files (rr)
- Change /rr/current symlink to 20090609
- Remove old boot/* kernels (rr)
- Similar bootroot clean-up after ensuring new shared root worked.
- Clean up SMW (/opt/xt-images).
- Clean up any other stale modules (like old xt-asyncpe)

The end result is less choices in 'module avail' and less likelihood users load a 2.1 set which does not function correctly under 2.2. The cloning and validation really should not be necessary. However, with no guidelines we were paranoid there could be inappropriate ties to previous versions and wanted to ensure a recovery mechanism. This cloning method can be done with the system up vs. `xthotbackup`. In practice, we did not need a recovery mechanism but a good (paranoid) System Administrator will always ensure one exists.

Some time has passed since we did this, but some command examples:

```
boot: sudo su - root # this is an exception condition for direct root
boot001:~ # cd /rr; mkdir 20090609
boot001:/rr # cd 20080707
boot001:/rr/20080707 # find . | cpio -pd ../20090609
boot001:/rr/20080707 # uals -RzZA -yMog >/tmp/200807rr.uals
boot001:/rr/20080707 # cd ../20090609
boot001:/rr/20090609 # uals -RzZA -yMog >/tmp/200906rr.uals
boot001:/rr/20090609 # exit

boot: wc -l 200*.uals # validate the clone
boot: sdiff -s 200*.uals | wc -l
       0

boot: sudo xtopview -r /rr/20090609 -m "start rpm clean-up"
default/:\w # rpm -qa | grep ^xt-[a-z-]*-2\.1\.
default/:\w # rpm -qa | grep ^xt-[a-z-]*-2\.1\. | xargs rpm -e
default/:\w # rpm -qa | grep catamount
default/:\w # rpm -e gcc-catamount
default/:\w # exit

boot: sudo su - root
```

```
boot001:~ # rpm -qa | grep ^xt-[a-z-]*-2\.1\.
boot001:~ # rpm -qa | grep ^xt-[a-z-]*-2\.1\. | xargs rpm -e
boot001:~ # cd /opt
boot001:/opt # rm -Rf xt-catamount

boot001:/rr/current/opt # mv xt-catamount /tmp/2.1rm/rr/

boot001:/rr/current/opt/modulefiles # mv gcc-catamount /tmp/2.1rm/rr/

boot001:/rr/current/opt/modulefiles/Base-opts # mv 2.1.* /tmp/2.1rm/rr/
```

Above is not complete, just the essence reconstructed from notes. When doing any kind of maintenance we do tend to use `script` to have a log of all activity, especially when electing to `'su - root'`. As stated up front, Cray provides no general guidelines on cleaning up old releases, for pointers ARSC is indebted again to our friends at NERSC.

### 7. Ongoing: Audit orphaned process

**Tool**: `pkg/chk/bin/chk_orphans.ksh`
**cron**: sysmon@boot chk_orphans (Daily)

Within `chk_etc` root ppid=1 processes are monitored via the hardware.txt file. Orphaned user processes should also be monitored for several reasons:

- `kftp`, `krlogin`, and `ssh` flaws leave stranded processes which can be auto-killed
- without process limits run-away processes can consume nodes
- detached (`nohup`) processes may represent user-daemons permitting some inappropriate action via an open port.

For multiple platforms ARSC runs the `chk_orphans` script which can auto-kill known residue and will report unknown and unfiltered ppid=1 processes.

### 8. Ongoing: Session life limits

**Technique:** policy, tod
**Alternative:** various

Our sponsors require idle session timeout (10 hours) and maximum session life (24 hours). Although some users are annoyed with these, they are in general a good idea. The concept is "abandoned" sessions (workstations) pose a risk as there is no viable means to ensure screen locking security. The enforcement mechanism for ARSC is a package named `tod`. At this time I cannot confirm if it is public domain or not. Prior to ARSC (late 1990s) I researched open source packages for similar requirements within UAF. Many relied on `utmp` records which are unreliable (long discussion). None were really effective, `tod` is effective. If there is interest I can pursue whether `tod` can be made available.

### 9. Ongoing: Establish process limits

**Technique:** policy, limits.conf, profile
**Example**: `CrayFiles/etc/security/limits.conf`

Native login nodes are "low resource" (only two CPUs with 8GB of memory, for us) and shared among many users for compiling as well as pre-|post-processing work. In a default XT configuration, they may also serve as PBS mom nodes. Without some form of limits enforcement, login nodes can easily be stressed (abused) by users. ARSC has implemented the resource limits on login nodes:

```
smw: cd $V/CrayFiles/etc/security/limits.conf; ls
backout  limits.conf.esl  limits.conf.login  limits.conf.template
smw: egrep -v '^#|^$' *login
*               soft    as              2097152
*               hard    as              2097152
*               soft    cpu             60
*               hard    cpu             720
*               soft    nproc           32
*               hard    nproc           64
```

This limits maximum processes, CPU, and address space size. Note, the process limits can become an issue if a user runs many small jobs and PBS mom and login nodes remain shared. There is also exposure within a PBS mom for abuse. Set PBS with `profile.local` and `csh.login.local`:

```
smw: cd  $V/CrayFiles/etc/profile.local; ls
backout  profile.local.boot001  profile.local.piman  profile.local.template
```

```
smw: more *template
...
  if [ -n "$PBS_JOBID" ]; then  # PBS mom address space limits  # R30756
        ulimit -a | /bin/grep " -v" >/dev/null
    if [ 0 = $? ]; then ulimit -v 2097152; ulimit -Hv 2097152   # bash
    else                ulimit -M 2097152; ulimit -HM 2097152   # ksh
    fi
  fi
...
smw: cd  $V/CrayFiles/etc/csh.login.local; ls
backout  csh.login.local.boot001  csh.login.local.piman  csh.login.local.template
smw: more *template
...
  if ( ${?PBS_JOBID} ) then     # PBS mom as limits     # R30756
        limit    vmemoryuse 2097152
        limit -h vmemoryuse 2097152
  endif
...
```

We are much more generous with limits on the fatter esLogin nodes.

### 10. Ongoing: Audit open ports (lsof, nessus)

**Tool**: `pkg/chk/bin/lsof_*`
**Example**: `CrayFiles/usr/local.adm/etc/lsof_ps.filter`
**Documentation**: uaklogin
**Alternative:** nessus, other
**cron**: sysmon@boot lsof_ports (Daily)

We mentioned exposure from open ports while discussing sdb and boot on network where `nessus` scans can topple some daemons. The `nessus` tool is a defense tool. Its purpose is to externally probe systems for open ports and analyze for risk of "dangerous" ports. Unfortunately, some daemons have not been coded to cope with the probing packets `nessus` sends. Daemons should be written to validate and cope with any input. A daemon which fails due to `nessus` is certainly suspect that its functional security may only be the lack documentation of acceptable message format. Nice guys playing by the rules do not send random packets at ports trying to cause damage. The networked world is not all nice. Many sites are protecting their systems with network firewalls and more rigid `iptables` definitions. Some sites employ packet sniffers trained to identify and alert on known suspicious traffic. Our sponsors require `nessus` scans monthly. Once-a-month scanning has no chance of identifying transient conditions. The `nessus` scans can trigger a denial-of-service for both poorly coded daemons and network appliances like printers which lack sophisticated logic.

Every open port represents an entry point into the system. Many are there for well defined and useful purposes, see `/etc/services`. Besides the defined ports, applications (users) can open unprivileged ports. While no privilege elevation can occur directly on unprivileged ports, the application written to listen on the port can act on the requests it sees which could execute commands in the context of the listening UID. A simple on-system check for open ports can be done with `'lsof -Pi'`. By running regularly (at least daily) and using a filter of known and accepted applications, one can stay aware of what doors are being left open into a system. If some young soul decides to establish a private `sshd` so her buddies can use her account, you might spot it. Of course, we all have policies prohibiting account sharing so we can wag our fingers and shut such activities down.

### 11. Ongoing: Audit locally created directories

**Tool**: `pkg/chk/bin/xts_dirs.ksh` , `pkg/chk/bin/chk_local_dirs.ksh`
**Example**: `CrayFiles/usr/local.adm/etc/local.dirs`
**cron**: sysmon@boot xts_dirs (2xWeek)

With traditional clusters and with workstation farms when a node needs to be rebuilt, one needs some means to recreate local customizations. ARSC uses ConfigFiles as mentioned, but we also need to recreate local directories and symlinks. We retain a registry of local created and modified (mode or ownership) directories. The `chk_local_dirs` script can validate and fix a node. The `xts_dirs` wrapper validates the cluster. In an XT environment the primary need for this tool is after system maintenance since CLE updates may regress some local settings.

### 12. Ongoing: Audit all system file changes

**Documentation**: ConfigFiles -- push -- chk_sanity -- xtvalidate
**Tool**: `pkg/push`
**Alternative:** xtopview
**cron**: sysmon@boot chk_sanity (Daily) invokes chk_etc

We already discussed practices for managing systems changes. The chk_sanity script runs daily as validation. This works very well

when System Administrators do what they are supposed to do and is supplemented with chk_etc (etc.txt) and `tripwire` at ARSC.

The `xtopview` tool Cray provides has good capabilities for managing the contents of `/etc` on the service nodes. Cray deserves recognition that they provide a tool for managing configuration files. Although `xtopview` is better than most things I have seen from other vendors, it only monitors `/etc`, does not manage the SMW or esLogin nodes, and is Cray specific and not applicable for our other platforms.

### 13. Ongoing: Audit system state: chkconfig

**Example**: `CrayFiles/var/local/logs/CF/chkconfig.txt`
**Tool**: `pkg/chk/bin/chk_etc.ksh`
**cron**: sysmon@boot chk_sanity (Daily) invokes chk_etc

With `chk_etc.ksh` invoked by chk_sanity in daily `cron` and as validation after maintenance, a `'chkconfig --list'` is written to a faux-CrayFiles entry to detect any changes.

The `chk_etc.ksh` script evolved from a similar script on a previous cluster to monitor system settings. For the Cray XT (and subsequent Scyld cluster) implementation, the script is generating faux-CrayFiles. These files are "pulled" vs. "pushed" to audit system state. ARSC also utilizes `tripwire` to audit filesystem changes, but live settings do not appear in files. The technique is still in evolution at ARSC, but the bottom line is one must be aware of any changes in dynamic system settings.

### 14. Ongoing: Audit system state: sysctl

**Example**: `CrayFiles/var/local/logs/CF/sysctl.txt`
**Tool**: `pkg/chk/bin/chk_etc.ksh` , `pkg/uak/src/rpmchk.c`
**cron**: sysmon@boot chk_sanity (Daily) invokes chk_etc

With `chk_etc.ksh` invoked by chk_sanity in daily `cron` and as validation after maintenance, a `'sysctl -A'` is written to a faux-CrayFiles entry to detect any changes. Some entries are volatile so there is a filter imbedded in the `chk_etc.ksh` script.

### 15. Ongoing: Audit system state: iptables

**Example**: `CrayFiles/var/local/logs/CF/iptables.txt`
**Tool**: `pkg/chk/bin/chk_etc.ksh`
**cron**: sysmon@boot chk_sanity (Daily) invokes chk_etc

With `chk_etc.ksh` invoked by chk_sanity in daily `cron` and as validation after maintenance, an `'iptables --list'` is written to a faux-CrayFiles entry to detect any changes. With CNL RSIP implementation this causes some minor "noise" when compute nodes die.

### 16. Ongoing: Audit system state: mtab, lsmod, nework, ppid=1

**Example**: `CrayFiles/var/local/logs/CF/hardware.txt`
**Tool**: `pkg/chk/bin/chk_etc.ksh`
**cron**: sysmon@boot chk_sanity (Daily) invokes chk_etc

The `chk_etc` script also generates `hardware.txt` faux-CrayFiles entry. This technique was started on a previous cluster and quickly adopted for our Cray XT to capture a number of things:

- `mtab etab`
- `lustre/health_check lustre/*/group_upcall`
- `lspci lsusb lsscsi`
- `/dev/*`
- `sysconfig/network[-scripts]/*`
- `lsmod`
- `root ppid=1`

This became a kitchen-sink collection and should be reworked, but it works well enough that it was deployed in our new Scyld cluster. The concept is the same as the `sysctl.txt`, `chkconfig.txt` and other `chk_etc` collections, audit system state. The filtered root ppid=1 has been useful for identifying missing and unexpected new daemons (such as `nfsserver` returning after CLE upgrades).

### 17. Ongoing: Audit modules state / history

**Example**: `CrayFiles/var/local/logs/CF/modules.txt`
**Tool**: `pkg/chk/bin/chk_etc.ksh` , `pkg/uak/src/modchk.c`
**Documentation**: xtModulesDefault
**cron**: sysmon@boot chk_sanity (Daily) invokes chk_etc

Formatted `'module avail'` lists are generated by `chk_etc` as faux-CrayFiles entries. These files can be used to track module changes over time. For example, following shows module changes over last 13 weeks:

```
smw: cd $V/CrayFiles/var/local/logs/CF/modules.txt/backout
smw: uals -yf --newer 13w *.login.* # identify files & dates
modules.txt.login.20100121
modules.txt.login.20100310
modules.txt.login.20100405
modules.txt.login.20100409
smw: modchk -f "`uals -yf --newer 13w *.login.*`" -match # show changes

Cray-XT

p p p p :    4 hosts
i i i i :   69 total modules
n n n n :  147 total versions
g g g g :
o o o o : * default; + available; - missing

- - - * :PrgEnv-cray           1.0.1
- - - * :cce                   7.2.1
+ + - - :gcc                   4.2.0.quadcore
+ + - - :gcc                   4.2.4
- - - + :hdf5                  1.8.4.1
- - - + :hdf5-parallel         1.8.4.1
- - - + :netcdf                4.0.1.3
- - - + :netcdf-hdf5parallel   4.0.1.3
- + + + :pbs                   10.2.0.93147
+ + - - :pgi                   7.0.7
+ + - - :pgi                   7.1.6
+ + - - :pgi                   8.0.2
+ + - - :pgi                   9.0.1
+ + - - :pgi                   9.0.2
+ + - - :xt-asyncpe            1.0c
- - - + :xt-asyncpe            3.7
+ + - - :xt-libsci             10.1.0
+ + - - :xt-libsci             10.2.1
+ + - - :xt-libsci             10.3.0
- - - + :xt-libsci             10.4.3
+ + - - :xt-mpt                3.0.2
+ + - - :xt-mpt                3.0.3.1
- - - + :xt-mpt                4.0.3

124 of 147 modules installed on all 4 hosts
```

### 18. Ongoing: Audit rpm state / history

**Example**: `CrayFiles/var/local/logs/CF/rpm.txt,rpmcnl.txt`
**Tool**: `pkg/chk/bin/chk_etc.ksh`
**cron**: sysmon@boot chk_sanity (Daily) invokes chk_etc

The `chk_etc` script also generates `'rpm -qa | sort'` as faux-CrayFiles entries. Like the module.txt files, the `rpm.txt` files can be used to track changes over time or compare that systems are at the same levels:

```
smw: cd $V/CrayFiles/var/local/logs/CF/rpm.txt
smw: rpmchk -f rpm.txt.ognipa,rpm.txt.pingob,rpm.txt.pingoc -match
#Linux x86_64 2.6.16.60-0.42.9-smp
#
#o p p :    3 hosts
#g i i :  970 total packages
#n n n :
#i g g :
#p o o :
#a b c :Name                 Version      Release
#
 - i i :kernel-smp           2.6.16.60    0.42.4
```

```
#
# 969 of 970 packages installed on all 3 hosts

smw: rpmchk -f rpm.txt.ognipa,rpm.txt.pingob,rpm.txt.pingoc -name kernel-smp
#Linux x86_64 2.6.16.60-0.42.9-smp
#
#o p p :    3 hosts
#g i i :    5 total packages
#n n n :
#i g g :
#p o o :
#a b c :Name       Version   Releas
#
 - i i :kernel-smp 2.6.16.60 0.42.4
 i i i :kernel-smp 2.6.16.60 0.42.5
 i i i :kernel-smp 2.6.16.60 0.42.7
 i i i :kernel-smp 2.6.16.60 0.42.8
 i i i :kernel-smp 2.6.16.60 0.42.9
#
# 4 of 5 packages installed on all 3 hosts
```

Looks like the esLogin node `ognipa` has one extra kernel, but the three esLogin nodes are otherwise synchronized.

As mentioned in culling unnecessary RPMs this can also be used to identify what has been removed or added to a system:

```
smw: uals -Z rpm.org.pingo1 rpm.txt.pingo1
- 0640  sysmon   crayman   31k 081003.0923 rpm.org.pingo1
- 0640  sysmon   crayman   26k 100409.0943 rpm.txt.pingo1
smw: rpmchk -f rpm.org.pingo1,rpm.txt.pingo1 -match !version
#    Lin
#    x86
#    2.6
#
#r  :p  :    2 hosts
#p  :i  : 1294 total packages
#m  :n  :
#   :g  :
#   :o  :
#   :1  :Name
#
 i  :-  :CheckHardware
 i  :-  :Crystalcursors
 i  :-  :Mesa-devel-32bit
 i  :-  :MozillaFirefox
 i  :-  :MozillaFirefox-trans
 i  :-  :MySQL
 i  :-  :OpenEXR
 i  :-  :OpenEXR-32bit
 i  :-  :SDL
 i  :-  :SDL-32bit
 i  :-  :a2ps
 i  :-  :aalib
 i  :-  :aalib-32bit
 i  :-  :account-ss
 -  :i  :alps-app-devel
...many omitted...
 -  :i  :xt-totalview
 i  :-  :yast2-control-center
 i  :-  :yast2-core-devel
 i  :-  :yast2-devel
 i  :-  :yast2-x11
 i  :-  :yelp
 i  :-  :zenity
 i  :-  :zmd-inventory
 i  :-  :zoo
```

```
#
# 949 of 1294 packages installed on all 2 hosts
```

Above generated the list of all RPMs removed or added between 2008-10 as delivered by Cray and current 2010-04.

### 19. Ongoing: Audit filesystem content

**Example**: `CrayFiles/var/local/logs/CF/etc.txt`
**Tool**: `pkg/chk/bin/chk_etc.ksh` , `pkg/permchk`
**Alternative:** tripwire
**cron**: sysmon@boot scripted (Daily) uals; chk_etc

We have discussed practices for managing system changes and auditing system file changes, but what happens when somebody is lazy or busy and edits a new file directly and forgets to copy it back? A fall back method is required. ARSC is using tripwire, but that does require good rules sets and careful scanning of what one is accepting. The `chk_etc` script also sums all files in `/etc` where most (not all) configuration files live, that provides a secondary check. This is redundant and may be retired as our `tripwire` use becomes more reliable.

Prior to `tripwire` on previous clusters, we ran periodic `'uals -RzZA --mount --fields Mogr'` (the `'r'` field is bsd sum) of all system filesystems, not just `/etc`, and added filtered compares.

### 20. Ongoing: Audit system state: disk storage

**Example**: `CrayFiles/var/local/logs/CF/lsi.txt,disk.txt,lun.txt`
**Tool**: `pkg/xts/bin/lsi_profile`
**Documentation**: Pingo_lsiost -- Pingo_diskfw
**cron**: sysmon@smw lsi_profile (Daily)

Storage controllers are sophisticated and require audit and configuration backup. The LSI controllers provide GUI tools for monitoring and configuration. When configuring many controllers, manual GUI use is subject to human error and inconsistency. For monitoring many hundreds of disks and storage components watching a GUI is not at all practical. LSI does provide an under documented command line interface. Fortunately, the cluster which preceded our Cray XT also utilized LSI controllers so we had an adaptable tool. The essence of the tool is dump as text and audit all storage profile components. This is pulled in by `chk_etc` and we know in the morning if we have a component which needs replacement or if we have excessive error counts to investigate.

With previous systems, we learned the hard way that replacement parts can have unpredictable firmware levels. In some cases older incompatible firmware may arrive on a replacement component. In one case, newer disk firmware on a replacment disk in an array resulted in LUN and filesystem corruption. Cray does better than some vendors in firmware tracking, but sites should still know exactly what is in their storage and investigate when they see component firmware change after replacement. We can immediately identify all storage firmware and associate the LUNs to the OSTs. You only need this information with unusual problems (none yet on our XT), but when you need it you really need it.

With multiple vendors in the past, we have been recipients of bad batches of disks. Having history of serial numbers, manufacture dates, and replacement has been critical for resolving problems like this. In one case we identified the failure trends well before the vendor.

### 21. Ongoing: Purge /tmp and lustre scratch

**Tool**: `pkg/fspurger`
**Alternative:** various
**cron**: root@all purge (WeekDays)

Users being users, temporary and work storage will grow until full if one does not implement policies, quotas, purgers, and other tools to address usage. Within ARSC (and at our peers) purging has always been a contentious issue. For `/workdir` we have a 30-day purge policy on most of our systems. When `/workdir` at 150TB consistently runs only 30-40% utilized staff and users have difficulty understanding purging, but:

- if filesystems fill it is too late (jobs will die)
- manually addressing after a threshold can require significant effort at inopportune times
- very large filesystems perform better and manage fragmentation better if usage is <60%
- excess space for shorter term demands is beneficial
- if purge time is too long, users forget to migrate to archive and lose data

We do maintain a large pool of unpurged space for users with known needs. We do have users running touch scripts to avoid purges. Some peers have implemented purges based on inode residency not access or modification time to in order to thwart touch scripts.

Our existing purger has decent configuration options to manage `/tmp`, `/scratch`, `/workdir`, and other requirements adequately. It does cope with special characters in filenames, but it is a complex and dated script not really designed for extremely large filesystems.

Attempts to redesign and rewrite (or borrow and adapt peer tools) invariably bog down into the *"why purge?"* discussion loops meaning the existing tool lives on because of inertia. Fortunately, it is "good enough"... but could be much better.

### 22. Ongoing: Health check and event paging, internal

**Tool**: `pkg/xts/bin/xtpage` , `pkg/xts/sbin/xtp*`
**Documentation**: xtOncallPaging -- OncallTriage -- xtpage_pingo
**Alternative:** various
**cron**: sysmon@smw,boot xtpage (2xHourly)

The following criteria were initially establish as oncall paging criteria:

- boot or sdb node failure
- multiple login node failures
- lustre unhealthy
- NFS to archive storage down on multiple nodes
- greater than 10% compute nodes down
- PBS stalled or flushing

On previous systems ad hoc scripts paged independently. Over the years this has created some confusion so a configuration file (xtpage_pingo) invoked by a wrapper (xtOncallPaging) was designed using concise test scripts:

- xtp_Compute - Check if too many compute nodes are unavailable
- xtp_Cron - Check if cron daemon is running and attempt restart
- xtp_Daemons - Check if critical login node daemons are running and attempt restart
- xtp_Health - Boot node `pdsh` initiated **xtpage** on login nodes
- xtp_Job - Check qsub and aprun on semi-idle system (**xtp_Usage** flagged)
- xtp_Lustre - Check if Lustre healthy and writable
- xtp_NFS - Check if NFS (archive/u?) writable
- xtp_PBS_alps - Check PBS, `alps`, and `flexlm` daemons and attempt restart
- xtp_Ping - Check if node is pingable
- xtp_PingUp - Check if node is pingable, available ssh, and not restricted
- xtp_Restrict - Check if system is Restricted
- xtp_Usage - Check if system is busy or if jobs can run

Having a designed process made it much easier to identify all pagable events and write general Oncall Triage procedures.

### 23. Ongoing: Health check and event paging, external

**Documentation**: xtpage_crays
**Alternative:** various
**cron**: sysmon@admin1 xtpage (2xHourly) BigBrother

The design of the XT Oncall Paging process described above was intended for use from multiple nodes. The SMW checks the boot node, the boot node checks the remainder of cluster, and a cluster external node (xtpage_crays) checks the SDB as well as simple checks of SMW and login nodes. The checks are all cron initiated, the cross node checks are capable of restarting cron if it is down.

Parallel to the development of this paging process, another ARSC group was implementing Big Brother for a consolidated view of all critical systems and servers and an integrated paging, call-back, and logging system. The XT Oncall Paging script was extended to send status files to Big Brother instead of paging directly.

### 24. Ongoing: Roll and protect system logs

**Example**: `CrayFiles/etc/logrotate*`
**Tool**: `pkg/xts/bin/xt5logcp`
**Alternative:** mazama
**cron**: sysmon@smw,boot,sdb,esl lxt5logcp (Daily) also logrotate

The Linux `/etc/logrotate.d` scripts handle log rotation and can be extended and configured for site requirements. The logs themselves should also be rolled off the cluster both for site wide consolidation and for security. We copy the logs nightly with the `xt5logcp` script which copies to the storage server. The script leaves local copies for a specified number of days as well as copying off. The syslog messages are routed automatically to an external log host as well as the local retention. The odds of a serious security violation are not very high with "defense in depth", but a non-casual attack will attempt to tamper with logs: protect them.

### 25. Ongoing: Archive external logs

**Tool**: `pkg/xts/bin/xt5logtar`

**Documentation**: xt_logs
**cron**: sysmon@admin1 xt5logtar (Monthly)

Besides rolling logs off host as described above, the mess which is accumulating on the storage server must also be managed. We utilize a monthly script to tar the various log directories on the archive server to reduce the clutter. We generally attempt to document logs and log retention policies (xt_logs).

### 26. Ongoing: Backup disk storage configuration (LSI)

**Tool**: `pkg/config_backup/bin/config_backup.pl`
**cron**: sysmon@smw config_backup (Daily)

Typically storage vendors (LSI) and their redistributors (Cray) will include instructions to perform configuration backups prior to controller firmware upgrades. Regular backups are also a good idea. We have yet to need them (*thank goodness*). However, it is conceivable that a catastrophic failure of redundant controller with a replacement running different and potentially incompatible firmware will cause problems. Recovery will be easier if a current configuration backup is available. Our LSI backup script is provided and is augmented with our disk storage audit.

### 27. Ongoing: Check L1 and L0 temperature, voltage, health

**Tool**: `pkg/xts/bin/chk_l0.ksh,chk_L1L0.ksh`
**Documentation**: Diag_L0
**Alternative:** sec
**cron**: sysmon@smw chk_L1L0 (Daily)

When we received our XT5 there were several early node failures. We learned to use hdt and probe L0 states and identified a set of marginal VRMs (Voltage Regulator Modules) for preventative replacement. We also had air flow concerns and were closely monitoring temperatures. After addressing the above our system has been stable for VRMs. We continue to run the `chk_L1L0.ksh` daily, it was running every four hours initially. The script serially queries all L0 and L1 which is acceptable under the SEDC limitations identified with FN#5629, but for a 5-cabinet XT this does take 7+ minutes to run. The result is an email summarizing recent temperatures, fan speeds, and any over-threshold voltages:

```
Date: Sun, 18 Apr 2010 10:12:07 -0800
To: crayman@arsc.edu
Subject: :>>:chk_L1L0  :piman   :   2: 2 warnings:
From: sysmon@arsc.edu (sysmon-piman)

c2-0c0s6 CNL-3  9 PROC1 VDDNB =  1321/  1200    10.1% !!high
c2-0c1s0 CNL-0  4 PROC0 VDDNB =  1324/  1200    10.3% !!high
!! 2 warnings, vsense.20100418.1005

#Date    Time:Air c0-0 c1-0 c2-0 c3-0 c4-0 :Fan c0-0 c1-0 c2-0 c3-0 c4-0
20100418.1005:     17C  17C  17C  17C  19C :    46Hz 46Hz 46Hz 46Hz 49Hz
20100418.0255:     17C  17C  17C  17C  19C :    46Hz 46Hz 46Hz 46Hz 49Hz
20100417.0255:     16C  16C  17C  17C  19C :    46Hz 46Hz 46Hz 46Hz 49Hz
20100416.0255:     14C  15C  15C  15C  18C :    46Hz 46Hz 46Hz 46Hz 49Hz
20100415.0255:     16C  15C  16C  16C  18C :    46Hz 46Hz 46Hz 46Hz 49Hz
20100414.0255:     17C  17C  17C  17C  19C :    46Hz 46Hz 46Hz 46Hz 49Hz
20100413.0255:     17C  16C  17C  17C  19C :    46Hz 46Hz 46Hz 46Hz 49Hz
20100412.0255:     16C  16C  17C  17C  19C :    46Hz 46Hz 46Hz 46Hz 49Hz
20100411.0255:     16C  16C  17C  17C  19C :    46Hz 46Hz 46Hz 46Hz 49Hz

----
Mail: crayman@arsc.edu
Job:  chk_L1L0.ksh
Log:  /var/local/scratch/L1L0/chk_L1L0.20100418.1005
```

Note the temperature history still shows c4-0 running higher temperatures, this has been an issue with room airflow since installation.

The `chk_l0.ksh` script is still used for data gathering when a node has problems, for example:

```
smw: chk_l0.ksh -n 295 # nid00295 failed recently...
- 0644  crayadm  crayadm  4236 100418.0954 consumer.1004051418
- 0644  crayadm  crayadm   16m 100418.1005 console.1004051418

   295    0x127  c2-0c1s1n3  compute      down       batch
# Date changed from 04-18 to 04-18
```

```
# c2-0c1s1n3 vsense

c2-0c1s1 CNL-3  0 VDDA        =  2420/  2500    -3.2%
c2-0c1s1 CNL-3  1 +12V        = 12384/ 12500*   -0.9%
c2-0c1s1 CNL-3  2 PROC0 VDDIO =  1795/  1800    -0.3%
c2-0c1s1 CNL-3  3 PROC0 VTT   =   905/   900     0.6%
c2-0c1s1 CNL-3  4 PROC0 VDDNB =  1259/  1200     4.9%
c2-0c1s1 CNL-3  5 PROC0 VDD   =  1224/  1200     2.0%
c2-0c1s1 CNL-3  6 PROC0 VLDT  =  1192/  1210    -1.5%
c2-0c1s1 CNL-3  7 PROC1 VDDIO =  1798/  1800    -0.1%
c2-0c1s1 CNL-3  8 PROC1 VTT   =   902/   900     0.2%
c2-0c1s1 CNL-3  9 PROC1 VDDNB =  1253/  1200     4.4%
c2-0c1s1 CNL-3 10 PROC1 VDD   =  1221/  1200     1.8%
c2-0c1s1 CNL-3 11 PROC1 VLDT  =  1198/  1210    -1.0%
c2-0c1s1 FeP-0  0 Tyco Bus    = 12465/ 12500*   -0.3%
c2-0c1s1 FeP-0  1 12V Bus     = 12421/ 12500*   -0.6%
c2-0c1s1 FeP-0  2 Disc Sw In  = 12465/ 12500*   -0.3%
c2-0c1s1 FeP-0  3 Disc Sw Mid = 12465/ 12500*   -0.3%
c2-0c1s1 Mez-7  0 LDT         =  1240/  1200     3.3%
c2-0c1s1 Mez-7  1 AVDD        =  2480/  2500    -0.8%
c2-0c1s1 Mez-7  2 CORE        =  1602/  1550*    3.4%
c2-0c1s1 Mez-7  3 5V BIAS     =  4984/  5000    -0.3%

# c2-0c1s1n3 hsense

c2-0c1s1:+ /opt/bin/hsense 3 5 7
c2-0c1s1:FR2 SMP node 3/FR2PIC (CrayPIC):
c2-0c1s1:       80 00000000  (HealthOK)
c2-0c1s1:Front-End Power/VERTYPIC 0 (CrayPIC):
c2-0c1s1:       00000080  (Health)
c2-0c1s1:Mezzanine/MEZZPIC (CrayPIC):
c2-0c1s1:       00800000  (Health)
c2-0c1s1:+ set +x

# c2-0c1s1n3 hdt

c2-0c1s1:+ /opt/bin/hdt -n 6:0 -m 0x411
c2-0c1s1:0x411 0x0000000000000000
c2-0c1s1:+ /opt/bin/hdt -n 6:0 -m 0x412
c2-0c1s1:0x412 0x0000000000000000
c2-0c1s1:+ /opt/bin/hdt -n 7:0 -m 0x411
c2-0c1s1:0x411 0x0000000000000000
c2-0c1s1:+ /opt/bin/hdt -n 7:0 -m 0x412
c2-0c1s1:0x412 0x00000005f5abd050
c2-0c1s1:+ set +x

# c2-0c1s1n3 consumer.1004051418 # 2 entries for c2-0c1s1n3

Sun Apr 18 09:54:44 2010 - rs_event_t at 0x805e4d8
ev_id = 0x040040e5 (ec_heartbeat_stop)
ev_src = ::c2-0c1s1
ev_gen = ::c0-0c0s0n0
ev_flag = 0x00000002 ev_priority = 0 ev_len = 160 ev_seqnum = 0x00000000
ev_stp = 4bcb4764.00077726 [Sun Apr 18 09:54:44 2010]
svcid 0: ::c2-0c1s1n3 = svid_inst=0x0/svid_type=0x0/svid_node=
c2-0c1s1n3[rsn_node=0x127/rsn_type=0x0/rsn_state=0x7], err code 65740 - node heartbeat fault
ev_data...
00000000: 01 00 00 00 00 00 00 00 00 00 00 00 00 07 27 01 *..............'.*
00000010: 08 00 11 03 00 00 00 00 01 00 00 00 cc 00 01 00 *................*
00000020: 6e 6f 64 65 20 63 32 2d 30 63 31 73 31 6e 33 20 *node c2-0c1s1n3 *
00000030: 6d 69 67 68 74 20 62 65 20 64 65 61 64 2c 20 6c *might be dead, l*
00000040: 61 73 74 20 68 65 61 72 74 62 65 61 74 20 30 30 *ast heartbeat 00*
00000050: 31 30 65 31 62 61 2c 20 48 54 20 6c 6f 63 6b 75 *10e1ba, HT locku*
00000060: 70 20 64 65 74 65 63 74 6f 72 20 61 62 6c 65 20 *p detector able *
00000070: 74 6f 20 72 65 61 64 20 6e 6f 64 65 20 6d 65 6d *to read node mem*
00000080: 6f 72 79 2c 20 48 54 20 6c 6f 63 6b 75 70 20 63 *ory, HT lockup c*
00000090: 72 69 74 65 72 69 61 20 6e 6f 74 20 6d 65 74 00 *riteria not met.*

Sun Apr 18 09:54:56 2010 - rs_event_t at 0x805e4d8
ev_id = 0x040040e5 (ec_heartbeat_stop)
ev_src = ::c2-0c1s1
```

```
ev_gen = ::c0-0c0s0n0
ev_flag = 0x00000002 ev_priority = 0 ev_len = 98 ev_seqnum = 0x00000000
ev_stp = 4bcb4770.0007a58f [Sun Apr 18 09:54:56 2010]
svcid 0: ::c2-0c1s1n3 = svid_inst=0x0/svid_type=0x0/svid_node=
c2-0c1s1n3[rsn_node=0x127/rsn_type=0x0/rsn_state=0x7], err code 131276 - node heartbeat fault
ev_data...
00000000: 01 00 00 00 00 00 00 00 00 00 00 00 00 07 27 01 *..............'.*
00000010: 08 00 11 03 00 00 00 00 01 00 00 00 cc 00 02 00 *................*
00000020: 6e 6f 64 65 20 63 32 2d 30 63 31 73 31 6e 33 20 *node c2-0c1s1n3 *
00000030: 63 6f 6e 73 69 64 65 72 65 64 20 64 65 61 64 2c *considered dead,*
00000040: 20 6c 61 73 74 20 68 65 61 72 74 62 65 61 74 20 * last heartbeat *
00000050: 64 65 61 64 64 65 61 64 2c 20 73 75 69 63 69 64 *deaddead, suicid*
00000060: 65 00                                           *e...............*

# c2-0c1s1n3  console.1004051418 # 90 of 1187 lines for 04-18

[2010-04-18 09:54:33][c2-0c1s1n3]Unable to handle kernel paging request at ffffffff69123b10 RIP:
[2010-04-18 09:54:33][c2-0c1s1n3][]
[2010-04-18 09:54:33][c2-0c1s1n3]PGD 103067 PUD 0
[2010-04-18 09:54:33][c2-0c1s1n3]Oops: 0010 [1] SMP
[2010-04-18 09:54:33][c2-0c1s1n3]last sysfs file: /devices/system/node/node1/nr_hugepages
[2010-04-18 09:54:33][c2-0c1s1n3]CPU 5
[2010-04-18 09:54:33][c2-0c1s1n3]Modules linked in:
blcr blcr_vmadump blcr_imports mgc lustre lov mdc lquota osc kptllnd ptlrpc obdclass lnet lvfs libcfs ipip
[2010-04-18 09:54:33][c2-0c1s1n3] krsip portals rca heartbeat
[2010-04-18 09:54:33][c2-0c1s1n3]Pid: 11272, comm: wrf.exe Tainted: PF M U 2.6.16.60-0.39_1.0102.4787.2.2.41-cnl #1
...additional entries omitted...

# Log: /var/local/scratch/L1L0/Summary.20100418.1023.c2-0c1s1n3
```

### 28. Ongoing: Check node response

**Tool**: `pkg/xts/bin/xtresponds.ksh` , `pkg/uak/src/xt_stat.c`
**Alternative:** various
**cron**: sysmon@boot xtresponds (4xHourly) also xt_stat

This is a technique which went through three generations of IBM/AIX clusters to be implemented on our Cray XT5 and has been implemented on our new Scyld cluster. Some technique is required to identify when compute nodes fail. For our XT the `xtresponds` script provides this as a regular `cron` script parsing `xtprocadmin` and respecting a flag when the system is restricted for maintenance.

The `pkg/uak/src/xt_stats.c` program also evolved from previous IBM/AIX `sp_stats` program to summarize system availability:

```
smw: xt_stats -m =2010-03 -d -t -c pingo
#   13.58 | 03-01.00:00 -> 03-01.13:35 | 2010-03-01.000501 1: 316
# 211.56 | 03-01.13:35 -> 03-10.09:08 | 2010-03-01.133501  : 284 316
#   10.19 | 03-10.09:08 -> 03-10.19:20 | 2010-03-10.090838  : -1 # restricted
# 160.75 | 03-10.19:20 -> 03-17.13:05 | 2010-03-10.192001  :    # unrestricted
#    0.25 | 03-17.13:05 -> 03-17.13:20 | 2010-03-17.130501  : 429 442
#   20.75 | 03-17.13:20 -> 03-18.10:05 | 2010-03-17.132001 1: 442
#    3.00 | 03-18.10:05 -> 03-18.13:05 | 2010-03-18.100501  : 331 442
# 113.00 | 03-18.13:05 -> 03-23.06:05 | 2010-03-18.130501  : 331 442 570
#    2.50 | 03-23.06:05 -> 03-23.08:35 | 2010-03-23.060501  : 331 442 520 570
#   78.25 | 03-23.08:35 -> 03-26.14:50 | 2010-03-23.083501  : 331 442 570
#    0.25 | 03-26.14:50 -> 03-26.15:05 | 2010-03-26.145001  : 39 331 442 570
#   21.75 | 03-26.15:05 -> 03-27.12:50 | 2010-03-26.150501 1: 331 442 570
#    0.25 | 03-27.12:50 -> 03-27.13:05 | 2010-03-27.125001  : 314 331 442 570
# 103.00 | 03-27.13:05 -> 03-31.20:05 | 2010-03-27.130501 1: 331 442 570
#    3.92 | 03-31.20:05 -> 04-01.00:00 | 2010-03-31.200501  : 258 331 442 570
# 743.00 2010-03

2010-03:Cluster   Node  Node   Node           30.96 days      743.00 hours
2010-03:   Down   Best  Worst  Total    432 compute  24 service nodes
2010-03:      1      0      1      9 #events | All_Up <=2Down <=12    Cluster
2010-03:   10.2   10.2  357.1   6086 #downHrs| 582.25  333.11   10.19   10.19
2010-03:          98.6%  51.9%       # up%   |  21.64%  55.17%  98.63%  98.63%
```

At first I was confused why March only had 743 not 744 hours: daylight savings time. The summary above shows we had 98.6% availability for March, the majority of lost time was the 10+ hour maintenance outage on 03-10. March was a bad month for us with 9 total node events, three were user induced. Our Change/Problem management system has details for each outage. February was certainly better:

```
smw: xt_stats -m =2010-02 -d -t -c pingo
# 206.33 | 02-01.00:00 -> 02-09.14:20 | 2010-02-01.000501 1: 443
#  18.75 | 02-09.14:20 -> 02-10.09:05 | 2010-02-09.142001  : 443 572
#   6.86 | 02-10.09:05 -> 02-10.15:56 | 2010-02-10.090501  : -1 # restricted
#  40.89 | 02-10.15:56 -> 02-12.08:50 | 2010-02-10.155622  :    # unrestricted
# 399.17 | 02-12.08:50 -> 03-01.00:00 | 2010-02-12.085001  : 316
# 672.00 2010-02

2010-02:Cluster   Node  Node   Node           28.00 days      672.00 hours
2010-02:   Down   Best  Worst  Total    432 compute  24 service nodes
2010-02:      1      0      1      2 #events | All_Up <=2Down <=12    Cluster
2010-02:    6.9    6.9  406.0   3769 #downHrs| 631.11    6.86    6.86    6.86
2010-02:          99.0%  39.6%       # up%   |   6.09%  98.98%  98.98%  98.98%
```

Being able to quickly summarize historical availability of our clusters is quite useful.

### 29. Ongoing: Usage allocation

**Tool**: `pkg/usage/sbin/parsepbscronjob` , `pkg/usage/bin/show_usage`
**Alternative:** csa acct
**cron**: sysmon@sdb parsepbscronjob (Daily,Monthly) also show_usage and infrastructure

Knowing who is using an HPC resource and managing allocations is a highly site specific topic. Each site has to find processes to cope with this. Our users are granted "allocations" of processor (wall clock) hours for our resources. We avoid shared-use of compute notes due to the difficulties of memory and IO management and the jitter that applications can inflict in shared situations. So, generally, nodes are allocated to a job whether it uses all the processors (8 on our XT) or memory (30+GB) or not. The "accounting" for this comes from roll-up of PBS logs into a general form job record generated on all our platforms. These records feed both our local database and our sponsor's databases tracking utilization. Enforcement of allocation can be done via several techniques: `qsub` submission wrappers, PBS prologue, or PBS hooks.

### 30. Ongoing: Audit CNL state (user perspective)

**Tool**: `pkg/xts/bin/envcnl*`
**cron**: sysmon@login2+peers envcnl (Daily)

The stateless compute nodes of Cray XTs pose a minor challenge for determining if something affecting users has changed with system maintenance. Since compute nodes are all the same (diskless), checking one is adequate since transient conditions like memory leaks *(none seen in CLE 2.2)* can be identified with node health check. The `envcnl` daily cron job *(or on demand)* initiates an aprun to identify any changes on a compute node. The script does use the 'Make CNL more Linux-like' process described earlier (except on ERDC/jade which has enabled DVS/DSL). Simplified slightly, the script gathers and filters:

```
/sbin/drop_caches
cat /proc/meminfo >>out.meminfo;
cat /proc/cpuinfo > out.cpuinfo;
env | sort        > out.env;
ulimit -a         > out.ulimit;
echo              >>out.ulimit;
ulimit -aH        >>out.ulimit
mount             > out.mount;
df -h             > out.df;
uaps -pl          > out.uaps;
for  On in / /var /tmp; do
  uals -RzZAL --mount --fields MogKr $TYPE $On
done              > out.uals
```

Why do we care? This script has identified:

- CLE 2.2 remounting `/tmp,/var 1777` (other+write)
- PBS upgrade voiding a locally customized variable
- presence of lnet_ptrace files stealing memory (clean-up added to NHC)
- profile typo preventing variable being passed to PBS
- regression of a local customization to CNL with an upgrade

As stated in the introduction: **understand what you have** and **know when something changes.**

Since I have accounts on two peers (NAVY and ERDC) and on NICS, I also run this script on those systems. In several cases I reported some minor CNL issues before their local staff (all talented) noticed. More importantly, I became aware when ERDC implemented DVS/DSL with CLE 2.2 and that NICS was trying CSA accounting. These are both features ARSC is interested in, but has not prioritized

effort. In both cases we can get feedback from their efforts/experience by...

### 31. Ongoing: Audit for unsafe user .files

**Tool**: `pkg/dotfiles`
**cron**: sysmon@login2 DotFiles (Hourly)

A long-lived tool at ARSC (pre-dates my 12 year tenure) is the `DotFiles` checker. It was born in the pre-kerberos era when `rsh|rlogin` and password authentication was still common. Kerberos has reusable tickets which saves users from continually re-entering complex passphrases and generated tokens. Previously users implemented `.rhosts` files. With a policy of no account sharing and general security, the user-controlled `.rhosts` were scanned periodically to ensure they were only permitting ARSC hosts and did not allow accounting sharing. The `.rhosts` need is long gone, but the tool was designed for other checks for general security recommendations to avoid group or other write on `.profile`-like files and group or other read on critical security files like `.ssh/id_rsa` private keys. This tool is supplemented by a separate tool to audit .k5login content.

### 32. Ongoing: Audit for excessive disk use

**Tool**: `pkg/0dskhog`
**cron**: sysmon@login2 0dskhog (Daily) also quotas

Managing disk usage is a combination of techniques:

- Filesystem quotas (lustre) on $HOME
- Auto-purge of `/tmp` and `/workdir`
- Daily usage reports on `/workdir`

We have enabled quotas on `/workdir` but not assigned quotas. Instead, we rely on a script reporting heavy users and manage by hand (so far). On a general Linux cluster we have a secondary script monitoring Lustre-OST usage to ensure they are not too imbalanced, to date that has not been necessary on our XT.

### 33. Ongoing: Synchronize passwords where required

**Documentation**: xt_passwords
**Alternative:** nis, ldap, other
**cron**: ids@smw shadow_xtpdate (WeekDays)

Our sponsors strongly discourage static passwords (for good reason). Unfortunately, for nodes which cannot reach kdc|ace authentication and for emergency use when those servers are not available, static passwords are still required for console login and `sudo` authentication. Staff with `sudoers` authority maintain passwords on the SMW. These passwords are replicated to the non-login XT nodes which do not have kdc|ace access. We have no static passwords on our XT login nodes. This is described in more detail in local Cray XT Passwords documentation. Note, we do use ssh Keys for internal cluster networks.

### 34. Ongoing: Audit passwd, shadow, group files

**Alternative:** nis, ldap, other
**cron**: sysmon@dispatch chk_passwd (daily) ids_tools

The ARSC-wide audit of passwd, group, and shadow files is from a `cron` script initiated from a general management system. The requirement is to ensure uid|gid and userid|group entries are appropriately synchronized across all platforms and systems and to retain a history of these for *all* systems. The back-copy of files is rolled into a database for additional reporting and validation.

### 35. Ongoing: Audit .k5login (or authorized_keys) content

**Documentation**: IdsOverview
**cron**: ids@dispatch k5login_check (daily)

Standard kerberos allows user control of the `~/.k5login` files much like the old `.rhosts`. Certainly kerberos is more secure, but with a policy of "no account sharing" (individual accountability) user control of `.k5login` files is still not tenable. The HPCMP release of kerberos at ARSC places `.k5login` files as root owned in a system directory and are managed via the `ids` userid creation and maintenance tools written at ARSC. The `ids` tools are not provided with this paper, typically site practices for user maintenance require site specific tools, but an overview is provided.

With our new Scyld cluster (an academic / non-HPCMP cluster) we are not using kerberos+securid for authentication. Initial implementation was securid-only, but resulted in dissatisfaction in the cycle time for a new code for consecutive ssh or scp requests. Use of ssh `PubkeyAuthentication` provides an alternative, but the default `sshd_config 'AuthorizedKeysFile .ssh/authorized_keys'` provides the same exposure of `.rhosts` or unprotected `.k5login` files for inappropriate account sharing. This opens considerable risk for lost or stole private keys. We are still considering options, but leaning towards something like:

- `sshd_config 'AuthorizedKeysFile /usr/local.adm/keys/%u'` system owned keys
- locally written `ssh-keycopy` which adds or appends user `authorized_keys` adding `from="host(s)"` directives to ensure access is only coming from approved domains and/or the host which generated the key. For example, the following `authorized_keys` only permits access from one host:

```
from="plagueis.arsc.edu" ssh-rsa AAAAB3N...xqHlgQ== kcarlson@plagueis.arsc.edu
```

No technique can completely prevent account sharing, but this mitigates risk of lost or stolen private keys. We have queries out to several forums for other alternatives. In the end, written policies, signed acceptance forms and some trust of users remain integral to security.

### 36. Ongoing: Establish system backups

**Documentation**: xt5Backup -- xthot -- Ext3Restore -- Dumps_admin1
**Alternative:** smwhotbackup xthotbackup
**cron**: backup@admin1 dump.platform (daily)

Backups have been mentioned in several tasks already and take multiple forms for storage controllers, cron, logs, CrayFiles, and `xthotbackup` and `smwhotbackup` prior to system upgrades. Various filesystems including user home directories also require backup. Different sites have different policies for user file backups and restores. Some do backups only for recovery of system failure, others (like ARSC) will restore from incremental or full backups for user errors.

ARSC backups are done across a private network using the sshd-adm established trust onto a storage server. This is a long-standing practice covering multiple platforms as demonstrated in the dump configuration file.

### 37. Ongoing: Audit cron usage

**Documentation**: crontabs
**cron**: sysmon@dispatch chk_crontabs (daily) also cron.deny

Some sites, like ARSC, freely allow users access to `cron`. Some prohibit `cron` or make it by request only. The `crontab` entries are under user control and often `/var` where they live has different backup policies as critical logs are rolled separate from filesystem backups. As such, especially for workstations which are rebuilt if needed and not backed up, users are responsible for their own `crontab` backup and recovery at ARSC. We do log all existing `crontab` entries across all our platforms via the `chk_crontabs` script so we can notify affected users if a system event (e.g., rebuilt workstation) causes loss of a `crontab`. The script also ensures we notice when a `crontab` remains when a user is inactivated.

### 38. Ongoing: Run external security scans

**Alternative:** nessus, STIGs, other
**cron**: sysmon@admin1 csa.xt5 (monthly) HPCMP csascripts

Through our sponsors ARSC is subjected to external security audits. There are two aspects to this:

1. We run "Comprehensive Security Audit" (CSA) scripts on a regular basis which analyze a broad range of recommended settings for DoD and industry *better practices* and feed results back our sponsor who will contact us if they see something which is either inappropriate from their perspective or they want us to formally document as an "accepted risk". The scripts are far less comprehensive than the tasks itemized in this paper and often focus on trees (e.g., `'chmod o-r /etc/aliases'`) rather than the forest (e.g., audit .k5login or audit suid) but they do provide another layer of checks.
2. Annually an external team comes on site for a week to re-review CSA script results, accepted risks, and to examine site security policy and practices, both documented and via staff and user interviews.

Nobody will say that a security audit is a *fun* event, but external reviews are an extremely good idea to ensure a site maintains good practices, trains staff, and stays aware of current threats and avoidance strategies.

### 39. Ongoing: Ongoing system regression tests (benchmarks)

**Technique:** xhpl_all
**Documentation**: xtcompute -- CommonStartUp
**Tool**: `pkg/xts/bin/xhpl_all.ksh`
**Alternative:** various

With any system maintenance (if time permits) and after any significant upgrade, regression testing is appropriate to:

1. validate hardware stability
2. validate system functionality
3. identify changes in system performance characteristics

With log auditing and Check L1 and L0 hardware stability has not been a major issue for our XT5. General system functionality is partially addressed by the wide variety of checks we do have in place and the modules environment allowing non-default usage for longer term pre-testing. Also, the TDS gets updates first and is used to perform testing which can be done not-at-scale.

The performance issues of regression testing are significantly more complicated. One can always re-run the same binaries and compare results. One should select a reasonable sub-set of representative applications. However, the hardware/firmware and OS release only represent a portion of concerns: which compiler, what compiler version, what compiler options. Add in the desire to keep maintenance windows short so the systems can do real work makes building a complete regression testing infrastructure a challenge. There are manual efforts to re-run a representative applications periodically as validation and those are run after a major upgrade, but the effort remains manual at ARSC.

Our Cray engineer proactively identifies high single-bit error nodes to preventatively admin down. We do launch single-node xhpl on suspect nodes which have one of the "transient" error conditions before returning them to service. We also launch a cluster wide all-single-node xhpl run after maintenance:

```
boot: ssh login2 "cd /lustre/small/hpl; ./xhpl_all.ksh -a xt5@arsc.edu -n"
```

This generates a summary mail message and retains a log:

```
login1: tail -4 /lustre/small/hpl/xhpl_all.log
# 20100210.1319 432=nodes range=6.276e+01,6.306e+01 passed=432  Nids 10,11,12,13,132,133,...
# 20100310.1220 432=nodes range=5.967e+01,6.347e+01 passed=431 low=1 Nids 4,5,6,7,8,9,...
# 20100401.1050  50=nodes range=6.322e+01,6.345e+01 passed=50   Nids 258
# 20100405.1447 432=nodes range=6.285e+01,6.348e+01 passed=432  Nids 4,5,6,7,8,9...
```

The 04-01 test was a 50 iteration test on nid00258 before returning it to service after a deadlock. We have tuned the xhpl run for 22 minutes which only uses 22GB memory because we do not want to devote the extra 18 minutes it would take to stress all 31GB of usable memory. As a test, this is flawed in only checking CPU and memory and not the interconnect or IO. Note the 03-10 run had a low node, but log checks and re-running showed no problem. What can one do?

ARSC has never fully achieved the level of regression testing we would like. Efforts have mostly been manual-on-demand only automated on one of our platforms (not our XT5). Perphaps this is an area somebody can help us grow?

### 40. Ongoing: Stay aware of what peers are doing

**Documentation**: xtModulesDefault
**cron**: sysmon@login2+peers module_chk (Daily) also Xtreme CUG

As stated with audit CNL state (`envcnl`), we have benefited in seeing what other sites are doing. Any system of this complexity is impossible to fully grasp by an individual, by any single site, or by the vendor. This is the whole purpose of CUG, finding ways share information easily can only help. Besides `envcnl` we have the `module_chk` script running on our peers at ERDC and NAVY and on NICS/kraken. The original impetus was driven by user-needs (baseline configuration) to track and understand the programming environment in use at ARSC, ERDC, and NAVY. We share a user base who can become more than a bit irritated if they cannot run successfully when the HPCMP allocation process moves them to a different center.

When Patricia Kovatch offered me an account on NICS/Kraken at CUG in Atlanta, I jumped on it:

- NICS (University of Tennessee) has academic/research ties much like ARSC (University of Alaska).
- Seeing a non-HPCMP peer site helps ensure we (HPCMP as well as ARSC) are not blind to possibilities unrecognized by a DoD perspective.

We have made a start at this. We need to find ways to go further and possibly involve additional sites.

The `module_chk` jobs are an extension of audit module state already discussed. For example:

```
login1: cd /usr/local/etc/modules
login1: modchk -f "`ls modules.*`" -name pgi # what pgi versions are out there?

Cray-XT

E A N E A N :    6 hosts
R R I R R A :    1 total modules
D S C D S V :   18 total versions
C C S C C Y :
- - - - - - :
s p k j o e :
a i r a g i :
p n a d n n :
```

```
. . . . . . :
0 0 0 0 0 0 :
2 4 4 2 4 4 :
2 0 0 2 0 0 :
5 8 8 5 8 8 : * default; + available; - missing

+ + + + + + :pgi 10.0.0
- - + - - - :pgi 10.1.0
- - + - - * :pgi 10.2.0
- - + - - + :pgi 10.3.0
- - - - - + :pgi 6.2.5
- - - - - + :pgi 7.0.7
- - - - - + :pgi 7.1.6
- - - - - + :pgi 7.2.2
- + - - + + :pgi 7.2.3
- - + - - - :pgi 7.2.5
- - - - - + :pgi 8.0.1
- - - - + - :pgi 8.0.2
* - + + - + :pgi 8.0.5
- * + - * + :pgi 8.0.6
- - + - + + :pgi 9.0.1
- - + - + + :pgi 9.0.2
- - + - - + :pgi 9.0.3
- + * * + + :pgi 9.0.4

1 of 18 modules installed on all 6 hosts
```

We discovered early that NAVY is much more aggressive in promoting default modules with some core users who aggressively want bug fixes. We have a more conservative user base who resist compiler changes. Both approaches are valid. What we also see above is NICS and NAVY have 10.3.0 installed and we can contact them or pre-test an issue there if we are not ready to perform a non-default install of that version.

---

## Conclusion (top -- intro -- index -- general -- install -- CNL -- ongoing -- conclusion)

That is a lot of work! I have described ***what*** and ***why*** for managing large complex systems providing ***how*** from ARSC as examples. We do not claim *"best practices"* for the above, most are *"good practices"* and often constrained by the complexity of systems and staff resources available to make them even better. The tasks above do have contributions from numerous **ARSC** staff members and concepts borrowed from at least **AWE**, **ERDC**, **NAVY**, **NERSC**, **NICS**, **ORNL**, **Cray**, and the **HPCMP**. If we (**CUG**) collectively share our practices we will make all of our systems better. The concept of this paper started as "25 tips for Cray XT systems". It was quickly apparent there were many other things to mention and could go well beyond the 80 tasks above if I had more time and better exploited the knowledge of other staff for the *many* areas I rely on them: lustre, hierarchical storage, network, database, PBS (reservations, hooks, scheduling), programming environment, software modules, optimization, performance tuning, regression tests, allocation management, policies, ....

## Acknowledgments

This work was supported in part by a grant of HPC resources from the Arctic Region Supercomputing Center at the University of Alaska Fairbanks as part of the Department of Defense High Performance Computing Modernization Program.

## About the Author

Kurt Carlson has worked with many facets of computer services at the University of Alaska since 1975 and for the last 12 years with ARSC. Kurt still calls himself a Systems Programmer, but has not written any assembly code since the Vax/VMS and IBM/MVS eras and feels fortunate when he finds an opportunity to write some C around more mundane tasks of System Administration and Project Management. Kurt still uses "University of Alaska" logo punch cards as note cards and his wife tolerates the four remaining computers with seven operating systems at home.

---

Last update: 2010-05-11 kac