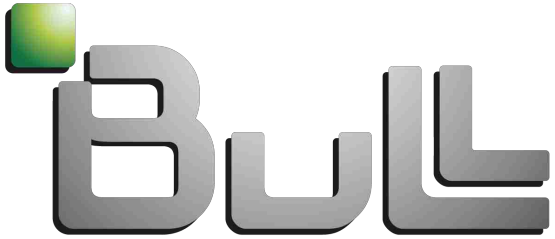


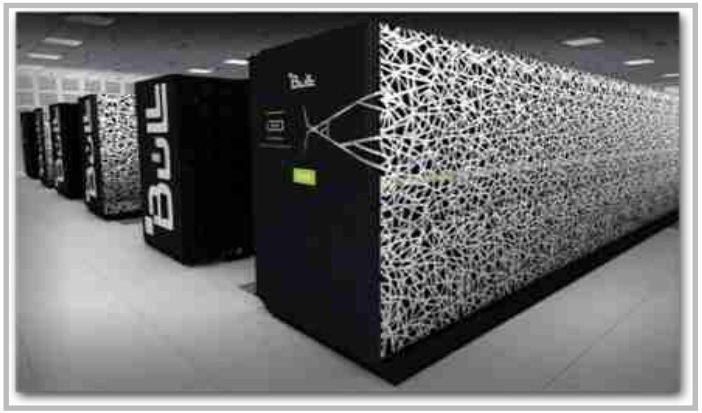
Lustre support in a Petaflopic environment

A.Louvet, B.Faccini

LIBERATE IT

Who is who ?

- On-site support engineers at Bull
- supporting high-end systems such as :
  - Tera-100 (Lustre 2.0)
  - TGCC (Lustre 2.0)
  - AWE (Lustre 1.6)
  - CCRT/GENCI (Lustre 1.4/1.6)
- Public and classified customers

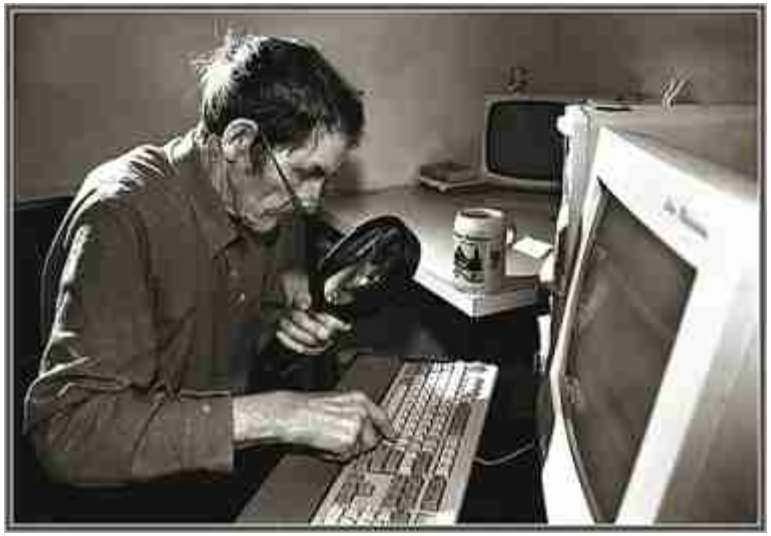


Caveats and difficulties

- Distributed architecture
- Huge numbers
  - Clients (Tera-100 is ~4324 Nodes)
  - OSSs (Tera-100 private storage has ~72)
  - OSTs (Tera-100 private storage has ~1080)
  - MDS hosting billions of inodes
- Tons of logs
- Large mix of many users applications exercising all of Lustre features
- Many software stacks (lvm/md, IB, eth, ...)
- Lustre sources becoming more and more complex, object oriented, lot of callbacks, ...
- Petaflopic lab machines are not that common

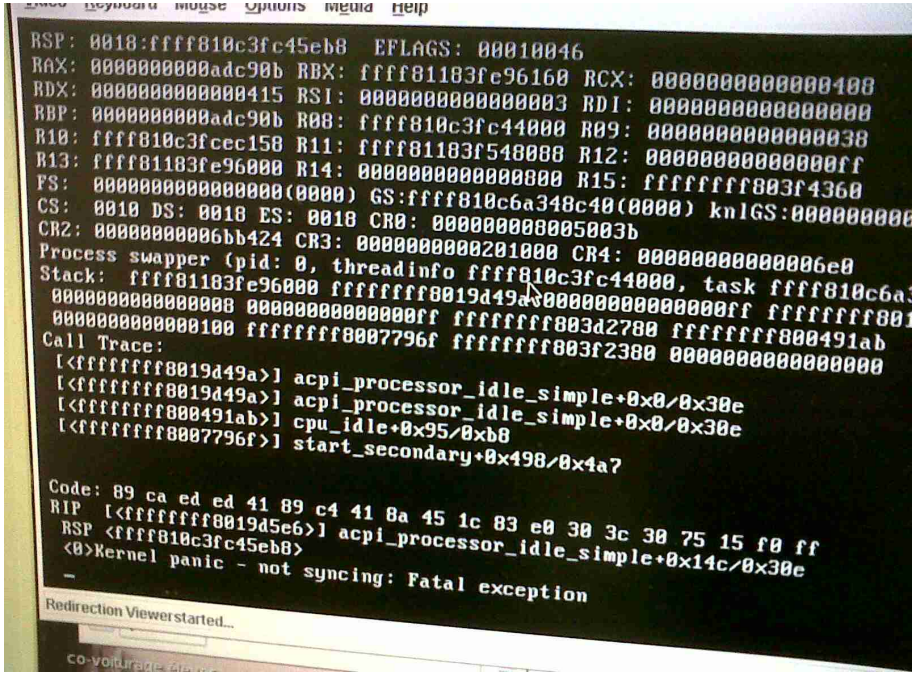
Problems families

- Right qualification of the problem is critical
  - LBUG
  - Crash (Oops, BUG)
  - Hang
  - Slow response triggering watchdog/time-out
  - Performance



Tools we use

- /var/log/*
- lctl dk
- top, slabtop
- sysrq [l, t, m, c]
- crash, makedumpfile
- oprofile, perf
- console/conman
- cscope / vi
- debugfs, tune2fs/tunefs.lustre
- fsck
- camera !



System configuration

- /proc/sys/kernel/sysrq
  - Make sure the system accept sysrq commands through /proc/sysrq-trigger or console
- panic_on_lbug, panic_on_oops
- NMI
- Console management
  - Usefull for sending sysrq commands when remote connection is not working anymore
  - Console recording is helfull (conman)

Kdump configuration

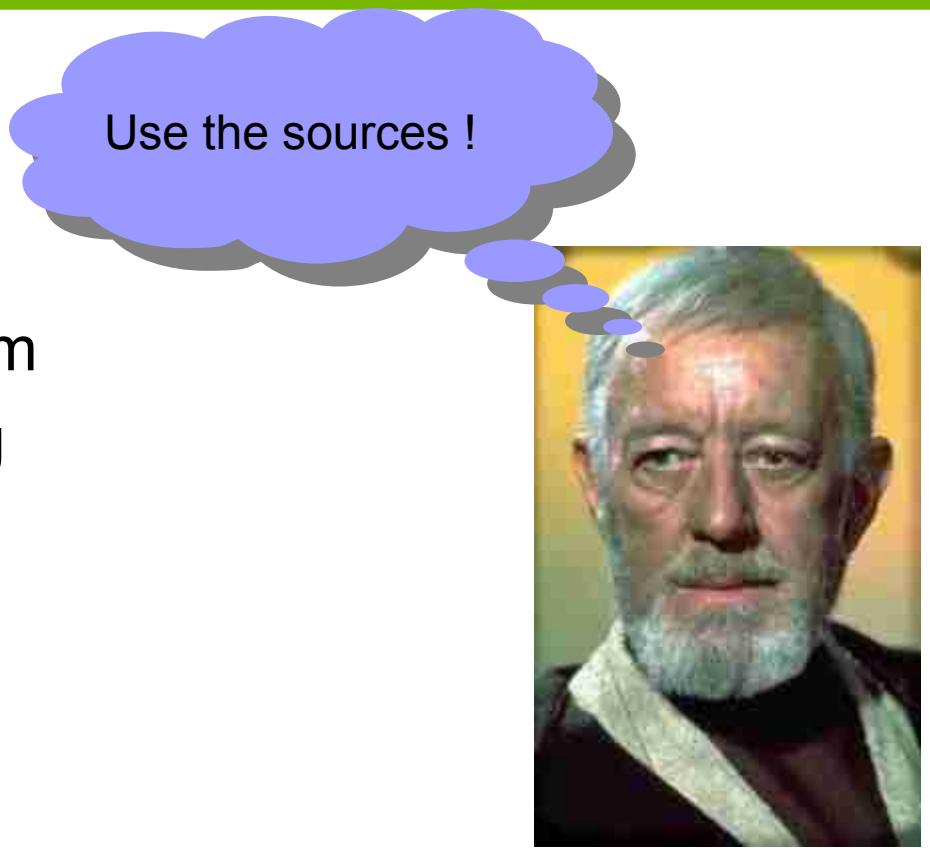
- Make it ready before something wrong cames up !
- Boot loader
  - crashkernel=auto
  - crashkernel=256M@64M
- Service turned on
  - chkconfig kdump on
- Need space to store dump
  - Compression
  - Filtering
- makedumpfile is the right tool to reduce dump size

crash

- Requiere to have the right kernel-xxx.debuginfo installed
- crash mapfile namelist dumpfile
  - crash /boot/System.map-$(uname -r) /usr/lib/debug/lib/modules/$(uname -r)/vmlinux
- By default only have knowledge on core kernel
  - mod -S is needed to get access to lustre infos (accept alternate location for .ko files)

Source of information

- Lustre & kernel sources
- Lustre manual
- [jira|wiki].whamcloud.com
- [bugzilla|www].lustre.org
- newsgroups



Caveats

- User expect the filesystem to be stable and available, if not they want it to be back on line ASAP.
- Collecting data for troubleshooting is fundamental
- Each case is unique
  - No magical rules.
  - Need a lot of experience and knowledge about how everything works together.
- Some issues are easier to troubleshoot than others
  - LBUG/OOPS has a clear starting point
  - Hang is less easier
  - Performance

Case Study 1

LIBERATE IT

Recovery issue

```
crash> sys
      KERNEL: /lib/debug/lib/modules/2.6.32-71.24.1.el6.Bull.23.x86_64/vmlinux
    DUMPFILE: vmcore  [PARTIAL DUMP]
        CPUS: 32
        DATE: Wed Aug  3 15:17:48 2011
      UPTIME: 01:10:42
LOAD AVERAGE: 1.05, 0.52, 0.20
       TASKS: 1513
    NODENAME: mds
     RELEASE: 2.6.32-71.24.1.el6.Bull.23.x86_64
     VERSION: #1 SMP Tue Jul 5 17:13:50 CEST 2011
     MACHINE: x86_64  (2000 Mhz)
      MEMORY: 128 GB
       PANIC: "kernel BUG at fs/jbd2/transaction.c:1030!"
```

… more

```
crash> bt
PID: 24472  TASK: ffff8808556011c0  CPU: 22  COMMAND: "tgt_recov"
 #0 [ffff88083370a9d0] machine_kexec at ffffffff8102e77b
 #1 [ffff88083370aa30] crash_kexec at ffffffff810a6cd8
 #2 [ffff88083370ab00] oops_end at ffffffff8146aad0
 #3 [ffff88083370ab30] die at ffffffff8101021b
 #4 [ffff88083370ab60] do_trap at ffffffff8146a3a4
 #5 [ffff88083370abc0] do_invalid_op at ffffffff8100dda5
 #6 [ffff88083370ac60] invalid_op at ffffffff8100cf3b
    [exception RIP: jbd2_journal_dirty_metadata+269]
    RIP: ffffffffa00518ed  RSP: ffff88083370ad10  RFLAGS: 00010246
    RAX: ffff881831c8b8c0  RBX: ffff881834107468  RCX: ffff8808512adc90
    RDX: 0000000000000000  RSI: ffff8808512adc90  RDI: 0000000000000000
    RBP: ffff88083370ad30   R8: 2010000000000000   R9: f790d737baaf2402
    R10: 0000000000000001  R11: 0000000000000040  R12: ffff8818343606d8
    R13: ffff8808512adc90  R14: ffff880859b81800  R15: 0000000000002000
    ORIG_RAX: ffffffffffffffff  CS: 0010  SS: 0018
 #7 [ffff88083370ad38] __ldiskfs_handle_dirty_metadata at ffffffffa04bb3fb [ldiskfs]
 #8 [ffff88083370ad78] fsfilt_ldiskfs_write_handle at ffffffffa09bede7 [fsfilt_ldiskfs]
 #9 [ffff88083370ae28] fsfilt_ldiskfs_write_record at ffffffffa09bf0fe [fsfilt_ldiskfs]
#10 [ffff88083370aea8] llog_lvfs_write_blob at ffffffffa05a018c [obdclass]
#11 [ffff88083370af58] llog_lvfs_write_rec at ffffffffa05a1732 [obdclass]
#12 [ffff88083370b038] llog_cat_current_log.clone.0 at ffffffffa059e14f [obdclass]
#13 [ffff88083370b118] llog_cat_add_rec at ffffffffa059e86a [obdclass]
#14 [ffff88083370b198] llog_obd_origin_add at ffffffffa05a51a6 [obdclass]
#15 [ffff88083370b1f8] llog_add at ffffffffa05a5381 [obdclass]
#16 [ffff88083370b268] lov_llog_origin_add at ffffffffa089a0cc [lov]
#17 [ffff88083370b318] llog_add at ffffffffa05a5381 [obdclass]
#18 [ffff88083370b388] mds_llog_origin_add at ffffffffa09d46f9 [mds]
#19 [ffff88083370b408] llog_add at ffffffffa05a5381 [obdclass]
#20 [ffff88083370b478] mds_llog_add_unlink at ffffffffa09d4de4 [mds]
#21 [ffff88083370b4f8] mds_log_op_orphan at ffffffffa09d5229 [mds]
#22 [ffff88083370b578] mds_lov_update_objids at ffffffffa09de7ef [mds]
#23 [ffff88083370b638] mdd_lov_objid_update at ffffffffa09f5cb2 [mdd]
#24 [ffff88083370b648] mdd_create_data at ffffffffa0a02c91 [mdd]
…
```

Symptom

- System crash at the end of recovery
- After a reboot and lustre mount, the system start recovery and …. crash again ! Death loop !
- A fsck didn't fix the problem

Basic analyse

- the system is asserting during jbd2 transaction add into a journal buffer :

```
J_ASSERT_JH(jh, handle->h_buffer_credits > 0);
```

- All credits in the transaction were used (h_buffer_credits = 0)
- The reservation was missing some actions, the buffer was not allocated big enough

Short time workaround

- Phone start ringing, users are waiting for the cluster !
- Trying to fix the issue will definitively take too much time.
- The problem comes up at the end of recovery, during orphan cleaning, so why not just clean them manually
  - `mount -t ldiskfs /dev/sdxx /mdt`
  - `mv PENDING PENDING.old`
  - `mkdir PENDING`
  - `umount /mnt`
  - `mount -t lustre /dev/sdxx /mdt`
- Will generate acceptable inconsistencies.
- Now have time to properly fix the root issue (LU-601)

Case Study 2

LIBERATE IT

MDS hang

- MDS stop answering from a lustre perspective
- Ping doesn't work
- Interactive console (bash login) doesn't work too, only sysrq answer
  - [alt] sysrq -l show that all cpus were fighting on a spinlock
  - livelock

    => NMI

Crash analysis

- No more free memory available
  - kmem -i says 99% of memory was used
  - kmem -s show that most of memory is in ldiskfs_inode_cache
- All cores are running a kiblnd thread
  - 31 cores out of 32 were running a kiblnd_sd_xx thread
    - Spinlock to access a buffer allocation code (kiblnd_pool_alloc_node)
  - 1 thread in kiblnd_connd
    - Same spinlock than the others threads
- Having all cores spinlock'ing is not an optimal solution

First attempt

- Reduced the number of kiblnd_sd_xx threads to keep some cores available for others threads to make progress.
- Observe how the ldiskfs_inode_cache slab cache grow during normal production
  - Attempt to shrink the cache doesn't work (echo 3 > /proc/sys/vm/drop_caches) , this slab is not shrinked
- Activating lustre debug (lctl dk) show that during reclaim, number of objects doesn't move against shrinker calls

  ... Shrink 128 objects

  ... 22857100 objects cached

- MDS finaly hang again

Second analyse

- The shrink is done into lu_site_purge
  - the kernel vm manager call lu_cache_shrink with nr_to_scan set to SHRINK_BATCH (kernel hard coded macro set to 128)
  - then call lu_site_purge with nr set to (at most) nr_to_scan.
  - lu_site_purge scan the nr first objects of the lru list, avoiding all objects with a h->loh_ref > 0 (busy objects)
- A scan of the first 100000 entries show that
  - The first 128 have a h->loh_ref > 0
  - There is 1500 objects with a reference count > 0

A way to the first workaround

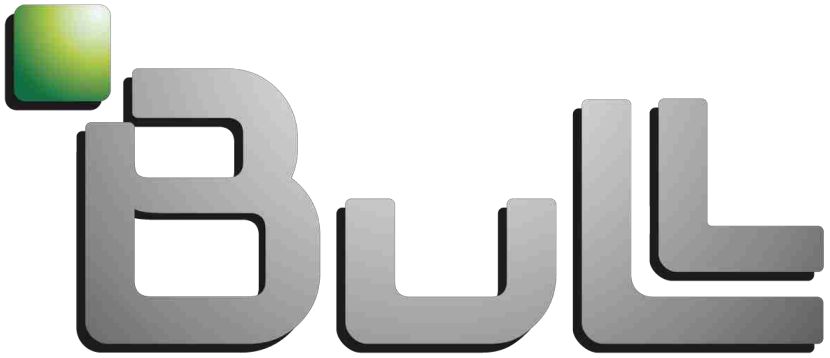
- we thought about 4 possible ways to work-around this issue :
  - simply raise the "SHRINK_BATCH" hard-coded value used in shrink_slab() to invoke the shrinkers
  - modify lu_site_purge to free nr objects in place of scanning nr objects
  - parse only "SHRINK_BATCH" objects but move each busy object encountered at the end of the LRU, to allow for non-busy objects to be found during the next search
  - move busy objects out of the lru

Final step

- Temporary workaround based on the third solution was implemented
  - Easier
  - Quicker
  - Less risky
- Jira 685 opened to get the cleaner solution

Conclusion

- Have to short time to return on production
- Have to provide temporary workaround
- Sometimes have limited access to traces or external assistance (classified sites)
- Stay on the official release as much as we can, doesn't want to have a local fork.



Architect of an Open World™

LIBERATE IT