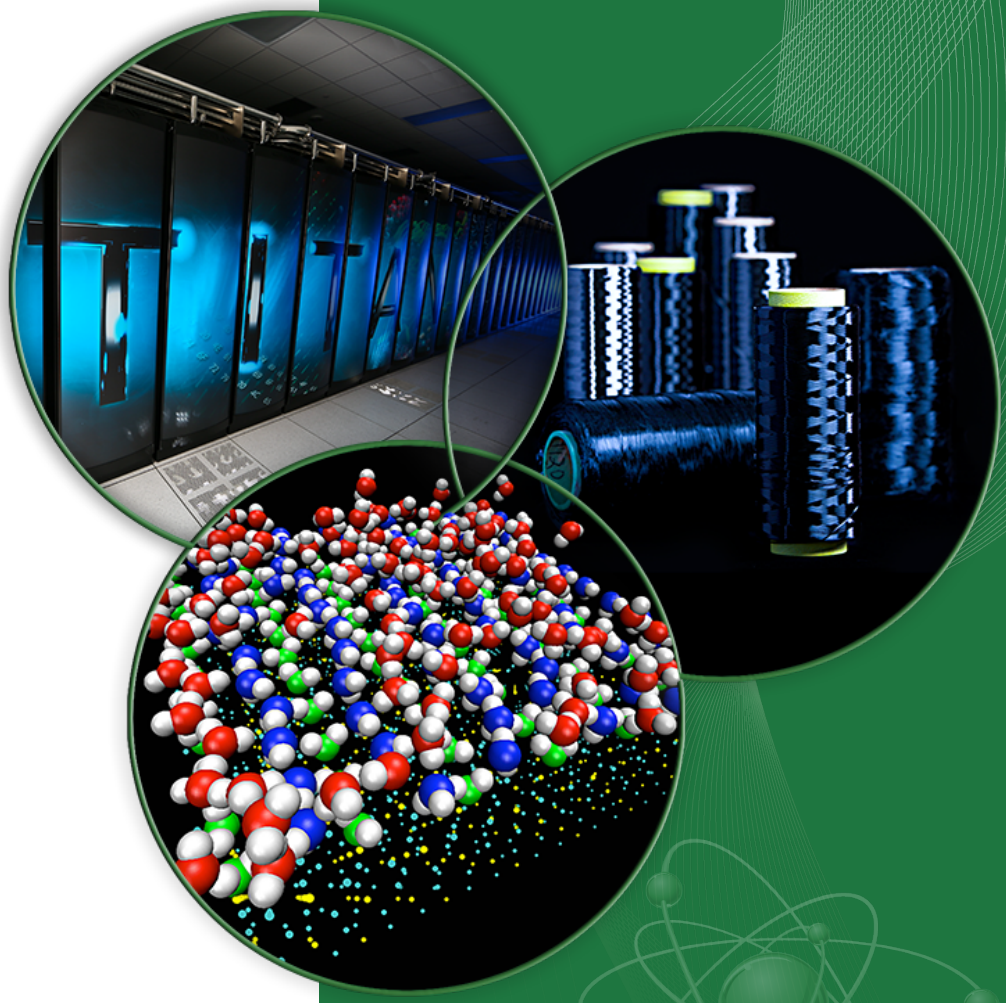
Oak Ridge National Laboratory

Computing and Computational Sciences Directorate



Lustre Crash Dumps And Log Files

Jesse Hanley
Rick Mohr
Sarp Oral
Michael Brim
Nathan Grodowitz
Gregory Koenig
Jason Hill
Neena Imam

November 2015

ORNL is managed by UT-Battelle
for the US Department of Energy

OAK RIDGE National Laboratory

Overview

- Crash Dumps
  - What is a crash dump
  - How does it fit into the Lustre flow
  - How to configure and generate crash dumps
  - How to use crash dumps
- System and Lustre debug log files
  - Standard messages
  - Tuning messages

Crash Dumps

Definition:

A portion of the contents of volatile memory (RAM) that is copied to disk whenever the execution of the kernel is disrupted

Potential Causes:

- Kernel Panic*
- Non Maskable Interrupts (NMI)
- Machine Check Exceptions (MCE)
- Hardware failure
- Manual intervention

* Some automatic, others only when explicitly activated

Relevance to Lustre

- With both clients and servers, Lustre kernel modules start multiple threads to process I/O requests
- Problems encountered during kernel execution can generate exceptions (known as LBUGs)
- System reboots are necessary to recover from LBUGs (the processing threads lock to help prevent further damage to the file system)
- Crash dumps provide a snapshot at the moment of the LBUG, allowing for analysis
- Analysis can help pinpoint code problems

Choices for dealing with LBUGs

- The default behavior is to lock the calling thread when encountering LBUGs, requiring a node reboot to resume normal operation
- Pass information about the LBUG to a custom tool
  - `lctl set_param upcall=/path/to/binary`
  - Called with 4 parameters: the string “LBUG”, the file where the LBUG occurred, the function name, and the line number in the file
  - The calling thread remains locked in this scenario
  - This method is not commonly used
- Create a kernel panic
  - `lctl set_param panic_on_lbug=1`

Configuring Crash Kernel

- Need a separate kernel (the “crash kernel”) to use when a kernel panic is detected
  - Enabling `panic_on_lbug` will generate kernel panics when an LBUG is encountered
  - Using `kexec`, this crash kernel executes, allowing access to the original kernel’s memory. Otherwise, this information would be inaccessible
- RHEL/CentOS/SUSE kernels support creating this kernel by adding a parameter to the boot kernel
- `crashkernel` option
  - Ex: `crashkernel=192M`
  - Ex: `crashkernel=auto`

Configuring Kdump

- The `kdump` tool handles automatically calling `kexec`, switching to the crash kernel
- Install the `kexec-tools` package
- Configure `kdump` through the `/etc/kdump.conf` file
  - By default, core dumps are stored in `/var/crash/`
  - Can specify an alternate location, such as an nfs mount:
    - `nfs my.nfsserver.example.org:/path/to/export`
  - Or an scp target (ssh must be configured):
    - `ssh user@my.server.example.org:/dest/path`
    - By default, uses sshkey at `/root/.ssh/kdump_id_rsa`
  - Will create a directory that includes the hostname and date, then write `vmcore` and `vmcore-dmesg.txt` files

Configure Kdump to use makedumpfile

- The `makedumpfile` utility dumps a kernel's contents into a file
  - As part of the crash workflow, `kdump` can call `makedumpfile` to generate a file with the original kernel's contents.
  - This file can then be debugged at some later point.
- Included within `kexec-tools` package
- Configured within `/etc/kdump.conf`:

```
core_collector makedumpfile <options>
```

makedumpfile Options

Make sure to pass the `-c` option (to enable compression)

The `dump_level (-d)` option:

- Allows omission of certain pages from the dump
- Is the sum of the chosen filtering levels:

Option	Description
1	Zero Pages
2	Cache Pages
4	Cache Private
8	User Pages
16	Free Pages

For example, to try including everything but free pages, but fall back to only collecting cache information:

```
-d 16,25
```

The `message_level` param allows the user to configure which outputs are included

- Follows a similar sum/bitmask setup as the `dump_level` parameter

Option	Description
1	Progress Indicators
2	Common Messages
4	Error Messages
8	Debug Messages
16	Report Messages

Defaults to 7: progress indicators, common messages, and error messages

Full kdump configuration example

`/etc/kdump.conf`:

```
nfs my.server.example.org:/data/kdump/dumps
core_collector makedumpfile -d 16 -c message_level 15
```

This will cause `makedumpfile` to generate a dump that contains the user, cache, and zero pages (all memory except free pages). Additionally, the kernel dump will include progress indicators, common messages, error messages, and debug messages.

Starting Kdump

- Ensure the kdump service is enabled:
  - `chkconfig kdump on` (RHEL/CentOS 6)
  - `systemctl enable kdump` (RHEL/CentOS 7)
- Start the service if not started already:
  - `service kdump start` (RHEL/CentOS 6)
  - `systemctl start kdump` (RHEL/CentOS 7)
- Verify that kdump is operational
  - `service kdump status` (RHEL/CentOS 6)
  - `systemctl status kdump` (RHEL/CentOS 7)

Using Crash

- The `crash` utility allows a user to interact with the contents of a kernel dump via a prompt
- Install `crash` package
- Use the appropriate kernel debuginfo and call the crash command:
  - `crash /usr/lib/debug/modules/kernel-${version}/vmlinux /path/to/vmcore`
- If unsure of the kernel version of the vmcore:
  - `strings vmcore | head`

Crash commands

Within the `crash` prompt, there are several useful commands for debugging:

```
crash> sys
      KERNEL: /usr/lib/debug/lib/modules/2.6.32-example.x86_64/vmlinux
    DUMPFILE: ./vmcore  [PARTIAL DUMP]
        CPUS: 12
        DATE: Sat Oct 17 16:48:21 2015
      UPTIME: 2 days, 03:47:00
LOAD AVERAGE: 161.82, 184.95, 219.81
       TASKS: 580
    NODENAME: host1
     RELEASE: 2.6.32-example.x86_64
     VERSION: #1 SMP Wed Aug 12 13:19:57 EDT 2015
     MACHINE: x86_64  (2499 Mhz)
      MEMORY: 256 GB
       PANIC: "[186653.379255] Kernel panic - not syncing: Watchdog detected hard LOCKUP on cpu 0"
crash>
```

Crash Commands

- "`bt`" – Display a kernel stack backtrace
- "`log`" – Dumps the kernel log_buf contents in chronological order
- "`ps`" – Displays process status for selected, or all, processes in the system
- "`files`" – Displays information about open files of a context
- "`dev`" – Dumps character and block device data

Making sense of crash data

- Information from `crash` can be hard to digest
- Not always immediately helpful
- If a reproducer is found, multiple crash dumps can be compared to find a common pattern (usually through backtraces)
- Often most useful to vendors/developers

Lustre Syslog Messages

- As Lustre software code runs on the kernel, single-digit error codes display to the application; these error codes are an indication of the problem.
- These messages are sent to syslog and are available in `/var/log/messages` by default.
- Messages that contain "`LustreError`", "`LBUG`", or "`assertion failure`" can be indicative of problems.
- Also, check the console log (`dmesg`)

Lustre Debugging Logs

- Increasing verbosity can help when a problem occurs
  - The overhead of enabling debugging can also hide problems related to race conditions
- Increasing verbosity can also decrease performance
- Controlled with:
  - `lctl {set,get}_param debug` or
  - `sysctl lnet.debug`
- Use "+" and "-" syntax to append/remove individual items instead of overloading entire setting
- The default log size is 5MB per CPU, but can be increased:
  - `lctl set_param debug_mb=1024`
  - Logs use an in-memory ring buffer, so when they fill up, the oldest records will be discarded

```
[root@host1 ~]# lctl get_param debug
debug=ioctl warning error emerg ha config console
[root@host1 ~]# sysctl lnet.debug
lnet.debug = ioctl warning error emerg ha config console
[root@host1 ~]# lctl set_param debug="+neterror"
debug=+neterror
[root@host1 ~]# lctl get_param debug
debug=ioctl neterror warning error emerg ha config console
[root@host1 ~]# sysctl lnet.debug
lnet.debug = ioctl neterror warning error emerg ha config console
```

Using lctl for debug logs

`lctl dk <filepath>` will dump the contents of the debug logs to `<filepath>`

Sample lctl run:

```
[root@host ~]# lctl
lctl > debug_kernel /tmp/lustre_logs/log_all
Debug log: 324 lines, 324 kept, 0 dropped.
lctl > filter trace
Disabling output of type "trace"
lctl > debug_kernel /tmp/lustre_logs/log_notrace
Debug log: 324 lines, 282 kept, 42 dropped.
lctl > show trace
Enabling output of type "trace"
lctl > filter portals
Disabling output from subsystem "portals"
lctl > debug_kernel /tmp/lustre_logs/log_noportals
Debug log: 324 lines, 258 kept, 66 dropped.
```

Debug daemon and lctl

- The debug daemon periodically dumps the ring buffer contents to disk in binary form
  - The contents need to be converted to human readable
- Example of using `debug_daemon` and interactive `lctl` to dump debug files to a 40 MB file:

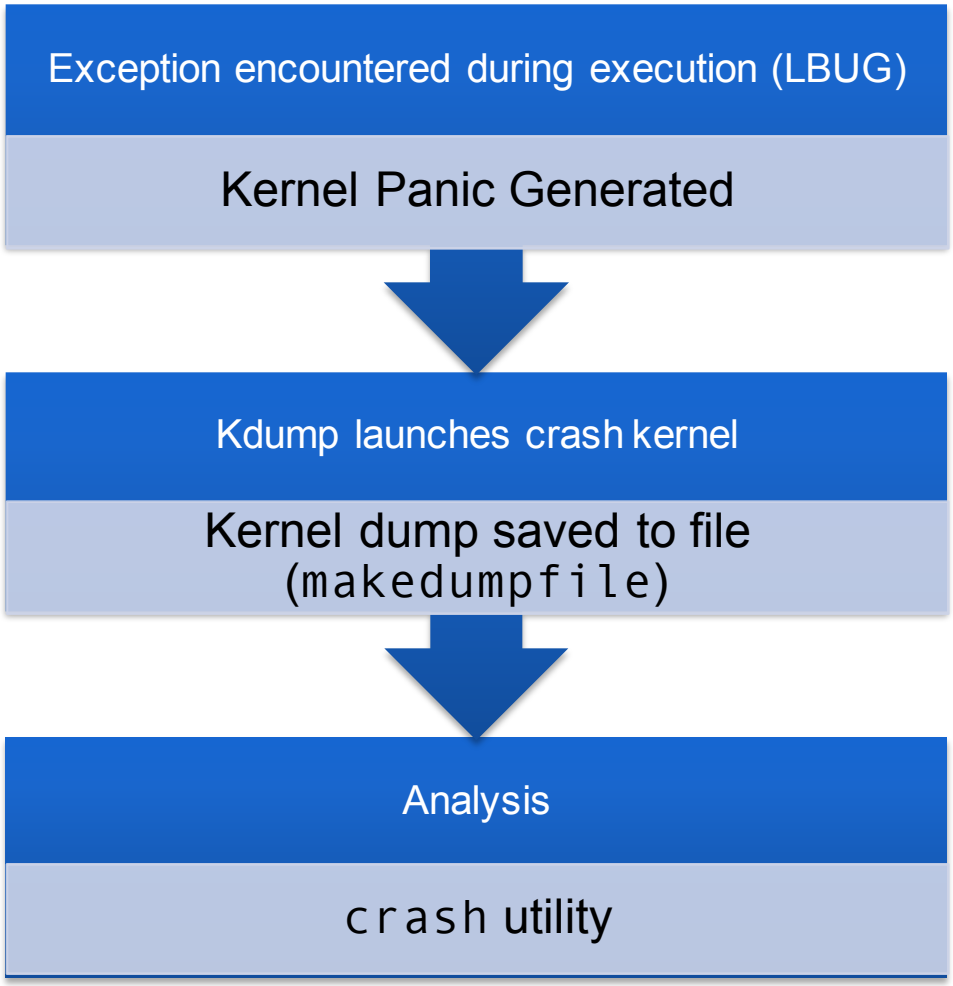
```
lctl
lctl > debug_daemon start /var/log/lustre.40.bin 40
<run filesystem operation(s) to debug (outside of the lctl session)>
lctl > debug_daemon stop
lctl > debug_file /var/log/lustre.40.bin /var/log/lustre.log
```

- Optionally, start a daemon with an unlimited file size:

```
lctl > debug_daemon start /var/log/lustre.bin
```

Conclusion

Crash Cycle



Debug information

- System logs
- Lustre debugging logs
  - Can specify debug targets
  - Can specify how much data to collect

Acknowledgements

This work was supported by the United States Department of Defense (DoD) and used resources of the DoD-HPC Program at Oak Ridge National Laboratory.