Online Patching for Lustre

Andriy Skulysh
Arshad Hussain

LUG 2017
Bloomington, Indiana

Agenda

Kpatch

Kpatch Apply Process

Kpatch Limitations

Lustre Issues Found

Lustre Changes Completed

Test Results

Open Issues and Future Work

Kpatch

Introduced in 2014 by RedHat

RedHat RHEL7, there are patches for CentOS 6

Kernel part and tools are GPLv2 licensed

Automated binary diff

‘ftrace’ + stop_machine()

Single switching point

Compatible with kdump/crash

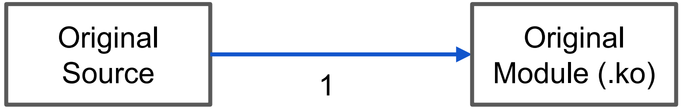
- Replacement functions are normal functions.

User load/unload hooks

- Initialize modified global/static variables
- Perform data modification

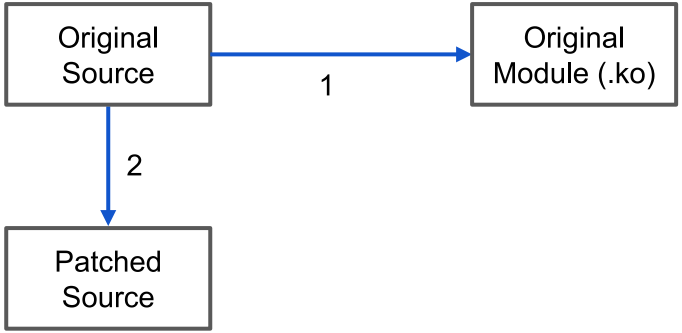
Kpatch Apply Process

1. Original source and module
2. Patch source
3. kpatch-build - prepares loadable patch module
   a. Compiles original and patched source with -ffunction-sections -fdata-sections
   b. Generates binary diff
   c. Results in a patch_module
4. kpatch-load - Load module into running kernel.
   a. ‘stop_machine()’ freezes all tasks
   b. Checks for patching function in all stacks
   c. Do actual functions replacement
   d. Runs user hooks
5. Finally New Patch Module Running
6. kpatch-load --unload - Uninstall module



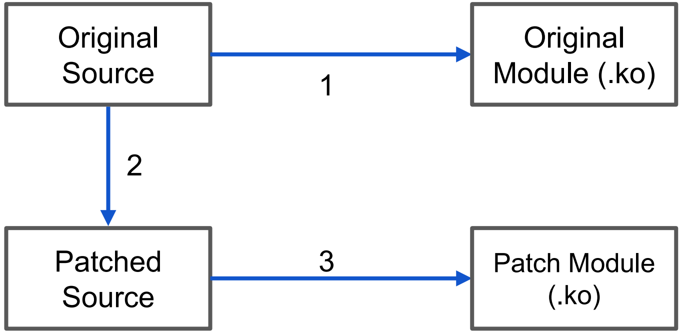
Kpatch Apply Process

1. Original source and module
2. Patch source
3. kpatch-build - prepares loadable patch module
   a. Compiles original and patched source with -ffunction-sections -fdata-sections
   b. Generates binary diff
   c. Results in a patch_module
4. kpatch-load - Load module into running kernel.
   a. ‘stop_machine()’ freezes all tasks
   b. Checks for patching function in all stacks
   c. Do actual functions replacement
   d. Runs user hooks
5. Finally New Patch Module Running
6. kpatch-load --unload - Uninstall module



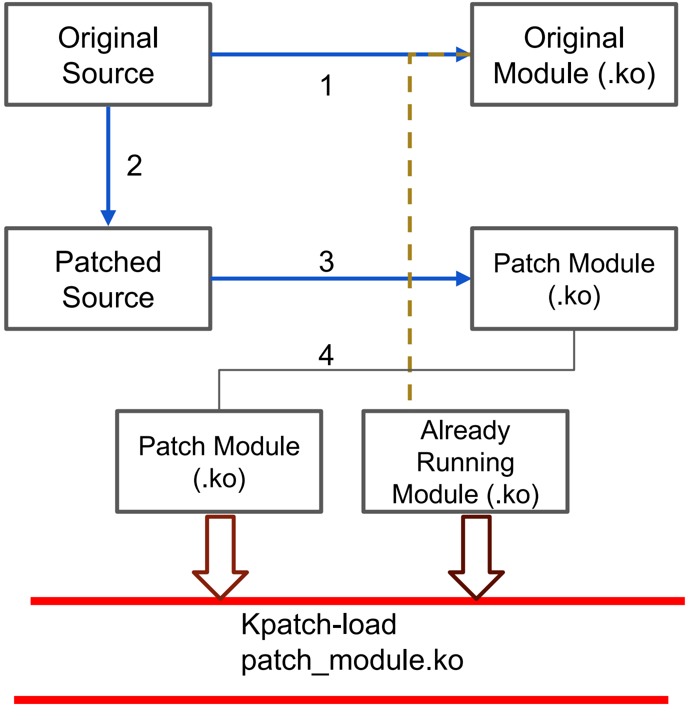
Kpatch Apply Process

1. Original source and module
2. Patch source
3. kpatch-build - prepares loadable patch module
   a. Compiles original and patched source with -ffunction-sections -fdata-sections
   b. Generates binary diff
   c. Results in a patch_module
4. kpatch-load - Load module into running kernel.
   a. 'stop_machine()' freezes all tasks
   b. Checks for patching function in all stacks
   c. Do actual functions replacement
   d. Runs user hooks
5. Finally New Patch Module Running
6. kpatch-load --unload - Uninstall module



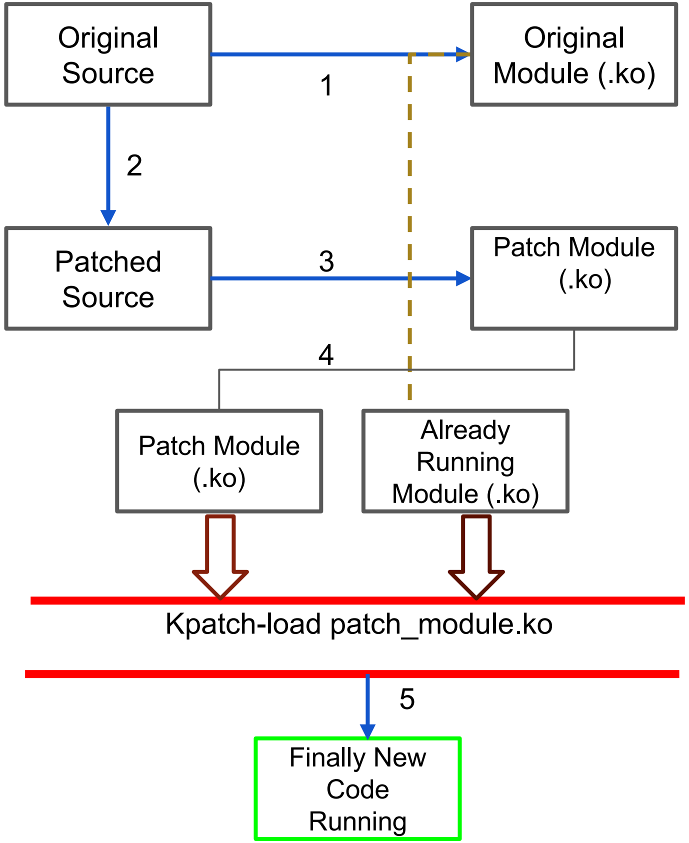
Kpatch Apply Process

1. Original source and module
2. Patch source
3. kpatch-build - prepares loadable patch module
   a. Compiles original and patched source with -ffunction-sections -fdata-sections
   b. Generates binary diff
   c. Results in a patch_module
4. kpatch-load - Load module into running kernel.
   a. ‘stop_machine()’ freezes all tasks
   b. Checks for patching function in all stacks
   c. Do actual functions replacement
   d. Runs user hooks
5. Finally New Patch Module Running
6. kpatch-load --unload - Uninstall module



Kpatch Apply Process

1. Original source and module
2. Patch source
3. kpatch-build - prepares loadable patch module
   a. Compiles original and patched source with -ffunction-sections -fdata-sections
   b. Generates binary diff
   c. Results in a patch_module
4. kpatch-load - Load module into running kernel.
   a. ‘stop_machine()’ freezes all tasks
   b. Checks for patching function in all stacks
   c. Do actual functions replacement
   d. Runs user hooks
5. Finally New Patch Module Running
6. kpatch-load --unload - Uninstall module



Kpatch Limitations

Replaces whole function

Cannot patch running function - Uses stop_machine() to freeze all process and threads before making switch to new function.

Does not allow modification of statically allocated data

- New field can be added via shadow variable
- Does not allow removal of global variables
- Global variable should remain in code unused
- Static variables are global

Lustre Issues Found

- Line numbers in debug logs

```
00000100:00100000:0.0:1490790862.421831:0:6948:0:
        (service.c:1941:ptlrpc_server_handle_req_in()) got req x1563202477584868
```

  - One line change dramatically increases number of modified functions. Since __LINE__ forces change with each new line

- Static variables in debug logs
  - Removal of static variables is not allowed
- Delays during stop_machine()
  - Lock and export timer can lead to evictions
- Minimize number of alive objects
  - RPC
  - Transactions

Lustre Changes Completed

- Remove line numbers from debug logs
  - Less debug info, but it is covered with git hash
- Do not use static variables in debug logs
  - Ability to rate particular message is lost
- Make calls from thread’s main function to non static functions only
  - Increases amount of patchable code
- Add ptlrpc barrier with timeout
  - Stop processing of incoming requests except BRW, cancels
  - Disable BRW
  - Disable cancels
  - Flush buffers, commit all transactions
  - Stop barrier and abort patching on timeout
  - There is an MDT barrier already used for creating snapshots
- Add load hook to refresh timers to avoid evictions

Test Results 1 - Simple Test to simulate 2000 clients

10 Clients, IB network

- 200 mounts per client
- Command Executed

```
run_parallel() {
          for N in `seq 1 $NUM_MOUNTS`; do
                    dd if=/dev/zero of=/mnt/lustre-$N/dd-$(hostname)-$N
                   bs=1K count=1800000 conv=notrunc &
          done
          wait
}
```

Server Config

- MDSCOUNT=2
- OSTCOUNT=2

Test Results 1 - Simple Test to simulate 2000 clients

```
# kpatch-load /tmp/kpatch/kpatch-kp1.ko
Mounted clients:
Mounted servers: /dev/md0
Stage_1 /* All Active Request */
0
Stage_2 /* Canceled + BRW requests */
289 /proc/fs/lustre/ost/OSS/ost_io/nreq_active
289
y
0
Stage_3 /* Canceled Requests */
0
Barrier took 1 second(s)
insmod /tmp/kpatch-kp1.ko
Total (un)load time 24 second(s)
```

Test Results 2 - Racer Test to simulate 500 clients

- 10 Clients, IB network
    - 50 mounts per client
    - Command Executed

```
run_parallel() {
     for N in $(seq 1 $NUM_MOUNTS); do
          mkdir -p /mnt/lustre-$N/racer-$(hostname)-$N
          DIR=/mnt/lustre-$N/racer-$(hostname)-$N NUM_THREADS=1 MAX_FILES=1000 sh
/usr/lib64/lustre/tests/racer/racer.sh &
    done
   wait
}
```

- Server Config
    - MDSCOUNT=2
    - OSTCOUNT=2

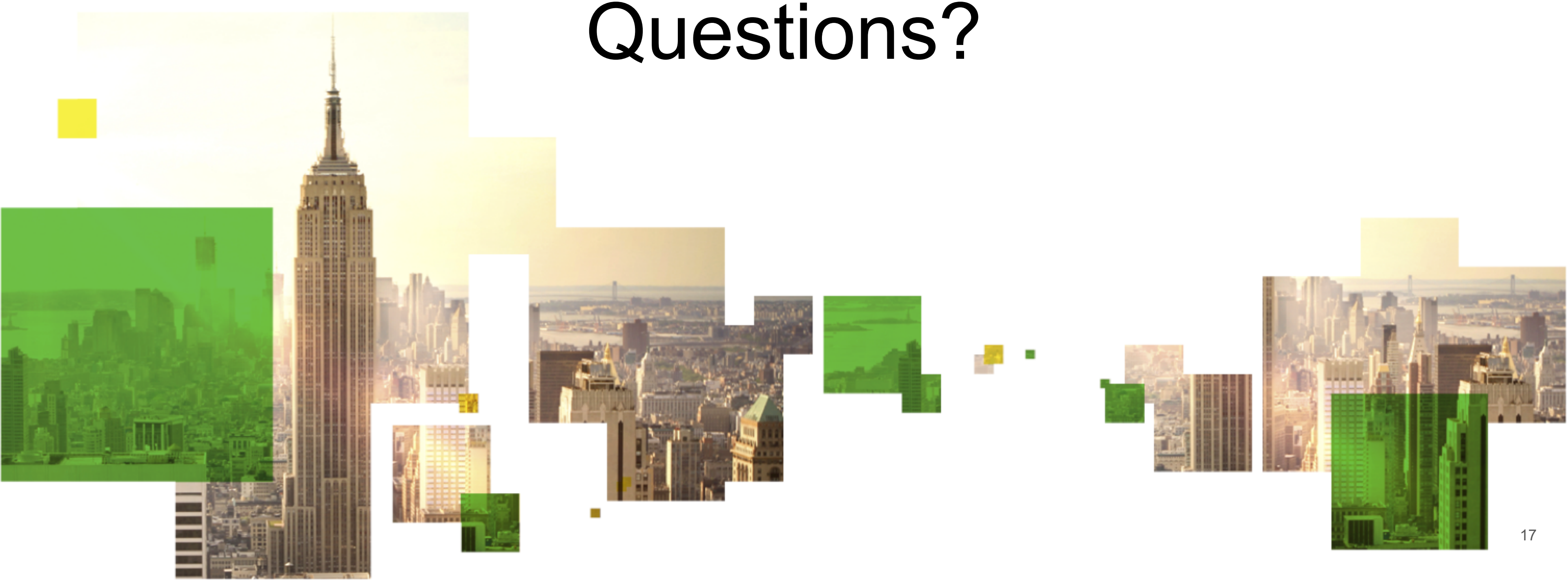
Test Results 2 - Racer Test to simulate 500 clients

```
# kpatch-load /tmp/kpatch/kpatch-kp1.ko
Mounted clients:
Mounted servers: /dev/md65
Stage_1 /* All Active Request */
32 /proc/fs/lustre/mds/MDS/mdt_readpage/nreq_active
279 /proc/fs/lustre/mds/MDS/mdt/nreq_active
311
y
0
Stage_2 /* Canceled + BRW requests */
0
Stage_3 /* Canceled Requests */
0
Barrier took 8 second(s)
insmod /tmp/kpatch-kp1.ko
Total (un)load time 23 second(s)
```

Open Issues and Future Work

- Test various configurations to refine eviction due to stop_machine()
- Test online patching for components beyond I/O path
  - MDS, LOD, Lnet, OSP, OSS
- Test with longer running I/O stress tests
  - Large sequential I/O
  - Complex racer.sh
- Test with complex setup. Larger Clients and OSS
- Address any issues related to release & deployment process

Questions?



Thank You !