Ksplice+: Rebootless kernel updates in a distributed system

Sanjay Kulhari
PhD Student, Dept. of CSE
U C Riverside
Outline

- Ksplice: Overview
- Distributed environment: Ksplice failure scenario
- Ksplice+: Solution by extending Ksplice
- Project activities/progress
- References and Future work
- Questions
Ksplice

- Prevents Loss of state/network connections
- Avoids downtime and unexpected problems due to restart

Running kernel with bug $\xrightarrow{\text{Ksplice}}$ Running kernel without bug

Update the kernel without restart
Ksplice

- Building an update
  - Find what has been changed
  - Build pre and post source code to get object code
  - Compare to find the list of changed functions

- Applying an update
  - Match pre code to running kernel
    - Discover symbol values
    - Safety check
  - Call stop_machine
    - Perform “safe time” check
    - Insert jmp instructions
Ksplice

Kernel

- foo
- bar

Primary module

- foo

- foo method is updated
- Object code of foo added to Ksplice’s primary module
Ksplice

- Resolves symbols in Post code
- Find foo in running kernel
- Find a safe time to insert jmp
An example scenario

- Signature of function *foo* is changed and it returns a different type
- Object code of updated foo function is added to Ksplice’s primary module, symbols are resolved and jmp statement inserted
- All the functions calling foo are updated to use new signature of foo

**Question**: What to do when the calling function is from a different kernel?
Distributed system

Remote procedure call

Kernel 1

Client machine

Client proc. → Client Stub

Kernel 2

Server machine

Server stub → Server Proc.

Expected behavior

- Two machines should continue to execute in parallel using old functionality until kernel on both of them is updated
Failure scenario

Kernel 2 updated, but client process on Kernel 1 fails due to incompatibility.
Success scenario

Kernel 2 is updated, but it keeps track of the state of the calling function. When calling function on client process gets updated, Kernel 1 jumps to new version otherwise not.
Required changes in Ksplice

- Ksplice core kernel module to track if the calling function is patched or not.

- Implement conditional jump based on state of calling function.
Ksplice+: Enhancement to Ksplice

- Allow normal execution to a call from unpatched Kernel.
- When the state of calling Kernel is ‘Patched’ allow the jump statement and execute patched version of the function.
- No failure as the systems communicate using unpatched functionality till both the systems are updated (Lazy update)
Project activities

- Built Linux kernel version 2.6.30.1 and added a custom system call.
- Updated the custom system call and created a patch using diff command.
- Executed Ksplice-create to create update file.
- Applied the patch by executing Ksplice-apply on the update file.
- Executed remote procedure call on unpatched/patched kernel.
- Simulated failure scenario in a distributed system.
- Tried to understand Ksplice code and make changes.
Future work

- Make appropriate code changes to Ksplice core kernel module that performs run-pre matching and inserts jump instruction and thoroughly test the functionality.

- Enhance solution for multi-node clusters.
References


Questions