Refuse to Crash with Re-FUSE

Swaminathan Sundararaman, Laxman Visampalli, Andrea C. Arpací-Dusseau, Remzi H. Arpací-Dusseau
Kernel-level File Systems

- Hard to develop FSes
  - Interact with OS components
  - Support variety of features
  - Difficult to debug

- **Require**: skilled developers

- New file system in 5-10 years
  - Adding user-desired features is difficult
FUSE: File system in USErspace

- Framework to run FS as a user-level process
  - Simplify development and deployment of FSes
  - FS interface to access underlying data
    - Database, email, ftp, http, ssh

- ~180 user-level file systems in few years
  - Can be customized to do just one thing
    - Compression, remote access (interface to cloud storage)
  - Raw device, pass through, and network-based
Refuse to Crash with Re-FUSE
User-level File System Issues

- Quick development
  - Not your typical file-system developer
  - No rigorous testing
  - No good documentation on FUSE
    - API, error scenarios

- Result in crashes
  - Require manual intervention to fix the problem
    - Repair & restart
  - Users feels that file system “does not work”
    - Decreases adoption chances
Re-FUSE: Restartable FUSE

- Framework to restart FS on crashes
  - A generic mechanism to restart in transparent and stateful
    - Applications are oblivious to FS crashes

- Novel techniques
  - Request tagging, system call logging, non-interruptible system calls
  - Performance: page versioning and socket buffer caching

- Evaluation (Linux 2.6.18, FUSE 2.7.4, NTFS-3g, AVFS, and SSHFS)
  - Generality: < 10 lines of code for each FS
  - Robustness: 60 controlled & 300 random fault injection
  - Performance: < 13% for both micro & macro benchmarks
  - Recovery time: < 300 milliseconds to restart FS
Outline

- Background
- Challenges
- Re-FUSE
- Evaluation
- Conclusions
Failure Model

- Only the user-level file system is unavailable
Can Simple Re-execution Work?

File System is left in an inconsistent state
How About Undoing Operations?

Undo not always possible

Need: alternate mechanism to restore FSes
Outline

- Background
- Challenges
- Re-FUSE
- Evaluation
- Conclusions
- Restartable FUSE
  - Designed to deal with file system crashes
  - Framework built inside FUSE and OS
- **Principle:** inconsistent → consistent state

Continue from the last completed operation
Components of Re-FUSE

- **Fault Detection**
  - Fail-stop, transient faults
  - Monitors FS crashes

- **Fault Anticipation**
  - Records file-system state

- **Fault Recovery**
  - Restart FS process and re-execute requests
Fault Anticipation

Additional work done in preparation of failure

- Simplified due to FUSE architecture
  - Separate process executes FS requests
    - In-flight requests are preserved
  - Only the user-level file system crashes on failure
    - Applications state and updates (data) are preserved

- **Need:** generic mechanism to track progress
  - Mimic re-execution with sufficient recorded state
Challenges In Tracking Progress

- Correlate FS actions with in-flight requests
  - Decoupled execution & multi threading
- Determine the state of individual sub tasks
  - Request splitting
Straw Man Approach

- Single request, single external operation
  - Require no additional support

- Single request, many external operations
  - Sequence number to correlate operations

- Many requests, many external operations
  - Request id and sequence number to correlate request and operation

What if parameters are the same?
Request Tagging

- Correlates FS operation with external calls
  - Attach the fuse request id to the work thread

```
User-level File System

Worker Thread

Libfuse

KFM

FUSE Driver

External Operations
```

4/19/11

Refuse to Crash with Re-FUSE
System Call Logging

- Track progress of individual operations
  - On replay: helps mimic execution of completed requests

```
User-level File System
```

```
Libfuse
```

```
System Call
```

```
<table>
<thead>
<tr>
<th>R_id</th>
<th>Type</th>
<th>Sequence #</th>
<th>Parameters</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Open</td>
<td>1</td>
<td>file name, flags, ...</td>
<td>fd, errno</td>
</tr>
</tbody>
</table>
```

```
R_id=5
Open file name, flags, ...
```

```
Components of Re-FUSE

- Fault Detection
- Fault Anticipation
- Fault Recovery

Re-FUSE

4/19/11 Membrane: Operating System Support for Restartable File Systems

19
- Restore FS state needed to execute requests
  - Leverage cached state at the VFS layer
- Re-execute in-flight requests on restart
  - Leverage request queue at the KFM layer

**Syscall Request-Response Table**

<table>
<thead>
<tr>
<th>R_id</th>
<th>Type</th>
<th>Sequence #</th>
<th>Parameters</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Open</td>
<td>1</td>
<td>file name, flags, ...</td>
<td>fd, errno</td>
</tr>
</tbody>
</table>
Outline

- Background
- Challenges
- Re-FUSE
- Evaluation
- Conclusions
Evaluation

- Address the following questions
  - Implementation effort to work with Re-FUSE?
  - Can Re-FUSE hide failures from applications?
  - Performance overheads during user workloads?

- Experimental setup
  - 2.2 GHz Opteron, 2GB Ram, 2 80GB WDC disk
  - Implemented in Linux 2.6.18, FUSE (2.7.4)
  - Re-FUSE: 3300 loc in Linux kernel, 1000 loc in FUSE
Generality of Re-FUSE

<table>
<thead>
<tr>
<th>File System</th>
<th>Original</th>
<th>Added</th>
<th>Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block-based interface</td>
<td>NTFS-3g</td>
<td>32K</td>
<td>10</td>
</tr>
<tr>
<td>Pass through interface</td>
<td>AVFS</td>
<td>39K</td>
<td>4</td>
</tr>
<tr>
<td>Network-based interface</td>
<td>SSHFS</td>
<td>4K</td>
<td>3</td>
</tr>
</tbody>
</table>

Code changes in individual FSes

- **Additions**
  - Daemonize user-level process
  - Notify local state (such as external file handles)

- **Modifications**
  - New mount interface that includes restart flag as parameter
Robustness of Re-FUSE

- Inject transient faults
  - Crashes in FS

- Inspect appl., FS state
  - ✗ - bad ✓ - good

- NTFS-3g results
  - SSHFS, AVFS in paper

- Random fault injection
  - 100 faults for all 3 FSes

<table>
<thead>
<tr>
<th>Operation</th>
<th>NTFS-3g Function</th>
<th>FUSE NTFS-3g</th>
<th>Re-FUSE NTFS-3g</th>
</tr>
</thead>
<tbody>
<tr>
<td>create</td>
<td>fuse_create</td>
<td>✗ ✓ ✗</td>
<td>✓ ✓ ✓</td>
</tr>
<tr>
<td>mkdir</td>
<td>fuse_create</td>
<td>✗ ✓ ✗</td>
<td>✓ ✓ ✓</td>
</tr>
<tr>
<td>symlink</td>
<td>fuse_create</td>
<td>✗ ✓ ✗</td>
<td>✓ ✓ ✓</td>
</tr>
<tr>
<td>link</td>
<td>link</td>
<td>✗ ✓ ✗</td>
<td>✓ ✓ ✓</td>
</tr>
<tr>
<td>rename</td>
<td>link</td>
<td>✗ ✓ ✗</td>
<td>✓ ✓ ✓</td>
</tr>
<tr>
<td>open</td>
<td>fuse_open</td>
<td>✗ ✓ ✗</td>
<td>✓ ✓ ✓</td>
</tr>
</tbody>
</table>

Re-FUSE successfully hides failures
NTFS-3g Performance Overheads

Reliability almost comes for free
Outline

- Overview
- Challenges
- Re-FUSE
- Evaluation
- Conclusions
Conclusions

“Failure is not falling down but refusing to get up.”
- Chinese proverb

- Reliability through restartability
  - A generic mechanism to restart user-level file systems

- **Principle**: inconsistent → consistent state
  - Inconsistency due to incomplete operations
  - Track progress of operations to continue from last execution

- Novel techniques
  - *Request tagging*: differentiate between serviced requests
  - *System call logging*: tracks sequence of operations
  - *Non-interruptible system call*: move threads to safe state
  - *Page versioning*: minimize logging overheads
Thank You!

Questions?

Advanced Systems Lab (ADSL)
University of Wisconsin-Madison
http://www.cs.wisc.edu/adsl