Improved Redundancy and Consistency beyond RAID-1

Roland Kammerer

Institute of Computer Engineering
Vienna University of Technology

March 3, 2011
Agenda

1. Introduction
2. Safe Storage
3. Implementation
4. Evaluation
5. Conclusion
Part I

Introduction
Part of the thesis “Linux in safety-critical applications”
Can we trust the way Linux (and FLOSS) is developed and tested
LTP\(^1\) was looking for RAID tests

---
\(^1\)Linux Test Project (http://ltp.sf.net)
How does it work?

- RAID: Redundant Array of Inexpensive/Independent Disks
- Different RAID levels (e.g.):
  - RAID-0: striping
  - RAID-1: mirroring
- RAID-1 creates virtual hard disk
- Data written to virtual disk is mirrored to multiple physical disks
- If one disk fails, whole system is still operational
How to test it?

- Separate test environment from system under test
- Physically switch on/off hard disks
- Use cryptographic hashes to verify consistent state of files on the disk
- Linux software RAID has proven to be very stable
What are the problems?

- In general RAID-1 improves availability, *but* availability of storage can be part of the safety argumentation.

- Consistency checks:

- Important properties are missing that qualify RAID-1 as safe storage.

  $$ \implies \text{RAID-1 like behaviour is fine, but additional properties are necessary} $$
Improved Redundancy and Consistency beyond RAID-1
Kammerer

Part II
Safe Storage

Definition Requirements

Safe Storage
What is safe storage

Informal definition:

*Storage is considered to be safe, if it provides a high degree of confidence that raw data that is read is exactly the same as the raw data that was written to it.*
Consistency

- Safe storage needs much stronger concept of consistency checking than standard RAID-1
- Whenever the safe storage detects an inconsistent state (e.g., a bit flip) it has to inform the reading application
- Of utmost importance for safety critical applications. If they are informed, a safe-state can be reached.
Diversity

- File system development is a difficult task. Modern file systems are getting more complex.

<table>
<thead>
<tr>
<th>File System</th>
<th>Lines of Code</th>
<th>Open Bugs²</th>
</tr>
</thead>
<tbody>
<tr>
<td>ext2</td>
<td>8965</td>
<td>4</td>
</tr>
<tr>
<td>ext3</td>
<td>16362</td>
<td>15</td>
</tr>
<tr>
<td>ext4</td>
<td>33979</td>
<td>45</td>
</tr>
<tr>
<td>xfs</td>
<td>74503</td>
<td>8</td>
</tr>
<tr>
<td>btrfs</td>
<td>57088</td>
<td>43</td>
</tr>
</tbody>
</table>

- Safe storage can benefit from the diversity in the file system sector
- Would be impossible to achieve in hardware (e.g., one file system on RAID-1 disk set)

²http://bugzilla.kernel.org
On-the-fly correction

- According to literature there is a large number of Undetected Disk Errors (UDEs).
- Use consistency and redundancy as a prerequisite
- Overwrite faulty copies on-the-fly with an agreed, consistent state (e.g., TMR + voting)
Simplicity

- As we have seen from the lines of code, file system development is a complex matter
- Implementation of safe storage should not introduce unnecessary complexity by itself
- ⇒ a simple layer above existing file systems
Part III

Implementation
Improved Redundancy and Consistency beyond RAID-1
Kammerer

Tools of the trade - FLOSS software

- **Linux:**
  - Widely used
  - A lot of different file systems which are proven in use
  - Necessary design diversity (different programmes, different companies)

- **FUSE:**
  - Upstream and used by many projects (sshfs, ntfs-3g, ...)
  - Really nice for rapid prototyping of crazy ideas
Tinysafefs

- Wrapper file system around existing file systems
- Has its own mount point (e.g., /mnt/tinysafefs)
- Reads/Writes data from/to existing mountpoints which should be the mountpoints of physical disks with distinct file systems
Two Disk Mode

- Simple “lowcost” version
- Similar to RAID-1, but with consistency checks (and file system diversity)
- On write: Data gets written to two mount points (i.e., two disks)
- On read: Data is read from both disks, gets compared and if (and only if) consistent gets forwarded to reading application
- If data not consistent: Return ENOENT (No such file or directory)
Two Disk Mode (2)

```
$ cd /tmp/tinysafefs
$ ls
testfile.txt
$ ls /disk*/
  /disk1:
testfile.txt
  /disk2:
testfile.txt
$ mkdir testdir
$ ls
testfile.txt
testdir
$ ls /disk*/
  /disk1:
testfile.txt
testdir
  /disk2:
testfile.txt
testdir
```
$ cd /tmp/tinysafefs
$ echo "testdata" > ./testfile.txt
$ ls /disk*
  /disk1:
  testfile.txt

  /disk2:
  testfile.txt
$ cat ./testfile.txt
  testdata
$ echo "destroy it" > /disk1/testfile.txt
$ cat ./testfile.txt
  cat: ./testfile.txt: No such file or directory
Three Disk Mode

- Wrapped around three disks
- On write: data is written to three disks
- On read: data is read from all copies and consistency gets checked
  - If at least two copies have a consistent state, data gets forwarded to user space application
  - If one copy is inconsistent, tinysafefs tries to overwrite it
  - If all copies are inconsistent: ENOENT
Three Disk Mode (2)

```bash
$ cd /tmp/tinysafefs
$ echo "testdata" > ./testfile.txt
$ ls /disk*
  /disk1:
  testfile.txt
  
  /disk2:
  testfile.txt
  
  /disk3:
  testfile.txt
$ cat ./testfile.txt
  testdata
$ echo "destroy it" > /disk1/testfile.txt
$ cat ./testfile.txt
  testdata
$ cat /disk1/testfile.txt
  testdata
$ echo "destroy it 1" > /disk1/testfile.txt
$ echo "destroy it 2" > /disk2/testfile.txt
$ cat ./testfile.txt
  cat: ./testfile.txt: No such file or directory
```
Part IV

Evaluation
Evaluation Setup

- **Hardware:**
  - Standard PC with Ubuntu Server 10.04.1
  - One disk with the OS, 3 usb drives with ext2, ext4, xfs
  - Used for performance tests and selected scenarios

- **Software:**
  - Simulator written in Python
  - Simulates three disk mode with on-the-fly correction
  - Worst/Best case scenarios (and randomly selected cases) used for hardware evaluation
Performance of tinysafefs

- `dd with fsync`
- Caches were dropped after each run to minimize cache influence
- Block sizes from 128 bytes to 8192 bytes.
Read performance of tinysafefs - 3 disk mode

Figure: Read performance
Write performance of tinysafefs 3 disk mode

**Figure:** Write performance
Corrections of tinysafefts in 3 disk mode

- Hard to assume realistic numbers of failures per read.
  Assumption: 1 failure per 50 reads
- 1000 runs ⇒ 1000 seeds for random generator
- 100 files per run
- After 50 reads, 1 random file on one random disk gets destroyed.
- Read until first uncorrectable fault

<table>
<thead>
<tr>
<th>Until First Error</th>
<th>TDMC</th>
<th>Single Disk</th>
<th>Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min. Repairs</td>
<td>0</td>
<td>None</td>
<td>-</td>
</tr>
<tr>
<td>Max. Repairs</td>
<td>849</td>
<td>None</td>
<td>-</td>
</tr>
<tr>
<td>Min. Reads</td>
<td>103</td>
<td>51</td>
<td>2.0</td>
</tr>
<tr>
<td>Max. Reads</td>
<td>42861</td>
<td>299</td>
<td>143.3</td>
</tr>
<tr>
<td>Overall Reads</td>
<td>5317517</td>
<td>117697</td>
<td>45.1</td>
</tr>
<tr>
<td>Overall Repairs</td>
<td>102033</td>
<td>None</td>
<td>-</td>
</tr>
<tr>
<td>Average Reads</td>
<td>5317</td>
<td>117</td>
<td>45.4</td>
</tr>
<tr>
<td>Average Repairs</td>
<td>102</td>
<td>None</td>
<td>-</td>
</tr>
</tbody>
</table>
Part V

Conclusion
Lessons learned

- Safe storage can be implemented by means of consistency, diversity, on-the-fly correction, and simplicity
- Safety critical systems can benefit from FLOSS
  - All the tools are already there
  - Trust is put on software that can be reviewed (no binary-blob firmware)
ETX and EOT

Thank you for your attention!