A Study of Linux File System Evolution

Lanyue Lu, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, Shan Lu

Motivation

Local file systems are important
- Desktop and laptop: Windows, Mac, Linux
- Data center / Cloud: Google FS, Hadoop DFS
- Mobile devices: Andriod, iPhone

Why study is useful
- Study drives system designs
- Answer important questions quantitatively
- Valuable for different communities
  - File system developers
  - System researchers
  - Tool builders

How We Studied ?

File systems are evolving over time
- Code base is not static
- New features, bug-fixing, performance, reliability

Patches describe such evolution
- How one version transforms to the next
- Every patch is available
- "System software archeology" is possible

Study with other rich information
- Source code, design documents
- Forum, mailing list

What We Did ?

Our method: manual inspection
- XFS, Ext4, Btrfs, Ext3, Reiserfs and JFS
- All Linux 2.6 major versions
- 5079 patches, multiple passes

Quantitatively analyze file systems
- Patch types, bug patterns, bug consequences
- Performance improvement
- Reliability enhancement

Provide an annotated dataset
- Rich data for further analysis

Key Questions

Q1: What do patches do ?

Q2: What do bugs look like ?

Q3: Do bugs diminish over time ?

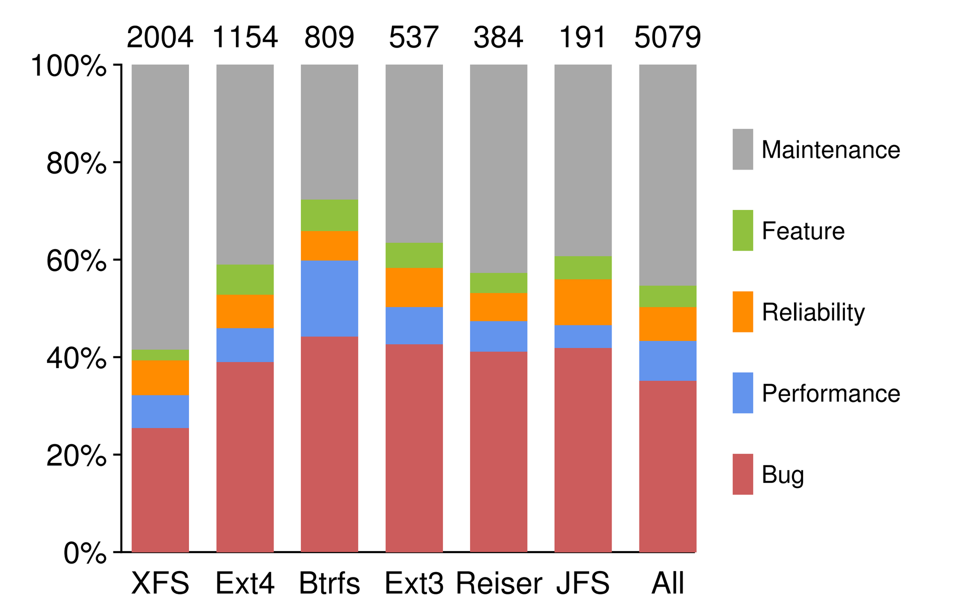
Q4: What consequences do bugs have ?

Q5: Where does complexity lie ?

Q6: Do bugs occur on normal paths or failure paths ?

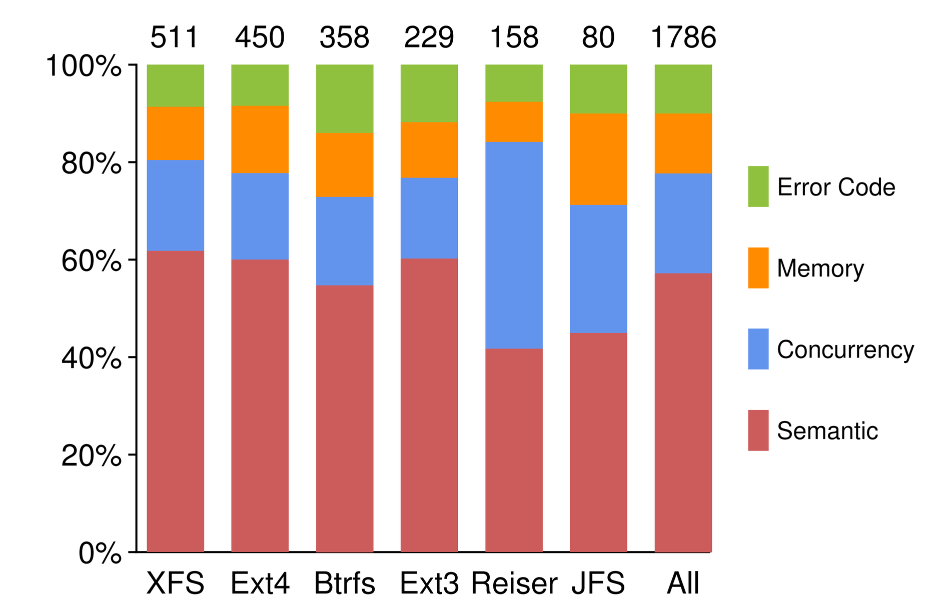
Q7: What performance techniques are used across file systems ?

1 Patch Overview



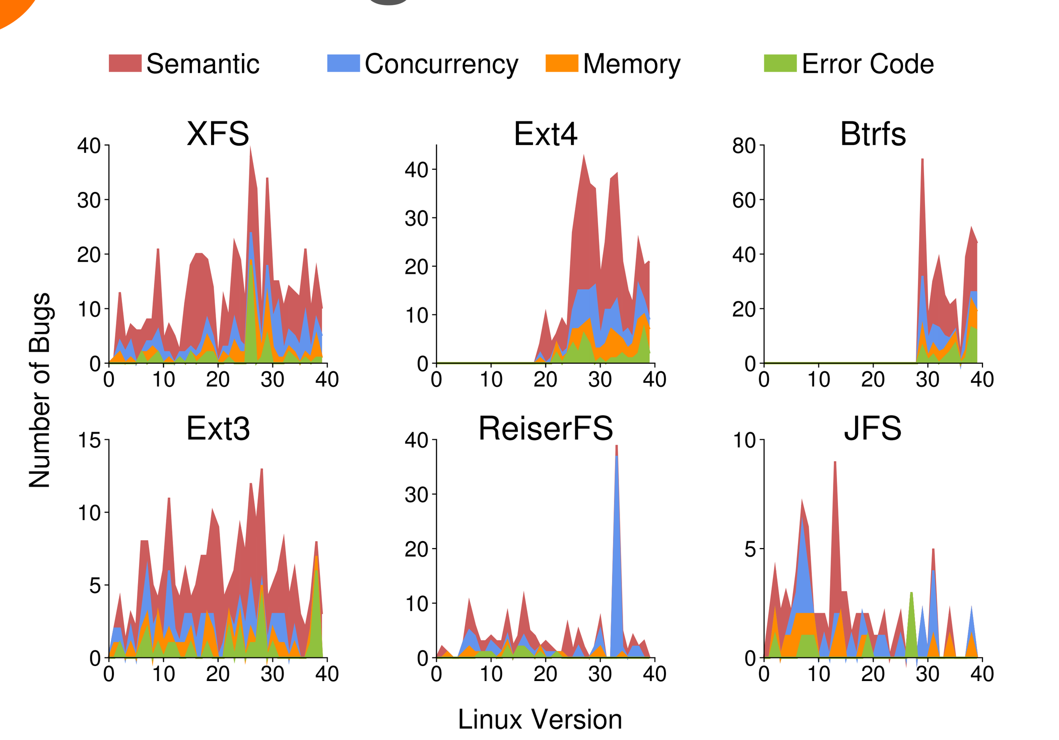
A1: 45% of patches are maintenance patches; 35% of patches are bug fixings. Even stable file systems, such as Ext3, have a large percentage of bug patches.

2 Bug Pattern



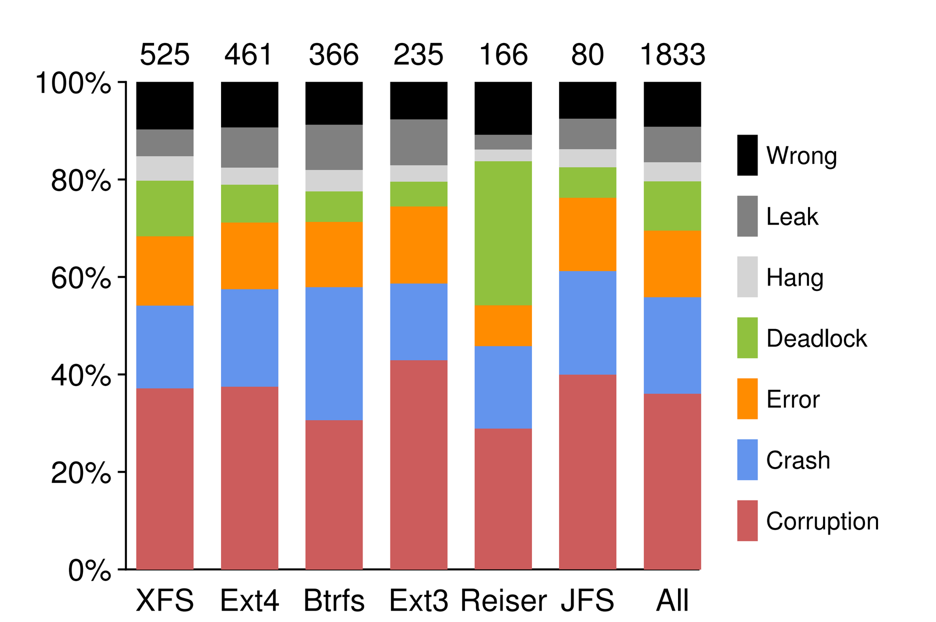
A2: Semantic bugs dominate other types (about 55% of total bugs). Concurrency bugs account for about 20% (higher than user-level software, which has about 3% of concurrency bugs)

3 Bug Trend



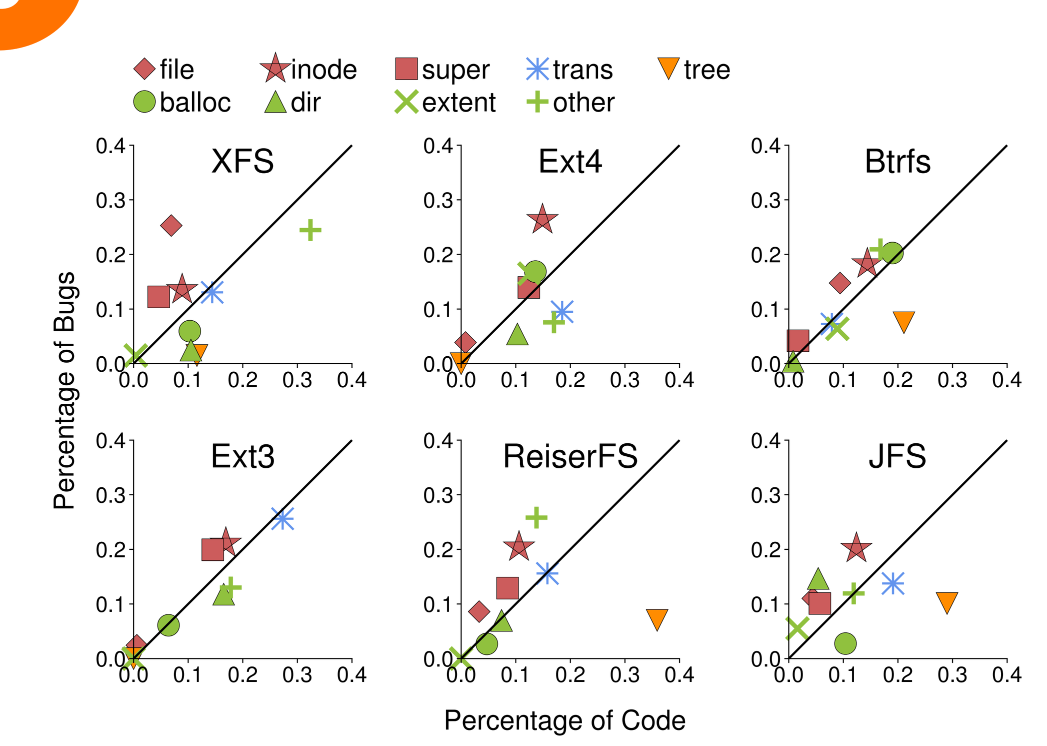
A3: The total number of bugs do not diminish over time (even for stable file systems), rather ebbing and flowing over time.

4 Bug Consequence



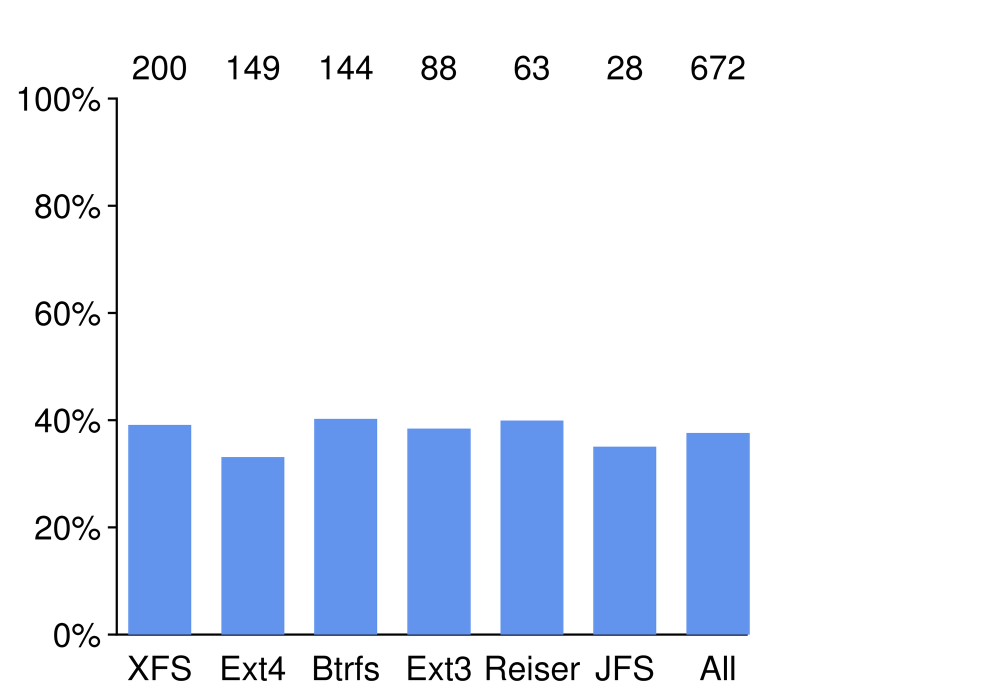
A4: Corruption and crash are most common, (about 40% and 20%). Wrong behavior accounts for only 5% to 10%, while it is dominant in user-level applications.

5 Correlation



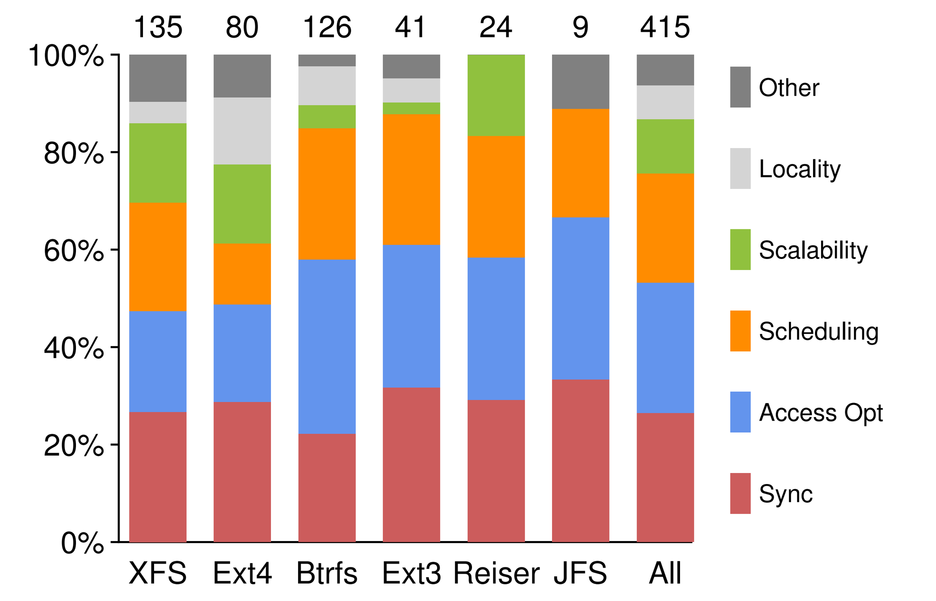
A5: Metadata management has high bug density (e.g., inode and super block). Tree structures are not particularly error-prone.

6 Failure Path



A6: 38% of bugs are on failure paths. Common bug examples: wrong or miss state update (semantic); miss unlock (concurrency); resource leak, null dereference (memory).

7 Performance



A7: A wide variety of performance techniques are used across file systems. Performance likely justifies the use of more complicated and time saving synchronization methods.

Conclusions

A large-scale study is feasible
- Time consuming, but manageable
- Similar study for other OS components

Research should match reality
- New tools are required for semantic bugs
- More attention for failure paths

History repeats itself
- Same mistakes, same performance improvement

Dataset is available
- http://research.cs.wisc.edu/adsl/Traces/fs-patch/
- More data in our paper and dataset