The Ongoing Evolution of Ext4 file system

New features and Performance Enhancements

Red Hat
Lukáš Czerner

October 24, 2011

Part I
Statistics

Agenda

1 Who works on Ext4?

2 Lines of code

redhat.

Agenda

redhat.

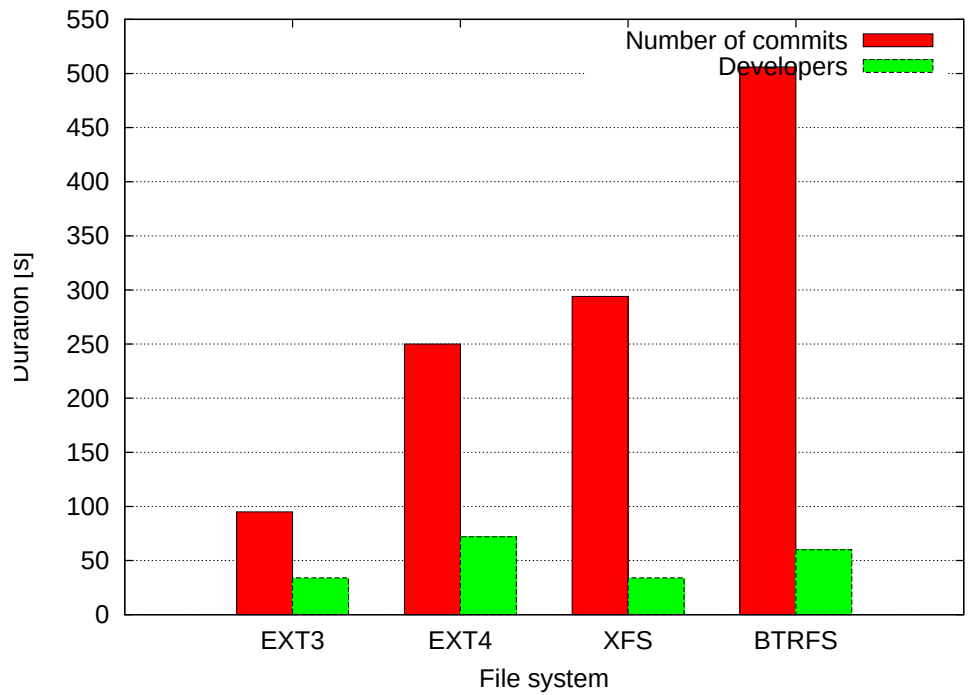
Last year of Ext4 development

- 250 non merge changes
- from 72 developers
- 9 developers has at least 10 commits
- 8512 lines of code inserted
- 5675 lined of code deleted

redhat.

Comparison with other local file systems

File system	Number of commits	Developers	Developers*
Ext4	250	72	9
Ext3	95	34	2
Xfs	294	34	4
Btrfs	506	60	11

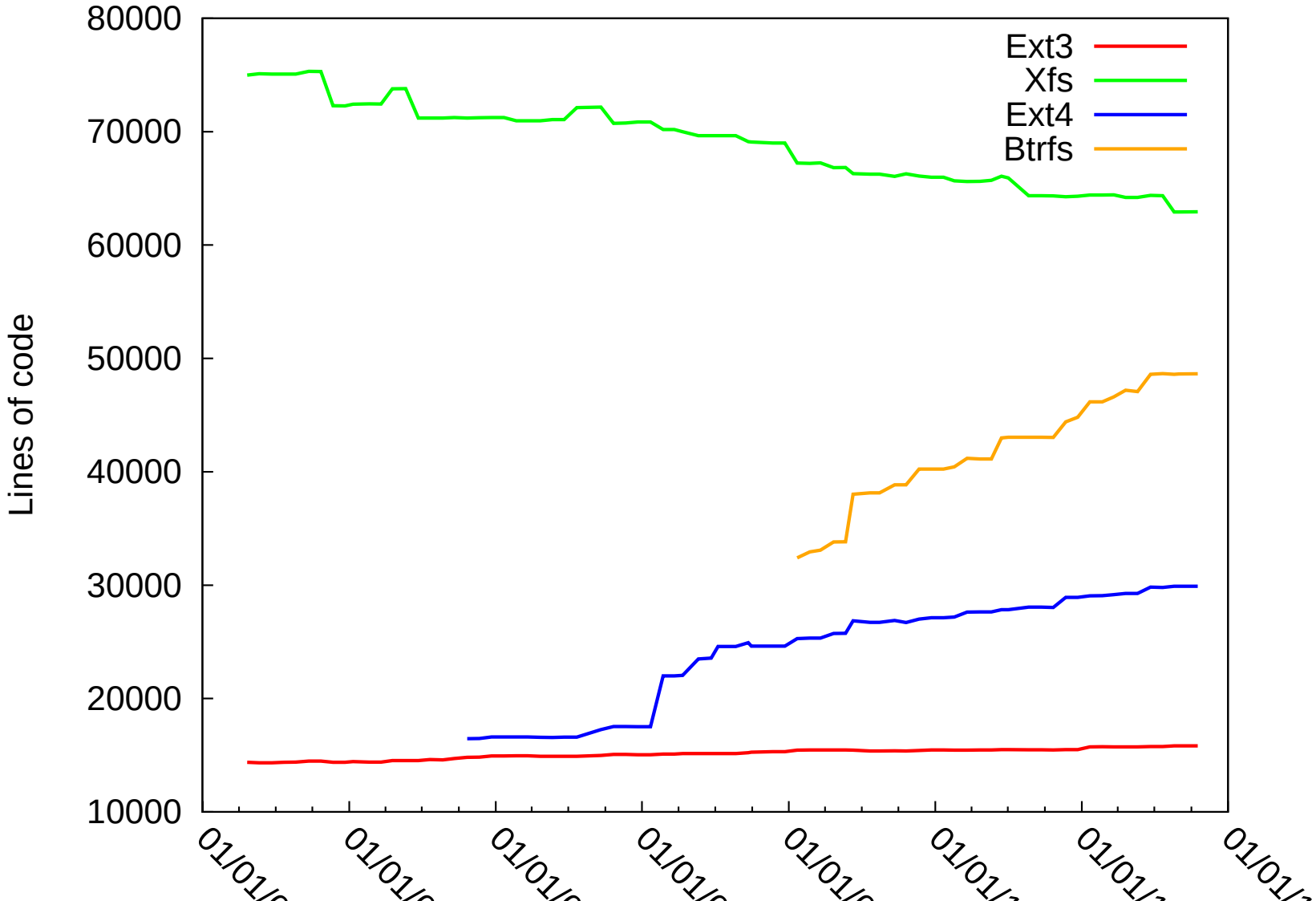


redhat.

Agenda

redhat.

Development of the number of lines



Part II

What's new in Ext4?

Agenda

3 Faster file system creation

4 Discard support

5 Support for file systems beyond 16TB

6 Punch hole support

7 Scalability improvements

8 Clustered allocation

redhat.

Agenda

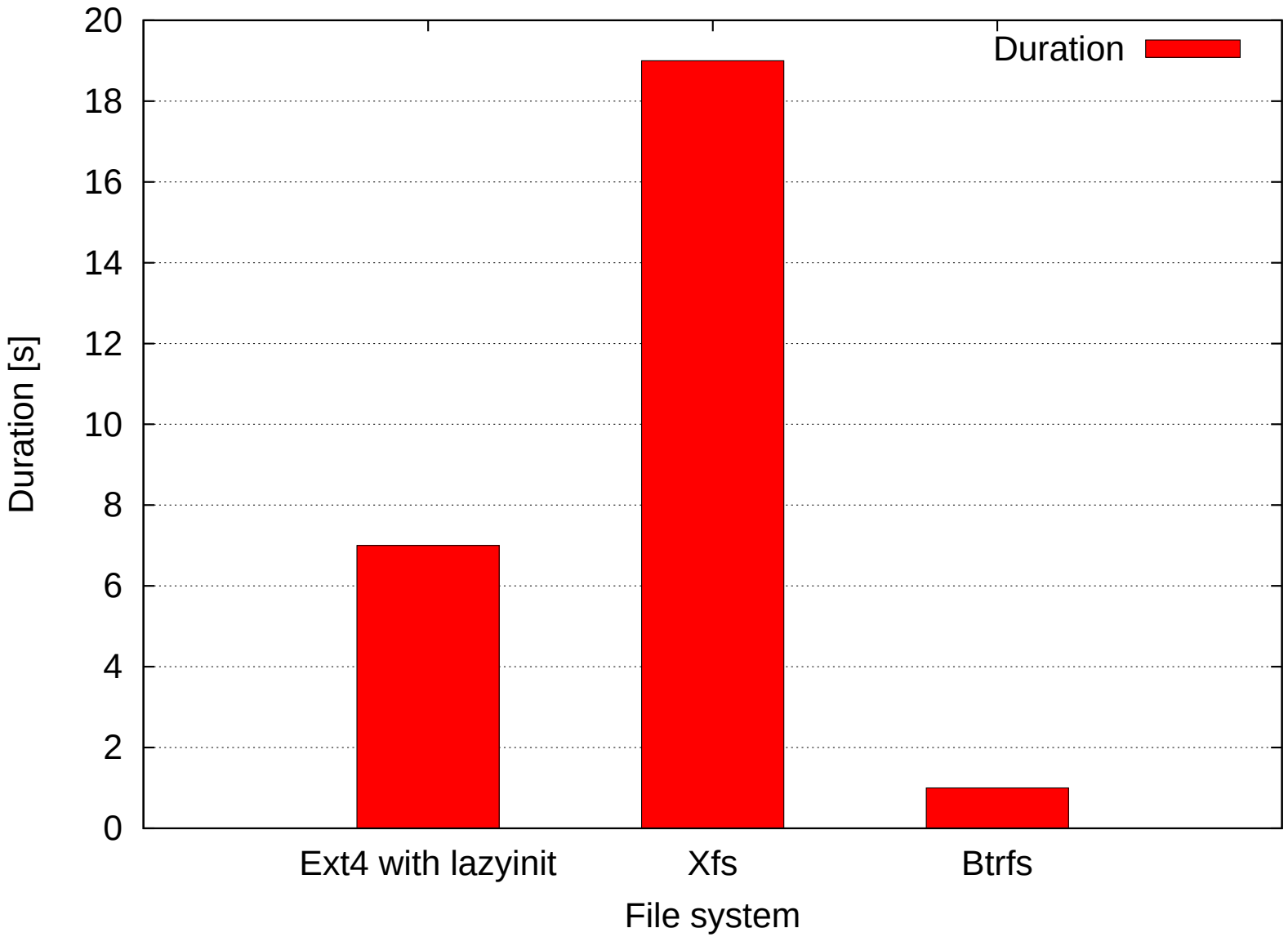
redhat.

Lazy inode table initialization

- This operation takes more than 80% of mkfs time
- For quite some time there is an option `-E lazy_itable_init`
  - Problems while repairing corrupted file system
- New kernel thread to initialize inode tables instead of mkfs
- Turned on by default
- e2fsprogs since v1.41.12-50-g210fd2c
- kernel since v2.6.36-rc6-12-gbfff687

redhat.

Ext4 mkfs time improvements



redhat.

Creating ext4 file system on SSD's

- Automatic detection of discard support
- Configurable via options and mke2fs.conf setting
  - `mkfs.ext4 -E [discard | nodiscard] device`
- If discard "zeroes data", there is no need to initialize inode table - speed-up

redhat.

Agenda

redhat.

Periodic discard

- Easy to implement
- File system support
  1. ext4 (v2.6.27-5185-g8a0aba7)
  2. btrfs (since upstream)
  3. gfs2 (v2.6.29-9-gf15ab56)
  4. fat, swap, nilfs
- `mount -o discard /dev/sdc /mnt/test`
- TRIM is non-queueable command - implications ?

redhat.

Batched discard support

- File system specific solution
- Provide ioctl() interface - FITRIM
- Do not disturb other ongoing IO too much
  1. Prevent allocations while trimming
  2. How to handle huge filesystem ?
- File system support
  1. ext4 (v2.6.36-rc6-35-g7360d17)
  2. ext3 (v2.6.37-11-g9c52749)
  3. xfs (v2.6.37-rc4-63-ga46db60)

redhat.

Agenda

redhat.

File system size over 16TB

- e2fsprogs support since 1.42-WIP-0702
- On-line resize for huge file system coming soon
- Off-line resize does not exists (yet?)
- Still experimental state
  - More testing needed
  - Scalability issues for huge file systems
  - Metadata overhead
  - Huge fsck time and memory requirements

redhat.

Agenda

redhat.

Punch hole support

- Allows to free space from the middle of the file
- Useful for trimming down the size of fs images
- Possibly mapping discard to punch hole in virtualization environment
  - qemu
  - loop device

redhat.

Agenda

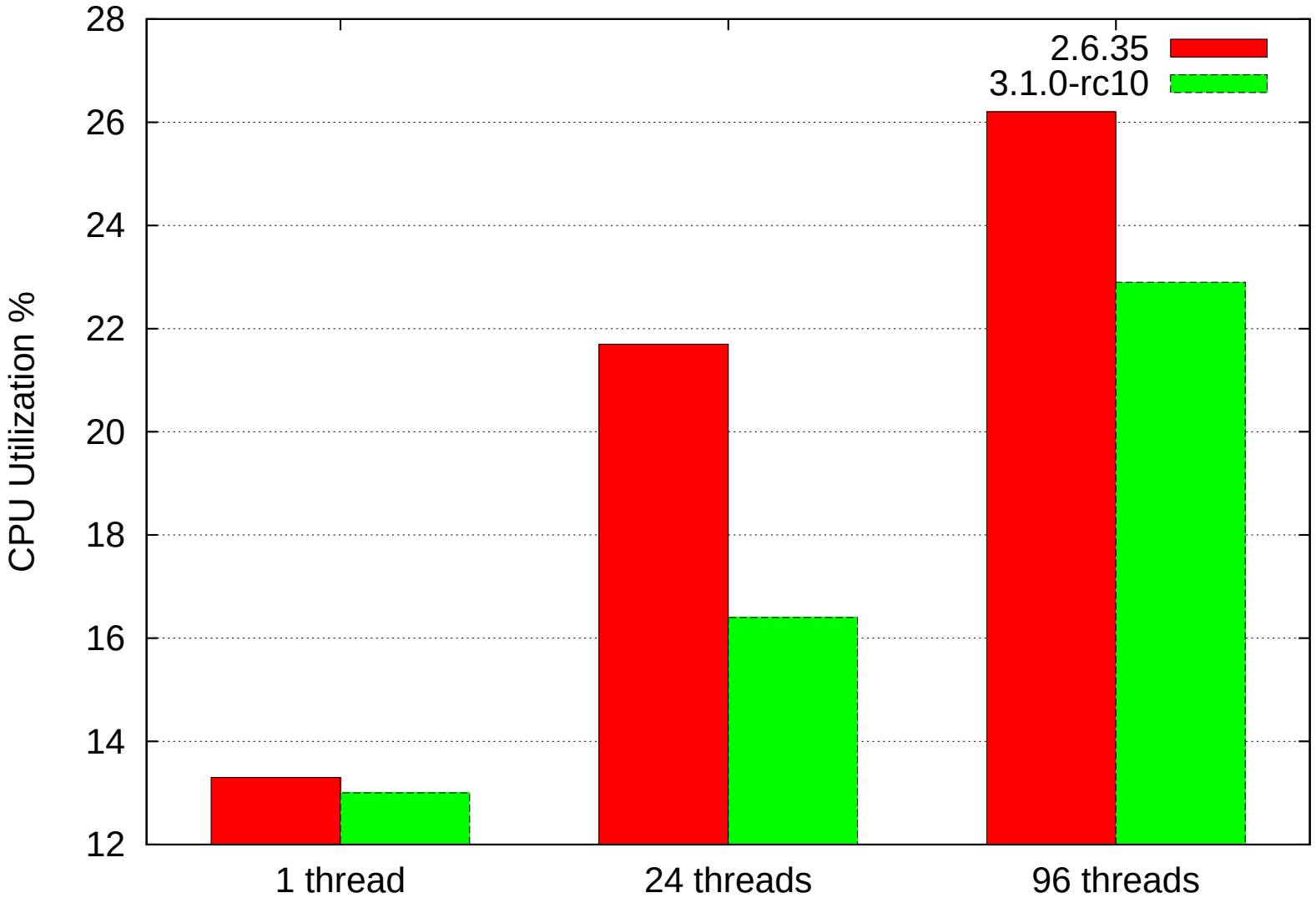
redhat.

Scalability improvements

- Many thanks to Eric Whitney for extensive testing
- Improvements in jbd2 layer (locking scalability)
- Using bio layer directly instead of buffer layer

redhat.

Scalability improvements



redhat.

Agenda

redhat.

Block size limitations

- File system block size limited by page size
- On bigger file systems, higher metadata overhead
- Higher block allocation overhead
- Ext4 block allocation
  - Ext4 has bitmap based allocation
  - Each bit represents one file system block
  - Bitmap stored in one file system block
  - One bitmap address 128MB (4096kB fs block)

redhat.

Clustered allocation

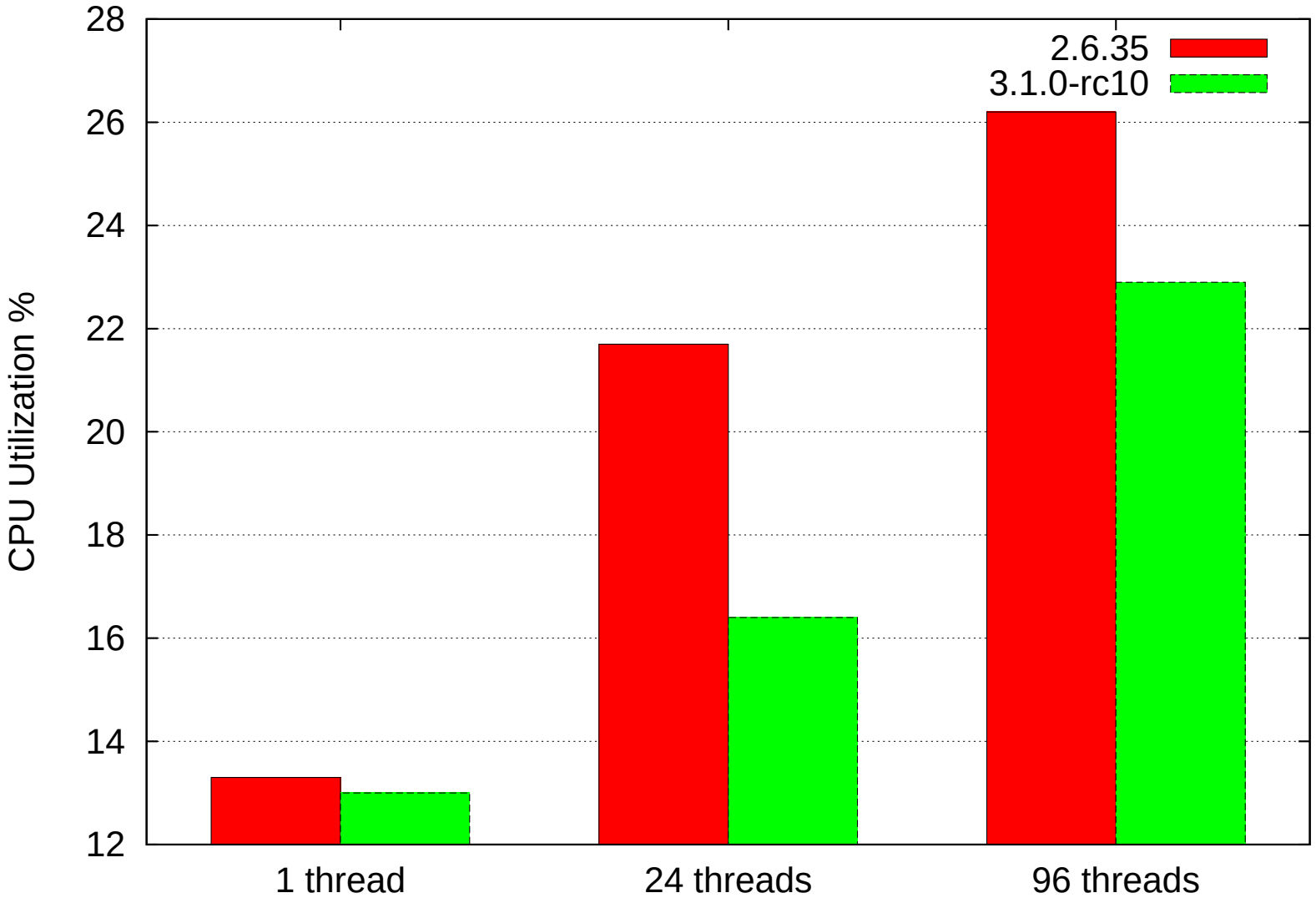
- Each bit represents power of two fs blocks
- Less allocation bitmaps to maintain - smaller overhead
- One bitmap can address 2GB (64KB cluster size)
- Downsides:
  - No sub-cluster allocation
  - Small files, directories, extent tree blocks will consume more space
- Still in dev branch of ext4

Part III

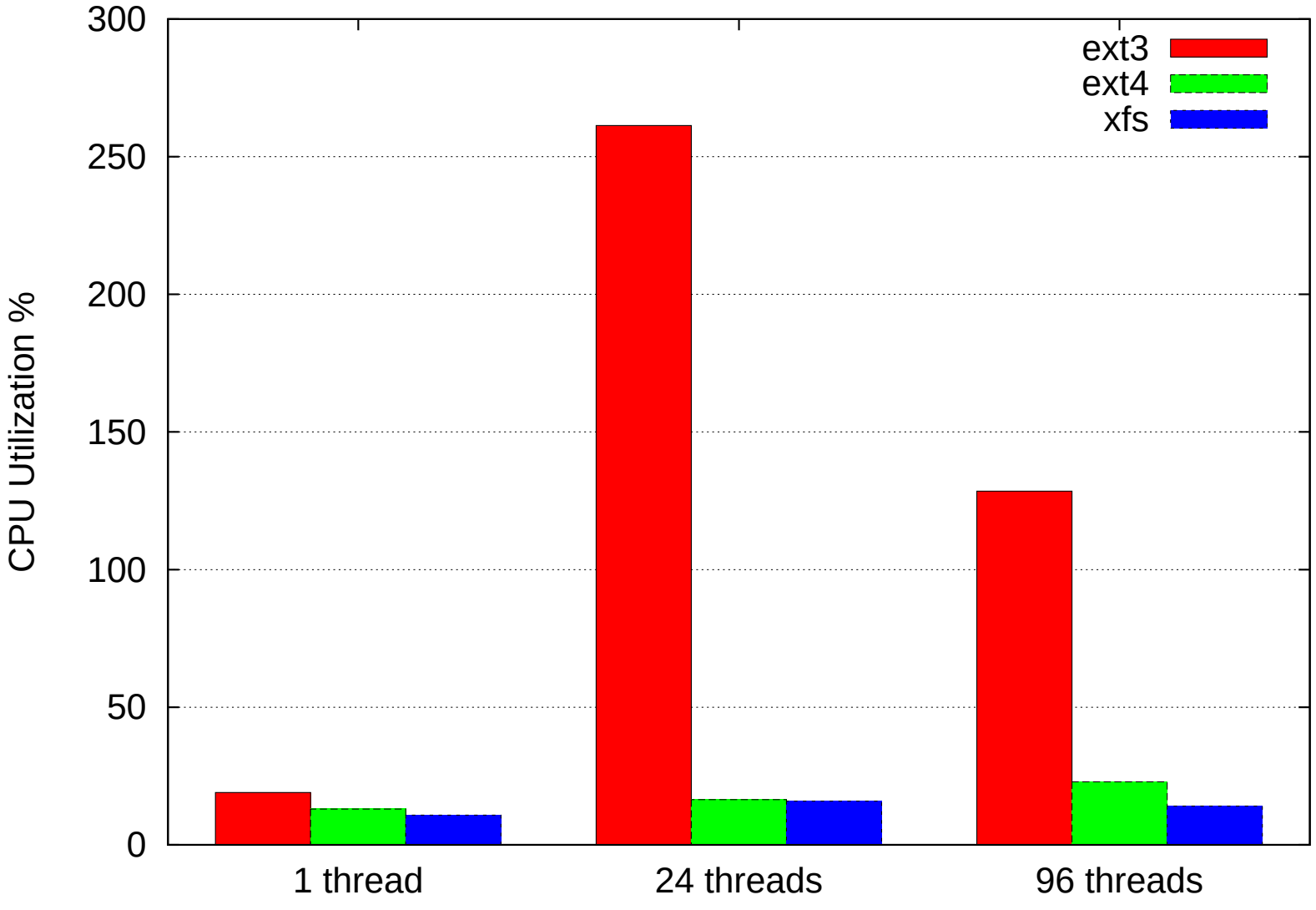
Some benchmark results

redhat.

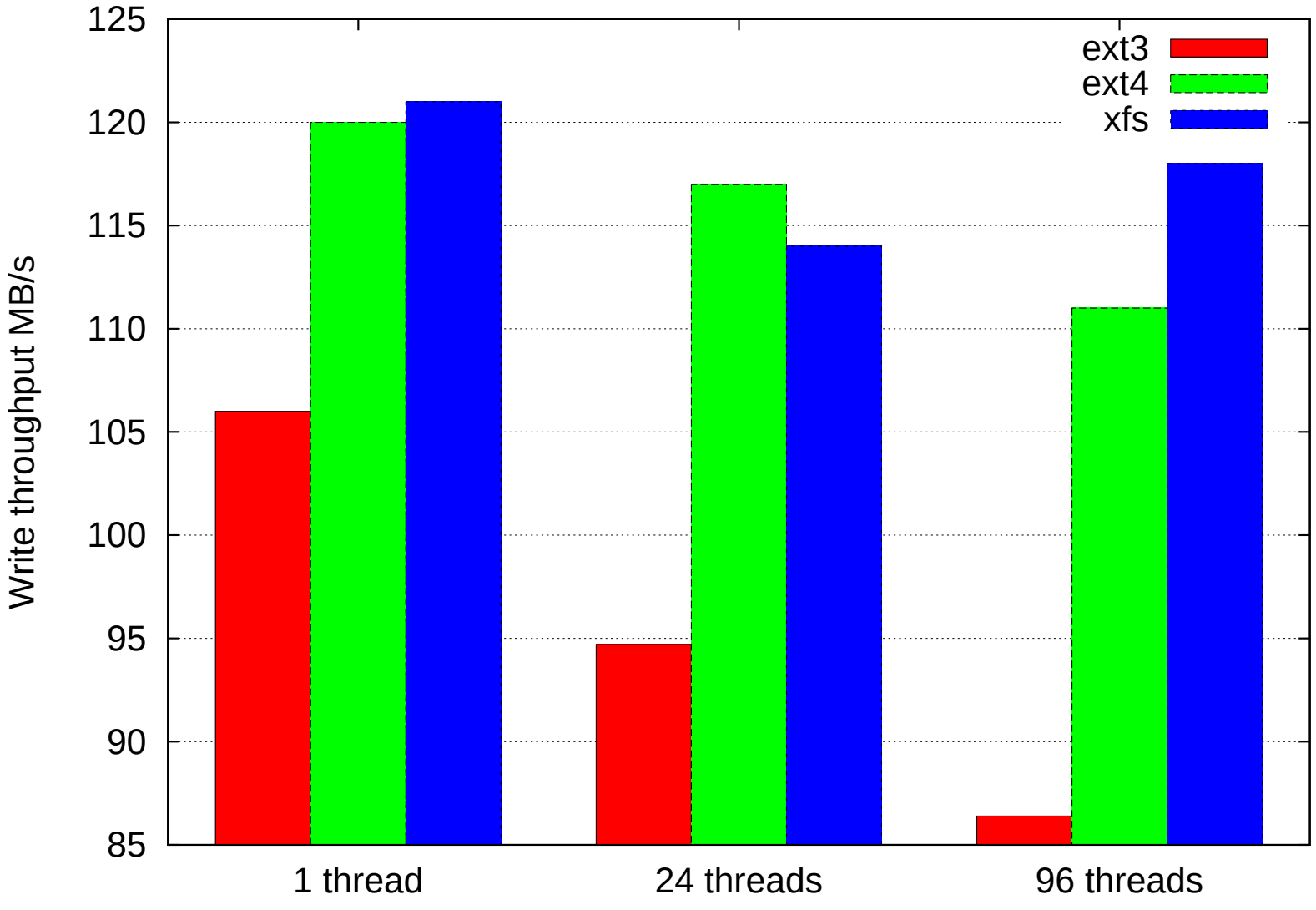
Scalability improvements



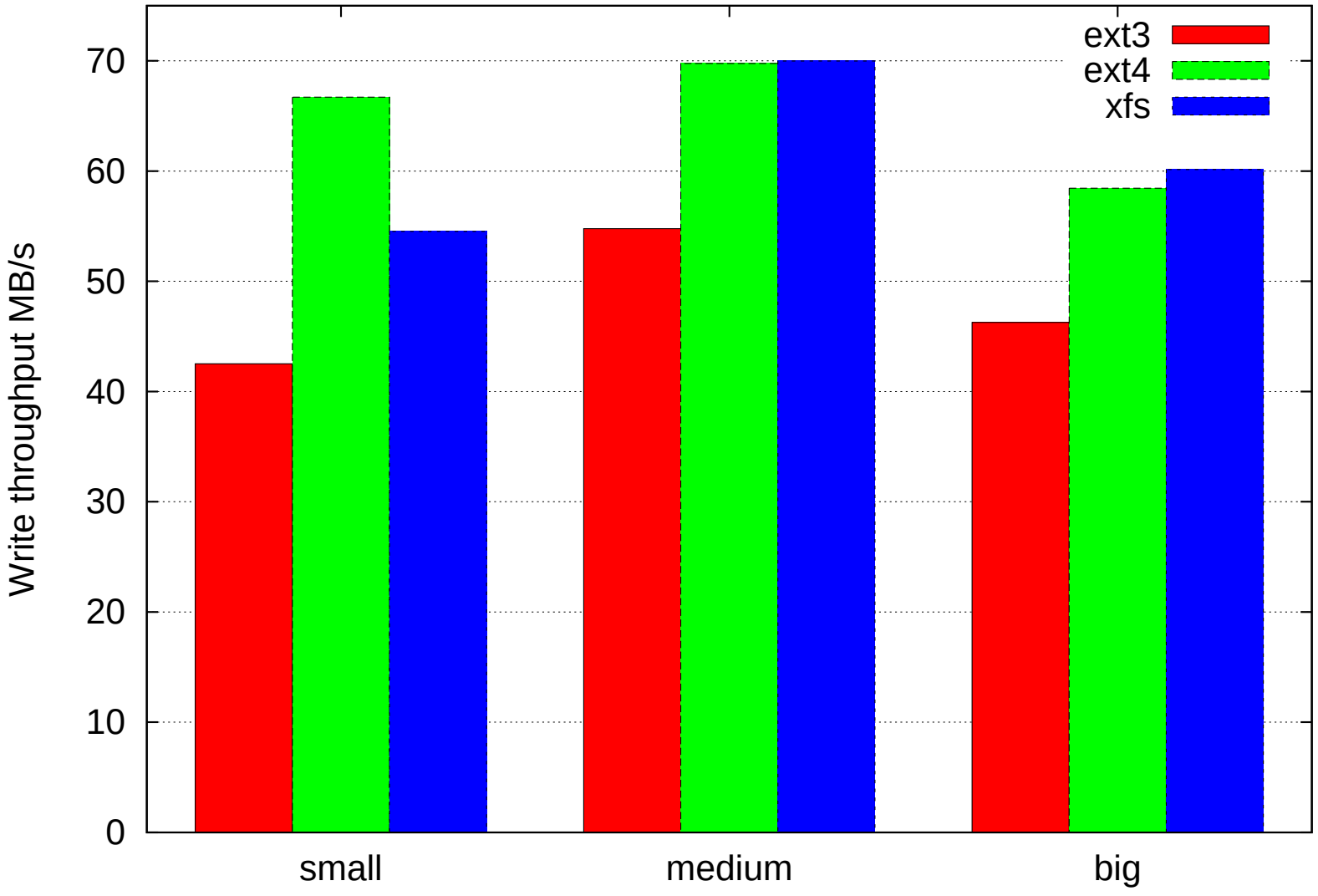
Scalability - comparison



Large file create - comparison



Metadata intensive workload - comparison



The end.

Thanks for listening.