XFS: Adventures in Metadata Scalability

Dave Chinner

<dchinner@redhat.com>

18 January, 2012

“If you have large or lots, use XFS”

-- Val Aurora, LCA 2007

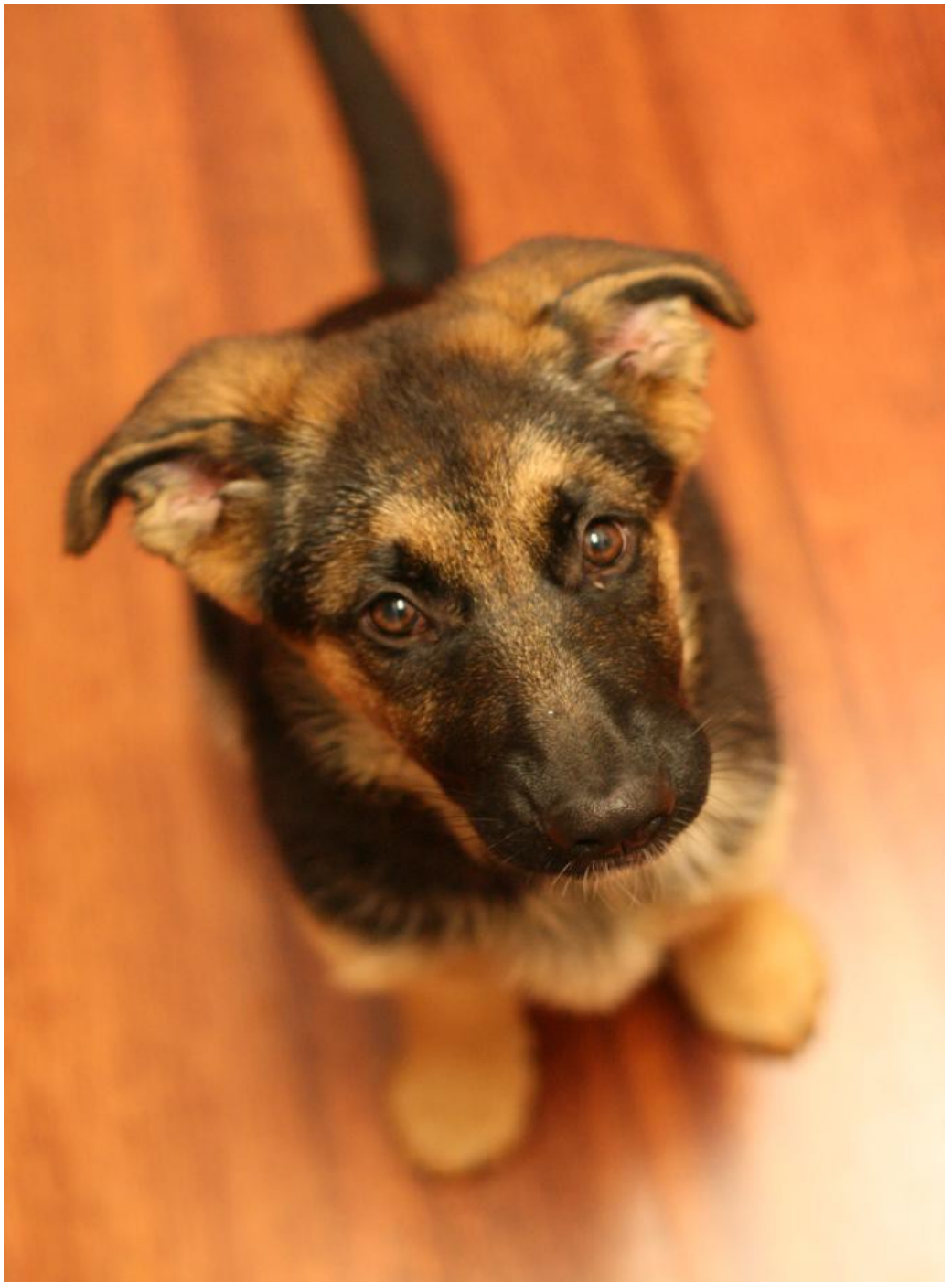
Overview

- What are the metadata performance problems?
- How were they solved?
- How well does it scale?
- Where do we go from here?
- Why are we changing the on-disk metadata format?
- What does this all mean?

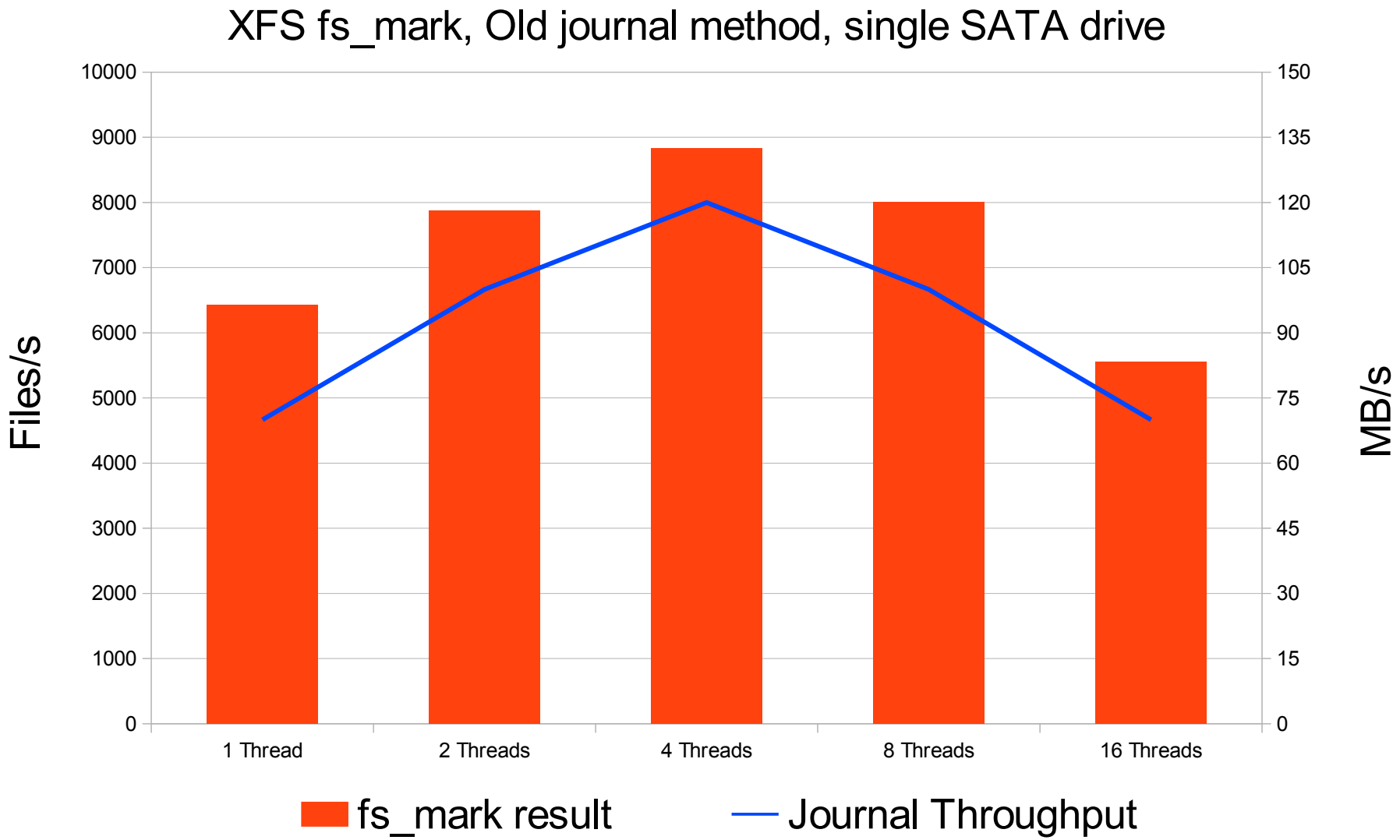
XFS's Metadata Problems

- Metadata read and lookup is fast and scales well.
- Metadata modification performance is **TERRIBLE.**
- Excellent transaction execution parallelism, little transaction commit parallelism.
- Typically won't scale past one CPU.
- Transaction commit throughput limited by journal bandwidth.
- Metadata writeback causes IO storms.
- Lots of locks to deal with.

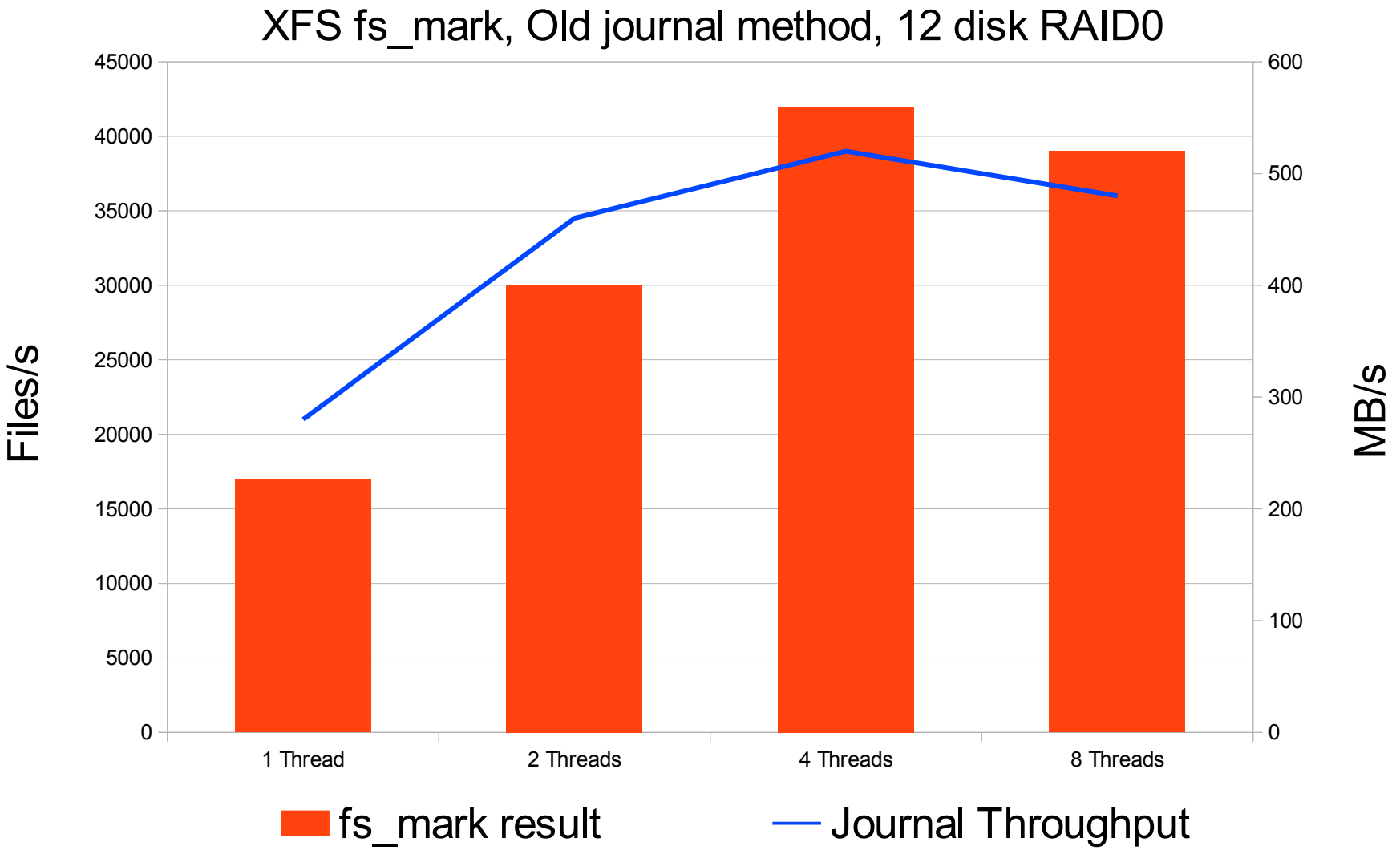
It's not that bad, is it?



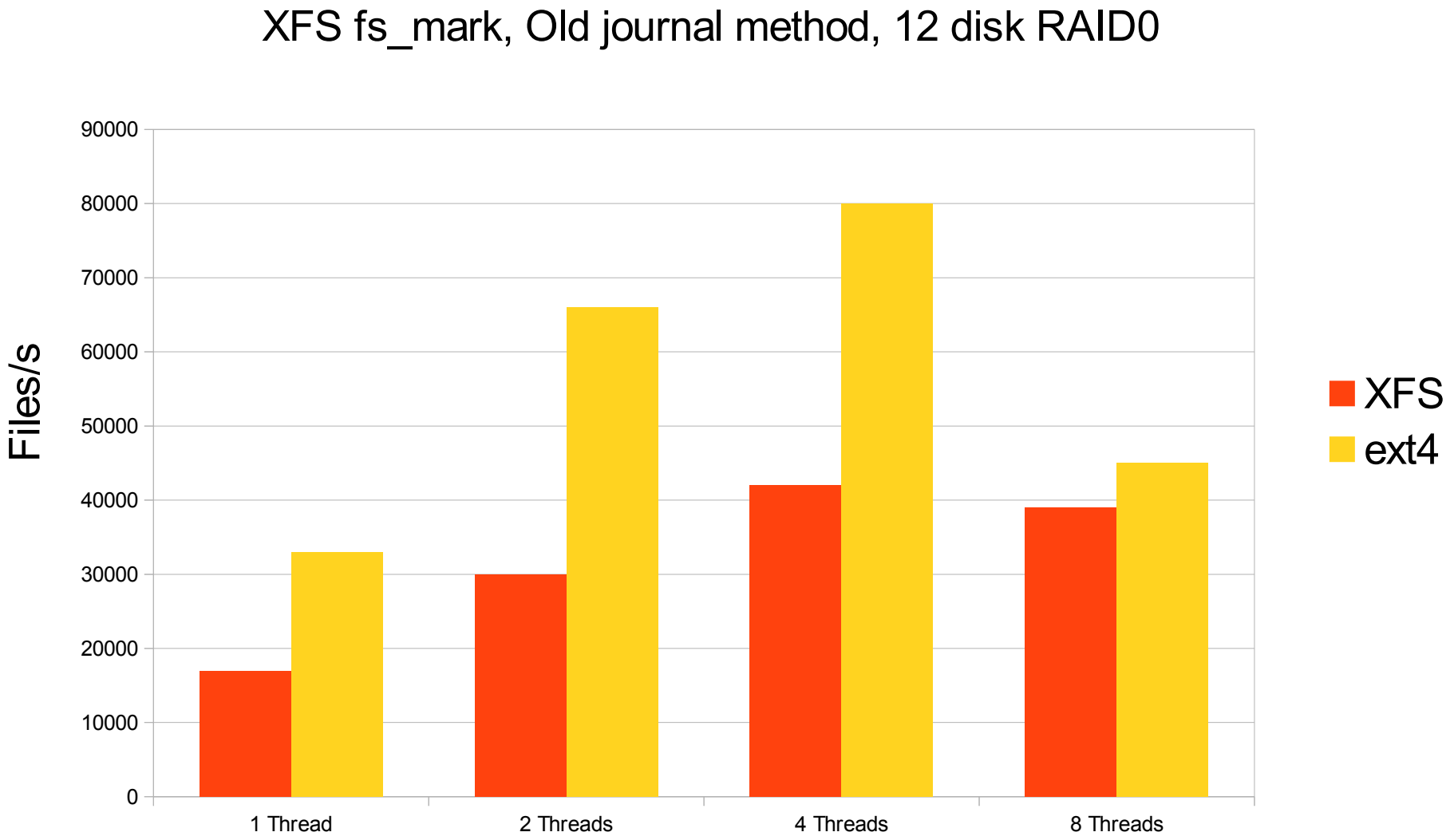
Just how bad?


XFS fs_mark, Old journal method, single SATA drive
Files/s
MB/s
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
150
135
120
105
90
75
60
45
30
15
0
1 Thread
2 Threads
4 Threads
8 Threads
16 Threads
fs_mark result
Journal Throughput

Just how bad?


XFS fs_mark, Old journal method, 12 disk RAID0
Files/s
MB/s
45000
40000
35000
30000
25000
20000
15000
10000
5000
0
600
500
400
300
200
100
0
1 Thread
2 Threads
4 Threads
8 Threads
fs_mark result
Journal Throughput

Just how bad?


XFS fs_mark, Old journal method, 12 disk RAID0
Files/s
90000
80000
70000
60000
50000
40000
30000
20000
10000
0
1 Thread
2 Threads
4 Threads
8 Threads
XFS
ext4

Pretty Bad, eh?

- Ext4 can be up 20-50x times than XFS when data is also being written as well (e.g. untarring kernel tarballs).
- This is XFS @ 2009-2010.
- Unless you have seriously fast storage, XFS just won't perform well on metadata modification heavy workloads.
- But I took solace in the signs that ext4 had some limitations showing up....

The Fix is in!

- One major algorithm change.
- Several significant optimisations.
- Lots of hot locks and structures to improve.
- No on-disk format changes!

Delayed Logging

- Original idea was floated by Nathan Scott back in 2005.
- Took 4 attempts over 5 years to design and implement a working solution.
- Solution came from considering how to solve a reliability problem (transaction rollback).
- Aggregates transaction commits in memory.

Delayed Logging

- Checkpoints the aggregated changes to the journal in a special transaction type.
- Utilises known algorithms, so no proofs required.
- Lots of information about the mechanism in Documentation/filesystems/xfs-delayed-logging-design.txt
- Only journalling method in 3.3

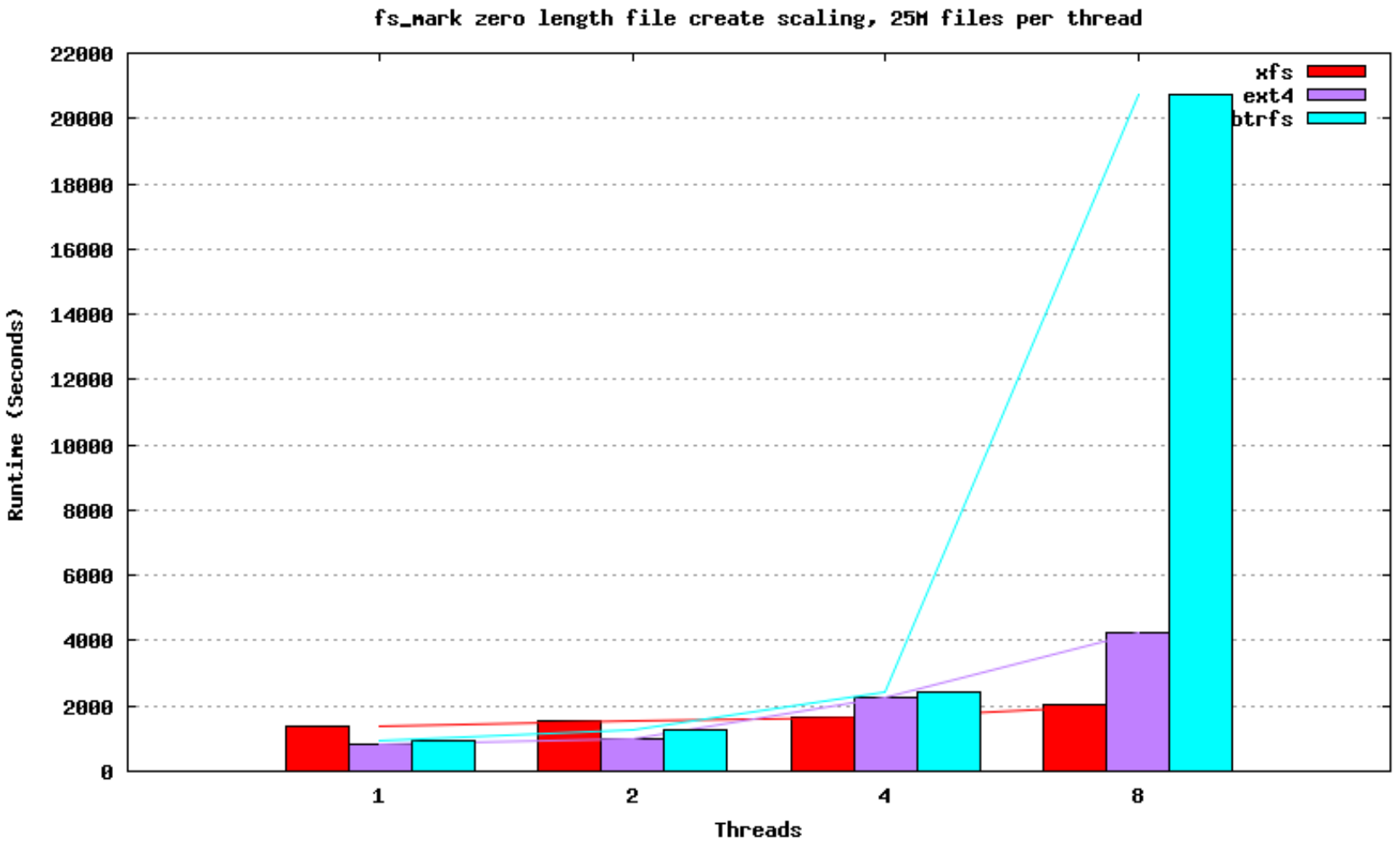
Other Improvements

- Lockless log space reservation fast path.
- Delayed write metadata.
- Extensive metadata sorting before IO dispatch.
- Batched active log item manipulations.
- Metadata caching divorced from the page cache, now uses a reclaim algorithm originally proven on Irix.
- Lockless (RCU based) inode cache lookups.

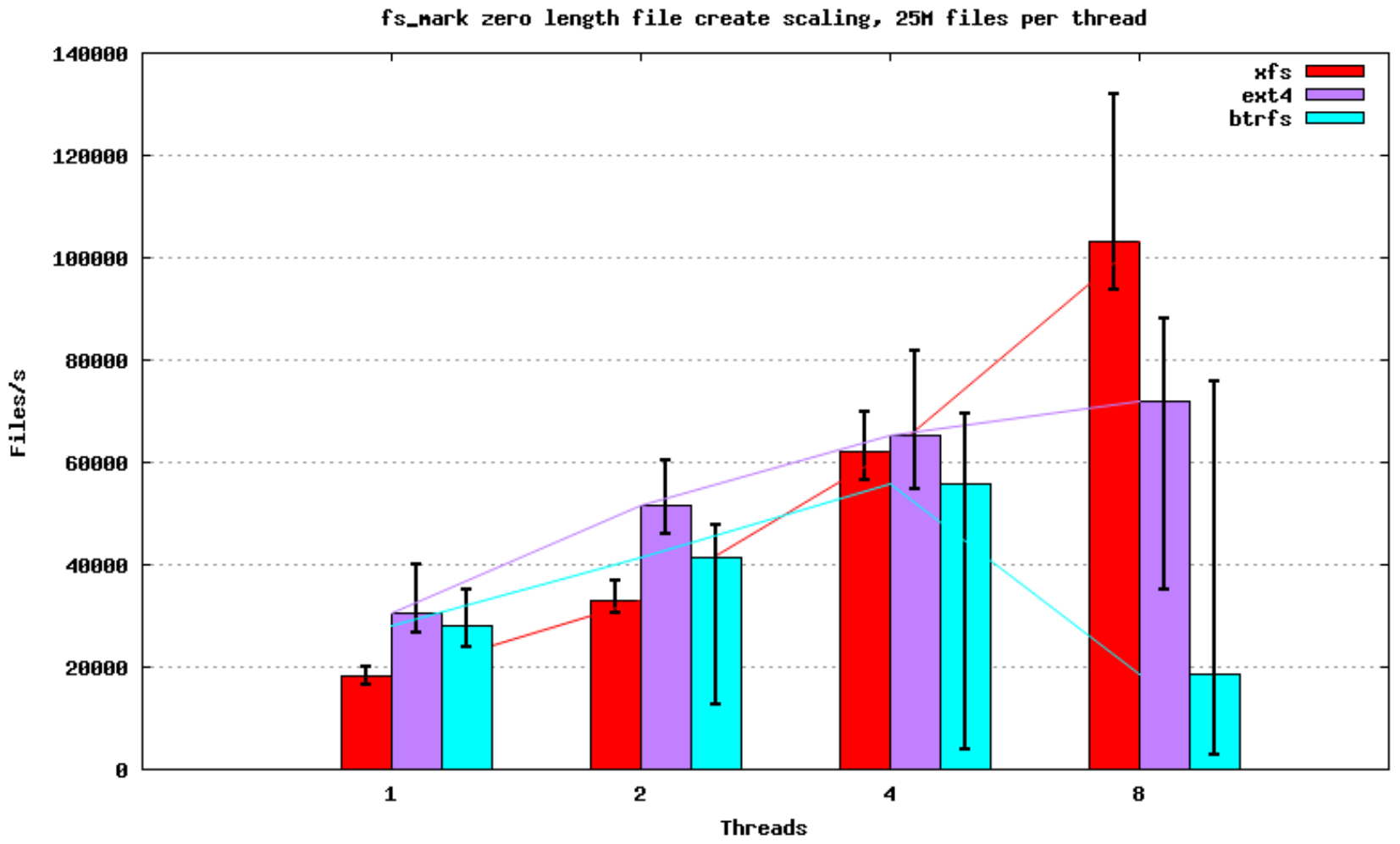
So how does XFS scale now?

- 8p, 4GB RAM KVM VM.
- Host has 17TB 12 disk RAID0 device, XFS filesystem, 17TB preallocated image file, virtio, Direct IO.
- Guest filesystem uses mkfs, mount defaults except for inode64,logbsize=262144 for XFS.
- Intent is to test default configurations for scaling artifacts.
- Parallel fs_mark workload to create 25 million files per thread.

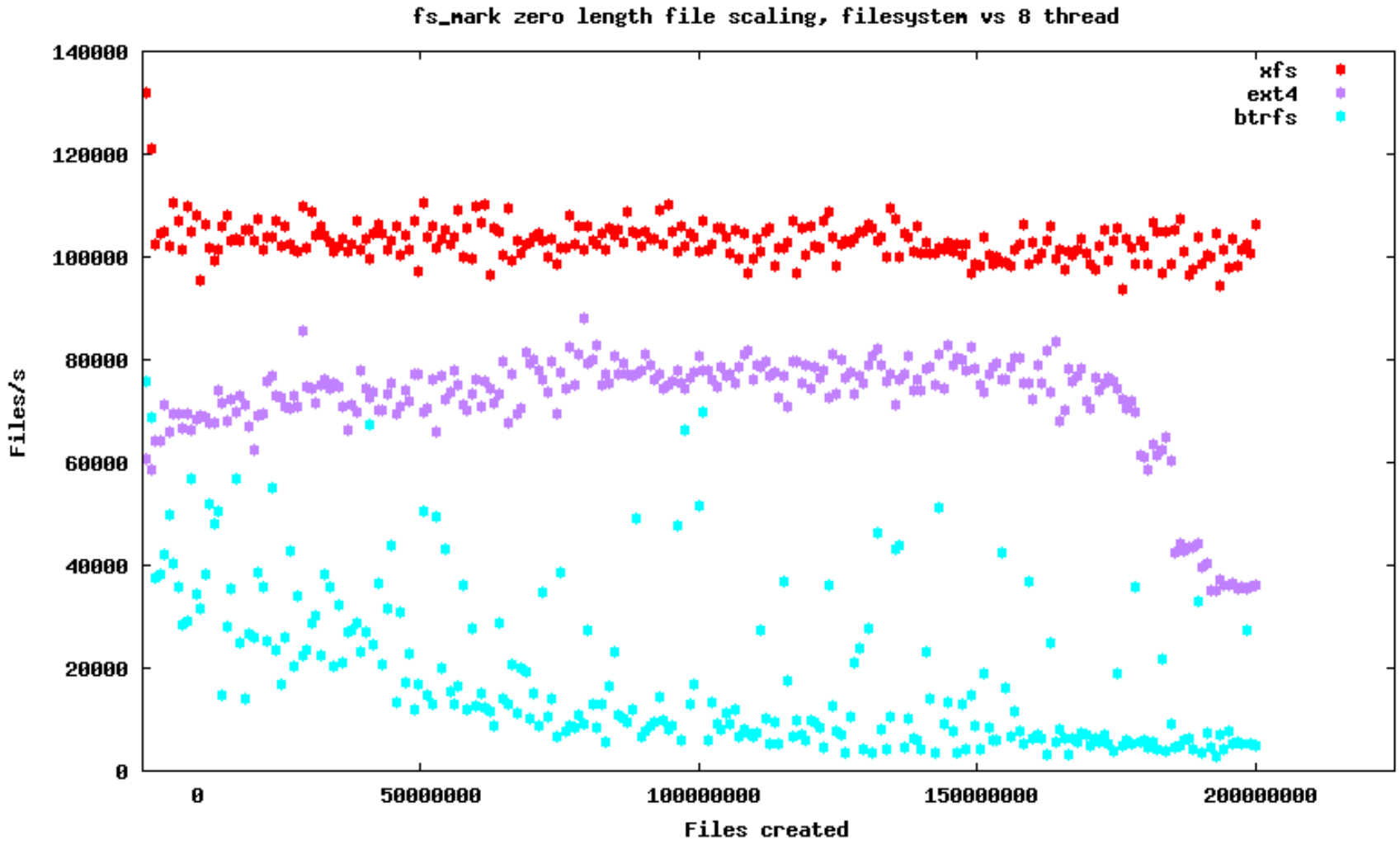
File Create Scaling



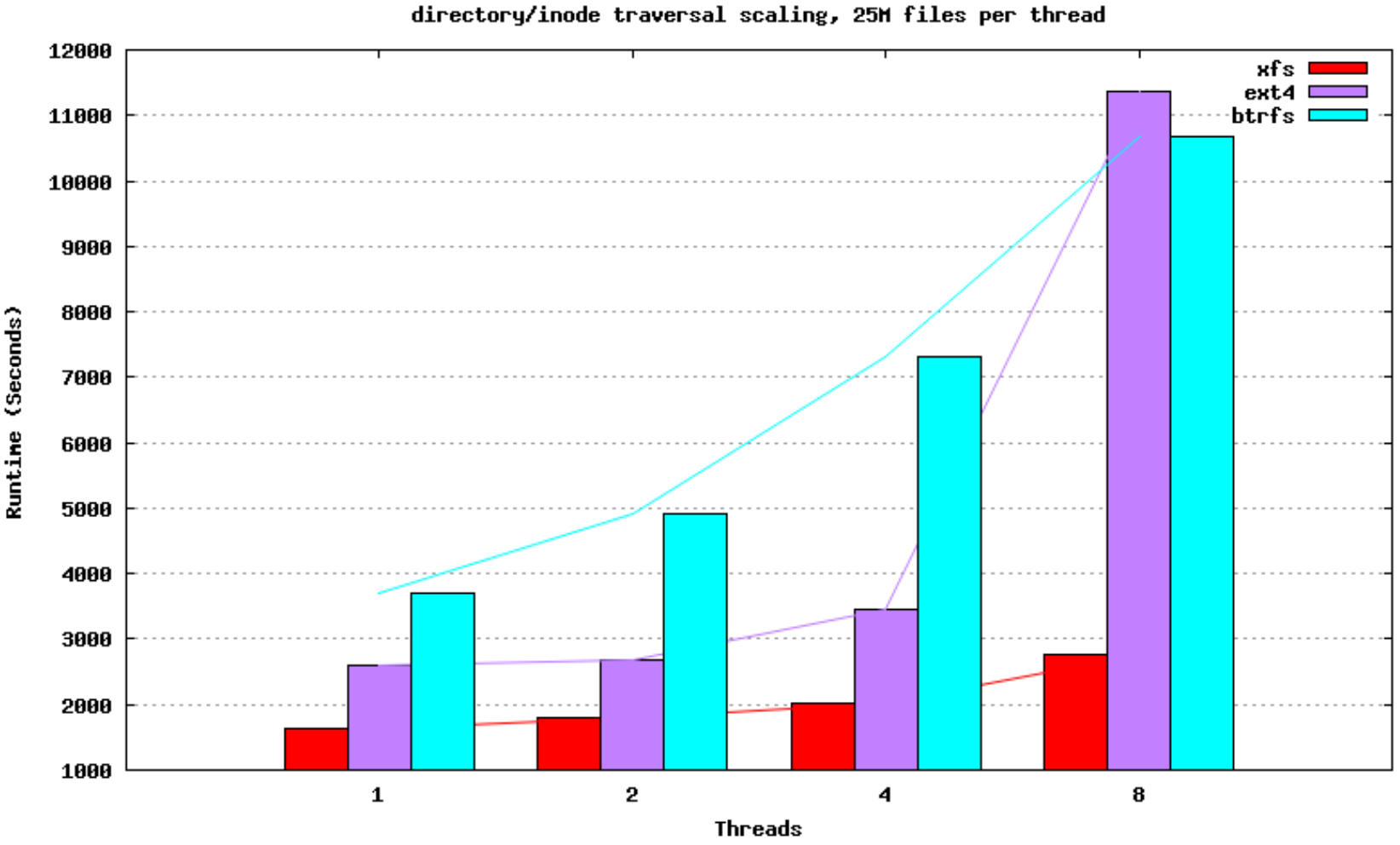
File Create Scaling



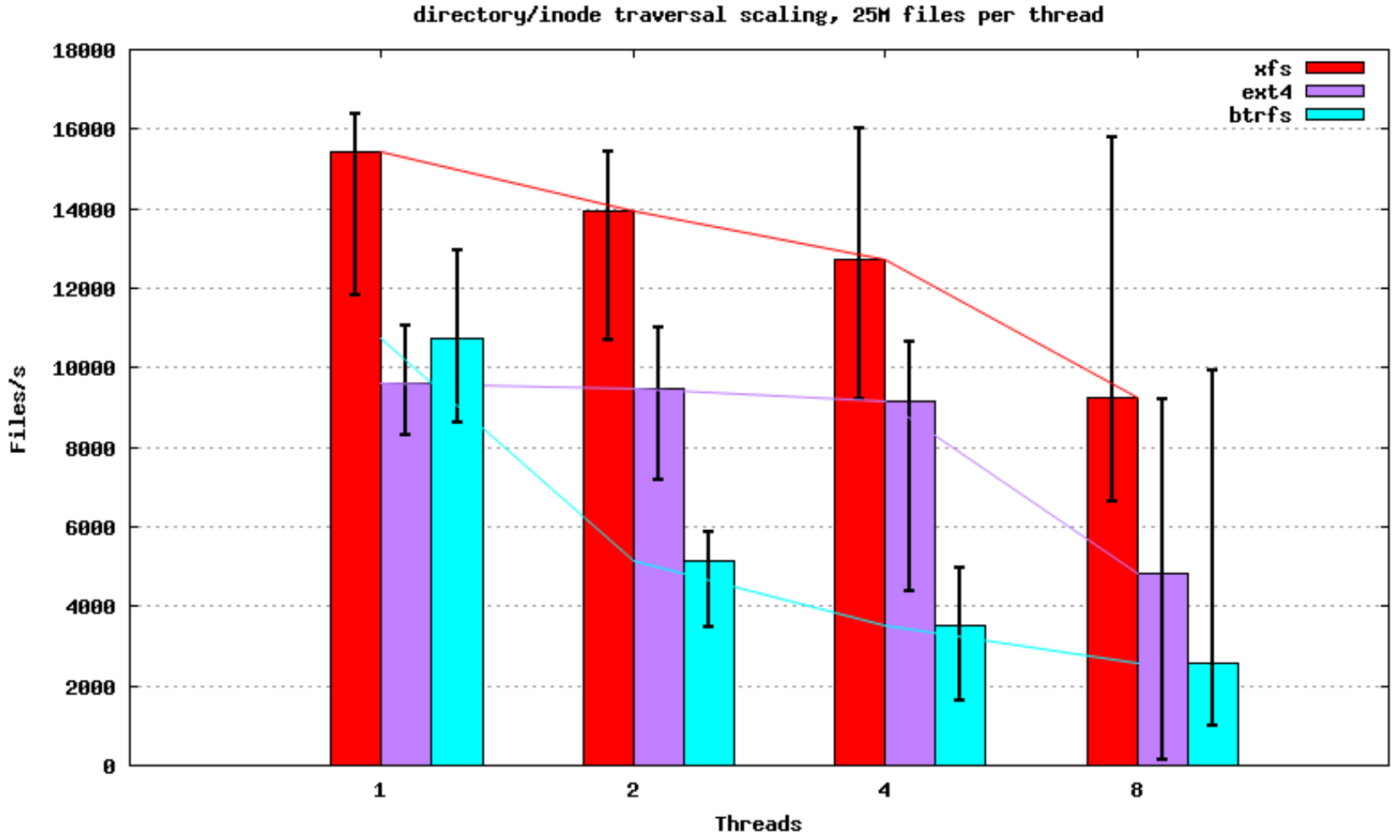
File Create Scaling



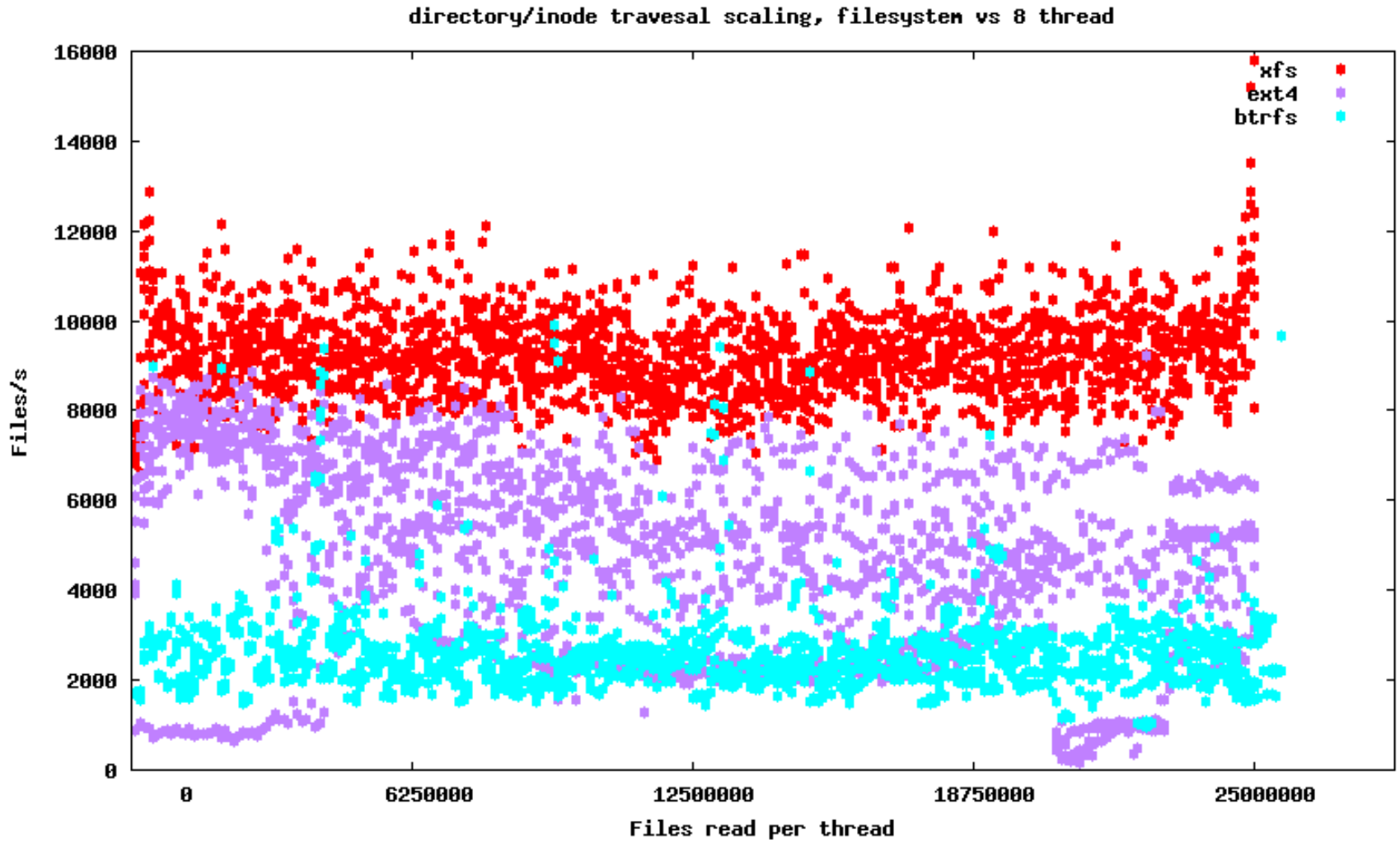
Directory Traversal Scaling



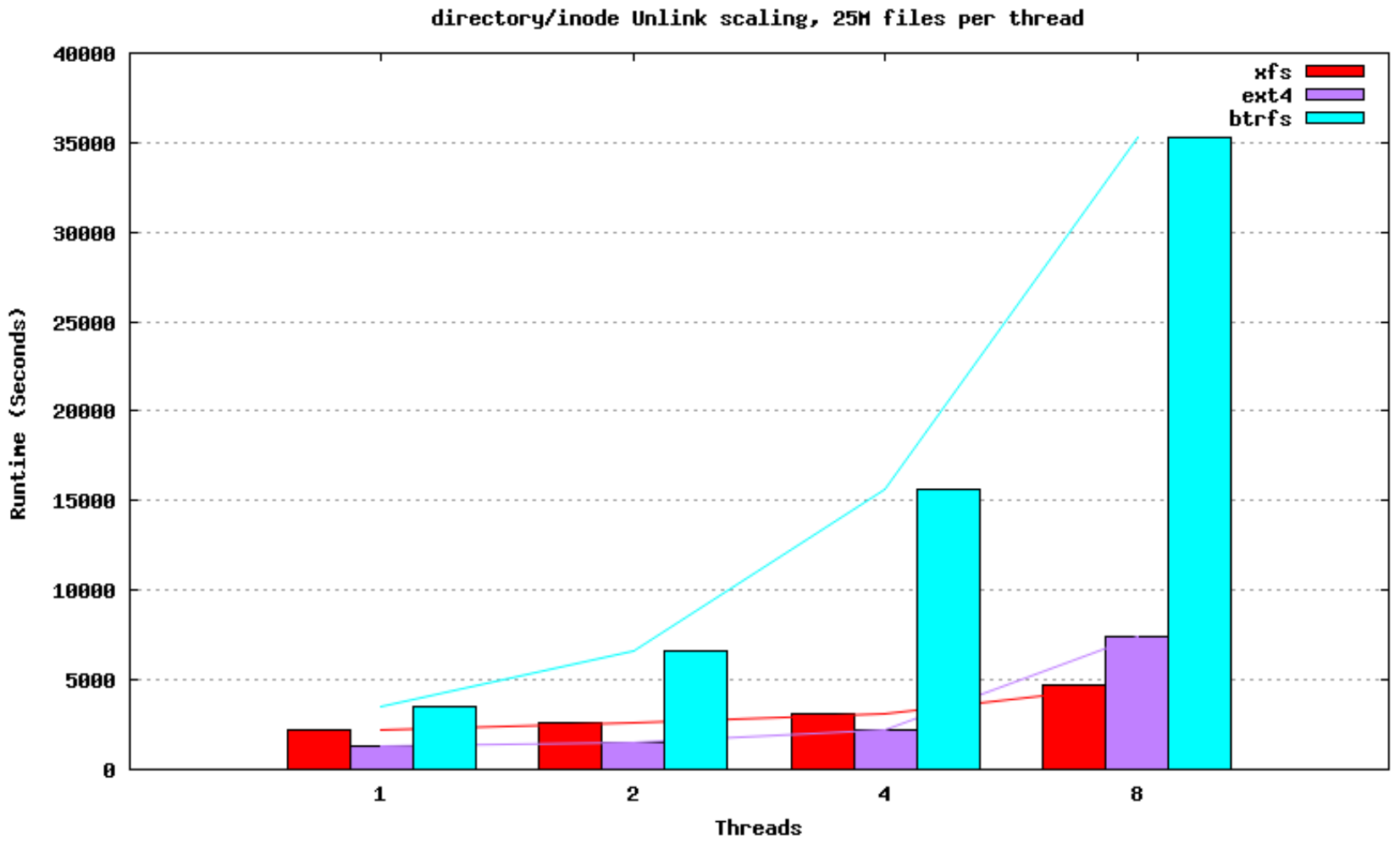
Directory Traversal Scaling



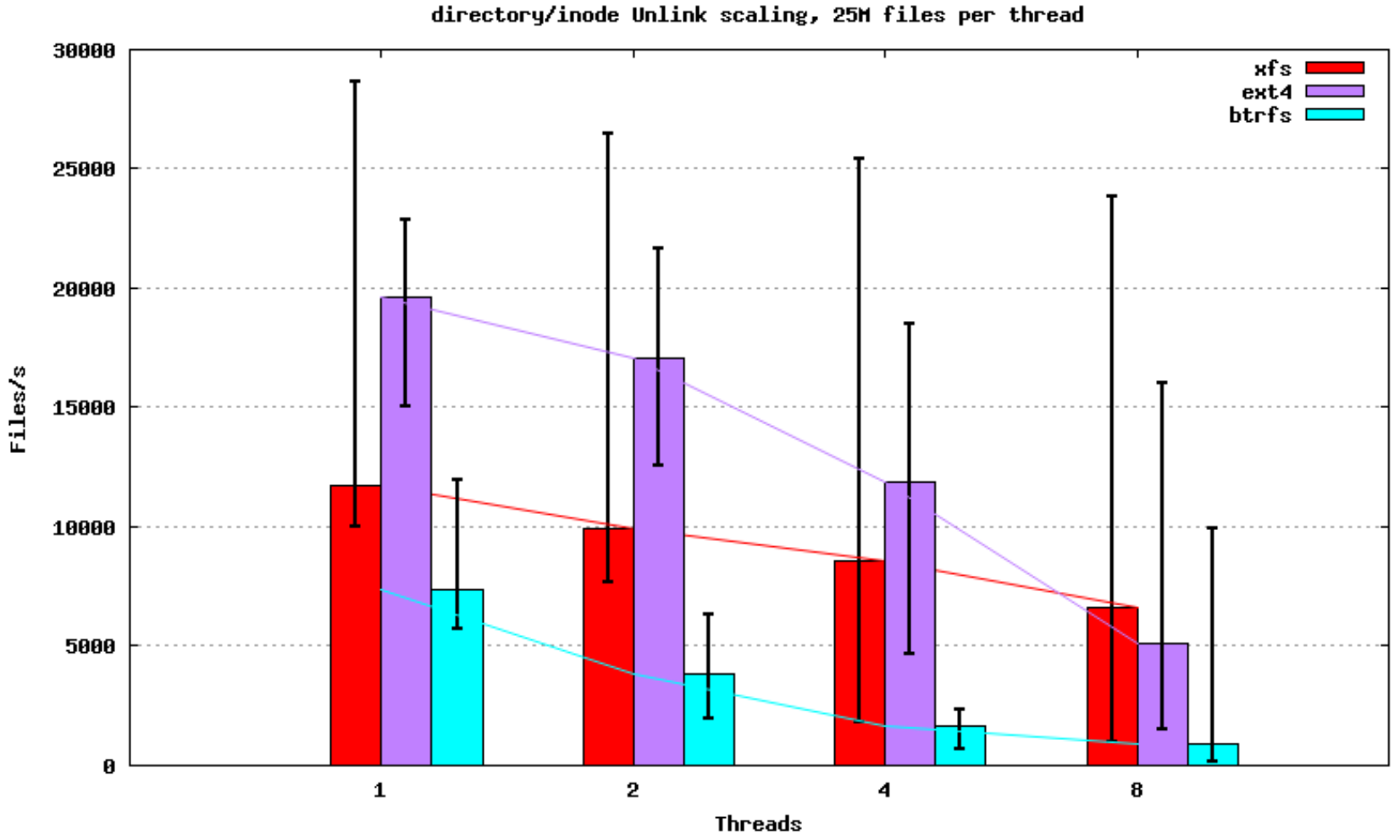
Directory Traversal Scaling



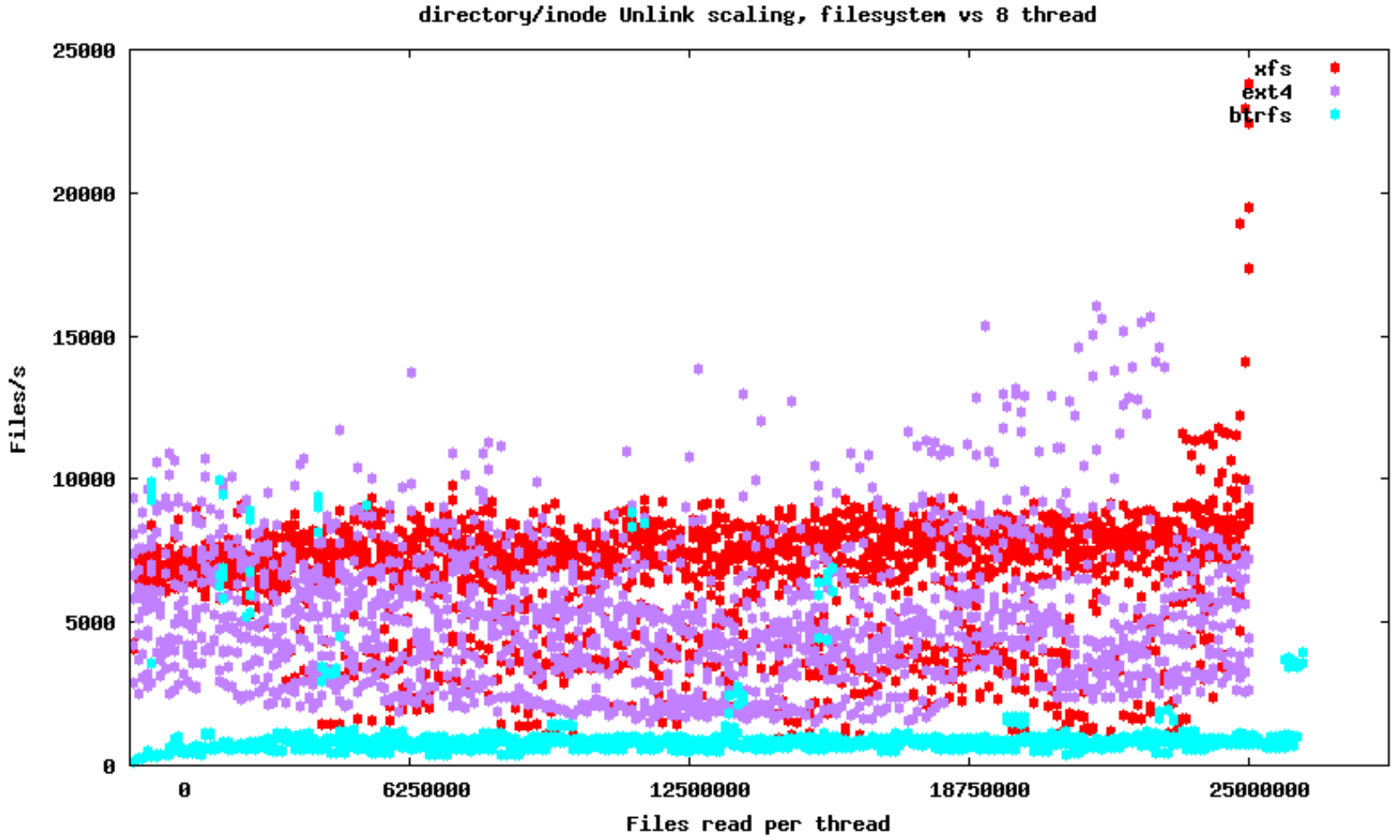
Unlink Scaling



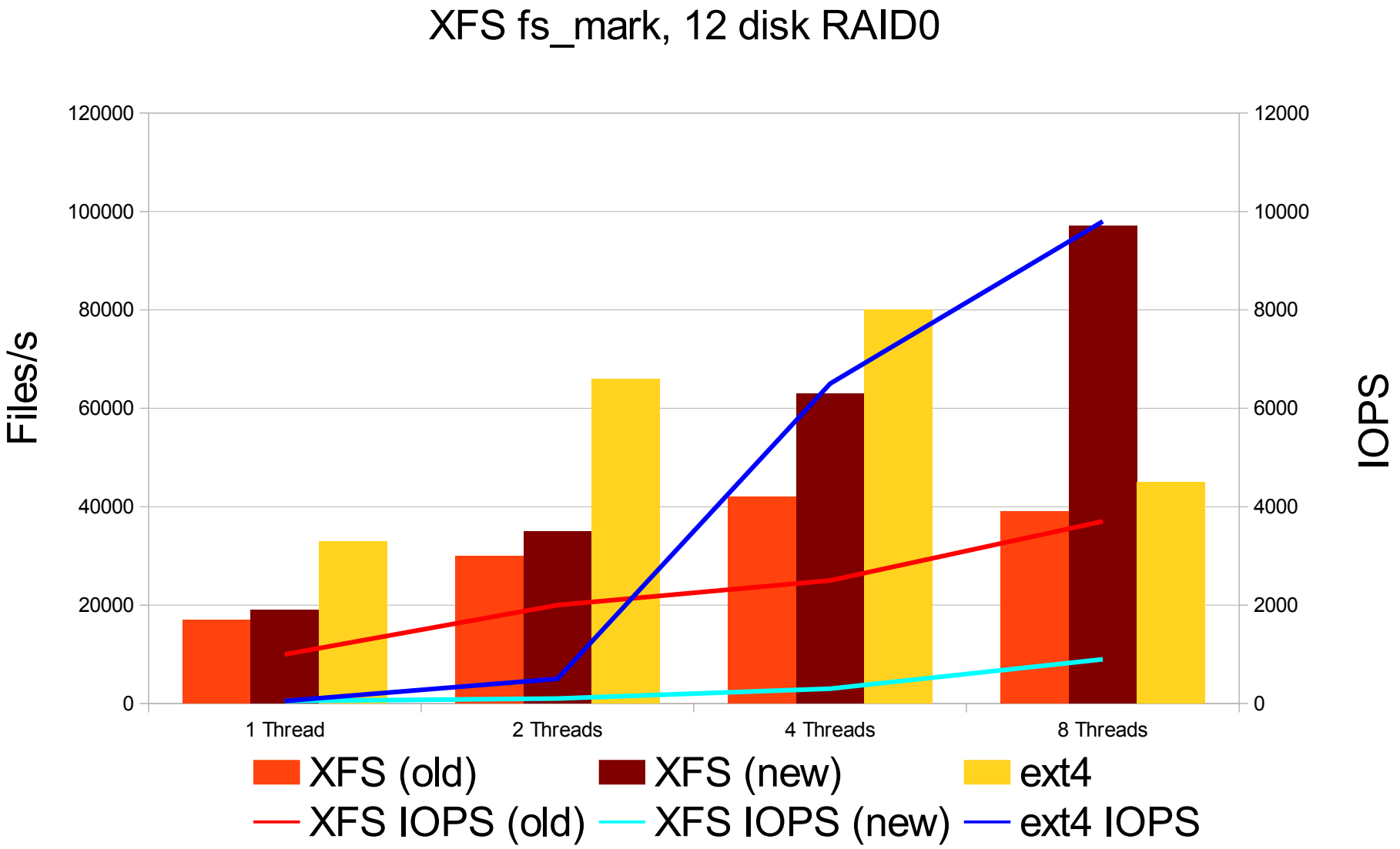
Unlink Scaling



Unlink Scaling



Just how bad? Revisited.



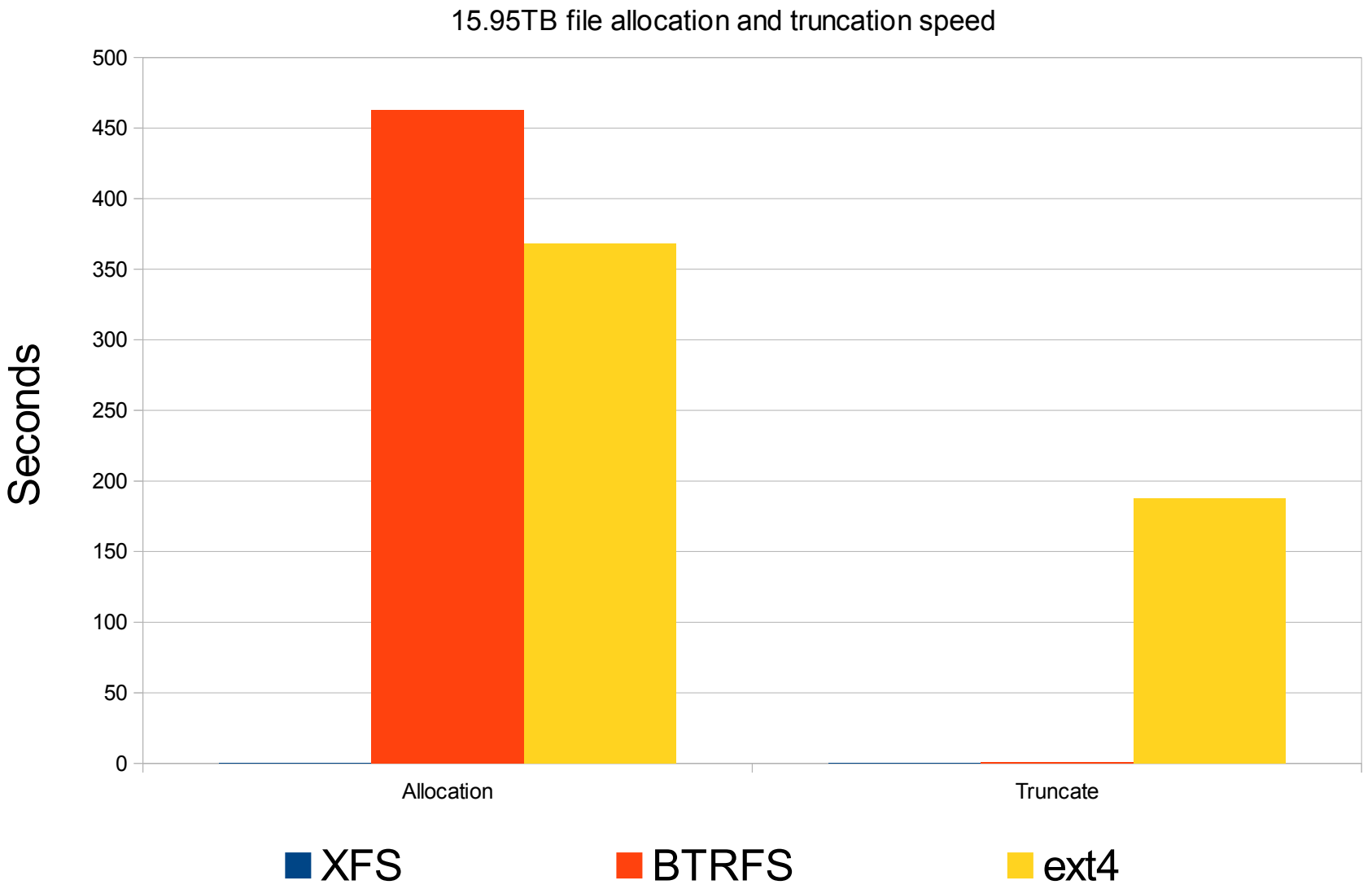
XFS isn't slow anymore

- Even on single threaded workloads, XFS is not much slower than ext4 and BTRFS anymore.
- XFS directory and inode lookup operations are faster and scale much better than ext4 and BTRFS.
- BTRFS is modification rate limited by metadata writeback during transaction reservation.
- XFS has the lowest IOPS rate at a given modification rate – both ext4 and BTRFS are IO bound at higher thread counts.

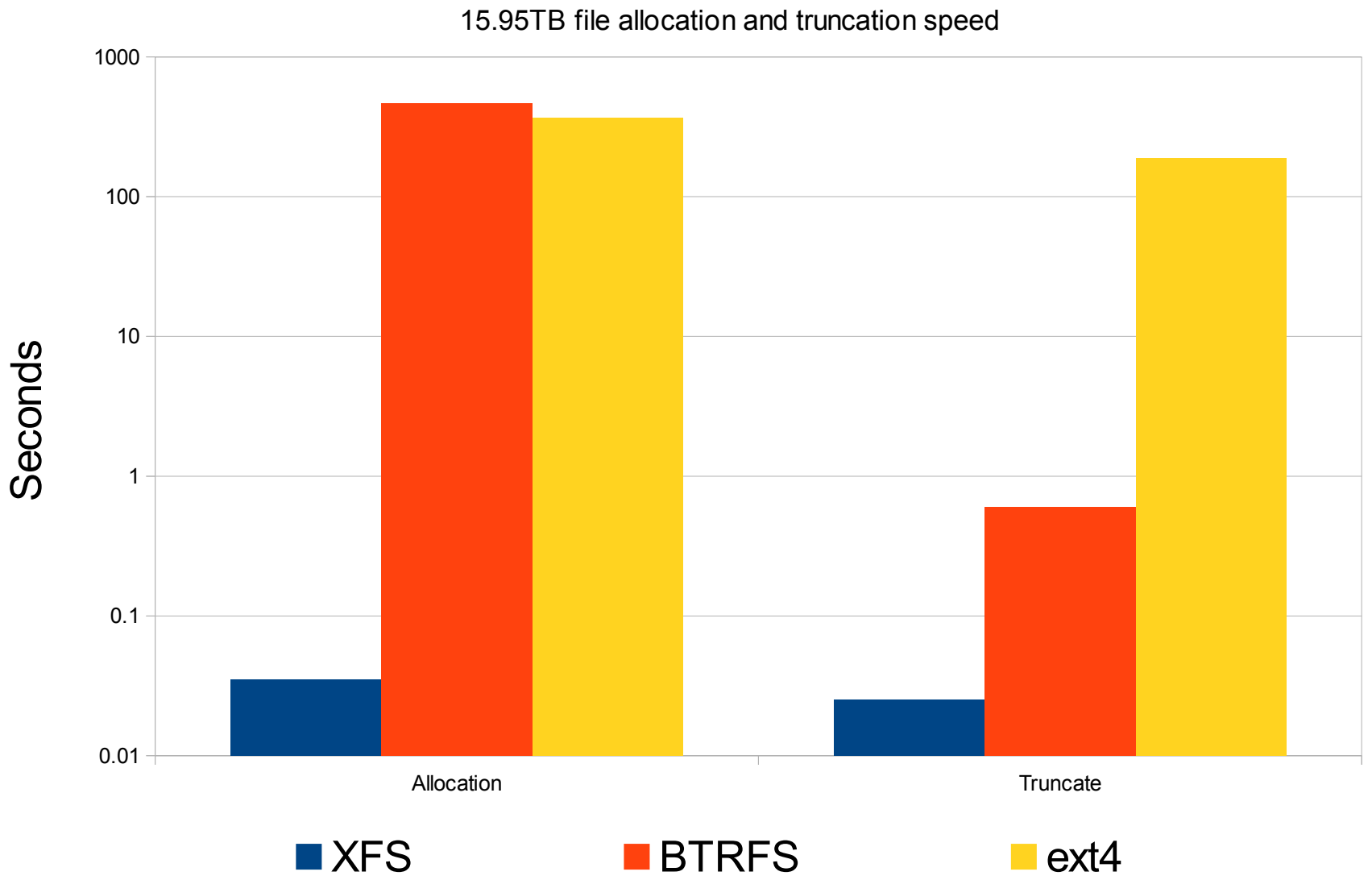
Another Look at Metadata Scalability: Allocation

- Free space indexing and allocation scalability is another aspect of metadata operations.
- XFS excels at large scale allocation.
- The other filesystems, not so much.
- Test limited to under 16TB because ext4 doesn't support file sizes more than 16TB!

Extent manipulation speed



Extent manipulation speed – logarithmic Y-axis



EXT4 Allocation scalability

- Ext4 is 4 orders of magnitude slower than XFS at large scale allocation!
- Ext4 “bigalloc” w/ 1MB cluster size
  - reduces overhead by ~2 orders of magnitude.
  - Increases small file space usage by ~2 orders of magnitude.
    - A single kernel tree now takes ~160GB of space.
  - Incompatible with various other ext4 options and functionality.
  - Introduces more complex configuration questions than it answers

Ext4 Allocation Scalability

- Architectural deficiency of a 80's era filesystem:
  - Free space is indexed by bitmaps.
  - Free space index manipulation time scales linearly with size of modification.
- **Cannot scale to arbitrarily large files and filesystems.**
  - 16TB file size limit is probably a sane maximum from this perspective.

BTRFS Allocation Scalability

- Large scale allocation is slow – slower than ext4.
  - CPU bound walking free space cache.
  - No architectural deficiency, just lack of algorithmic optimisation of the free space cache.
- Freeing is almost as fast as XFS.
  - Confirms that there is no architectural deficiency.
- Smaller allocations that don't stress free space cache lookups are extremely fast.
- Will scale to arbitrarily large filesystems with further optimisation.

Where to go from here?

- XFS metadata performance and scalability is good, can be considered mostly a solved problem.
- Further performance improvements from upcoming VFS lock scalability work.
- Validating performance scalability on high IOPS storage (e.g. PCIe SSDs).
- Improving reliability and failure resilience is the next major challenge.

Reliability is the Key to Future Scalability

- Petabyte scale filesystems could contain terabytes of metadata:
  - offline check and repair might be impossible due to time and memory requirements.
  - we need to move to online validation and repair.
- Confidence in the structural integrity requires improvements in error detection and correction.
- Also need to be able to handle transient errors (e.g. ENOMEM) better:
  - Transaction rollback support is needed here to avoid unnecessary filesystem shutdowns.

Improving Reliability and Resilience

- Robust failure detection is the most important aspect of the process.
- The first step is that metadata needs to be fully validated as correct.
- Data validation (e.g. data CRCs) is an application or storage subsystem problem, not a filesystem problem.
- Similarly, data transformation which can provide validation (e.g. compression, deduplication, encryption) are also considered to be an application or storage subsystem problem.

Improving Reliability and Resilience

- On disk format changes needed to fully validate metadata.
- No attempt to provide backwards or forwards compatibility for format changes.
  - Avoids compromising the new on-disk format design

Why Do We Need On-disk Format Changes

- CRCs are not sufficient by themselves to provide robust failure detection and recovery.
- Not enough free space in XFS metadata to add all the necessary fields without significant change.
- There is other functionality we need on-disk format changes to provide, too.
- Flag day!

Why Do We Need On-disk Format Changes

- Metadata needs to be self describing to:
  - Protect against misdirected reads and writes.
  - Detect stale metadata (e.g. from hosted filesystem images).
  - Have enough information to be able to reconnect the information to its parent if it becomes disconnected due to uncorrectable errors.
  - Be able to quickly identify the parent of a random block when the storage reports errors e.g. bad sector.

What are we adding for reliability

- Filesystem UUID to determine what filesystem the metadata came from.
- Block/inode number so we know the metadata came from the correct location.
- CRCs to detect bit errors in the metadata.
- “Owner” identifier to be able to determine who owns the metadata.
- Last modified transaction ID to ensure recovery doesn't replay modifications that have already been written to disk.
- Reverse mapping allocation btree.

What else are we adding?

- Taking advantage of the “flag day” format change to add additional feature changes:
  - d_type field in directory structure.
  - Version counters for NFSv4.
  - Inode create time.
  - Increase maximum directory sizes (up from 32GB).
  - Track directory sizes internally to allow speculative preallocation to reduce fragmentation.
  - New inode log item format so unlinked list logging is done via the inode items rather then buffers.

Why so much change?

- These are forward looking changes – not everything will initially be used.
- CRCs only prove what is read from disk is what was written.
- Other information proves it is the correct metadata.
- On-disk format changes are not particularly useful by themselves:
  - It is what we do with the additional information once it is on disk that is important.
  - Need to get it on disk first, however.

So what can we do?

- Proactive detection of filesystem corruption via online metadata scrubbing – the filesystem will find it before your application does.
- Reverse mapping allows us to:
  - Locate disconnected blocks due to corruption of structures.
  - Identify objects containing blocks that the storage says is corrupted and unrecoverable.
- Individual metadata blocks can tell us their owner.
- Enables on-line, application transparent detection and repair of certain common types of corruptions.

What does it all mean?

What does all this mean?

- From the XFS perspective:
  - Historical weakness is gone.
  - Scalability of data and metadata is unmatched.
  - Scalability of user space utilities is unmatched.
  - Feature development focused on reliability:
    - Aim to be comparable to BTRFS metadata reliability features.
  - No compromise approach to improvements.
    - Keeps implementation and testing simple.
    - Greatly limits scope for performance regressions.
  - XFS well placed to remain the “large and lots” goto Linux filesystem.

What does all this mean?

- From a BTRFS perspective:
  - Clearly not yet optimised for filesystems with large amounts of metadata.
  - About what is expected for a filesystem under heavy feature development and not yet fully stable.
  - Shows some deficiencies that might take some time to overcome (locking complexity, lookup speed).
  - Reliability features already well developed.
    - Just need to scale now!
  - Definitely capable of supporting the expected storage capabilities of the next few years.

What does all this mean?

- From a ext4 perspective:
  - Has metadata scalability issues
    - Architectural/historic deficiencies in free space indexing and directory implementation.
    - Does not handle increasing concurrency gracefully.
    - Isn't the fastest mainstream filesystem for metadata intensive workloads anymore.
  - Planned reliability improvements fall short of BTRFS and XFS.
  - On-disk format is showing its age.
  - Already struggles to handle the storage capability of the new few years.

There's a White Elephant in the Room....

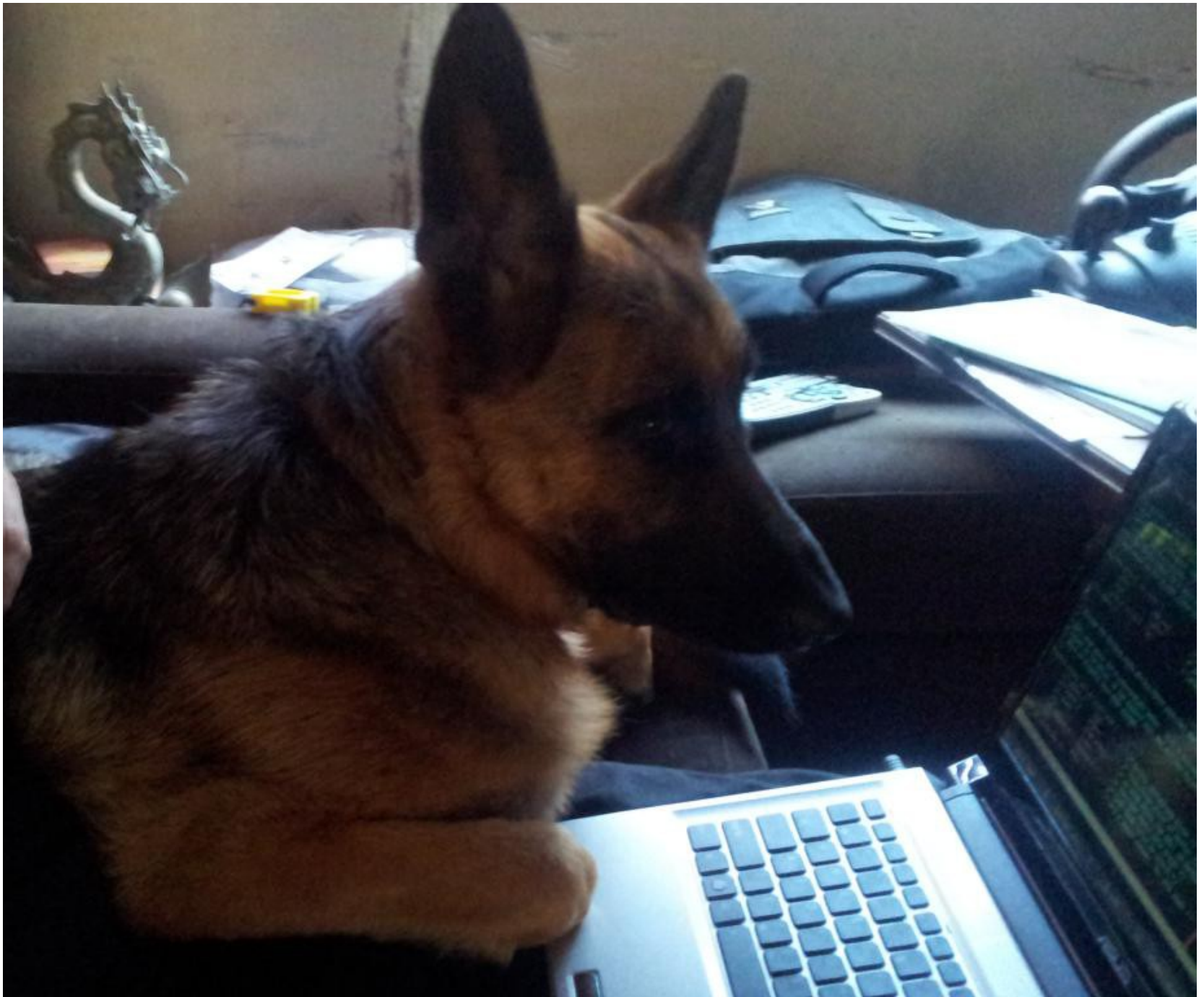
- BTRFS will soon replace ext4 as the default Linux filesystem thanks to its unique feature set.
- Ext4 is now being outperformed by XFS on its traditionally strong workloads, but is unable to compete with XFS where it is traditionally strong.
- Ext4 has serious scalability challenges to be useful on current, sub-$10,000 server hardware.
- Ext4 has become an aggregation of semi-finished projects that don't play well with each other.
- Ext4 is not as stable or as well tested as most people think.

There's a White Elephant in the Room....

- With the speed, performance and capability of XFS and the maturing of BTRFS, why do we need EXT4 anymore?

Questions and Flames?

XFS Code Review in Progress



Other recent XFS Kernel Features

- Background discard (FITRIM)
- Online discard
- All fallocate modes supported
- greatly simplified syscall->page cache IO path
- greatly simplified writeback (pagecache->disk) IO path
- factored and simplified internal allocation interface
- Many other cleanups and simplifcations
- Speculative allocation improvements to minimise fragmentation in concurrent write workloads.

Recent Userspace features

- major libxfs update to sync with 2.6.38 kernel code
- xfs_repair now checks everything xfs_check does
- xfs_repair has significant improvements in error detection and correction, as well as some memory usage reductions.
- xfs_check is effectively deprecated
- mkfs.xfs 4k sector support
- mkfs.xfs TRIM support
- xfsdump multi-streamed dump support
- Many bug fixes, translation fixes and other minor changes