ORACLE

Filesystem Features and Performance

Chris Mason

Filesystems

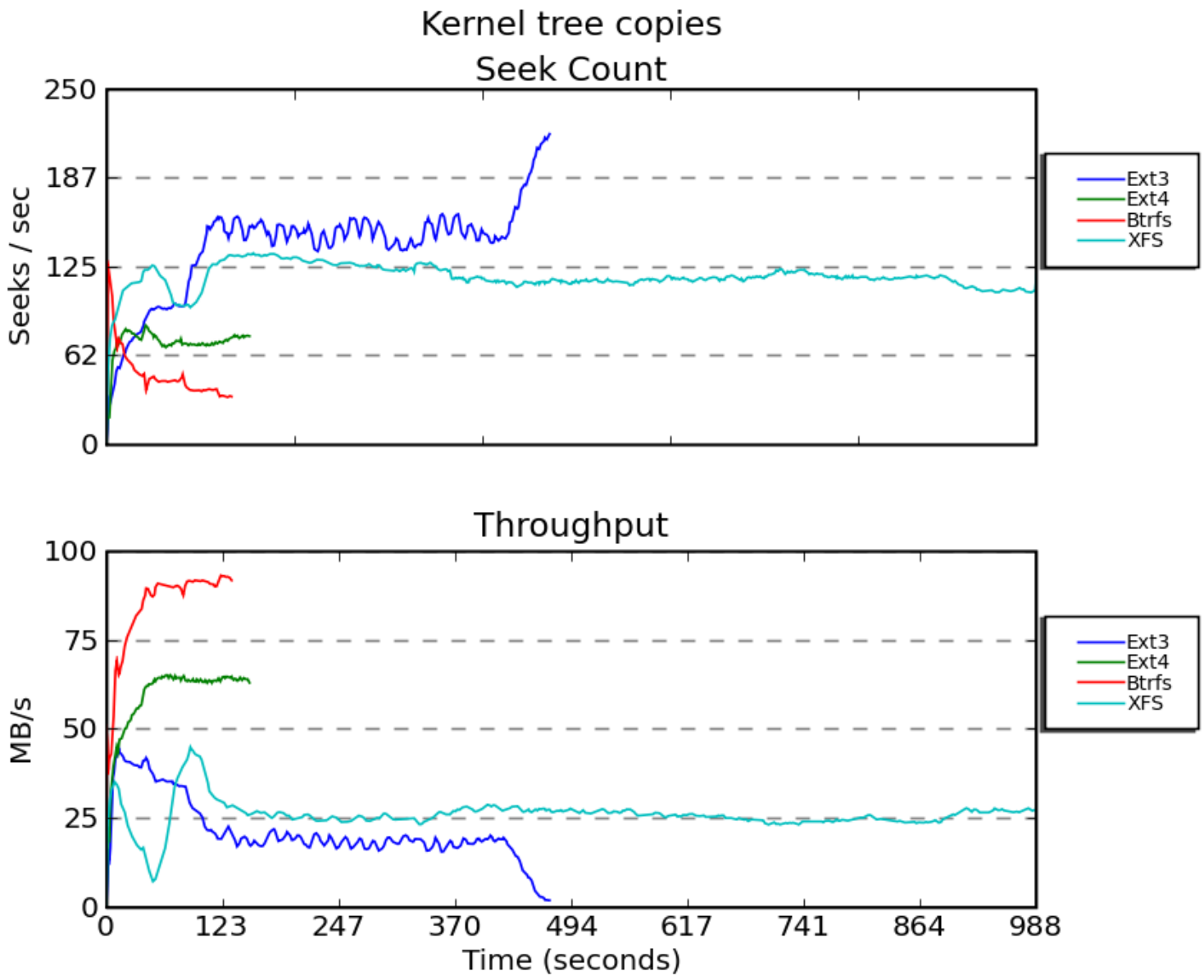
- XFS
  - Well established and stable
  - Highly scalable under many workloads
  - Can be slower in metadata intensive workloads
  - Often the best option for production use today
- Ext4
  - Can read ext3 filesystems
  - Includes delayed allocation and extents
  - Lifts many limits from ext3
- Btrfs
  - Many new features

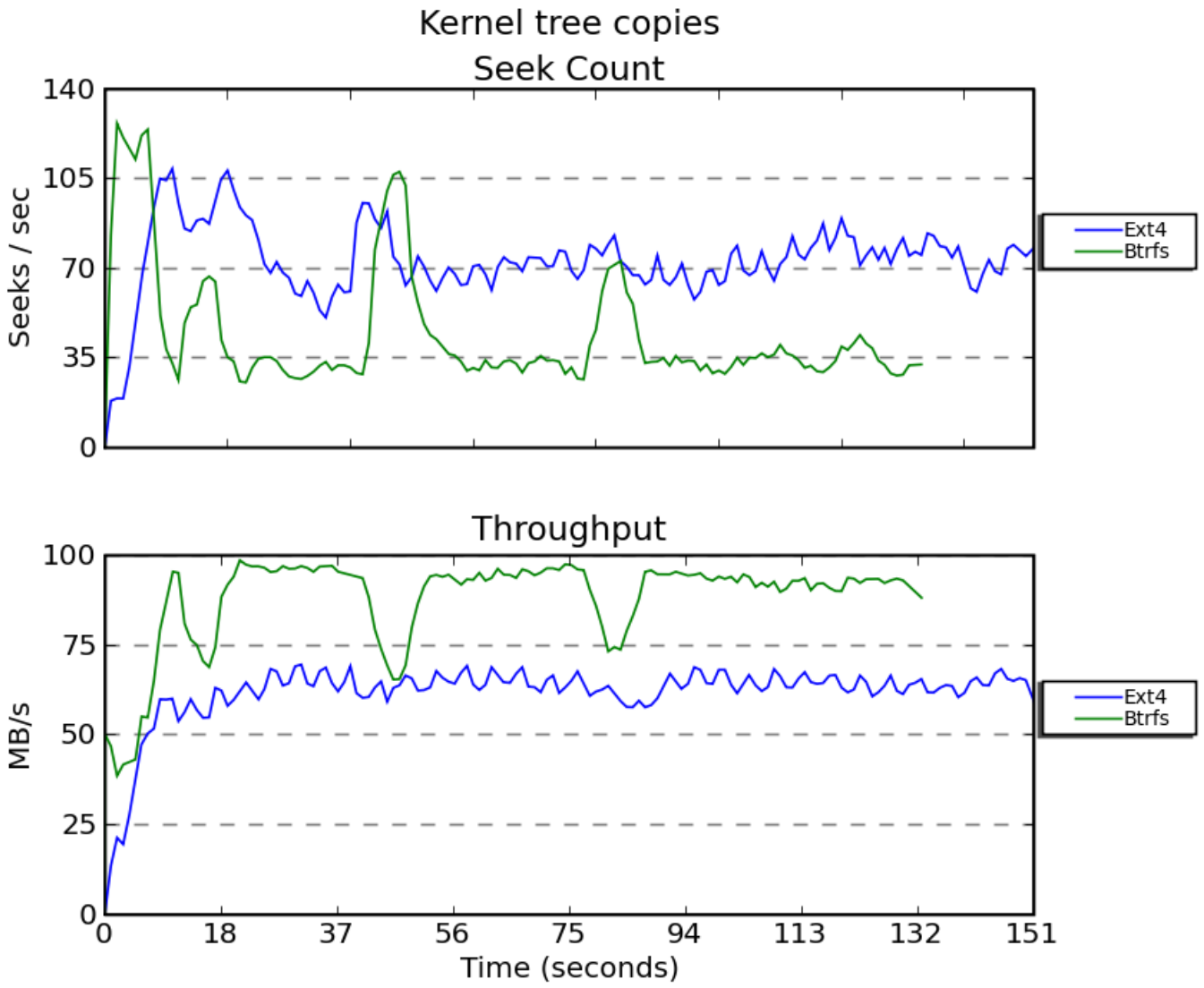
Filesystem Performance Metrics

- File allocation and writeback
  - Extents are much more efficient
  - Crash recovery log can be a major performance factor
- Directory indexing
  - Every FS does this differently
  - Major impact on read and write performance
- Synchronous Writes
  - Expose bottlenecks in many different parts of the FS

File allocation and Writeback

- Delayed allocation
  - Don't allocate blocks on disk until just before dirty pages are written. All 3 filesystems do this
- Extent based storage
  - Allocates a range of bytes instead of a fixed unit
  - Much more efficient for large files
- Transaction log
  - XFS uses a logical log of operations
  - Ext3 and Ext4 use a write ahead block log
  - Btrfs uses copy on write (COW)

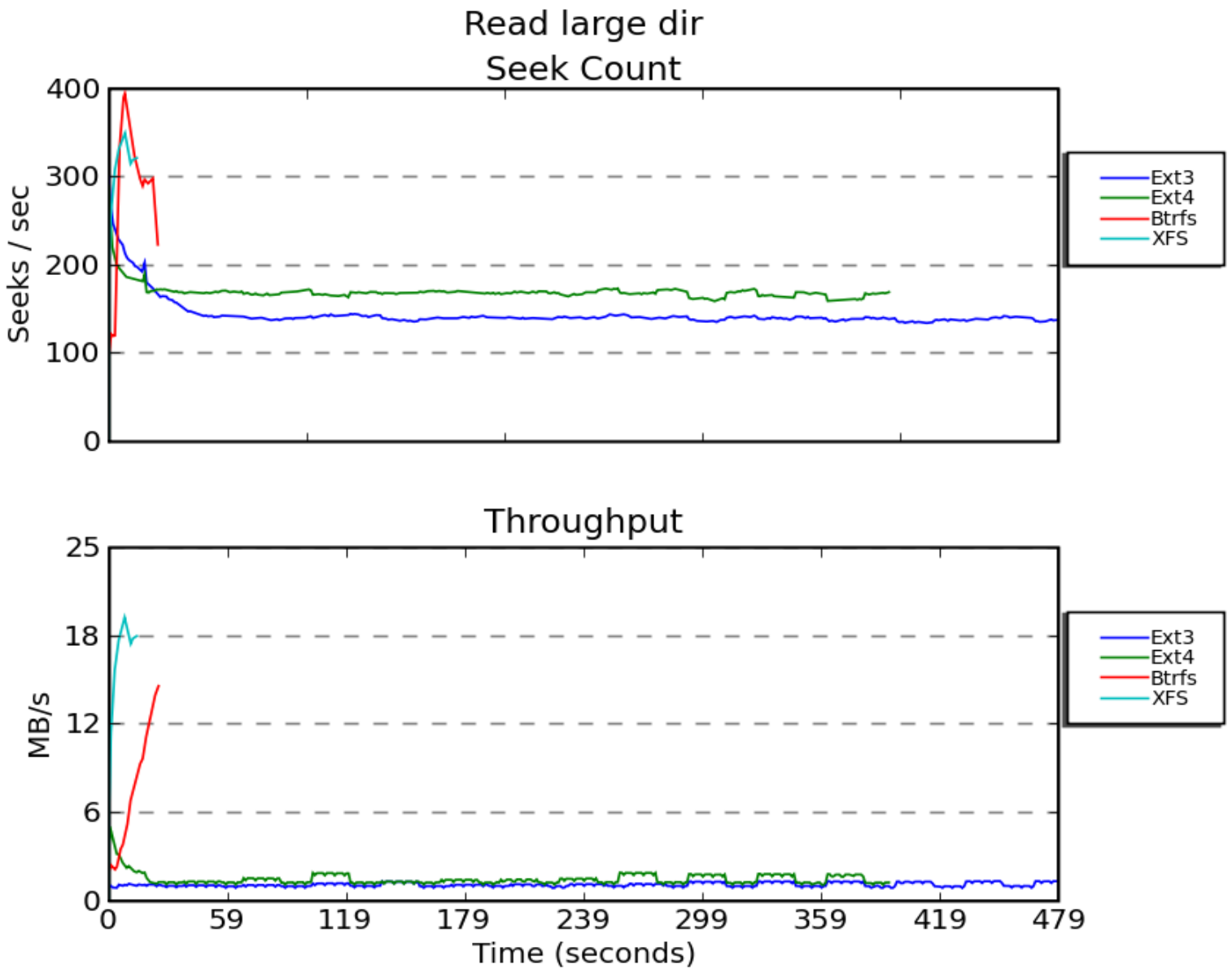

Kernel tree copies
Seek Count
Seeks / sec
250
187
125
62
0
Ext3
Ext4
Btrfs
XFS
Throughput
MB/s
100
75
50
25
0
0
123
247
370
494
617
741
864
988
Time (seconds)
Ext3
Ext4
Btrfs
XFS


Kernel tree copies
Seek Count
Seeks / sec
140
105
70
35
0
Ext4
Btrfs
Throughput
MB/s
100
75
50
25
0
0
18
37
56
75
94
113
132
151
Time (seconds)
Ext4
Btrfs

Kernel Tree Copy Movies

Directory Indexing

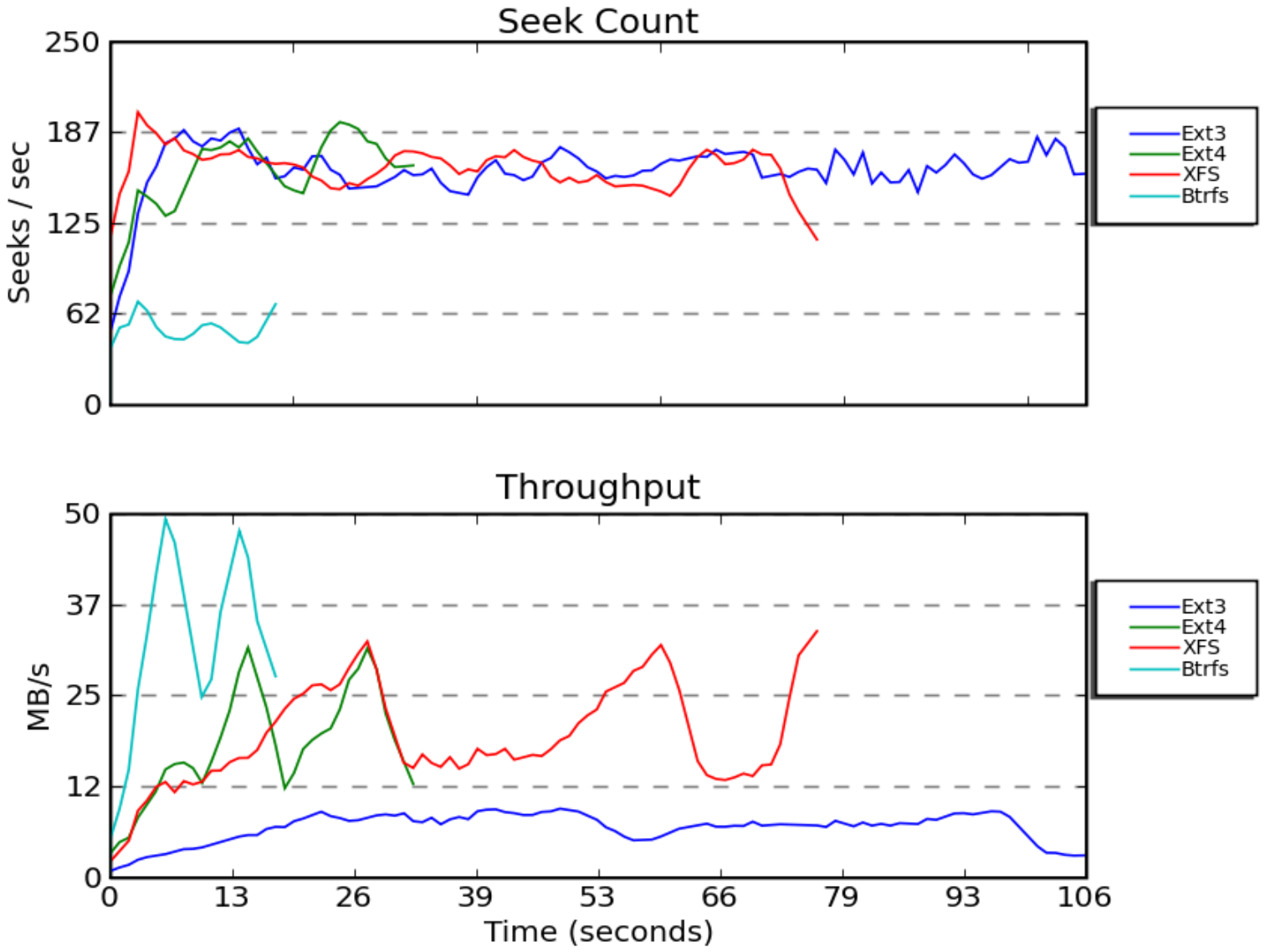
- Directory indexes provide better performance in large directories.
- They also determine the order the filesystem returns things to readdir
  - Used by backup programs to read all the files in the directory
- XFS indexes with a specialized btree
- Btrfs indexes once by name and once by sequence number
- Ext3 and Ext4 use htree
  - Very bad results for readdir on large directories


Read large dir
Seek Count
Seeks / sec
400
300
200
100
0
Ext3
Ext4
Btrfs
XFS
Throughput
MB/s
25
18
12
6
0
0
59
119
179
239
299
359
419
479
Time (seconds)
Ext3
Ext4
Btrfs
XFS

Directory Read Movies

Synchronous Writes

- Performance depends on logging subsystem, data writeback and data allocation policies
- Synchronous writes can have a big impact on other work being done by the filesystem
- Barriers are used by default on Ext4, Btrfs, and XFS to protect against corruption on power failures
- Btrfs uses a specialize log to improve synchronous performance


Seek Count
Seeks / sec
250
187
125
62
0
Ext3
Ext4
XFS
Btrfs
Throughput
MB/s
50
37
25
12
0
0
13
26
39
53
66
79
93
106
Time (seconds)
Ext3
Ext4
XFS
Btrfs

Synchronous write movies

Btrfs Goals

- General purpose filesystem that scales to very large storage
- Fast development cycle to attract developers and users early
- Feature focused, providing features other Linux filesystems cannot
- Administration focused, easy to run and very fault tolerant
- Perform well in a variety of workloads

Btrfs Features

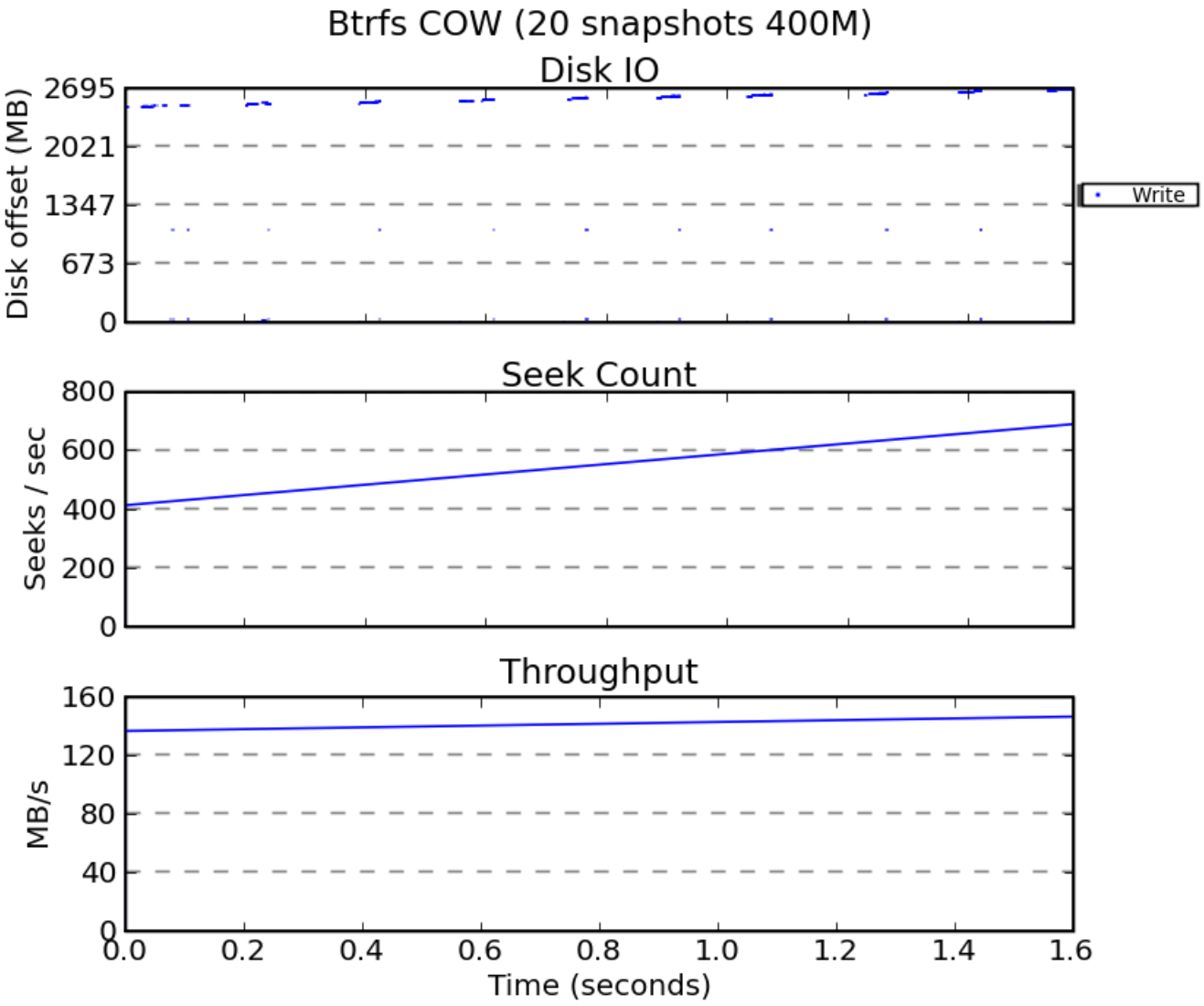
- Extent based file storage
- Copy on write metadata and data
- Space efficient packing of small files
- Optional transparent compression (zlib)
- Integrity checksumming for data and metadata
- Writable snapshots
- Efficient incremental backups
- Multiple device support
- Offline conversion from Ext3

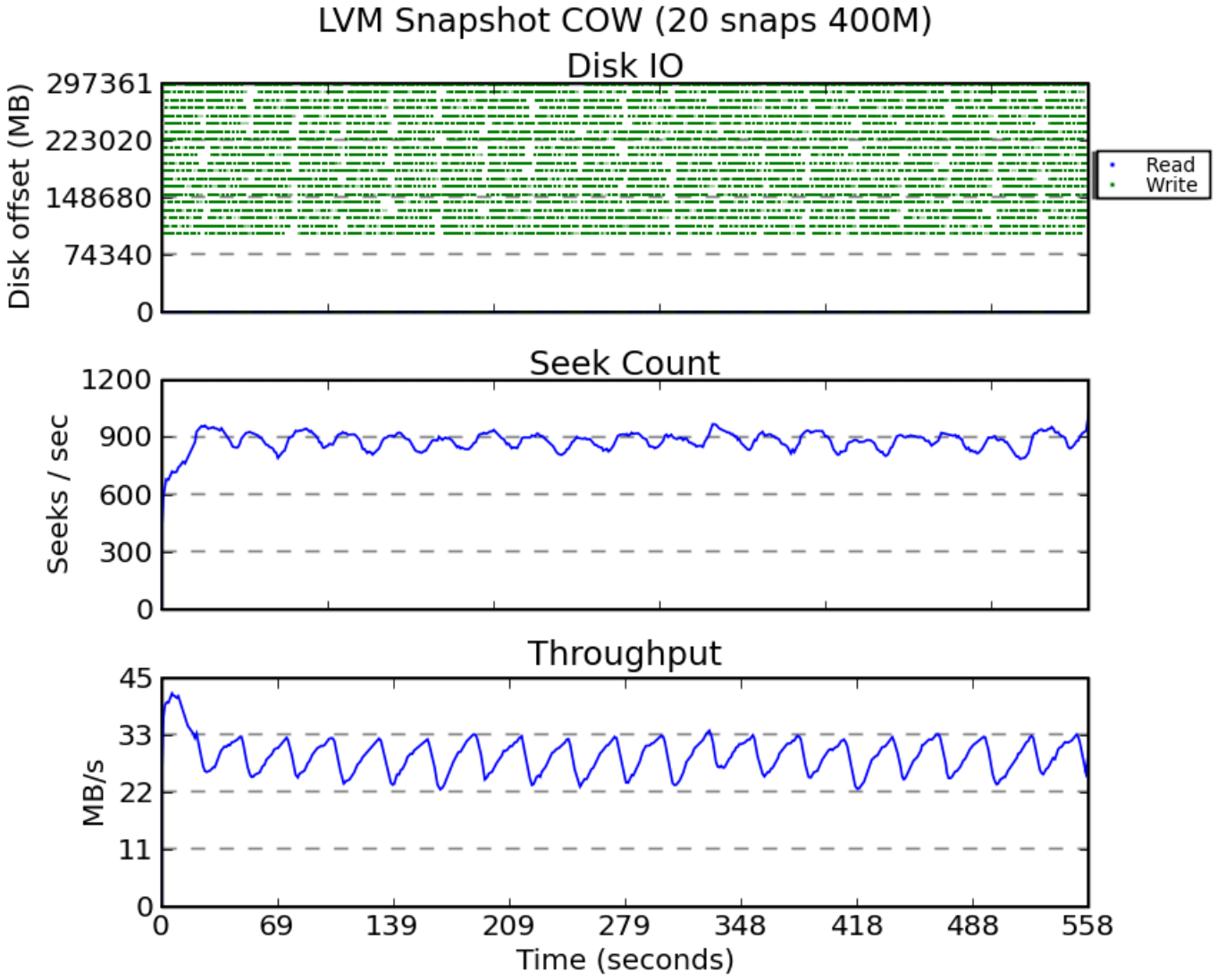
Btrfs Status

- Included in 2.6.29-rc1
- Generally usable in many workloads
- Generally stable

Snapshots and Subvolumes

- Subvolume is the unit of snapshotting
- As efficient as directories on disk
- Every transaction is a new unnamed snapshot
- Every commit schedules the old snapshot for deletion
- Snapshots use a reference counted COW algorithm to track blocks
- Snapshots can be written and snapshotted again
- COW leaves old data in place and writes the new data to a new location


Btrfs COW (20 snapshots 400M)
Disk IO
Disk offset (MB)
2695
2021
1347
673
0
Write
Seek Count
Seeks / sec
800
600
400
200
0
Throughput
MB/s
160
120
80
40
0
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
Time (seconds)


LVM Snapshot COW (20 snaps 400M)
Disk IO
Disk offset (MB)
297361
223020
148680
74340
0
Read
Write
Seek Count
Seeks / sec
1200
900
600
300
0
Throughput
MB/s
45
33
22
11
0
0
69
139
209
279
348
418
488
558
Time (seconds)

Multi-device Support

- Devices are added into a pool of available storage
- New logical address space is allocated with a specific RAID configuration and data storage flags
  - System (used by the volume management code)
  - Metadata
  - Data
  - Raid0, raid1, raid10, single-spindle-dup
- Space is allocated from the storage pool in large chunks (1GB or more)
- Devices can be mixed in size and speed

Multiple Device Managment

- Create the filesystem
  - mkfs.btrfs /dev/sdb
  - Mount /dev/sdb /mnt
- Add some devices
  - Btrfs-vol -a /dev/sdc /mnt
  - Btrfs-vol -a /dev/sdd /mnt
- Redistribute the data to the new devices
  - Btrfs-vol -b /mnt
  - This also restripes as required
- Remove a device
  - Btrfs-vol -r /dev/sdc /mnt

Online Resizing

- Shrink online by 1GB
  - Btrfsctl -r -1g /mnt
- Shrink a specific device
  - Btrfs-show # find device ids
  - Btrfsctl -r 2:-1g /mnt
    - 2: indicates to shrink devid 2
- Grow a device online
  - Btrfsctl -r 2:+1g /mnt
- Interfaces will be expanded to allow raid reconfiguration and advanced device management online

Synchronous Operations

- COW transaction subsystem is slow for frequent commits
  - Forces recow of many blocks
  - Forces significant amounts of IO writing out extent allocation metadata
- Simple write ahead log added for synchronous operations on files or directories
- File or directory Items are copied into a dedicated tree
  - File back refs allow us to log file names without the directory
  - One log btree per subvolume

Synchronous Operations

- The log tree uses the same COW btree code as the rest of the FS
- The log tree uses the same writeback code as the rest of the FS, and uses the metadata raid policy.
- Commits of the log tree are separate from commits in the main transaction code.
  - fsync(file) only writes metadata for that one file
  - fsync(file) does not trigger writeback of any other data blocks

Mixed Fsync Workload

- 8 Processes creating, writing and fsyncing 20k files
- 1 Process creating 30 copies of the linux kernel sources in sequence

	Kernel tree I/O Rate	Fsync Creates
Btrfs	83MB/s	15,680
Ext4	122MB/s	1280
Ext3	66.85MB/s	3200

data=ordered

- Implemented to make sure that after a crash, files don't point to extents that have not been written
  - Prevents exposing stale data from deleted files
- Ext3 and Ext4
  - Write all dirty data before transaction commits
  - A single fsync will sync all dirty file non-delalloc data in the FS
- XFS and Btrfs
  - Update file metadata only after data is on disk

SSD

- Very fast for reads, no seek penalties
- Becoming very fast for writes, much smaller seek penalties
- Changing the design of storage subsystems, databases and filesystems