Silicon Graphics, Inc.

Exploring High Bandwidth Filesystems on Large Systems

Presented by:

David Chinner

Jeremy Higdon

August 9, 2006

sgi

Background

- History of filesystem performance comparisons on small machines.
- Indicate the maximum performance achievable by a small number of disks, not the maximum the kernel or the filesystem can achieve.
- Increasing bandwidth requirements from customers:
  - Past: < 300MiB/s from 2.4.x kernels.
  - Present: 1-2GiB/s from early 2.6.x kernels.
  - Future: > 5GiB/s from current 2.6.x kernels.

Background, Part 2

- Dual- and multi-core CPUs are leading to higher parallelism on mainstream servers and desktops.

- Memory Capacity:
  - Increasing substantially faster than disk transfer rates.
  - More disks required to fill memory at the same proportional rate.

- Storage per disk:
  - Increasing faster than transfer rates.
  - Less disks are required for a given storage capacity.
  - Need to maximise per-disk throughput.

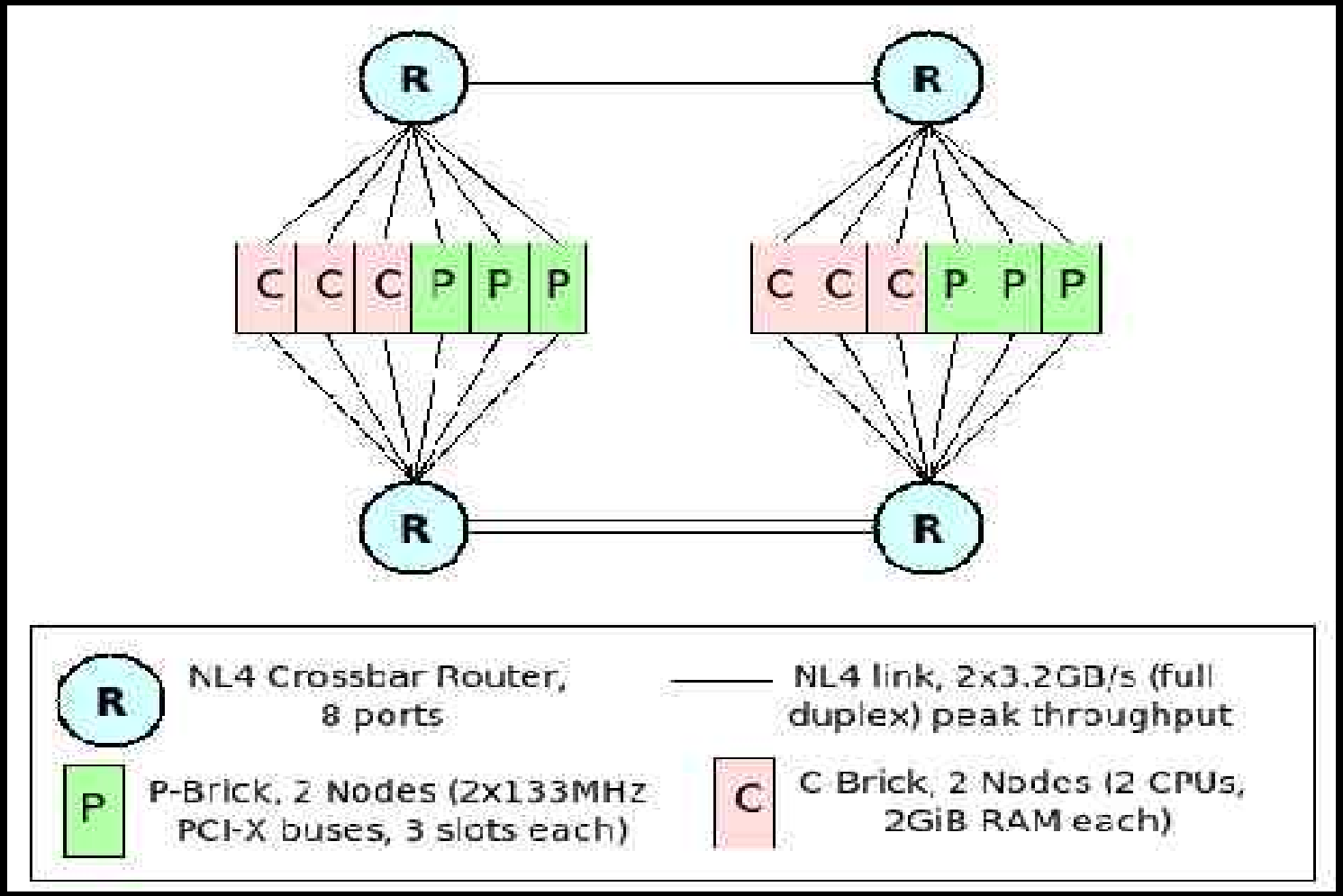
Need To Know.....

- Can we reach the physical hardware limits on a big machine?
- Is Linux stable under these loads?
- What VM issues arise under these loads?
- Are there NUMA issues we need to address?
- Do we have new file fragmentation problems?
- Are there other bottlenecks we do not know about?
- Can we expect customer workloads to achieve the same results?

Test Hardware

- Altix 3700:
  - 24x 1.5GHz Itanium 2 CPUs.
  - 24GiB RAM.
  - Cache coherent NUMALink 4 interconnect between nodes.
  - 36 x 133MHz PCI-X slots directly attached to the interconnect.
  - 64 HBAs, U320 SCSI and Fibre Channel.
  - 256 disks, 4 to each controller.
- Theoretical maximum sustained disk throughput is approximately 11.5GiB/s.
- Memory bandwidth and bi-sectional interconnect bandwidth substantially higher than this.

Machine Topology



sgi

Test Methodology

- Many configuration variables to be explored such as:
  - Volume and block device parameters.
  - different filesystems and filesystem parameters.
  - different I/O sizes and thread counts.
  - memory and process placement.
- PCP (`http://oss.sgi.com/projects/pcp`) used to archive monitored parameters for off-line analysis.
- Many parameters to be monitored including:
  - CPU, disk and memory utilisation.
  - VM behaviour.
  - Filesystem and volume manager behaviour (if supported)

Test Methodology

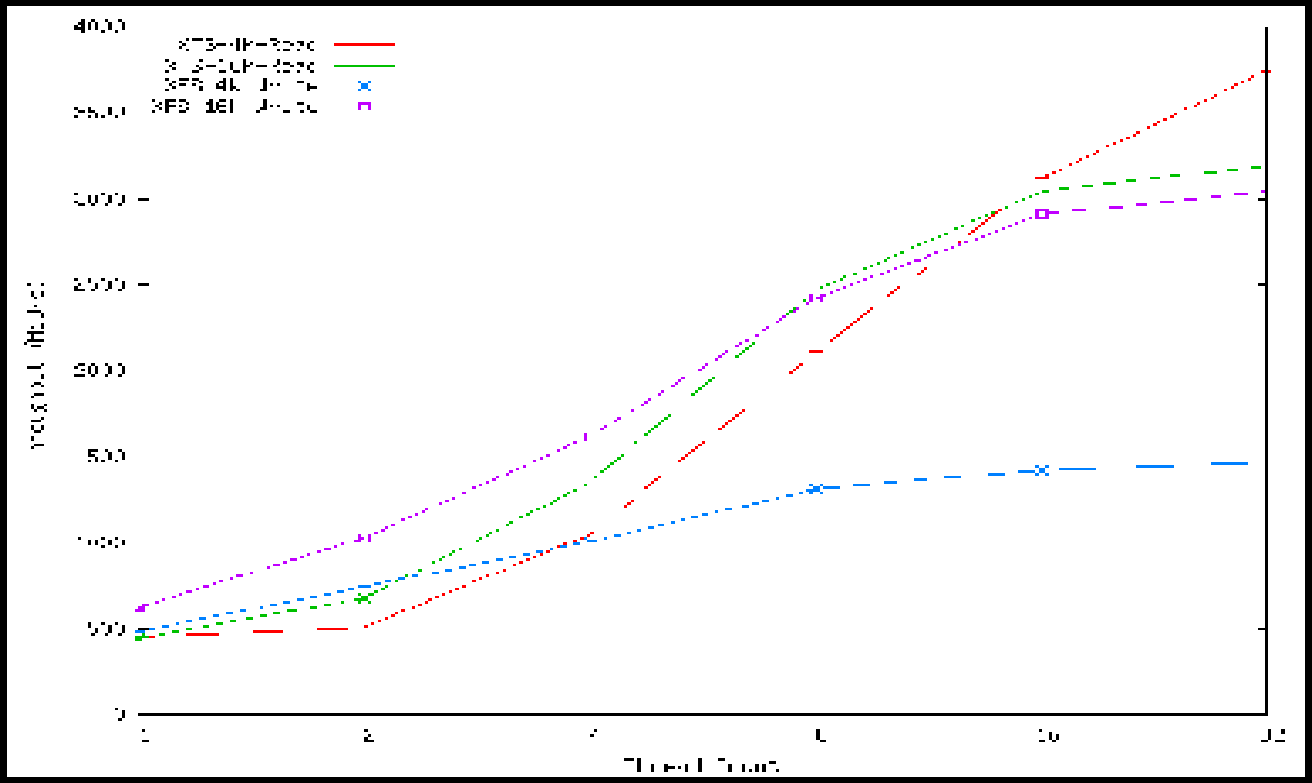
- Test Preparation:
  - Configure volume and filesystem with specific test parameters.
  - Mount filesystem, write out N “read” files with increasing levels of parallelism, then unmount filesystem.
- Repeat for different numbers of threads and I/O block sizes:
  - Mount filesystem.
  - Read n “read” files using I/O size B.
  - Unmount, remount filesystem.
  - Write n “write” files using I/O size B after first truncating them to zero length.
  - Unmount filesystem.

Volume Layout

- 1024 character line length limit for `dmsetup`:
  - Could not set up a flat 256 disk RAID-0 stripe.
  - Roughly 90 disks to a stripe was the limit.
- 2-tier RAID-0 stripe using MD and DM:
  - 4x64 disk DM RAID-0 stripes.
  - MD RAID-0 stripe of DM volumes.
  - Same stripe unit and stripe width as a flat stripe.
- SGI's XVM volume manager was used to confirm that the 2-tier stripe gave equivalent performance to a flat stripe.
- Stripe unit and width were varied to find the configuration that resulted in optimal throughput.

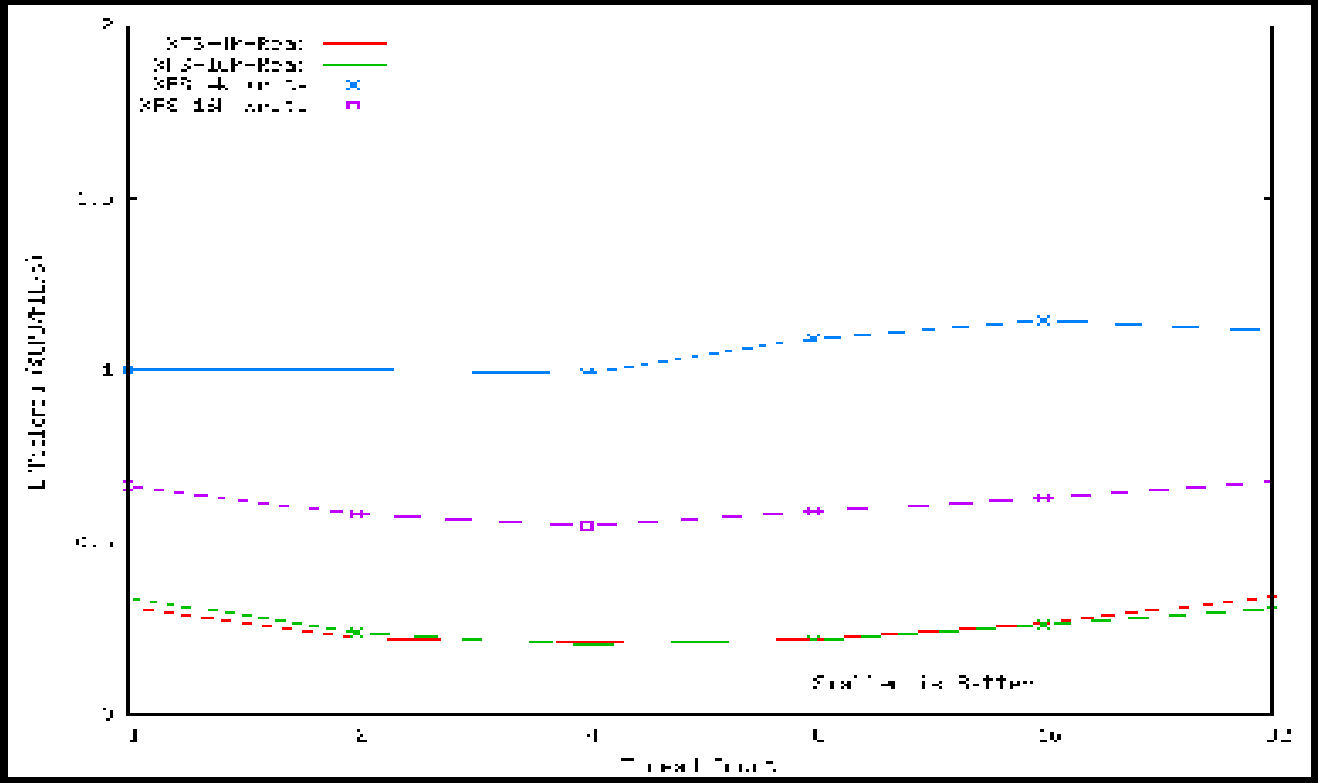
Results: Baseline XFS

- SuSE Linux Enterprise Server 9, Service Pack 2.
- Buffered I/O, 128KiB I/O size.



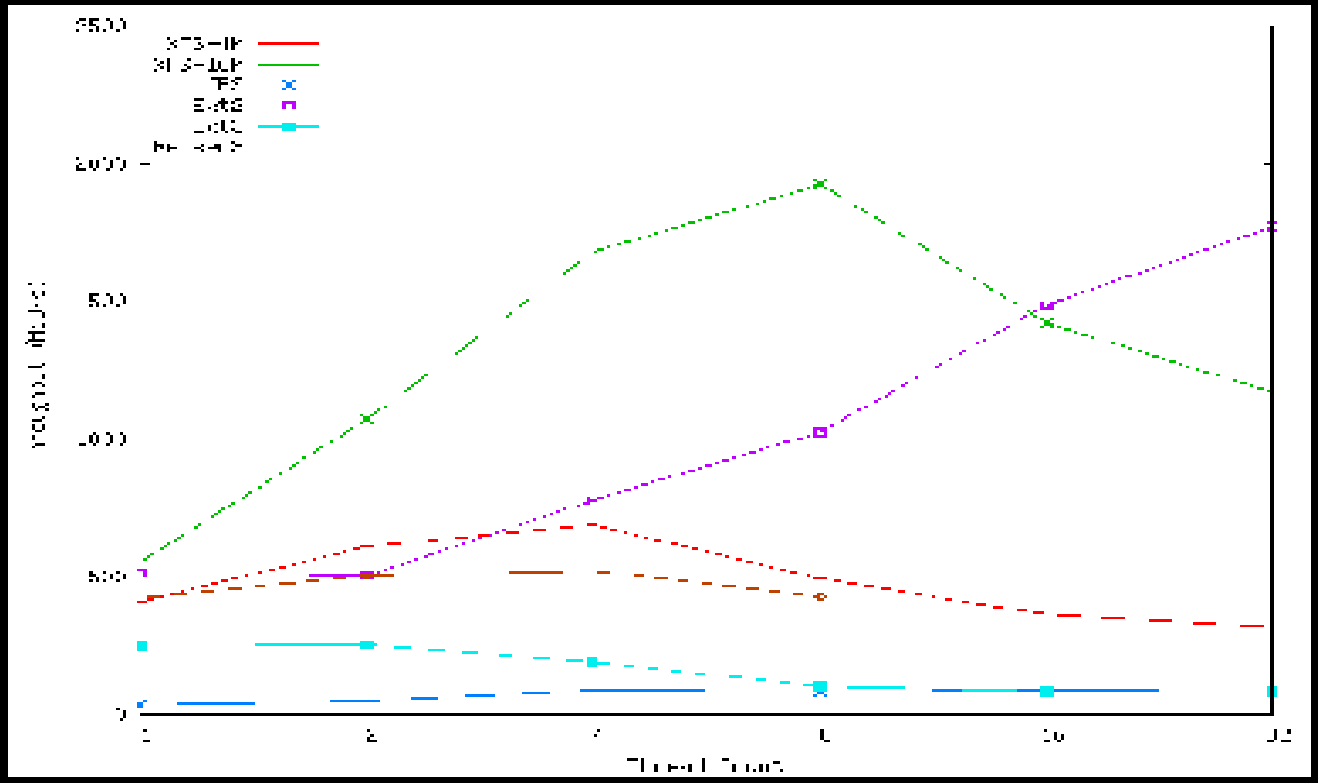
Results: Baseline XFS

- SuSE Linux Enterprise Server 9, Service Pack 2.
- Buffered I/O, 128KiB I/O size.



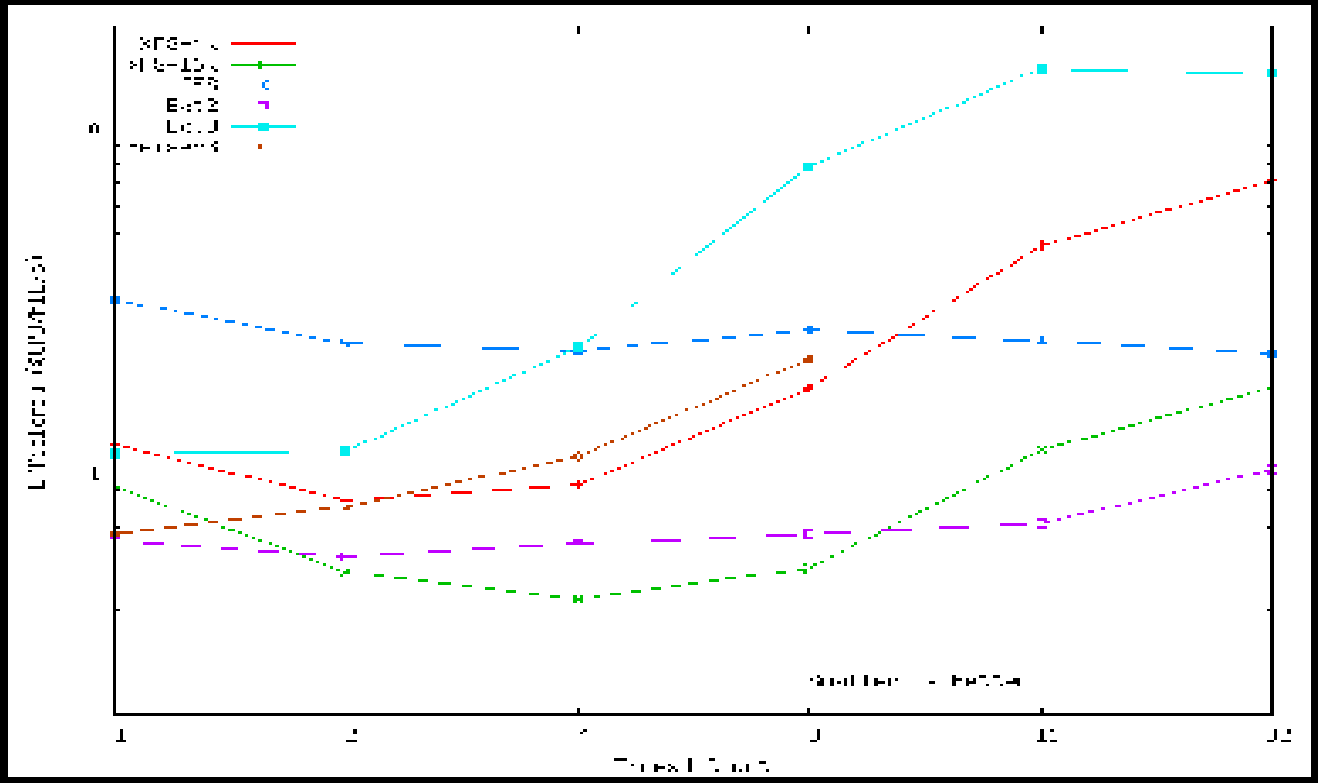
Results: 2.6.15-rc5 Write

- 2.6.15-rc5, sn2_defconfig kernel build, no debugging options, kdb on.
- Buffered write I/O, 256KiB I/O size.



Results: 2.6.15-rc5 Write

- 2.6.15-rc5, sn2_defconfig kernel build, no debugging options, kdb on.
- Buffered write I/O, 256KiB I/O size.

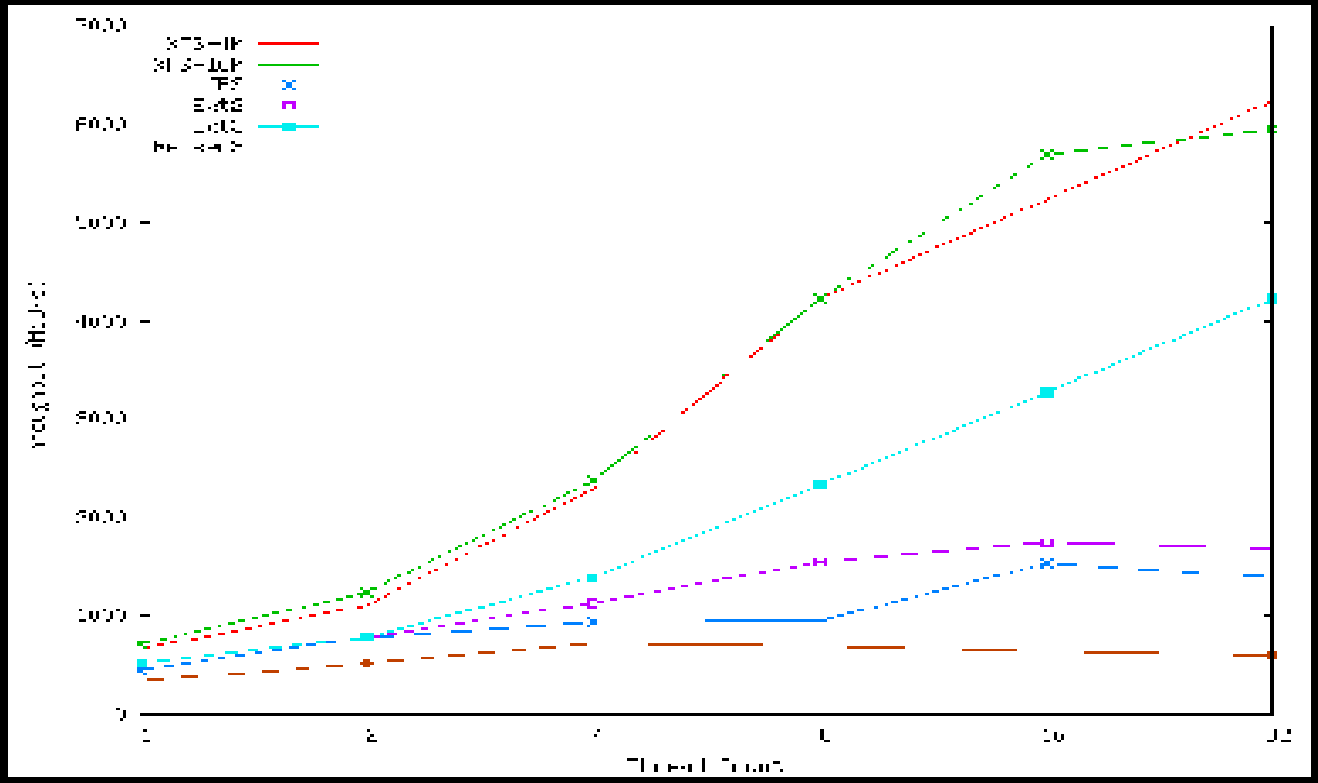


Buffered Write: Not so Good in 2.6.15-rc5

- XFS substantially slower in 2.6.15-rc6 than SLES9 SP2.
  - contention on the in-core superblock lock at >= 4 parallel write threads.
- Only EXT2 shows increasing throughput across tests, but even this was suboptimal:
  - At peak throughput the disks were sustaining >70,000 small write I/Os per second.
  - Disks were IOP saturated, not bandwidth saturated.
- JFS, ReiserFS and EXT3 do not scale to 2 parallel write threads due to lock contention.
- ReiserFS made extremely progress at greater than 8 parallel write threads
  - Tests were terminated at less than 10% completion after allowing several hours of additional runtime.

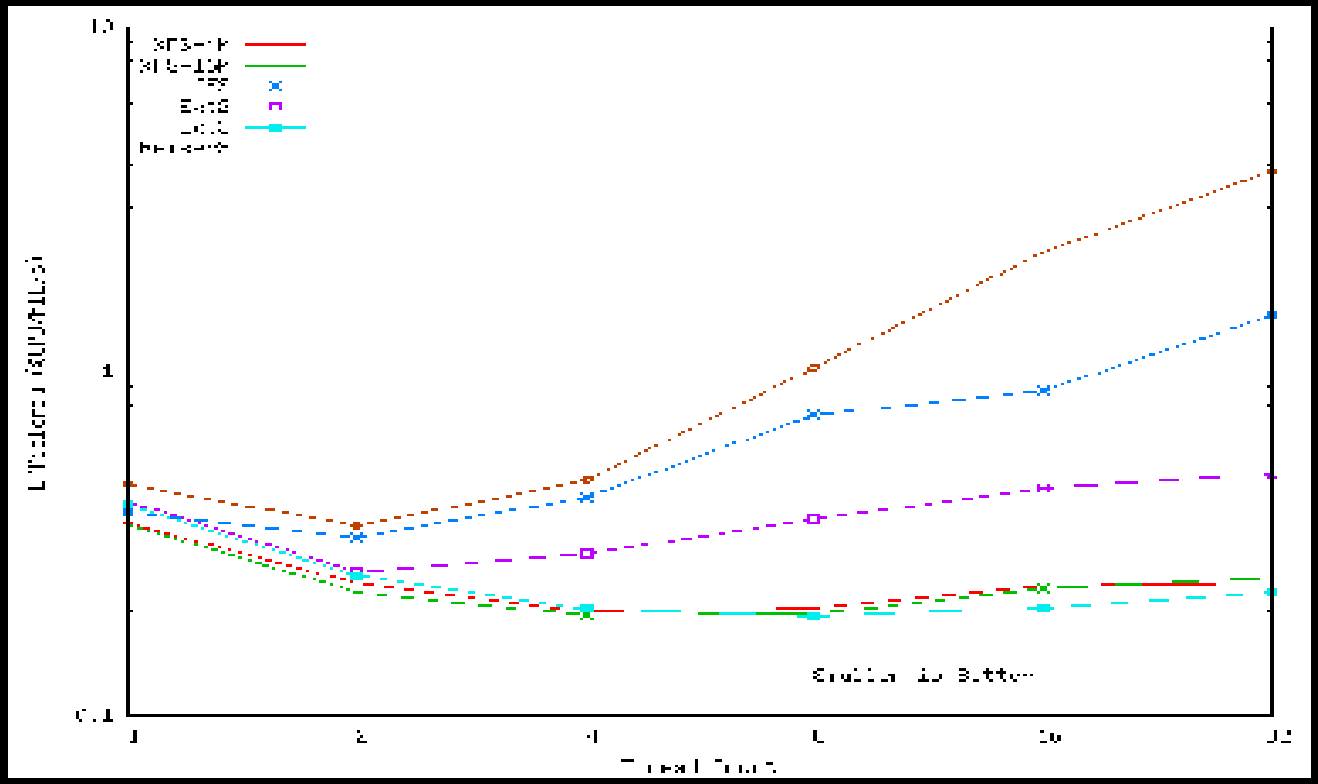
Results: 2.6.15-rc5 Read

- 2.6.15-rc5, sn2_defconfig kernel build, no debugging options, kdb on.
- Buffered read I/O, 256KiB I/O size.



Results: 2.6.15-rc5 Read

- 2.6.15-rc5 sn2_defconfig kernel build, no debugging options, kdb on.
- Buffered read I/O, 256KiB I/O size.

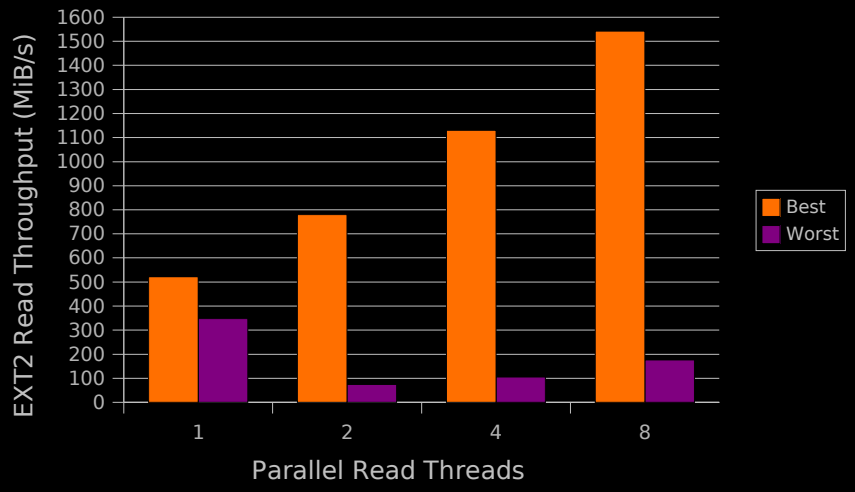


Buffered Read: Better, but still problems

- XFS was substantially faster than the SLES9 SP2 baseline:
  - Disks approaching 90% utilisation.
  - sustaining 13,000 I/Os per second to reach 6.3GiB/s.
- EXT3 has a similar throughput curve to XFS, but:
  - disks are IOP saturated.
  - sustaining > 60,000 I/Os per second to reach 4.5GiB/s.
- XFS and EXT3 have equally CPU efficient read paths.
- ReiserFS and JFS showing signs of lock contention limiting read throughput.
- EXT2 does not scale as well as expected from the buffered write results.

Issue #1 - Fragmentation

- Over multiple test runs, EXT2 and EXT3 had widely varied read throughput despite relatively consistent write throughput.
  - A result of fragmentation during the preparation stage.
  - Different fragmentation patterns each run.
  - Creates hot spots in the disk subsystem.
  - Increases disk seeks and decreases I/O sizes.
  - Run to run reproducibility is non-existent.
- EXT2 variation shows an order of magnitude between best and worst case results.
- This is on a fresh filesystem!



Issue #1 – Fragmentation

- Only XFS, JFS and ReiserFS consistently gave the same results over repeated test runs.
- Hard to draw any conclusions about file layout and fragmentation from the JFS and ReiserFS results:
  - Throughput limited by lock contention, not the disk subsystem.
- XFS rarely fragmented the files:
  - Typical results after parallel writes were 50-100GiB extents on disk guaranteeing large, sequential disk I/O.
  - Separate test runs often ended with identical allocation patterns.
  - Run to run throughput variation was within +/- 3%.

Issue #2 - Spinlocks

- All filesystems except EXT2 and JFS showed signs of spinlock contention in their write paths
  - JFS had sleeping lock contention in the write path.
- JFS and ReiserFS also showed spinlock contention in their read paths.
- Analysis of contention performed on XFS:
  - Fundamental problem is a `->prepare_write()` call for every filesystem block being written.
  - 175,000 calls/s at 700MiB/s for a 4KiB block size filesystem.
  - ~5.6us to obtain a spinlock and execute a function call, a memory read, two likely branches, a subtraction, a memory write and then drop the spinlock.
  - Does not scale past 4 concurrent write threads.

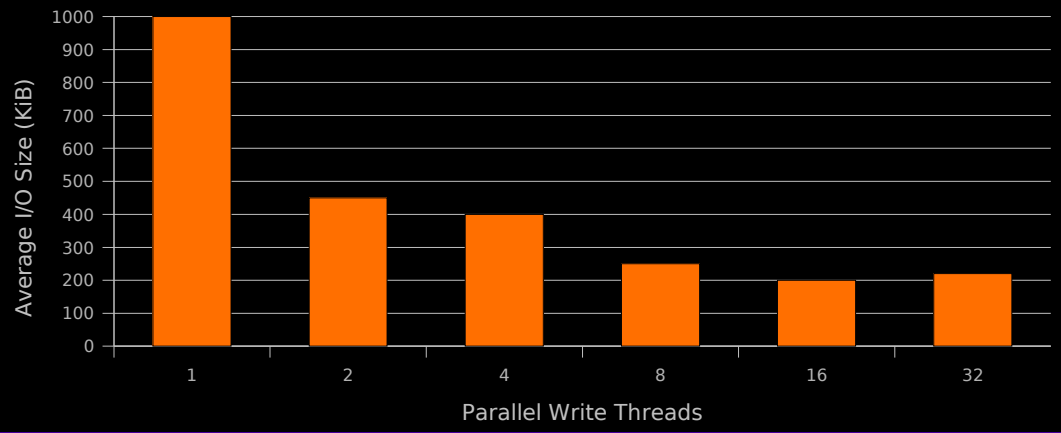
Issue #2 – Spinlocks

- Need to batch contiguous `->prepare_write()` calls to reduce the number of calls and lock traversals.
- Only a partial solution as contention will still occur given enough CPUs all writing at the same time.
- Each filesystem needs to be analysed separately to determine and fix the contention causing bottlenecks.
- For high end scalability, spinlocks on globally contended structures need to be avoided entirely.
  - Use sleeping locks if lock contention cannot be avoided.

Issue #3 – Memory Reclaim Behaviour

- Main problem is ensuring pdflush can keep up with the rate pages are being dirtied.

- With multiple write threads, we end up with kswapd doing all the writeback rather than pdflush:
  - Results in random offset writeback.
  - I/O sizes decrease due to less effective write clustering.
  - Delayed allocation cannot prevent fragmentation entirely.



Issue #3 – Memory Reclaim Behaviour

- pdflush running at 100% CPU with only 4 parallel write threads running.
- At 8 parallel write threads, pdflush almost never ran:
  - kswapd consuming 70-80% of a CPU per node.
  - On aggregate, the kswapd threads consume more CPU time than the writing processes.
- Making background write-back scale better would help prevent the kswapd write-back overload condition:
  - `wb_kupdate()` is single threaded so is limited in the amount of work it can do.
  - write-back contends for cache lines with concurrent writers and kswapd threads

Improvements

- Improvements were made to:
  - XFS
    - to solve lock contention problems.
    - to achieve large I/O sizes on small block size filesystems.
  - VM
    - To improve pdflush and memory reclaim interactions.
    - SN2 specific improvements.
  - Tuning
    - to improve memory locality and interconnect loading.

XFS Improvements #1

- Need to solve in-core superblock lock contention:
  - we need a distributed free space counter.
  - accuracy a requirement due to the counter being used to detect full filesystem conditions.
  - no global lock or cache line sharing in the fast path.
  - but still needs locking in the fast path to enable accurate summation of counters when required.

- The implementation uses:
  - per-CPU counters and locks.
  - a balancer that does accurate summation and distribution across all CPUs.
  - When the filesystem is full, it falls back to the existing slow method to ensure accuracy without requiring repeated, costly rebalancing.

XFS Improvements #2

- XFS's buffered write I/O size has been limited to the number of filesystem blocks it can put in a single `bio` due to it's use of `submit_bh()`:
  - 512KiB maximum I/O size on 4KiB block size filesystems.
  - 2MiB maximum I/O size on 16KiB block size filesystems.
  - due to using a `bio` vector per `bufferhead`.
- Christoph Hellwig and Nathan Scott converted XFS to build it's own `bio` vectors and use `submit_bio()`.
- Maximum I/O size XFS can issue is now determined by:
  - Maximum number of vectors in a `bio`.
  - Page size.
  - Underlying hardware limits.

VM Improvements #1

- A kernel upgrade from 2.6.14 to 2.6.15-rc5 showed a dramatic improvement in buffered read speeds from the same filesystem configuration:
  - 2.6.14 sustained approximately 4GiB/s (same as baseline results).
  - 2.6.15-rc5 sustained 6.3GiB/s.
- Dean Roe had, independently, been working on improving TLB flushing on the SN2 platform:
  - Uses hardware based global TLB flush calls rather than generic global flush.
- Global TLB flush speed has a major impact on memory reclaim speed.
- Buffered read I/O scalability appears to be limited by memory reclaim speed and scalability.

VM Improvements #2

- Christoph Lameter's node local memory reclaim was the main area of interest:
  - reduces cross-node cache line traffic during reclaim.
  - prefers to reclaim clean pages on a specified node before trying to allocate from a different node.

- Major differences in behaviour:
  - kswapd never ran:
    - All memory reclaim occurred directly in the allocation path from clean pages.
  - pdflush throughput increased by an order of magnitude:
    - Now sustains >5GiB/s write-back using less than half a CPU.
  - CPU usage was greatly reduced at low thread counts as there were no kswapd processes running.
  - Buffered reads were now clearly I/O bound.

Tuning Improvements

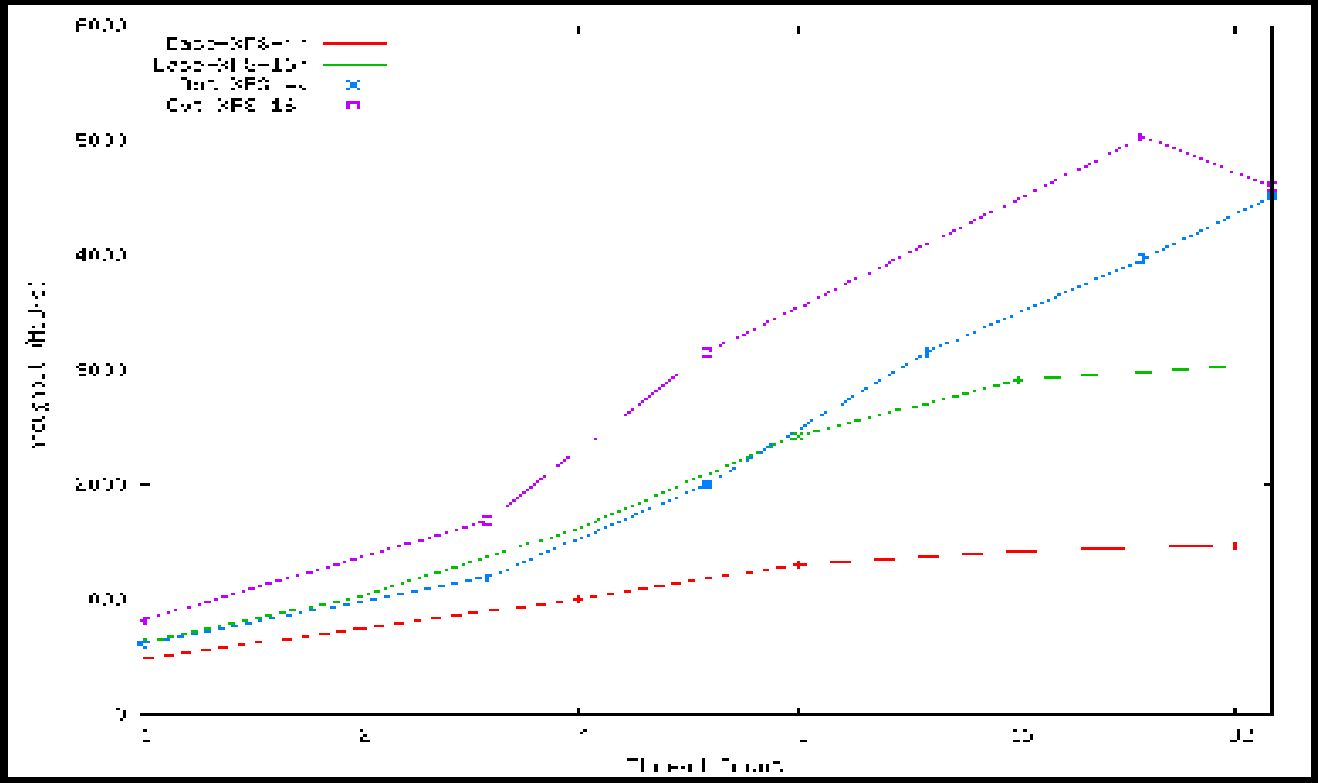
- Memory distribution was the main problem:
  - I/O hardware was spread evenly throughout the interconnect.
  - but page cache or direct I/O buffers were not.
  - unbalanced interconnect traffic.
  - both hot and idle nodes in the machine.

- '`numactl -i all ......`':
  - spreads page cache and direct I/O buffers evenly across all nodes.
  - balanced interconnect traffic.
  - improved throughput by up to 75%.

- Block device read ahead needed increasing to keep all disks in the stripe busy on buffered read I/O.

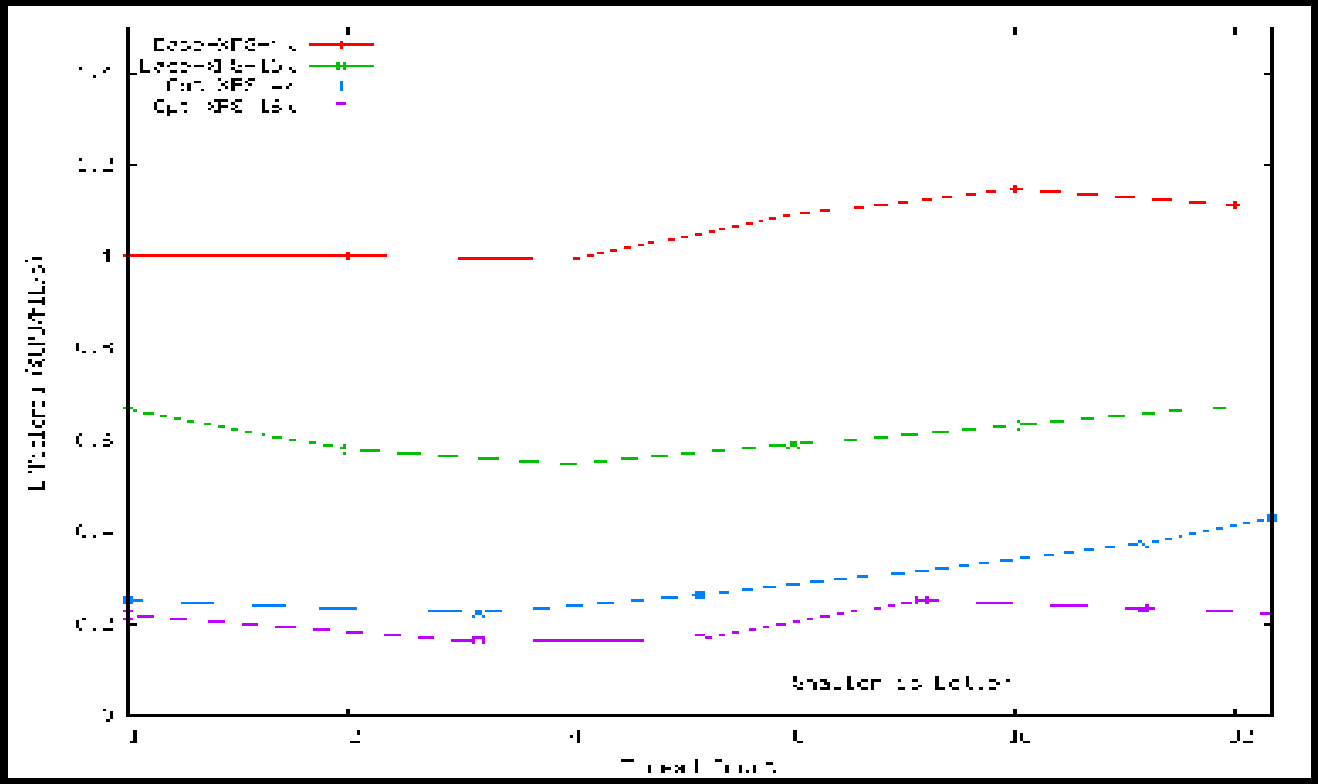
Results: Improved XFS, Buffered Writes

- 2.6.15-rc5 plus all listed improvements
- Baseline results from SLES9 SP2
- Buffered write I/O, 256KiB I/O size.



Results: Improved XFS, Buffered Writes

- 2.6.15-rc5 plus all listed improvements
- Baseline results from SLES9 SP2
- Buffered write I/O, 256KiB I/O size.

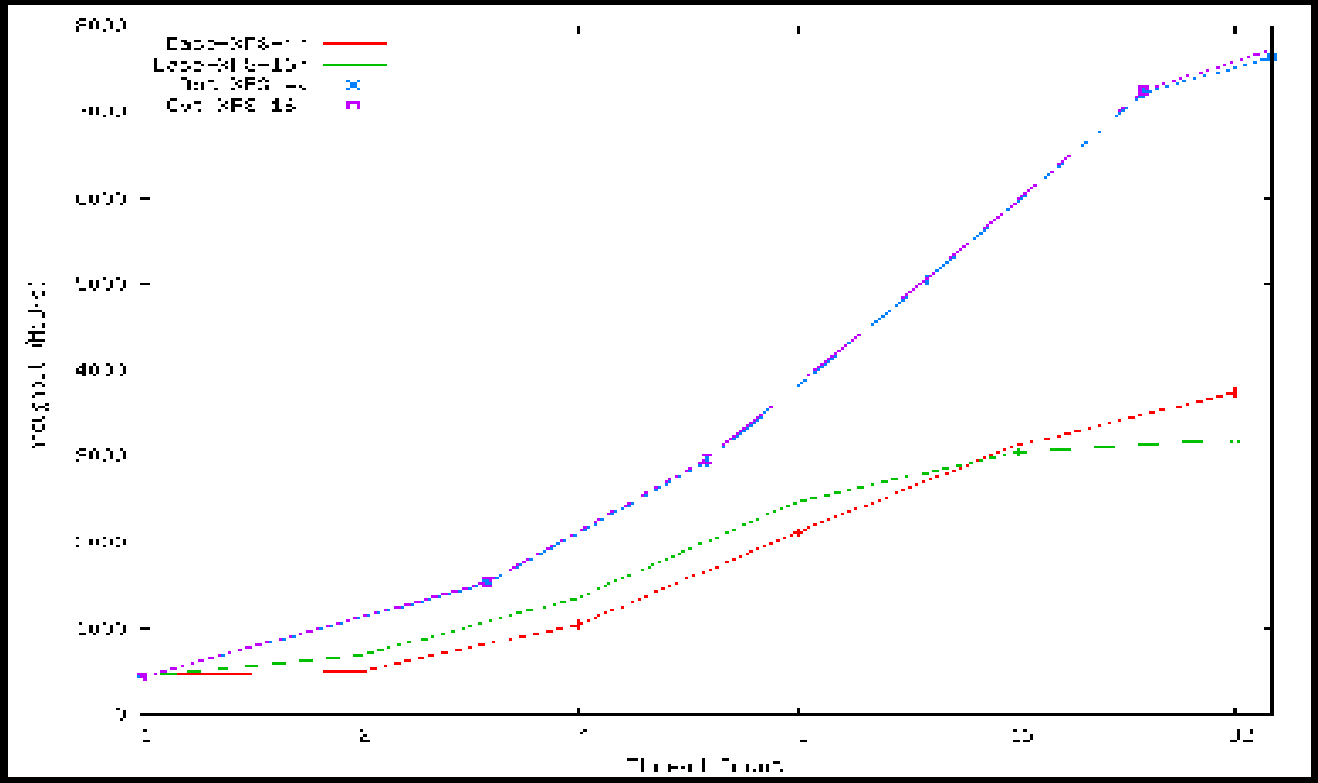


Results: Buffered Writes

- No adverse memory reclaim behaviour.
- No decrease in I/O sizes as load increases.
- pdflush no longer stressed by writeback.
- XFS throughput scales independently of filesystem block size.
- No apparent contention points at > 5GiB/s write throughput.
- Substantial reductions in CPU usage.

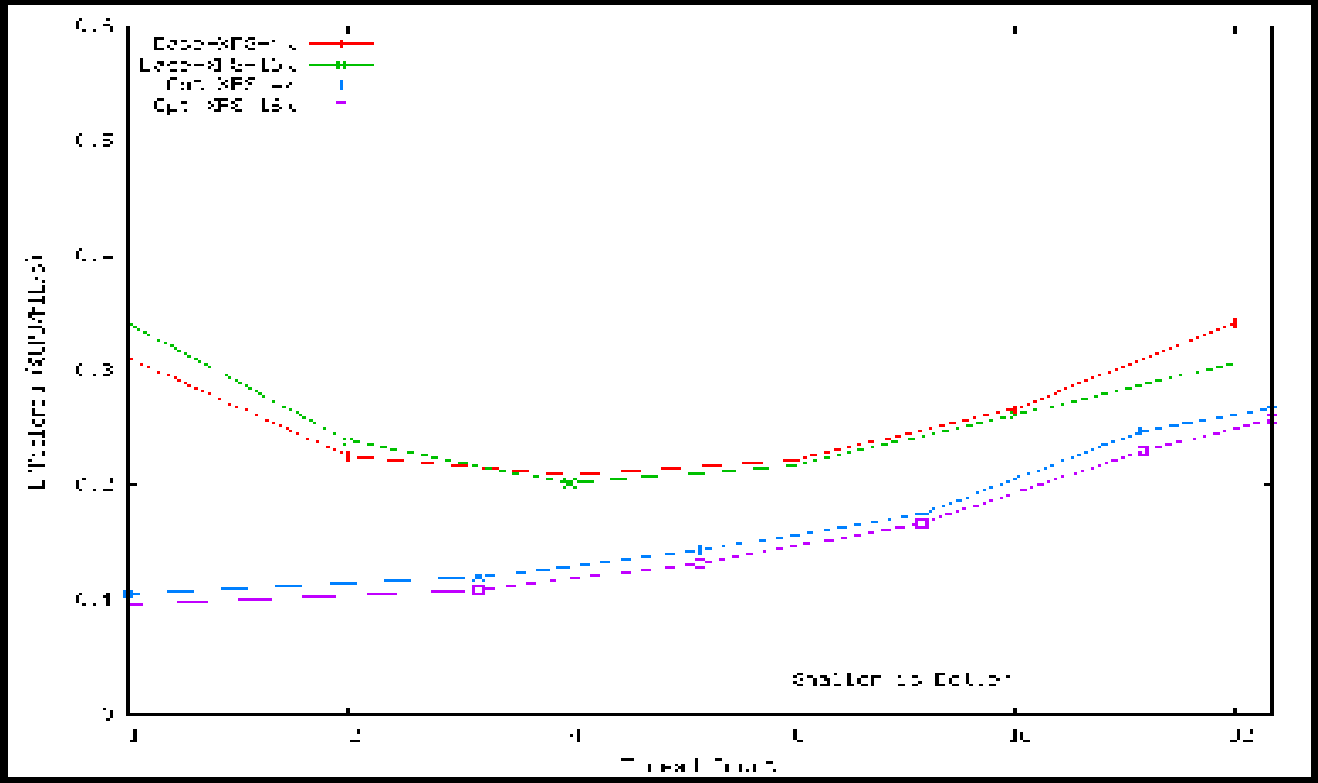
Results: Improved XFS, Buffered Reads

- 2.6.15-rc5 plus all listed improvements
- Baseline results from SLES9 SP2
- Buffered read I/O, 256KiB I/O size.



Results: Improved XFS, Buffered Reads

- 2.6.15-rc5 plus all listed improvements
- Baseline results from SLES9 SP2
- Buffered read I/O, 256KiB I/O size.



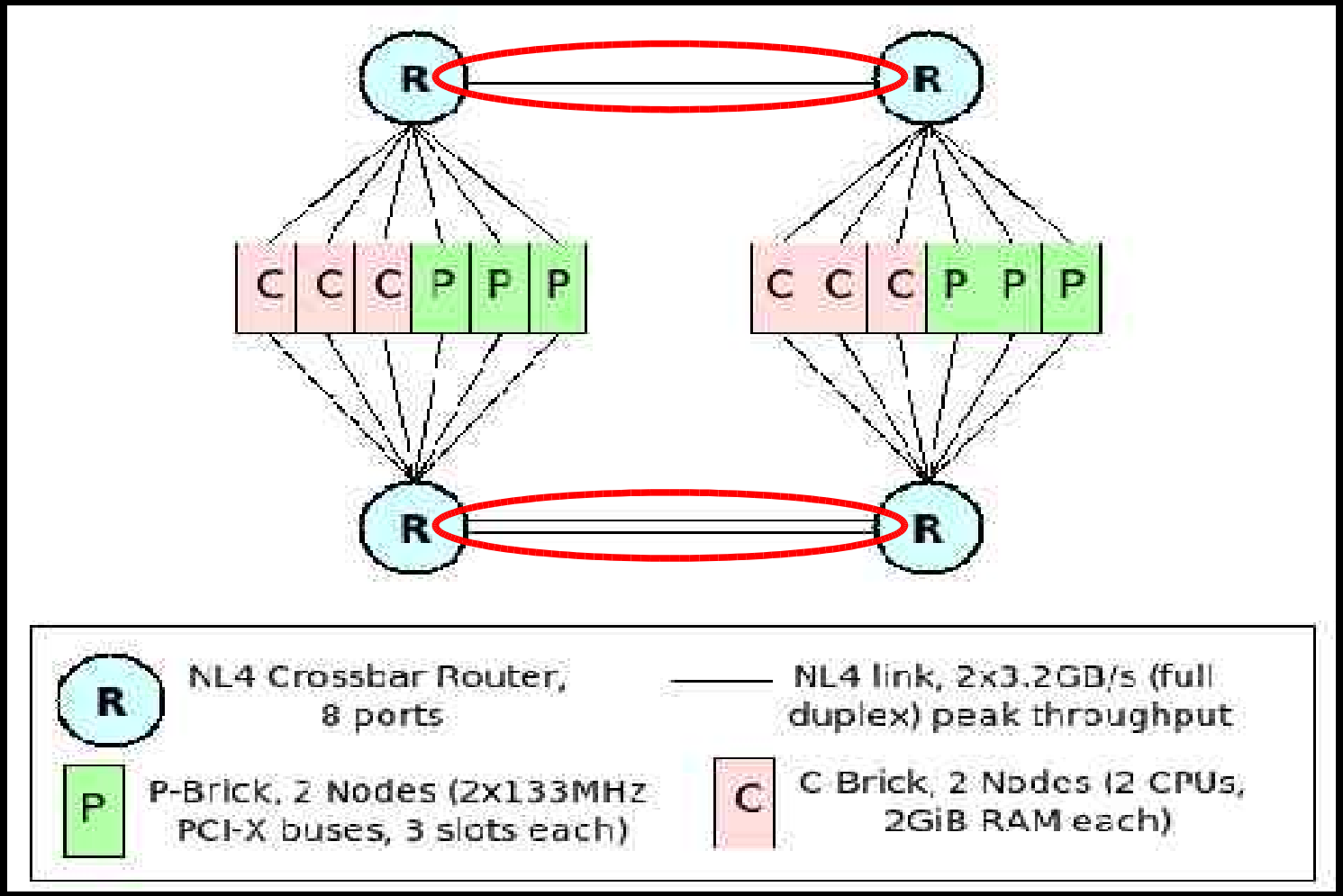
Results: Buffered Read

- 7.6GiB/s sustained read throughput.
- No apparent memory reclaim issues.
- XFS scales independently of filesystem block size.
- No apparent scalability bottlenecks, but efficiency is reducing as the number of concurrent readers and throughput increases.
  - indicates some limit is being approached.

What is limiting throughput?

- Buffered read I/O sustained 7.6GiB/s
  - Disk subsystem theoretically capable of > 11GiB/s.
- Memory access patterns on read is:
  - FC HBA -> memory -> CPU -> memory.
  - 11/12ths of the memory operations are remote due to memory placement.
  - ~21GiB/s of bi-sectional interconnect bandwidth utilised at this throughput rate.
- Interconnect saturation!
- Need a bigger machine with greater bi-sectional interconnect bandwidth to achieve higher throughput from buffered I/O.

Interconnect Saturation

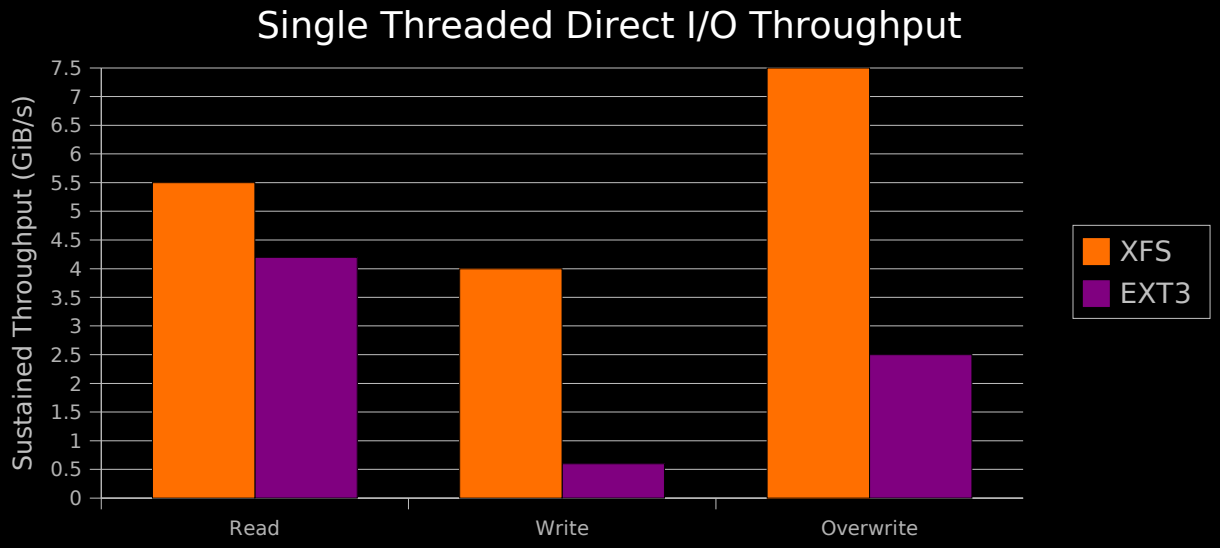


sgi

Direct I/O?

- With Direct I/O, we can avoid two of the three memory operations involved with buffered I/O if we don't touch the data buffers with the CPU
  - Should be able to achieve maximal throughput from disk subsystem

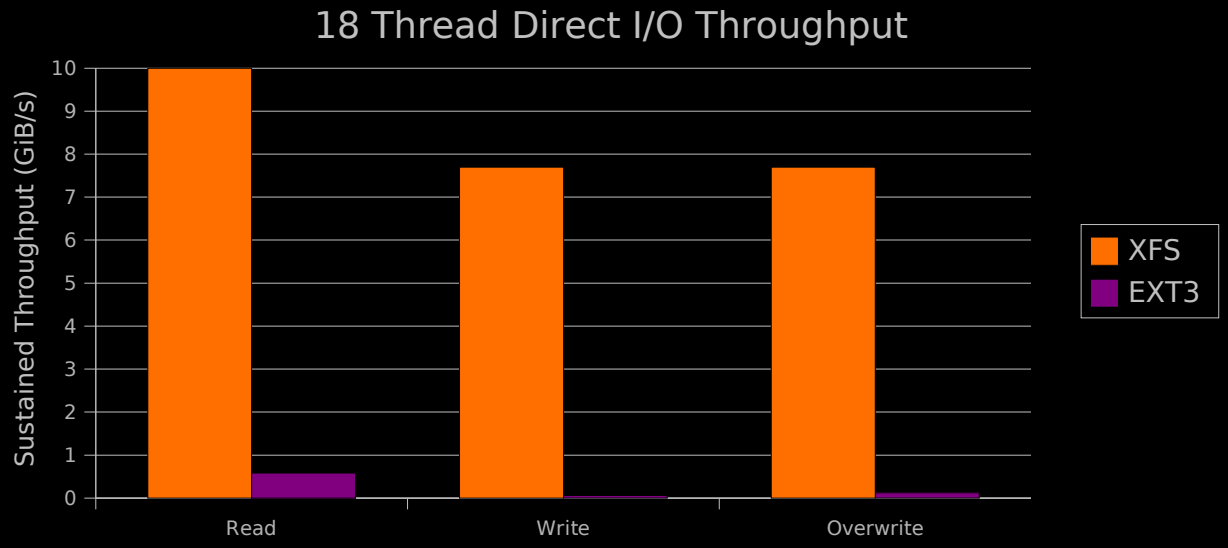
- 2.6.15-rc5, 512MiB I/O size, memory interleaved across all nodes.



Direct I/O?

- Multiple read threads on XFS results in sustained throughput of 10GiB/s and peak at over 10.7GiB/s of throughput.

- EXT3 suffers from severe fragmentation under parallel direct I/O write threads and only achieves 60MiB/s write speed.



Futures

- SAS, PCI-e, multi-core CPUs and NUMA all moving rapidly into mainstream:
  - issues seen in this investigation will become mainstream problems in the next 1-2 years.
  - need to ensure Linux operates effectively on systems of this size sooner rather than later.

- Simple jobs scale well but we are already seeing interactions with high bandwidth I/O and independent cpuset constrained jobs.
  - Indicates the need for filesystems to become NUMA and I/O path topology aware.
  - Filesystem placement and hence I/O bandwidth locality needs to be more finely controlled.

Futures

- Need to minimise disk seeks at all costs:
  - Disk sizes increasing faster than seek rates.
  - Smarter allocation algorithms are the key to achieving this goal.
  - Problems seen (and solved) at the high end are now becoming mainstream issues.
- Intrinsic parallelism of the average computer is increasing, so effective parallel allocation algorithms are becoming a necessity in Linux filesystems.

Conclusion

- Achieved throughput close to physical limits:
  - of the disk subsystem with direct I/O on XFS.
  - of the NUMALink interconnect with buffered I/O on XFS.
- Uncovered a set of generic filesystem scalability issues that affect every filesystem we tested.
- Solved the scalability problems we encountered in XFS and the VM running buffered I/O.
- Proved that XFS is the best choice for our customers: both on the machines they use and the common workloads they run.

Questions?