Filesystem Performance on FreeBSD

Kris Kennaway
kris@FreeBSD.org

BSDCan 2006, Ottawa, May 12

Introduction

- Filesystem performance has many aspects
- No single metric for quantifying it
- I will focus on aspects that are relevant for my workloads (concurrent package building)
- The main relevant filesystem workloads seem to be
  - Concurrent tarball extraction
  - Recursive filesystem traversals
- Aim: determine relative performance of FreeBSD 4.x, 5.x and 6.x on these workloads
  - Overall performance and SMP scaling
  - Evaluate results of multi-year kernel locking strategy as it relates to these workloads

Outline

- SMP architectural differences between 4/5/6.x
- Test methodology
- Hardware used
- Parallel tarball extraction test
  - Disk array and memory disk
- Scaling beyond 4 CPUs
- Recursive filesystem traversal test
- Conclusions & future work

SMP Architectural Overview

- FreeBSD 4.x; rudimentary SMP support
  - Giant kernel lock restricts kernel access to one process at a time
  - SPL model; interrupts may still be processed in parallel
- FreeBSD 5.x; aim towards greater scalability
  - Giant-locked to begin with; then finer-grained locking pushdown
    - FreeBSD 5.3; VM Giant-free
    - FreeBSD 5.4; network stack Giant-free (mostly)
    - Many other subsystems/drivers also locked
  - Interrupts as kernel threads; compete for common locks (if any) with everything else
- FreeBSD 6.x;
  - Consolidation; further pushdown; payoff!
  - VFS subsystem, UFS filesystem Giant-free

FreeBSD versions

- FreeBSD 4.11-STABLE (11/2005)
  - Needed for amr driver fixes after 4.11-RELEASE
- FreeBSD 5.4-STABLE (11/05)
  - No patches needed
- FreeBSD 6.0-STABLE (11/05)
  - patches:
    - Locking reworked in amr driver by Scott Long for better performance
    - All relevant changes merged into FreeBSD 6.1
  - A kernel panic was encountered at very high I/O loads
    - Also fixed in 6.1

Test aims and Methodology

- Want to measure
  - overall performance difference between FreeBSD branches under varying (concurrent process I/O) loads
  - scaling to multiple CPUs
- Avoid saturating hardware resources (e.g. disk bandwidth) with single worker process
    - Or there is no SMP performance gain to be measured
    - Easy to saturate a single disk; need **array** (software/hardware) to provide excess capacity
    - Other resources: CPU, memory bandwidth
- Measure both on real disk hardware and using memory disk
    - MD is useful in its own right for certain applications (mine)
    - Also a model of very high I/O rates; eye on the future

Test Methodology (2)

- UFS1 vs UFS2 (5.x and above)
  - UFS2 writes ~10% more data to disk for same FS workload, and is also ~5% faster **if the disk is not saturated;** but greater I/O rate saturates disk 10% sooner.
  - UFS1 used to avoid premature saturation, compare to 4.11
- Various UFS mount modes
  - Sync (slowest)
    - All writes synchronous
  - Noasync
    - Data asynchronous, metadata synchronous
  - Soft Updates
    - Dependency ordering of writes to ensure on-disk consistency
      - Pointless for md, but models high I/O rates on a very fast disk array
    - Is expected to perform between noasync and async speeds
  - Async (fastest)
    - All writes asynchronous; may corrupt on power failure

Test Hardware

- 2 CPU i386 2.8GHz Nocona Xeon
  - 4 GB RAM
  - LSI MegaRAID 320-2x RAID0 controller (amr driver)
  - 4 Ultra160 SCSI drives
  - FreeBSD 4.11/5.4/6.0
- 4 CPU amd64 Opteron 846 (2GHz)
  - 16 GB RAM
  - Used as memory disk; models high I/O rates
  - FreeBSD 5.4/6.0 only (no 4.x for amd64)
- 14-CPU sparc64 E4500
  - 16GB RAM (slow!)
  - 400MHz CPUs (slow!)
  - FreeBSD 6.0 only
  - Probe scaling of UFS beyond 4 CPUs
    - Severely limited by CPU and memory bandwidth

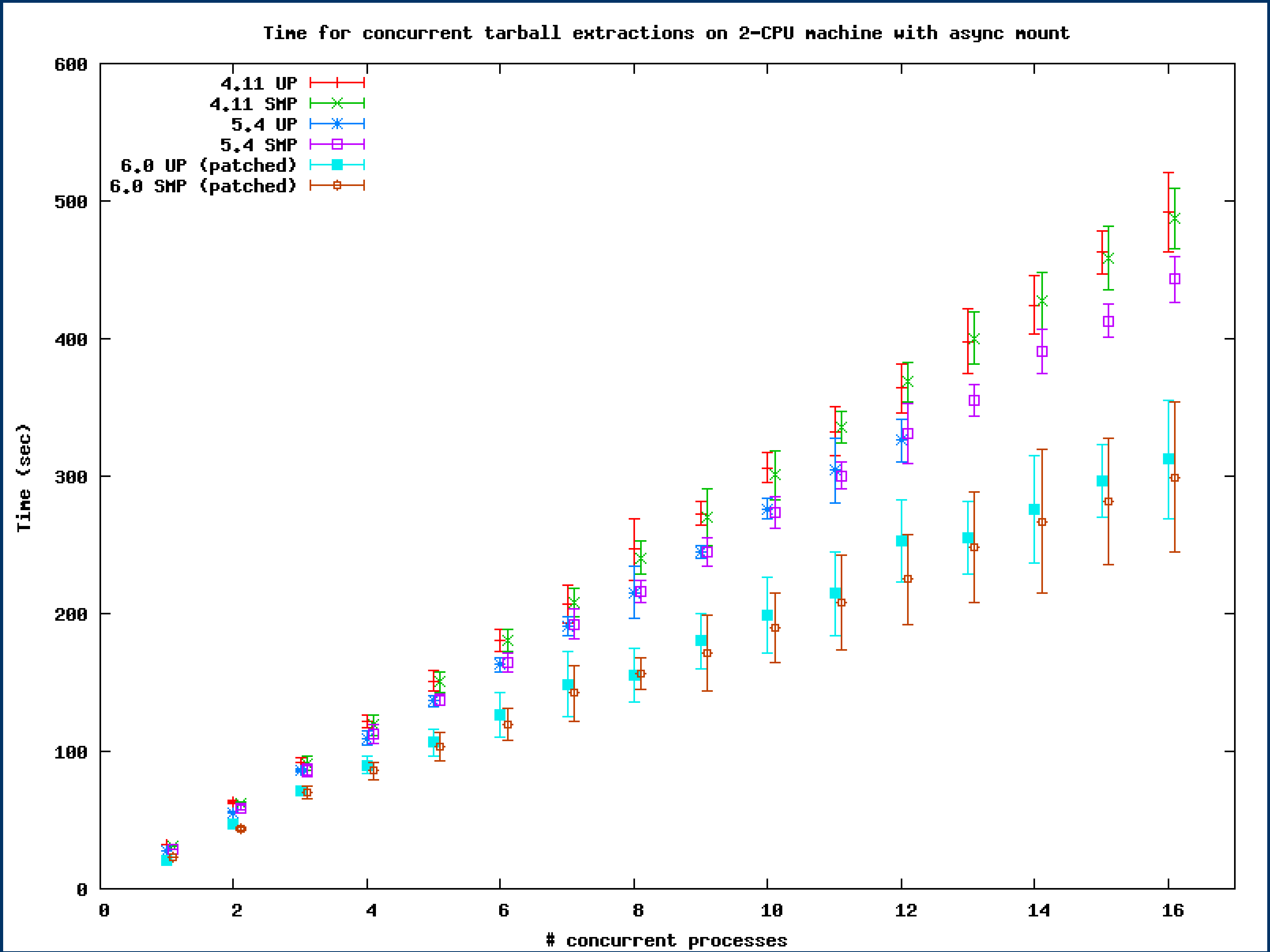
Tarball extraction test

- BSDtar is less efficient (30% slower) than GNUtar, and it has odd I/O patterns during extraction (write, then read)
- Use GNUtar with uncompressed tarball of ports tree (270 MB tarball, 93862 files, 4 levels deep)
- Extract to disk, and to swap-backed md to avoid disk overhead (higher I/O rates)
  - Swap backing is faster than malloc backing and doesn't use swap unless there is memory pressure
- Run multiple extractions to different subdirectories of filesystem root; destroy and recreate filesystem/backing store between tests

Tarball extraction (2)

- Measurements:
  - Time for completion of each process (real/sys/user)
  - Total I/O written to backing device (iostat -i)
- Care needed; all processes not scheduled equally! e.g. on 4 CPU system 4 gtar processes will perform I/O to the near-exclusion of all others, until they complete; then another 4, etc.
  - 4.11 also, but not as marked as on 5.x/6.x
  - ULE scheduler is fairer about I/O scheduling, but performs worse except under minor load
- Use average runtime instead

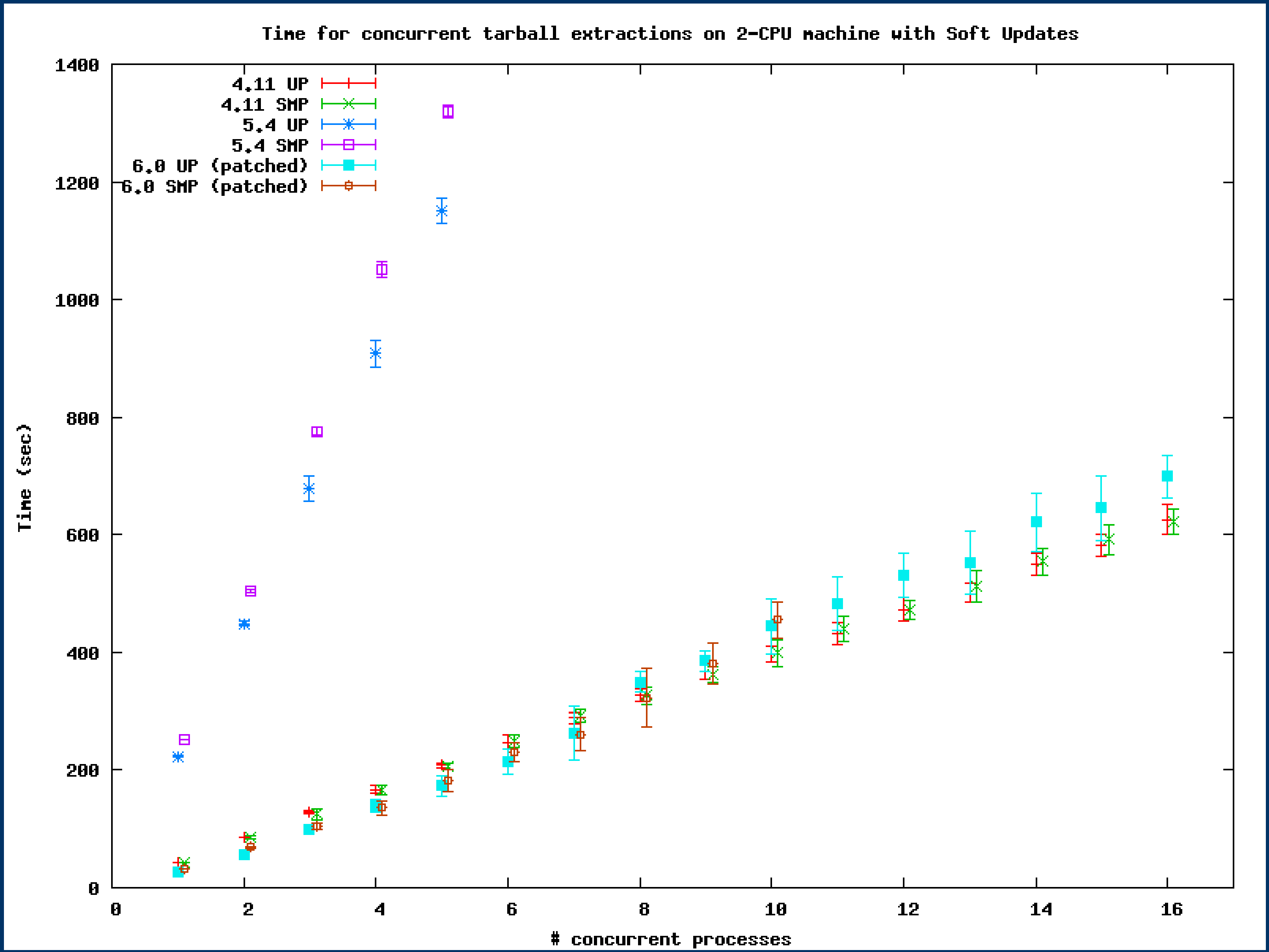
Async mounts (2-CPUs, amr)



Async Analysis

- FreeBSD 6.0 has 30% better overall performance compared to 4.11 on this hardware
- Small (5-8%) net performance benefit from second CPU on 6.0
  - still moderate contention within driver; hardware does not seem to easily accommodate concurrent I/O.
- sync/noasync results (not shown) are comparable.

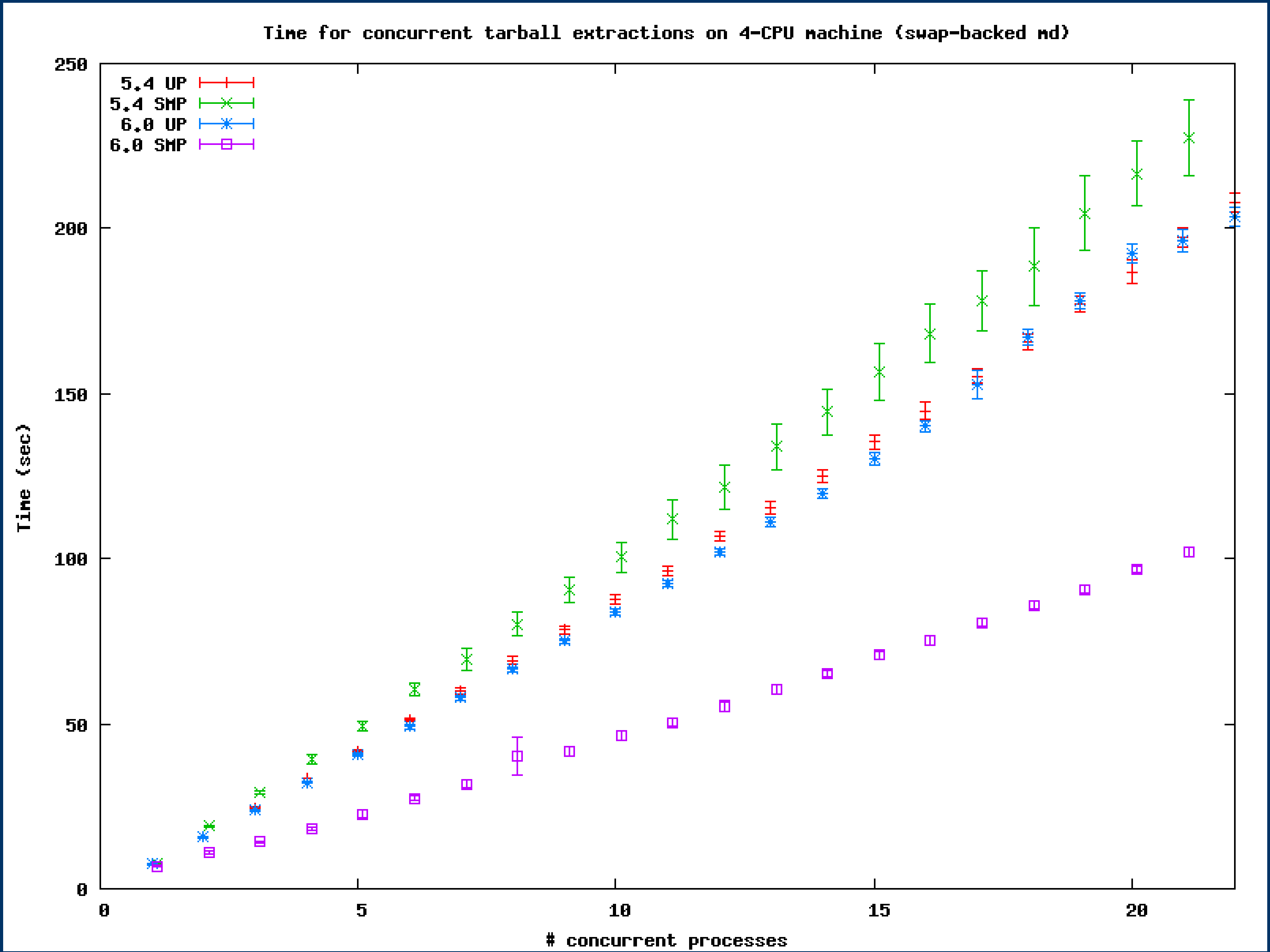
Soft Updates (2-CPU, amr)



Soft Updates Analysis

- FreeBSD 5.4 is 7 times slower than 6.0!
  - Mutex profiling shows that this is due to massive contention for the Giant lock between multiple writing processes and bufdaemon;
  - **Much worse** than serialized!
- Soft Updates on FreeBSD 6.0 is slightly slower than **sync** mounts at moderate load (~8 writers)!
  - Seems to be due to poor scaling of bufdaemon; later

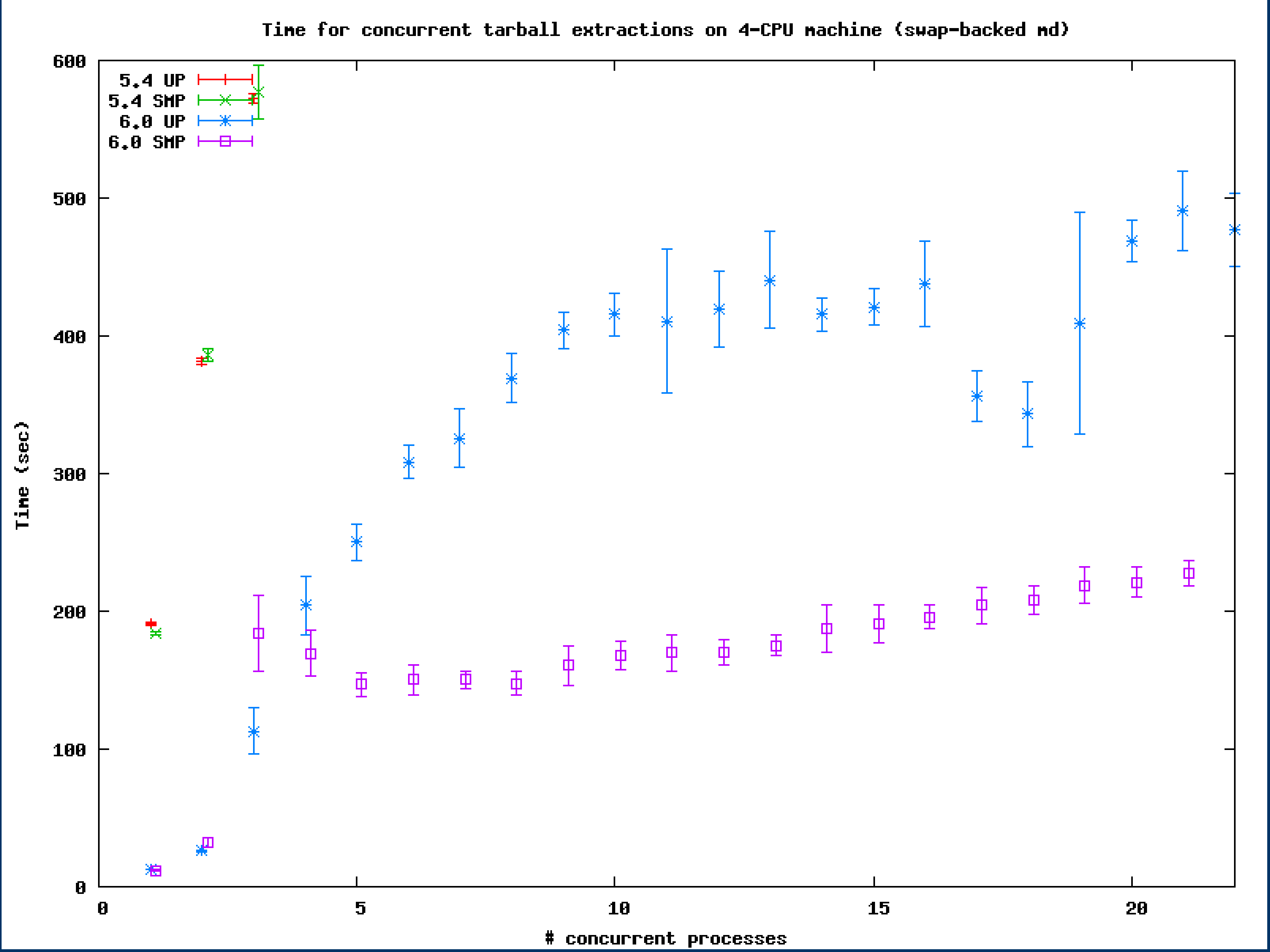
Async mount (4-CPU, md)



Analysis

- 5.4
  - is only slightly slower than 6.0 on UP, but...
  - ...performs **worse** on SMP than UP due to all the Giant contention
- 6.0
  - Takes 6.9 seconds to extract 93000 files to md!
  - SMP is ~1.9 times faster than UP on 6.0
  - This compares reasonably well to the 4-CPUs in the machine since the md worker thread, bufdaemon, g_up and g_down worker threads together consume nearly 150% of a CPU.
    - Still some overhead lost to locking contention

Soft Updates (4-cpu, md)



Bufdaemon (non-)scalability

- md allows higher I/O rates; even more work is pushed onto bufdaemon, which easily saturates 100% of a CPU.
- This is a major bottleneck for high-end I/O performance with soft updates, and it is already manifested on real disk hardware.
- In 6.0 bufdaemon is Giant-locked, but mutex profiling shows this is not a factor in this workload
  - In 7.0 bufdaemon is Giant-free
- Other mount modes scale well, although bufdaemon is still (less of) a factor.

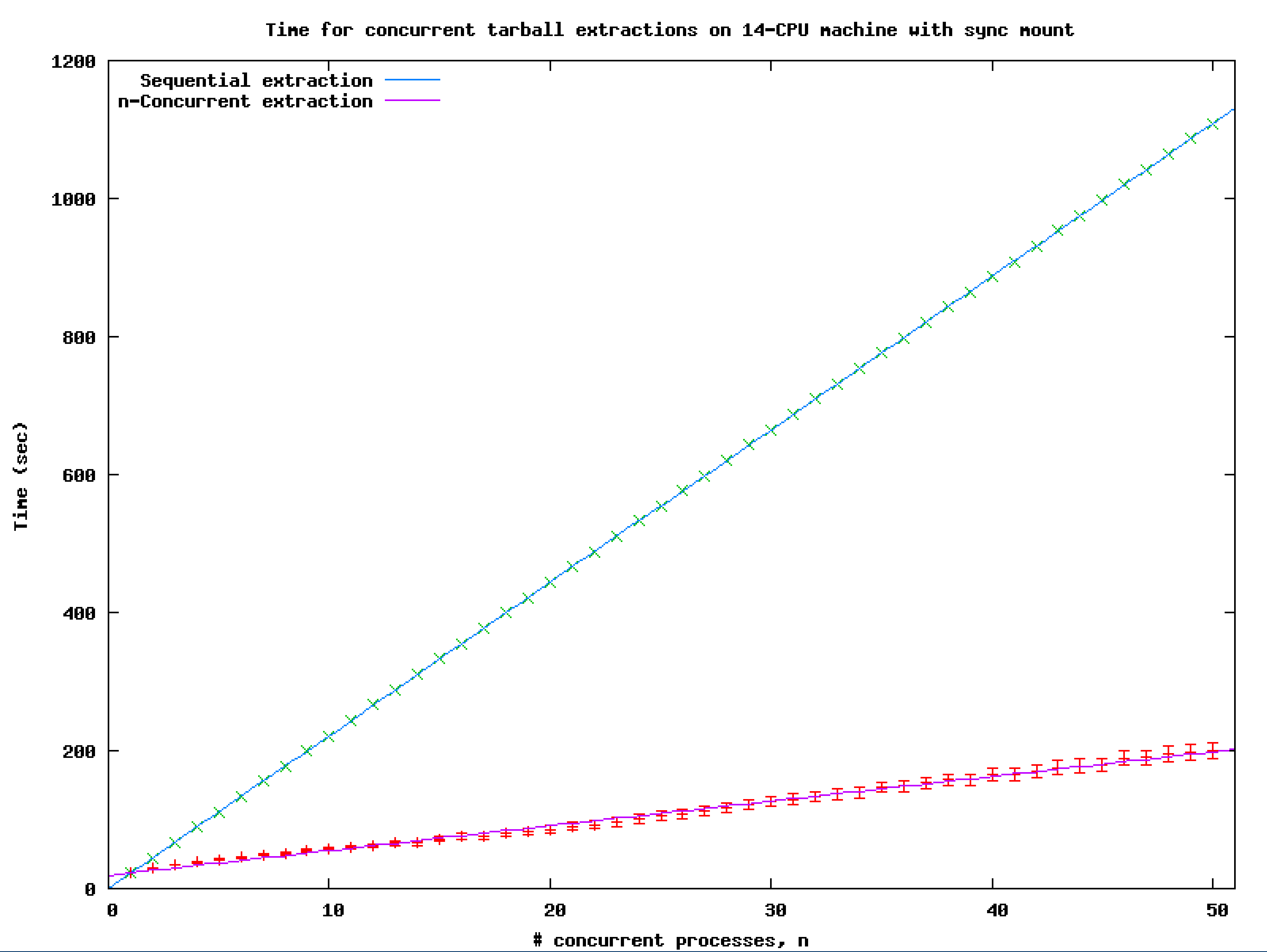
Scaling beyond 4 CPUs

- How far does FreeBSD 6.0's UFS scale?
- Use 14 CPU E4500
  - Severe drawbacks
    - slow CPUs (easily saturated by a single kernel thread)
    - Low memory bandwidth (md worker thread saturates CPU doing bcopy())
  - Use gstripe with 5 md devices to spread backing store load over 5 CPUs; but then:

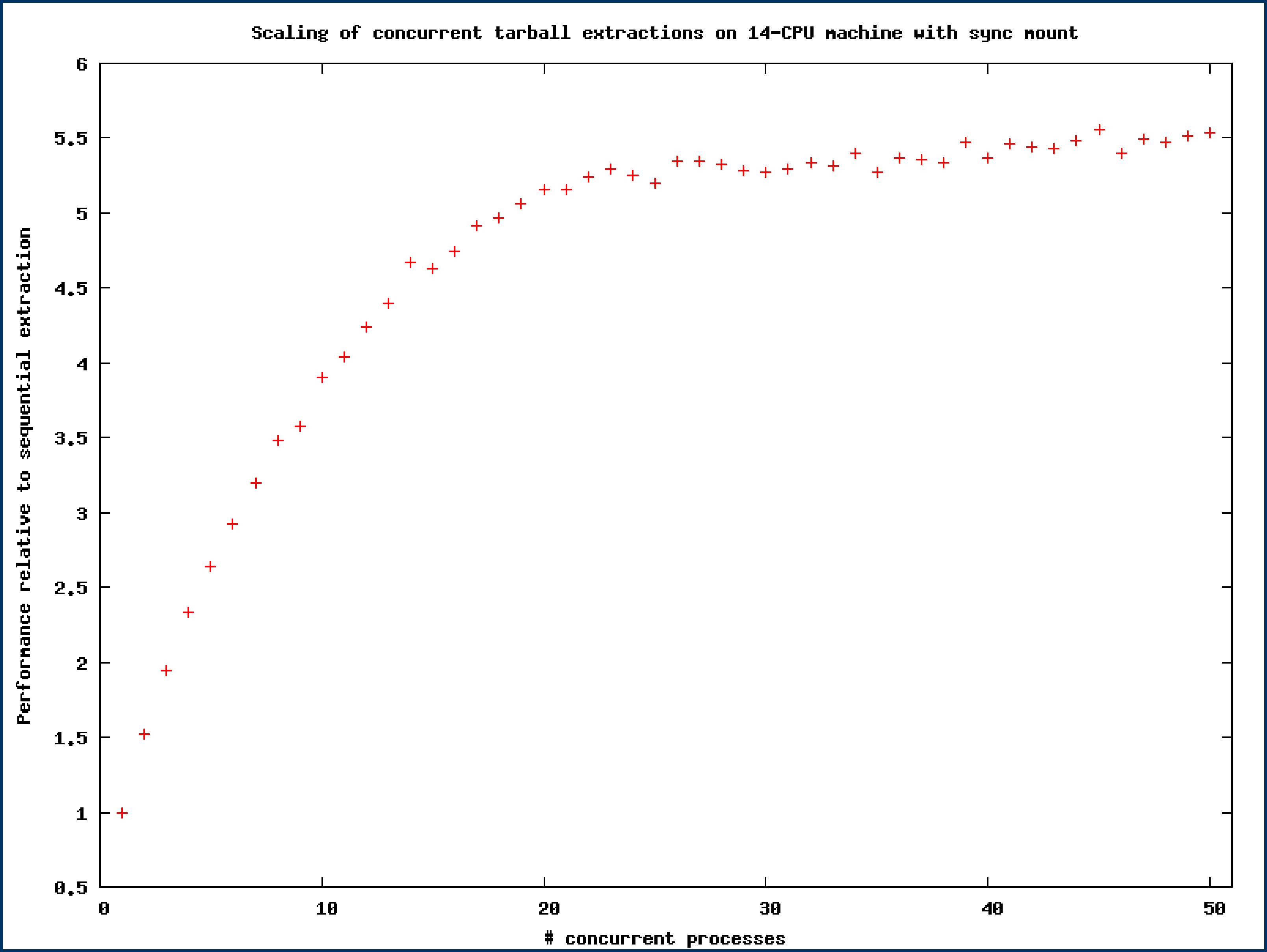
```
  PID USERNAME  THR PRI NICE   SIZE    RES STATE  C   TIME   WCPU COMMAND
    3 root        1  -8    0     0K    32K CPU8   8 662:56  95.41% g_up
10040 root        1  -8    0     0K    32K CPU6   1   0:53  66.19% md4
10038 root        1  -8    0     0K    32K CPU4   b   0:53  65.01% md3
10036 root        1  -8    0     0K    32K CPU12  4   0:52  63.44% md2
10034 root        1  -8    0     0K    32K CPU5   7   0:53  62.50% md1
10032 root        1  -8    0     0K    32K CPU1   3   0:53  62.16% md0
    4 root        1  -8    0     0K    32K -      c 514:51  56.74% g_down
```

  - Nevertheless, achieve impressive scaling with multiple concurrent extractions:

Scaling on 14 CPUs



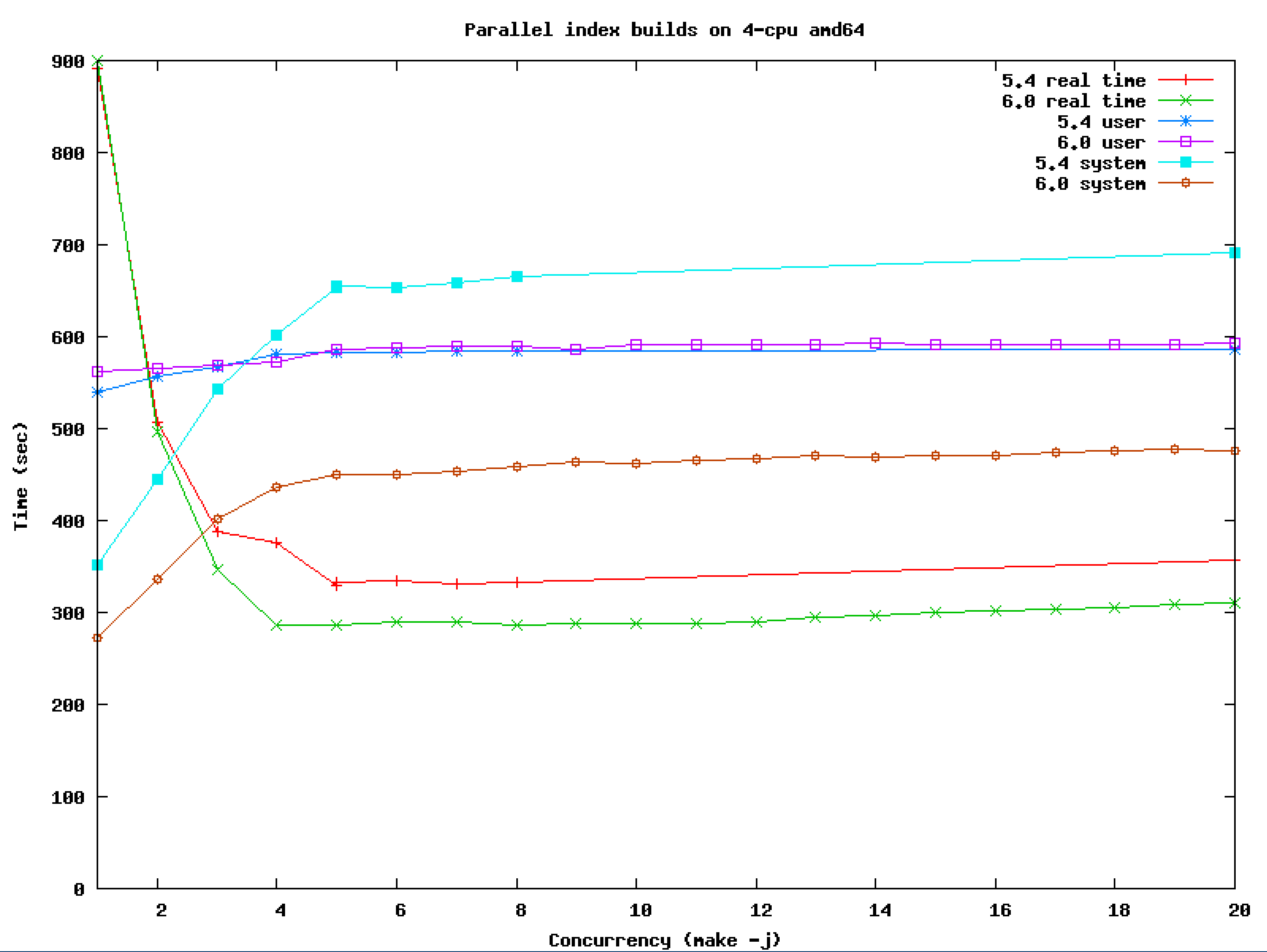
Concurrent/serialized ratio



Parallel filesystem recursion

- Benchmark: ports collection INDEX builds
  - Parallel, recursive traversal of ~15000 directories and reads of comparable number of files
  - Also forks 10000's of processes, so not a “pure” test; but interesting results
- 4-CPU amd64 system;
  - Therefore could only test 5.4 vs 6.0
- /usr/ports pre-cached by throwing away first run.

Parallel recursion on 4 CPUs



Status of the Giant lock

- 6.0 is ~15% faster than 5.4 on this test
- Profiling shows this is mostly due to contention on the Giant lock in 5.4
  - Under FreeBSD 5.4 Giant was acquired in 43366070/1181505200 = 3.6% of mutex acquisitions
  - Furthermore, 47% of these operations contended for the Giant lock (failed to acquire on first try).
  - On FreeBSD 6.0 Giant was only acquired in 782961/780550666 = 0.100% of all mutex ops
  - 36 times lower than on 5.4!
  - of these, only1.43% caused contention.
- For this and many other realistic workloads on FreeBSD 6.x, the Giant lock is no longer a significant factor in kernel performance.

Mission Accomplished?

...Not Yet!

- Locking work is ongoing
  - Some subsystems still Giant-locked (SCSI, TTY, IPv6)
  - Locking optimization and further pushdown work
- Nevertheless, a significant validation of the work of many FreeBSD developers over the past 6 years.

Conclusions

- Poor scaling of bufdaemon should be studied
  - For some applications it may be appropriate to use noasync/async mounts for better performance.
- Use an async swap-backed md if you can!
  - Crazy fast
- FreeBSD 6.0 performs much better on the test hardware than all previous versions tested
  - 30% faster than 4.11 for concurrent writes
  - 15% faster than 5.4 for concurrent reads
    - Also faster than 4.11, but not shown
- Revisit scaling tests on 16-core amd64/32-thread Ultrasparc T1 as we plan development priorities for existing and future commodity hardware.