Scalability Update, 2007

Kris Kennaway
The FreeBSD Project
kris@FreeBSD.org

May 17, 2007

Progress on FreeBSD SMP performance and scalability since BSDCan Dev Summit 2006

Outline

1. A case study: SQL database performance
2. Major SMP changes since May 2006
3. Remnants of the Giant lock
4. Future work and new SMP goals

A Case Study: SQL database performance

Why Database?

- Off-the-shelf benchmark (sysbench; `/usr/ports/benchmarks/sysbench/`)
- Seems to be a reasonable benchmark (as opposed to super-smack; 1-byte I/O!)
- FreeBSD did not perform well compared to Linux; excellent motivator for performance improvements

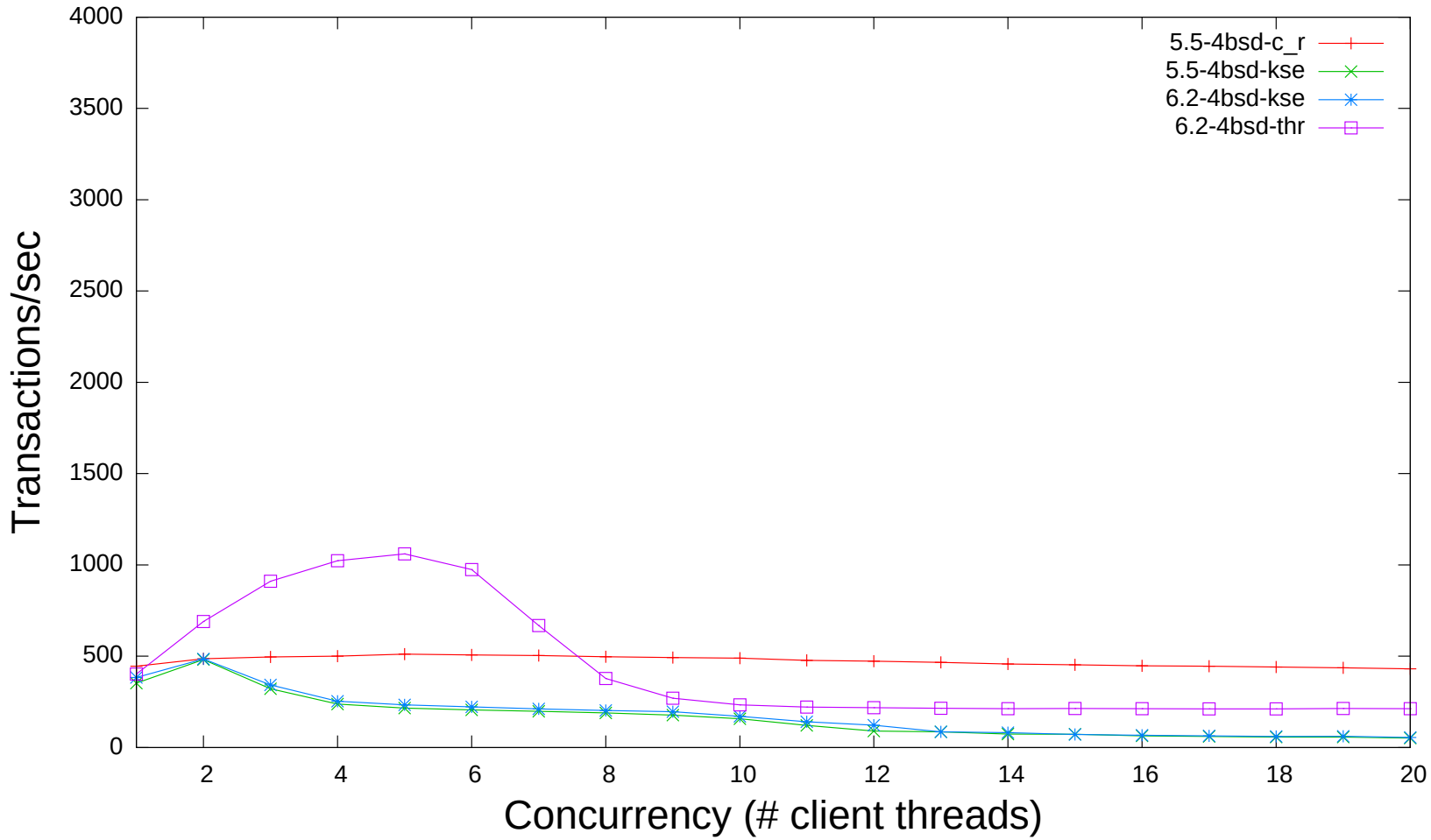
What Database?

- mySQL 5.0.37 (thread-based)
- PostgreSQL 8.2.4 (process-based + SysV IPC)

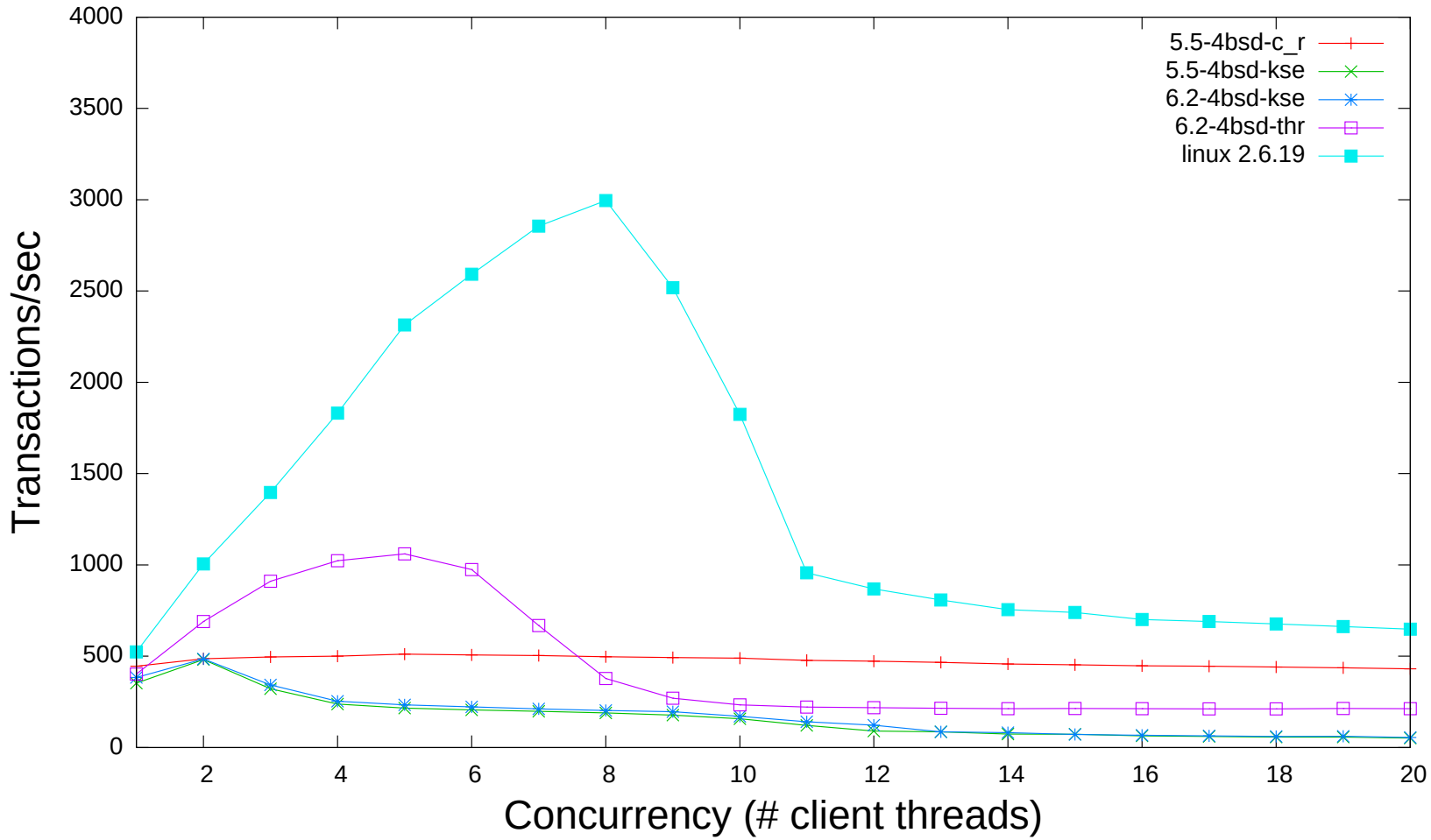
Configuration details

- amd64 system, 4*dual-core 2.2GHz CPUs
- 3GB RAM (working set fits in RAM)
- sysbench OLTP test
  - Complex transaction-based queries
  - Read-only; no disk access to avoid benchmarking disk performance
- multithreaded benchmark client
- clients and servers on the same system
- communication via UNIX domain socket
- 2 minute runs, after a warm-up run with 8 clients.

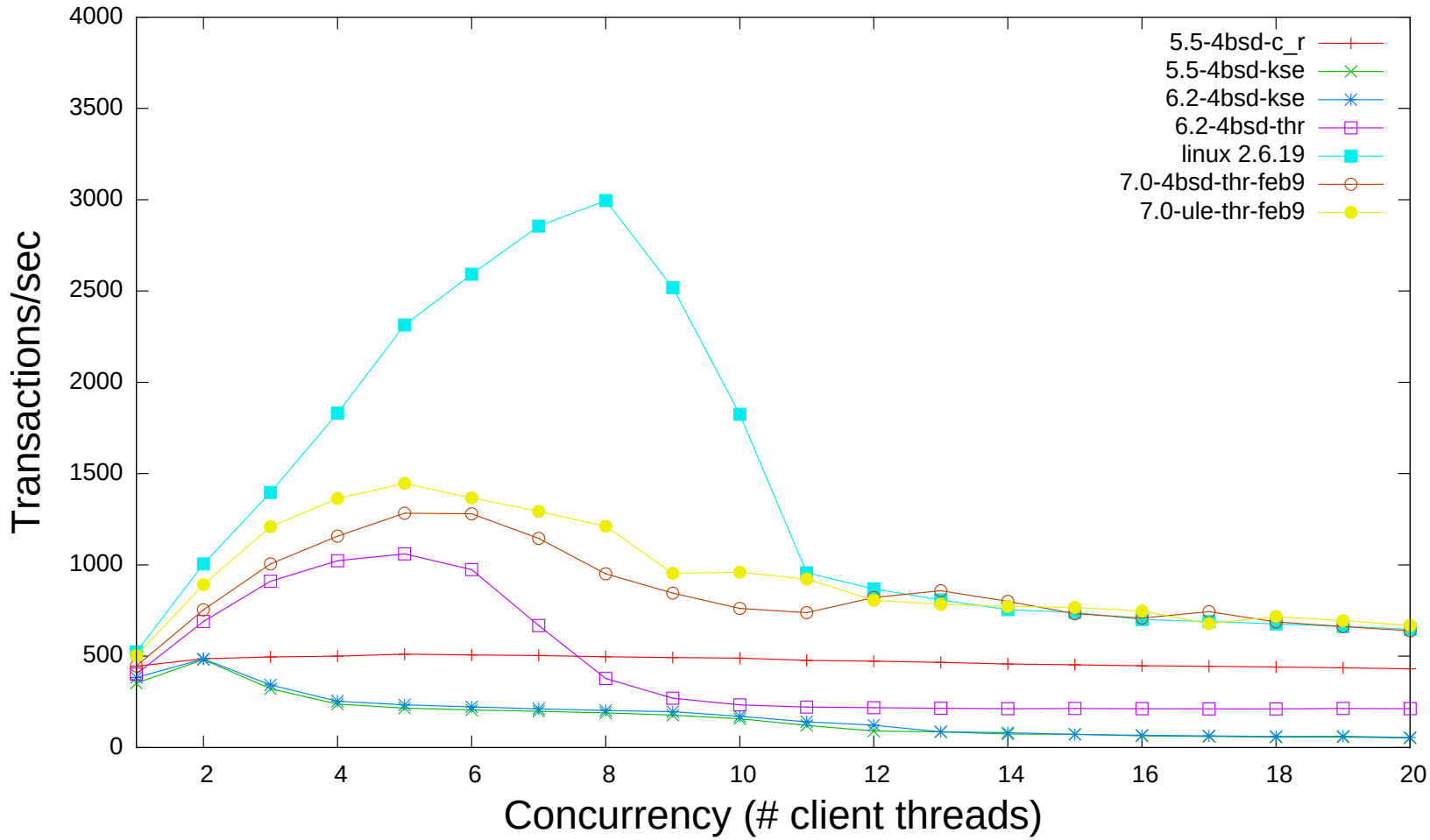
MySQL: Progress in FreeBSD 5.x and 6.x



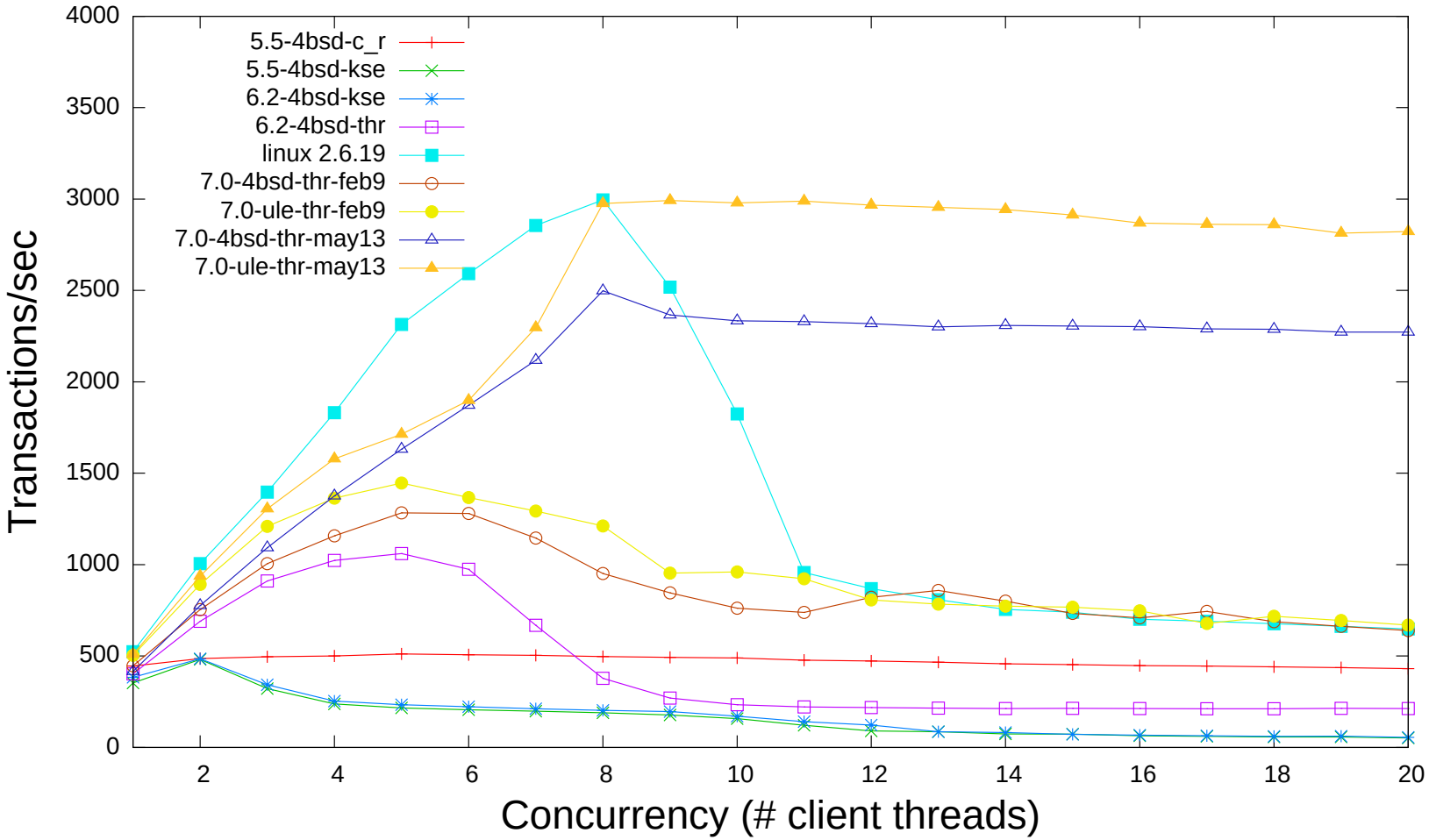
MySQL: The competition (FreeBSD vs Linux)



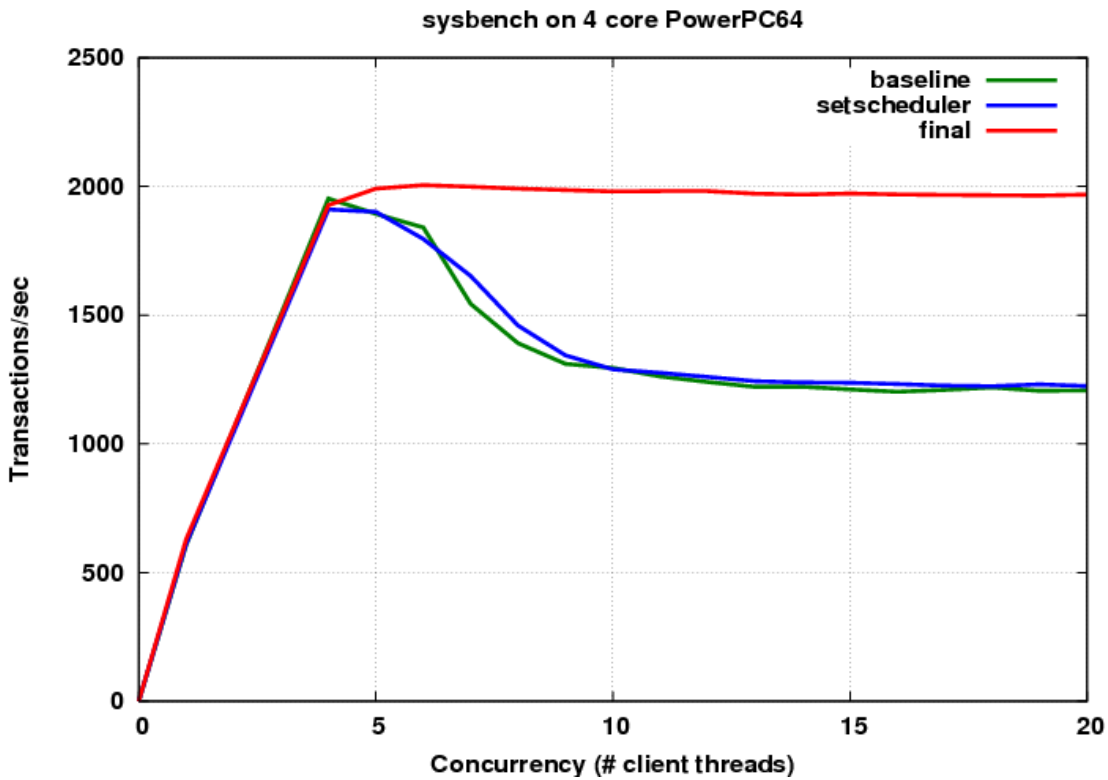
MySQL: Progress in FreeBSD 7.0 through February 2007



MySQL: Where we are today



Linux: Blame it on malloc? glibc vs tcmalloc



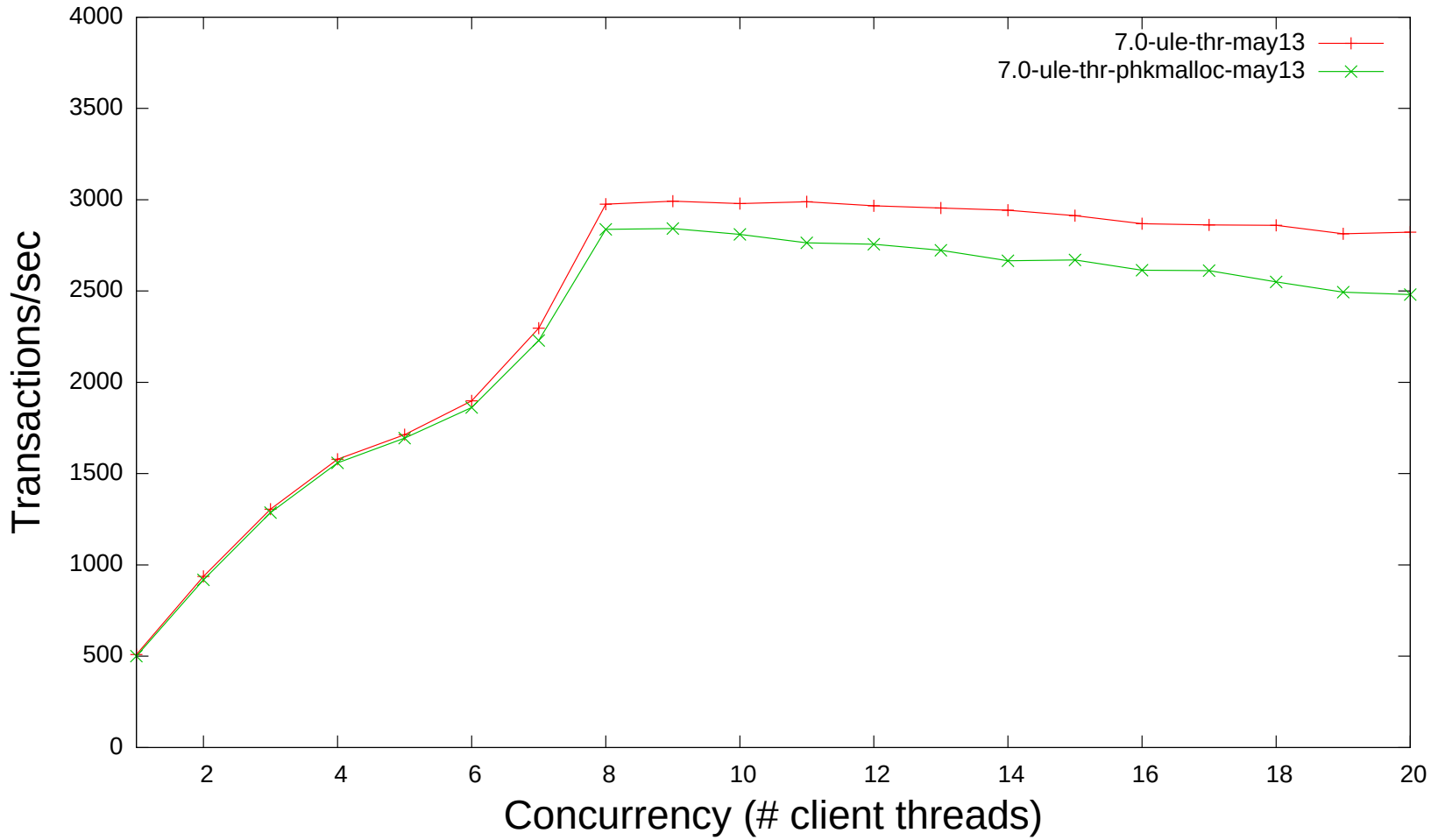
Source: `http://ozlabs.org/~anton/linux/sysbench/`

Linux: Not the whole story?

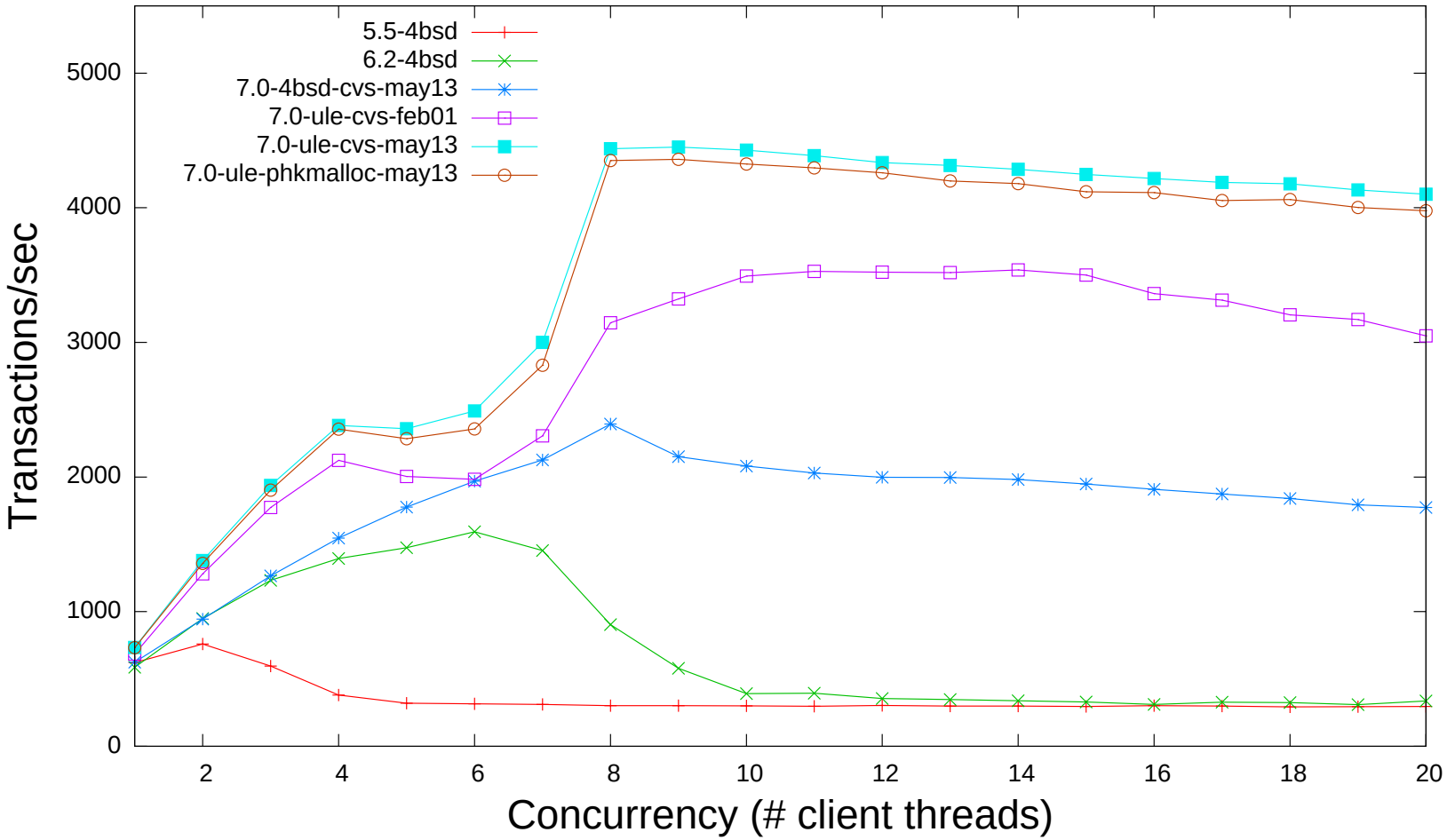
- Some reports on LKML of difficulty reproducing the tcmalloc result on 8-core systems
- tcmalloc claimed not to be a good general-purpose allocator
- A process-global semaphore acquired by all linux futex operations.
  - Agrees with the asymptotically single-threaded performance measured on amd64

...not our problem!

MySQL: jemalloc vs phkmalloc



PostgreSQL


5.5-4bsd
6.2-4bsd
7.0-4bsd-cvs-may13
7.0-ule-cvs-feb01
7.0-ule-cvs-may13
7.0-ule-phkmalloc-may13
Transactions/sec
0
1000
2000
3000
4000
5000
2
4
6
8
10
12
14
16
18
20
Concurrency (# client threads)

Progress since BSDCan 2006 (highlights)

- More efficient lock profiling
  - Still very inefficient; explore a worker thread model?
- UNIX domain socket locking; fine-grained locking for better concurrency
- Experimental scheduler locking work (on-going)
- reader-writer (rw) locks (non-sleepable)
- Optimized sx locks (sleepable); now an efficient low-level primitive
- File descriptor locking; home-brewed msleep lock $\longrightarrow$ sx

Progress since BSDCan 2006 II

- ULE 2.0. Now significantly out-performing 4BSD on the SMP workloads that have been benchmarked.
  - A remaining performance anomaly: poor CPU affinity at intermediate loads on dual-core systems
  - Benchmarking on other workloads required
- Giant pushdown
- Network stack optimization and regularization
- libthr is now the default thread library in 7.0 (finally!)
  - Should also be default on 6.x
- CAM subsystem locking (`scottl`)

Conclusion of the SMPng project

The SMPng project was launched in June 2000 and was a major focus of development in FreeBSD 5.x and above.

> *"The goal of the SMPng Project is to decompose the Giant lock into a number of smaller locks, resulting in reduced contention (and improved SMP performance)."*

The SMPng project was formally concluded in February 2007 after achieving this goal.

Remnants of the Giant lock

The FreeBSD 7.0 kernel is mostly Giant-free. Remaining Giant-locked systems are:

- TTY locking. In most configurations the Giant lock has now effectively devolved onto a Giant TTY lock.
- Some drivers (some CAM SIMs, pseudo-devices)
- Some filesystems (MSDOS, SMB, NFSv4, ...)
- Some network protocols (IPX over IP, IPv6 ND6/MLD6, netatm, ...)
- newbus (`scottl`; work in progress)
- USB, Firewire
- Parts of VM (contigmalloc, ...)
- Some scattered use elsewhere: sysctls, file operations, sysarch(), ...

See `http://wiki.freebsd.org/SMPTODO` for the up-to-date list of Giant-locked code and project ownership.

Danish axe time!

Legacy Giant-locked code to be removed in 7.0

- KAME IPSec; in favour of Fast IPSec (gnn, bz)
- I4B ISDN stack (rwatson)
- Legacy drivers: an arl awi cnw ce cp cs ctau cx en ex fe hfa idt ie if_ic oltr mn pcf pdq ray sbni sbsh snc sr tx wl xe (rwatson)
- netatm? (rwatson, Skip Ford)

Future work I: locking primitives

Legacy/home-grown locking primitives

- lockmgr
  - Proof of concept from `ups`
  - 1000 concurrent stat calls of the same file on an 8-core: reduces system time by 95% and real time by 82%.
  - lockmgr to be rewritten by Attilio Rao for Google SoC 2007
- Any others lurking?
  - msleep() is a pessimal synchronization/serialization primitive!
    - **Lesson of SMPng**: when the lock owner is running on another CPU it is almost always much better to spin instead of always going to sleep immediately.
    - Use standard primitives instead of rolling your own.
- Consolidation of primitives; too many?
  - sema()
  - msleep()/tsleep()/wakeup()/... → cv_*()?
  - ...?

Future work II: General

- Look for opportunities for shared locking - but profile carefully!
  - e.g. namei() is an "obvious" candidate for using a rwlock, but this slows it down significantly.
    - probably because it exposes lockmgr to concurrent access.
- select() locking (giant select lock)
  - How can it be optimized? This is the major remaining source of lock contention/wait time for the SQL benchmarks (contended on 50-75% of acquisitions)
- SysV IPC; POSIX advisory locks
  - Sanitize, explore finer-grained locking
- Remove non-MPSAFE subsystem crutches
- VM
  - vm_page_queue mutex is heavily contended

Future work III: Filesystem and Scheduler

Filesystem work

- More MPSAFE filesystems needed
  - MSDOS: Brian Chu, Google SoC 2007
- Shared lookups for UFS
  - Enables other work: rwlock for namei()

Scheduler

- scheduler lock is abused for non-scheduler purposes (rusage, ldt, vmmeter, timers)
  - `//depot/user/attilio/attilio_schedlock`
- global scheduler lock appears to be a barrier to scaling on ≥ 8 CPUs.
  - WIP to explore per-CPU scheduler locks with ULE; `jeffr`, `attilio`

Future work IV: Network stack

- Explore methods for improving parallelism in network processing
- Driver work
  - e.g. bce driver performs poorly with concurrent send/receive
  - others not yet evaluated
- multiple netisr worker threads
  - e.g. loopback transport; very easy to saturate a single netisr
    - important for services that cannot use unix domain socket for local transport
- Optimization work to support 10ge (`kmacy`)

New SMP Objectives: consolidation and optimization

1. Identify additional workloads to be optimized.
   - **You can help!** We need to identify good benchmarks modelling real-world performance cases.
   - Ideally should be simple to set up and operate.
   - If you have a test case I will be happy to work with you to profile it.
2. Consolidate performance on 8-core systems without regressing performance on < 8 CPU cores
3. Ready to push scaling upwards to 16 and 32-core systems
   - ...but need access to hardware
   - sun4v? Not currently stable.
   - Need access to 16-core amd64 hardware

An empirical observation

The scaling of FreeBSD n.x appears to be governed by

$$N_{\text{CPUs}} \sim 2^{n-4}, \quad n \geq 4$$