Database Hardware Benchmarking

Greg Smith

Truviso

05/21/2009

About this presentation

- The master source for these slides is
  `http://www.westnet.com/~gsmith/content/postgresql`
- Slides are released under the Creative Commons Attribution 3.0 United States License
  `http://creativecommons.org/licenses/by/3.0/us`

Why should you always benchmark your hardware?

- Many useful tests will only run when the server isn't being used yet
- Software stacks are complicated
- Spending money on upgrades only helps if you upgrade the right thing usefully
- Vendors lie

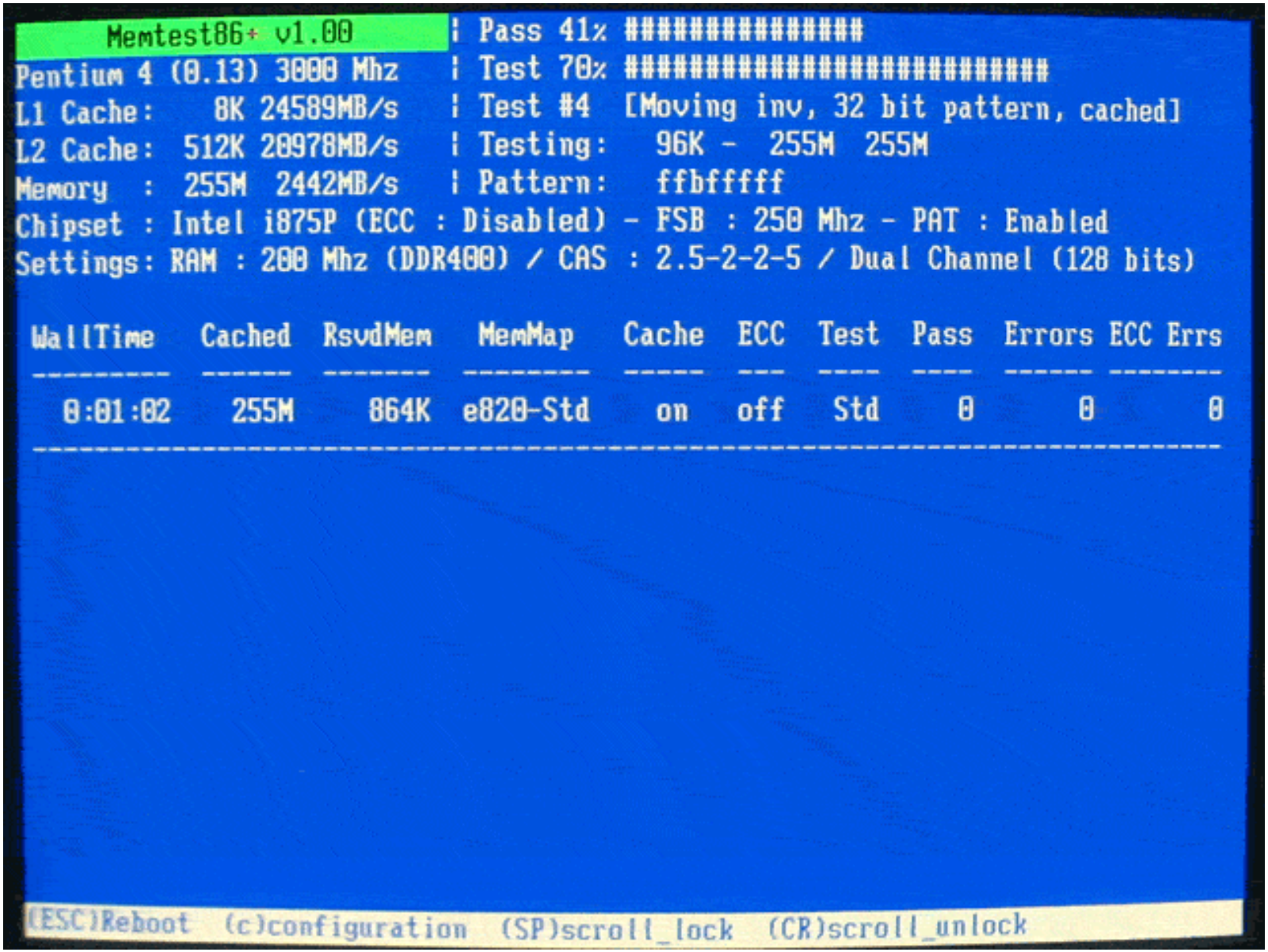
Systematic Benchmarking

- Memory
- CPU
- Disk
- Database server
- Application

Memory Tests

- DOS: memtest86+ (also on many Linux CDs, like the Ubuntu installer)
- Windows: SiSoftware Sandra
- UNIX: sysbench
- Linux: hdparm (only sometimes!)

memtest86+ sample



Memory test comparisons

CPU	Frequency	RAM Speed	memtest86+	sysbench
Q6600	2.4GHz	DDR2/667	2678	-
Q6600	2.4GHz	DDR2/800	3352	2009
T9400	2.53GHz	DDR2/1066	3743	2226
Xeon X5450	3.00GHz	DDR2/1333	3575	2487

- In Q6600 examples, DDR2/800 should be 20% faster than DDR2/667
- The 25% gain is because of improved clock multipliers

Sources for slow memory results

- Single channel RAM/slot mistakes
- Incorrect SPD/timing/voltage
- Bad RAM/CPU multiplier combination
- Poor quality RAM

PostgreSQL and the CPU

- PostgreSQL uses only a single CPU per query
- Queries executing against cached data will bottleneck on CPU
- COPY is CPU intensive

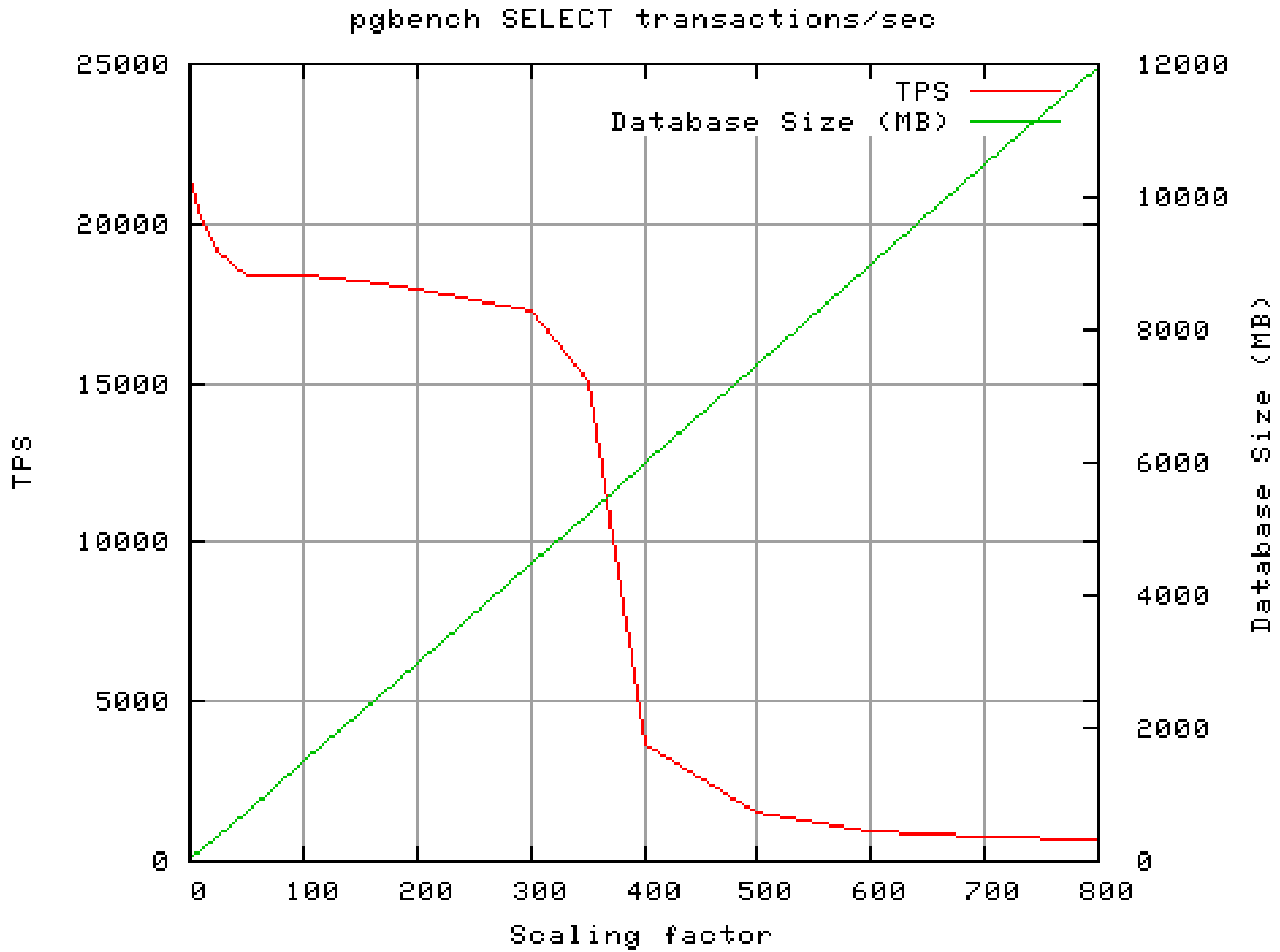
CPU Tests

- Windows: SiSoftware Sandra
- UNIX: sysbench CPU test
- Custom test with timing and generate series
- pgbench select-only on small database
- Vary pgbench client count to test single or multiple CPUs

sysbench CPU comparisons

Processor	Frequency	sysbench CPU seconds
Intel Q6600	2.40GHz	19.6
Intel T9400	2.53GHz	12.0
Intel Xeon X5450	3.00GHz	8.4

pgbench Read-Only Scaling - 8GB server



Sources for slow CPU results

- Slow memory
- Power management throttling
- Linux: /proc/cpuinfo shows 1000MHz suggests you need to adjust the CPUFreq Governor to "performance"

Disk Tests

- Sequential write: INSERT, COPY FROM (when not CPU limited)
- Sequential read: SELECT * FROM and similar table sequential scans
- Seeks: SELECT using index, UPDATE
- Commit fsync rate: INSERT, UPDATE

dd test

- Compute 2X the size of your RAM in 8KB blocks
- blocks = 250,000 * gigabytes of RAM

```
time sh -c "dd if=/dev/zero of=bigfile bs=8k count=X &&
sync"
time dd if=bigfile of=/dev/null bs=8k
```

- Watch vmstat and/or iostat during disk tests
- vmstat's bi and bo will match current read/write rate
- Note the CPU percentage required to reach the peak rate

bonnie++

```
./bonnie++
```

```
bon_csv2html
```

- Ignore the per-character and create results, look at the block output/input ones
- Random Seeks:
- The test runs SeekProcCount processes (default 3) in parallel, doing a total of 8000 random seek reads to locations in the file. In 10% of cases, the block read is changed and written back.

bonnie++ ZCAV

```
./zcav -f/dev/sda > t500
```

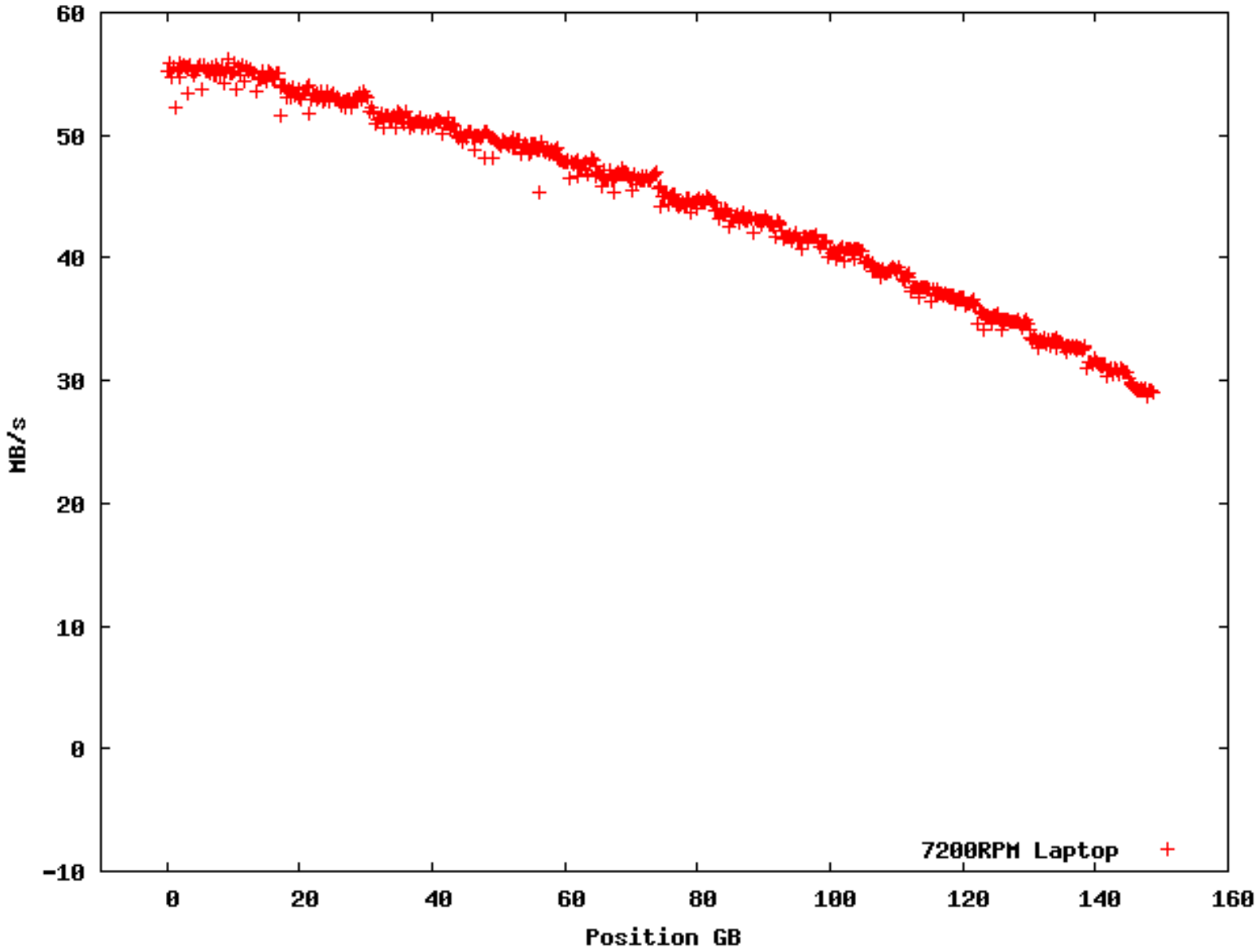
- Must get a recent version of bonnie++ for ZCAV to scale properly for TB drives (1.03e works)
- Download somewhat broken gnuplot script sample and typical results from:

`http://www.coker.com.au/bonnie++/zcav/results.html`

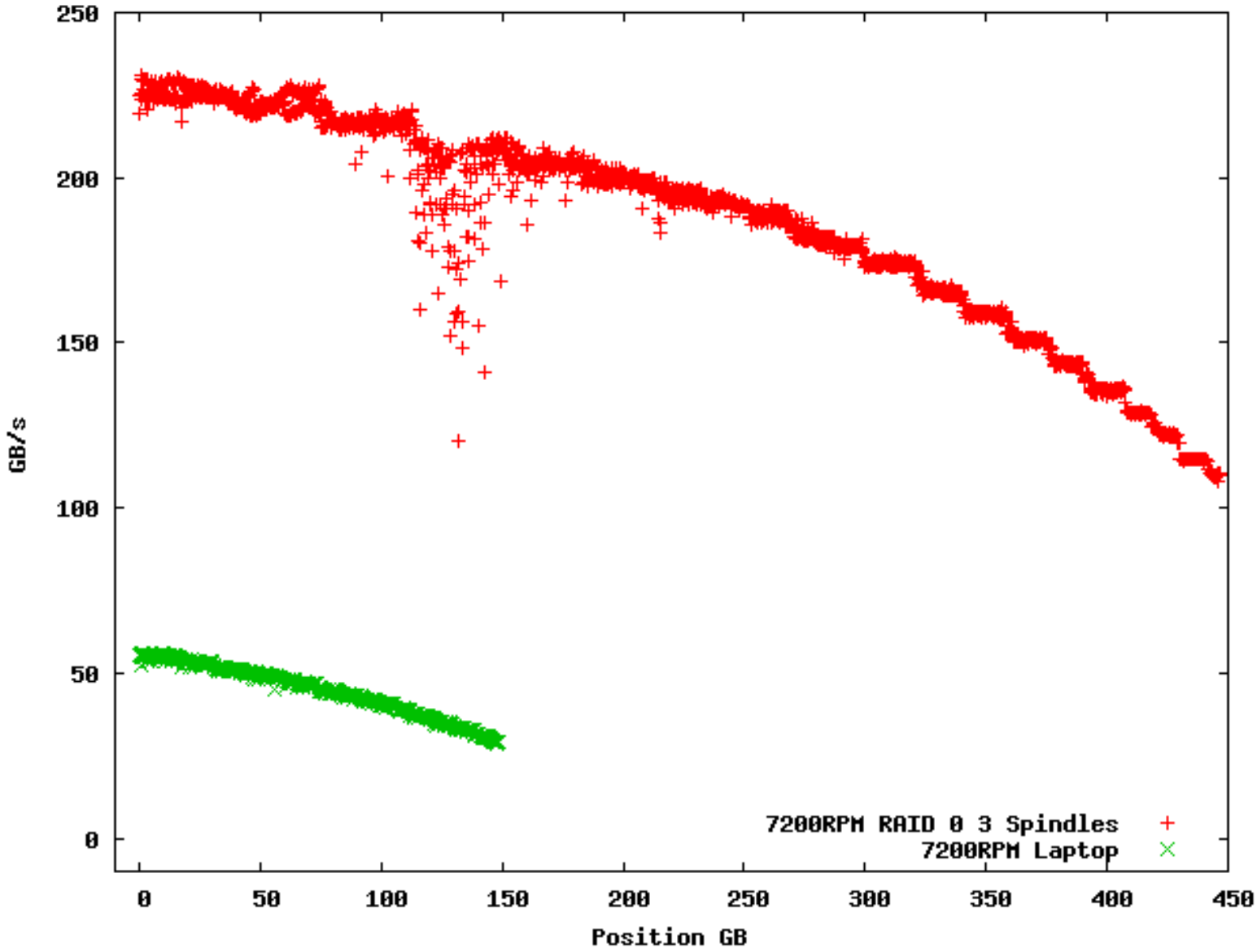
Improved bonnie++ ZCAV gnuplot script

```
unset autoscale x
set autoscale xmax
unset autoscale y
set autoscale ymax
set xlabel "Position GB"
set ylabel "MB/s"
set key right bottom
set terminal png
set output "zcav.png"
plot "raid0" title "7200RPM RAID 0 3 Spindles",
 "t500" title "7200RPM Laptop"
```

bonnie++ ZCAV: Laptop 7200RPM Disk



Comparison with 3-Disk RAID0 of 7200RPM SATA Disks



Read seeks/second - sysbench

```
THREADS=1
GB=10
MODE=rndrd
OPTIONS="--test=fileio --num-threads=$THREADS
 --file-block-size=8K --file-test-mode=$MODE
 --file-num=$GB --file-total-size=${GB}G
 --file-fsync-freq=0 --file-fsync-end=no"
sysbench prepare $OPTIONS
sysbench run --max-time=60 $OPTIONS
sysbench cleanup $OPTIONS
```

Sample sysbench random read results

```
Read 78.125Mb Written 0b

Total transferred 78.125Mb (1.0059Mb/sec)

128.75 Requests/sec executed
```

- That's 128.75 seeks/second over 10GB, resulting in a net throughput of 128.75 * 8KB/s = 1.01MB/s
- Consider both the size of the disk used and the number of clients doing seeks

More customizable seek tests

- bonnie++ experimental (currently at 1.95)
- iozone
- fio
- Windows: HD Tune does everything but commit rate

Sources for slow disk results

- Poor mapping to underlying hardware
- Buggy driver
- Insufficient bandwidth to storage
- Bottlenecking at CPU/memory limits
- Bad performing filesystem or filesystem misaligned with stripe sizes
- Writes faster than reads? Probably low read-ahead settings somewhere.

`http://it.toolbox.com/blogs/database-soup/the-problem-with-iscsi-30602`

`http://blog.endpoint.com/2008/09/filesystem-io-what-we-presented.html`

fsync tests

```
sysbench --test=fileio --file-fsync-freq=1 --file-num=1
 --file-total-size=16384 --file-test-mode=rndwr run
 | grep "Requests/sec"
```

- pgbench insert-only test
- PostgreSQL contrib/test_fsync might work, but isn't really reliable

Sample laptop disk specification

- ST9160823AS Momentus 7200.2
- 7200 RPM
- 8MB Cache
- Average seek: 11ms
- Average rotational latency: 4.17ms
- Transfer rate: 59MB/s

Computed parameters

- Rotational latency = 1 / RPM / 60 / 2
- IOPS=1/(latency+seek)
- IOPS = 1/(((1/(RPM/60))/2) + S)
- IOPS = 1/(4.17ms + 11ms) = 65.9 IOPS

IOPS Calculators and Info

http://www.wmarow.com/strcalc/

http://www.dbasupport.com/oracle/ora10g/disk_IO_02.shtml

http://storageadvisors.adaptec.com/2007/03/20/sata-iops-measurement/

Sample disk results

Disks	Read	Write	bonnie++ seeks	RO seeks	Commit
1	41	35	178 @ 6GB	129/s @ 10GB	2653/s or 58/s
3	125	119	371 @ 8GB	60/s @ 100GB	10855/s

- Two commit rates for 1 disk setup (the laptop drive) are with/without an unsafe write cache
- 3 disk RAID0 includes a 256MB battery-backed write cache

Custom PostgreSQL tests

- Quick CPU test (about 1140ms on my laptop):

```
\timing
select sum(generate_series) from
 generate_series(1,1000000);
```

- Quick insert/plan test:

```
\timing
CREATE TABLE test (id INTEGER PRIMARY KEY);
INSERT INTO test VALUES (generate_series(1,100000));
EXPLAIN ANALYZE SELECT COUNT(*) FROM test;
```

Benchmarking functions

http://justatheory.com/computers/databases/postgresql/benchmarking_upc_validation.html

http://justatheory.com/computers/databases/postgresql/benchmarking_functions.html

- pg_stat_user_functions handles this specific job in 8.4
- The general technique is applicable for all sorts of custom benchmarks

pgbench select only test

```
\set naccounts 100000 * :scale
\setrandom aid 1 :naccounts
SELECT abalance FROM accounts WHERE aid = :aid;
```

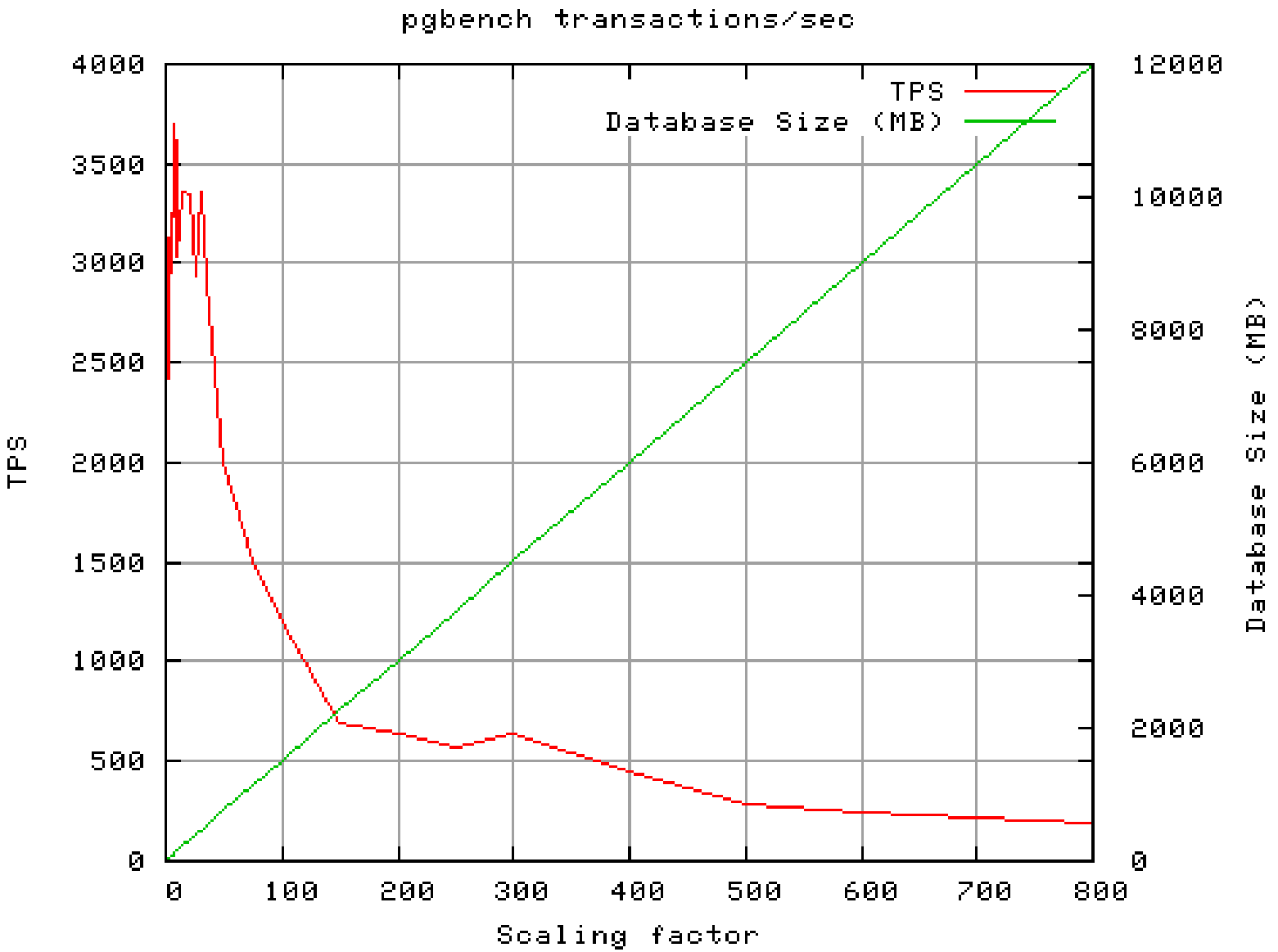
pgbench custom test: insert only, to measure commit rate

```
\set nbranches :scale
\set ntellers 10 * :scale
\set naccounts 100000 * :scale
\setrandom aid 1 :naccounts
\setrandom aid 1 :naccounts
\setrandom bid 1 :nbranches
\setrandom tid 1 :ntellers
\setrandom delta -5000 5000
BEGIN
INSERT INTO history (tid, bid, aid, delta, mtime)
  VALUES (:tid, :bid, :aid, :delta, CURRENT_TIMESTAMP);
END;
```

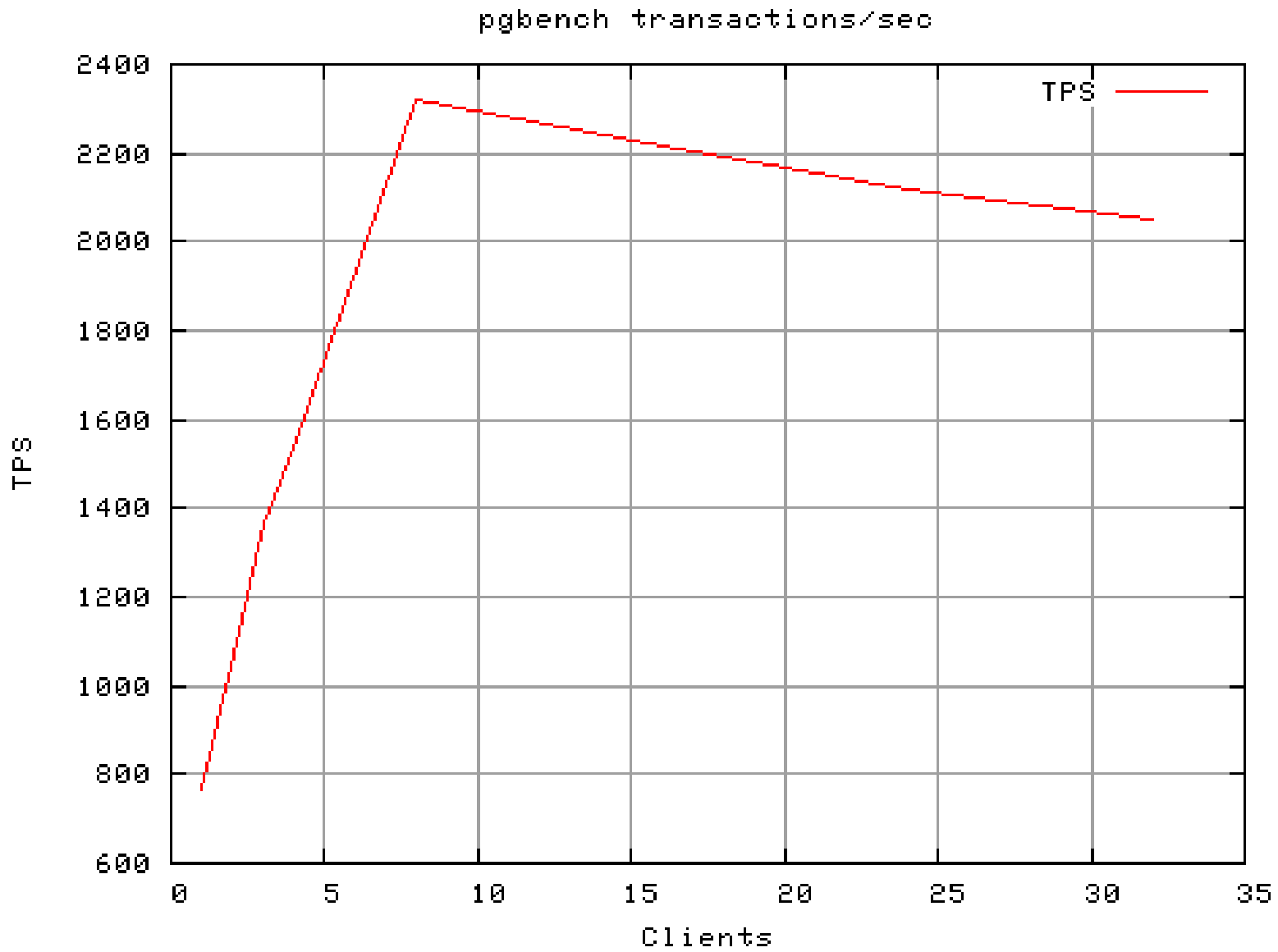
pgbench standard test

```
BEGIN;
UPDATE accounts SET abalance = abalance + :delta
  WHERE aid = :aid;
SELECT abalance FROM accounts WHERE aid = :aid;
UPDATE tellers SET tbalance = tbalance + :delta
  WHERE tid = :tid;
UPDATE branches SET bbalance = bbalance + :delta
  WHERE bid = :bid;
INSERT INTO history (tid, bid, aid, delta, mtime)
  VALUES (:tid, :bid, :aid, :delta, CURRENT_TIMESTAMP);
END;
```

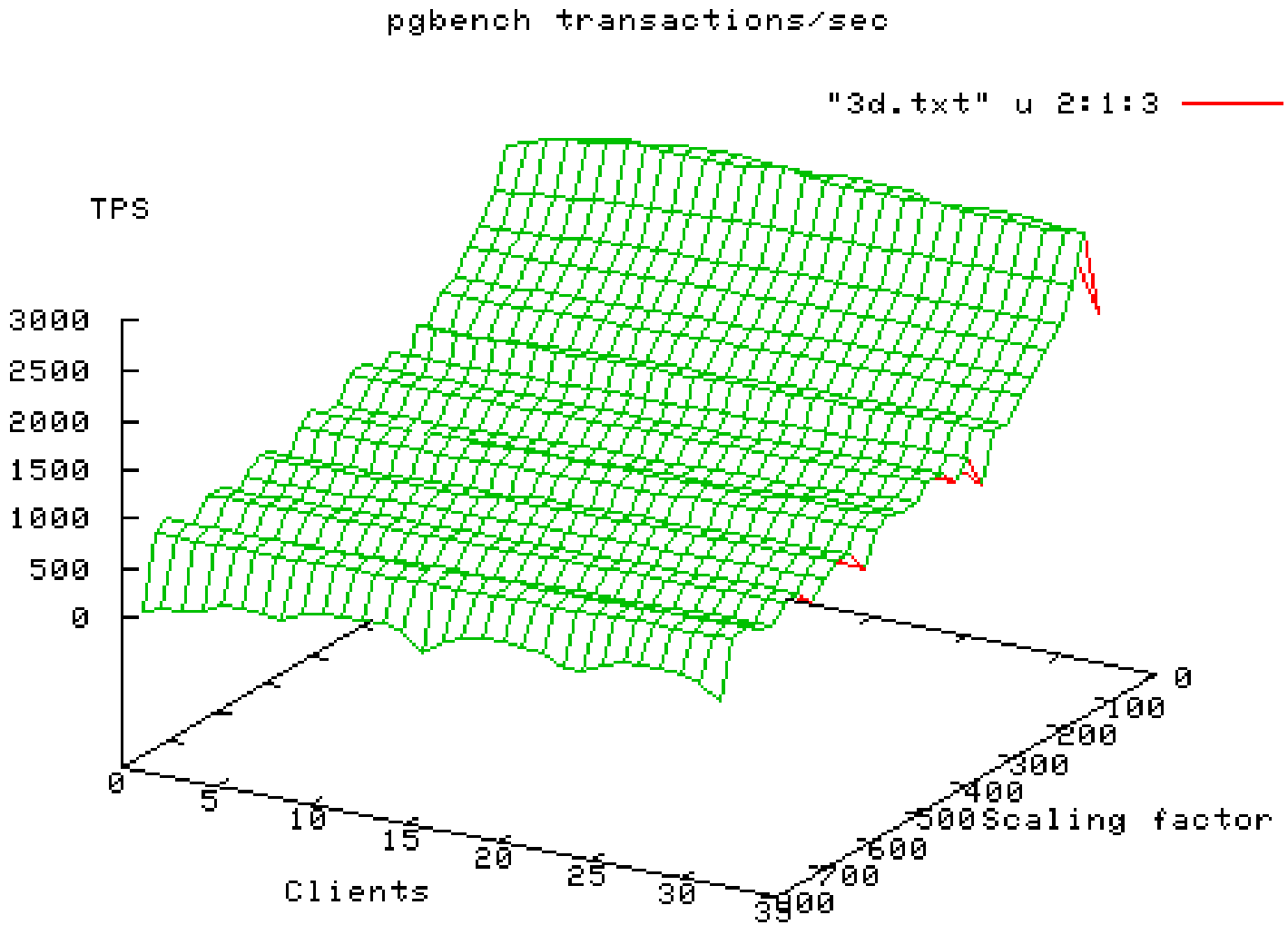
pgbench TPC-B Size Scaling



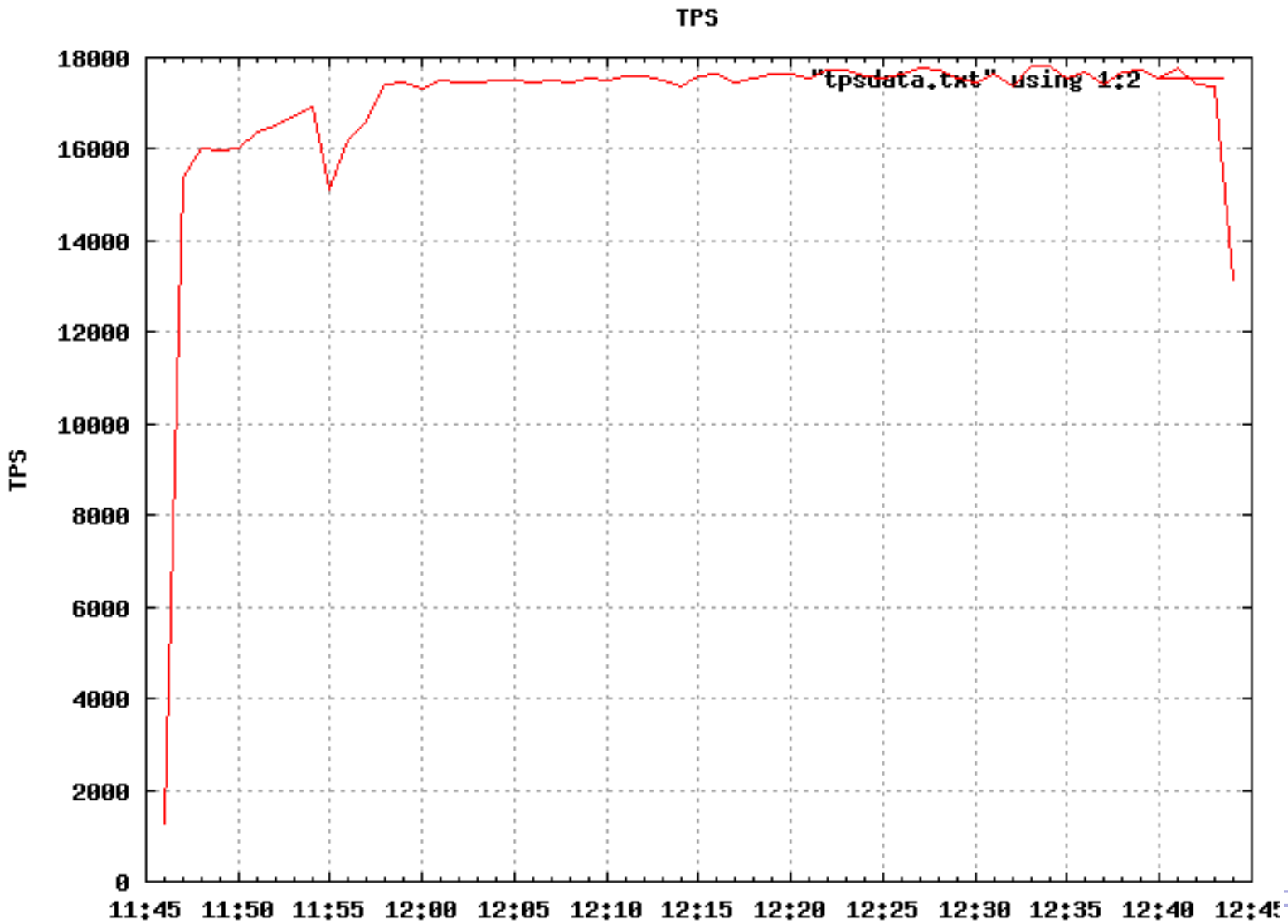
pgbench TPC-B Client Scaling



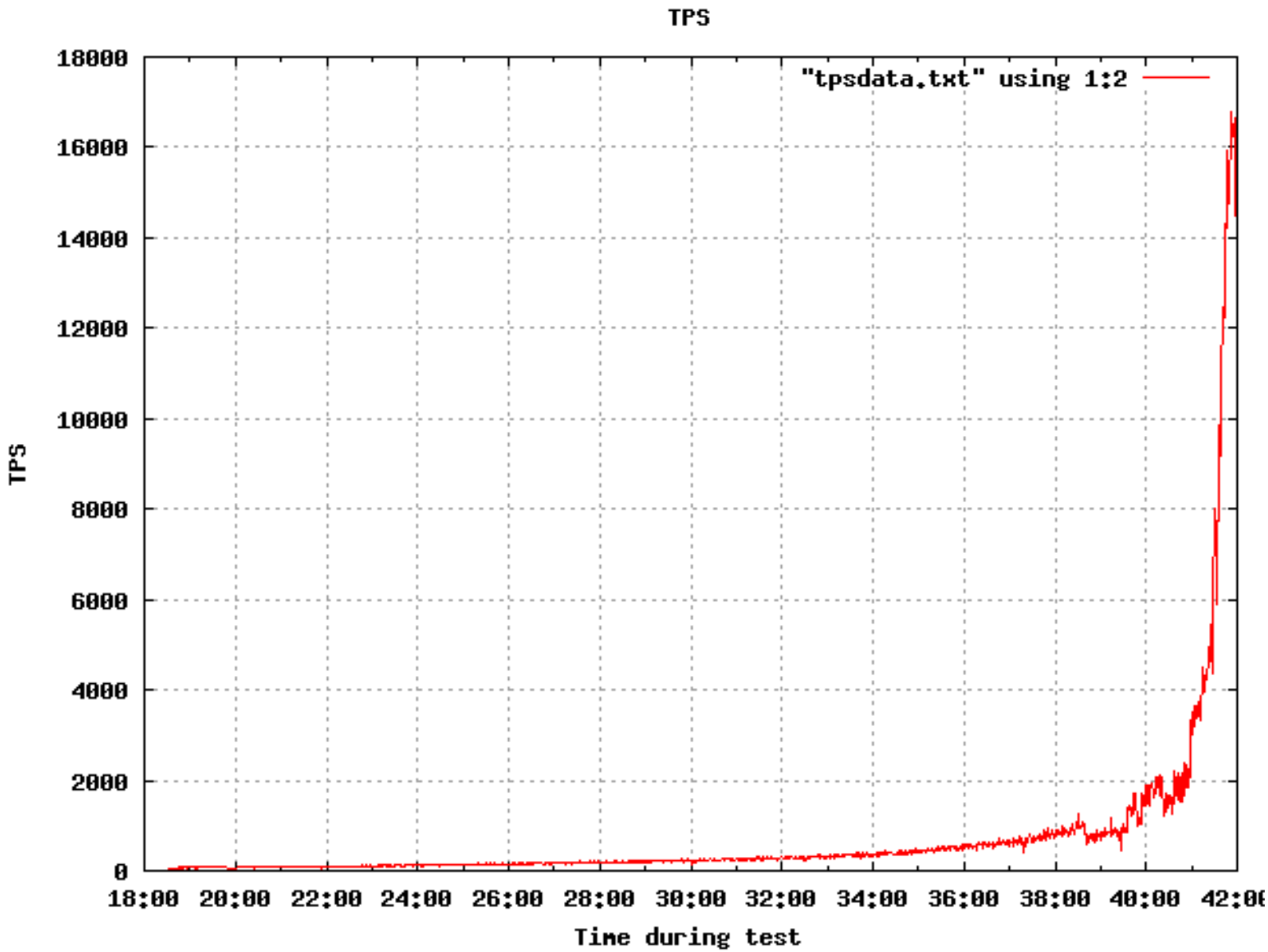
pgbench TPC-B Size and Client Scaling (3D glasses not included)



pgbench select-only warm cache: scale=100, clients=4, 1M transactions



Cold cache: scale=100, clients=4, 1M transactions



Custom test: insert size

```
create table data(filler text);

insert into data (filler) values (repeat('X',:scale));

SCALES="10 100 1000 10000"
SCRIPT="insert-size.sql"
TOTTRANS=100000
SETTIMES=1
SETCLIENTS="1 2 4 8 16"
SKIPINIT=1
```

Insert size quick test

	Scale			
Clients	10	100	1000	10000
1	1642	1611	1621	1625
2	2139	2142	2111	2232
4	3196	3306	3232	3296
8	4728	5243	5445	5038
16	9372	8309	8140	7238

What should you do?

- Trust no one
- Don't start on application benchmarks until you've proven basic performance
- Vendors alternate among lying, misunderstanding what you want, and trying to make you feel dumb
- Use simple, standard tools whenever possible to minimize vendor disputes
- Be prepared to translate to your vendor's language and subvert their agenda
- Never spend real money on hardware unless you can return it if it sucks