Introduction to pgbench

Greg Smith

Truviso

04/05/2009

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
- See http://creativecommons.org/licenses/by/3.0/us

pgbench History

- TPC-B: `http://www.tpc.org/tpcb/`
- State of the art...in 1990
- Models a bank with accounts, tellers, and branches
- pgbench actually derived from JDBCbench
- `http://developer.mimer.com/features/feature_16.htm`
- But aimed to eliminate Java overhead
- The JDBCbench code is buggy
- Early pgbench versions inherited some of those bugs

Database scale

- Each scale increment adds another branch to the database
- 1 branch + 10 tellers + 100000 accounts
- Total database size is approximately 16MB * scale

```
createdb pgbench
pgbench -i -s 10
```

- scale=10 makes for a 160MB database

Database schema

```
CREATE TABLE branches(bid int not null,
  bbalance int, filler char(88));
ALTER TABLE branches add primary key (bid);
CREATE TABLE tellers(tid int not null,bid int,
  tbalance int,filler char(84));
ALTER TABLE tellers add primary key (tid);
CREATE TABLE accounts(aid int not null,bid int,
  abalance int,filler char(84));
ALTER TABLE accounts add primary key (aid);
CREATE TABLE history(tid int,bid int,aid int,delta int,
  mtime timestamp,filler char(22));
```

Filler

- Filler itended to make each row at least 100 bytes long
- Fail: null data inserted into it doesn't do that
- Rows are therefore very narrow
- Not fixed because it would put new pgbench results on a different scale

Scripting language

```
\set nbranches :scale
\set ntellers 10 * :scale
\set naccounts 100000 * :scale
\setrandom aid 1 :naccounts
\setrandom bid 1 :nbranches
\setrandom tid 1 :ntellers
\setrandom delta -5000 5000
```

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;
```

No teller/branch test

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

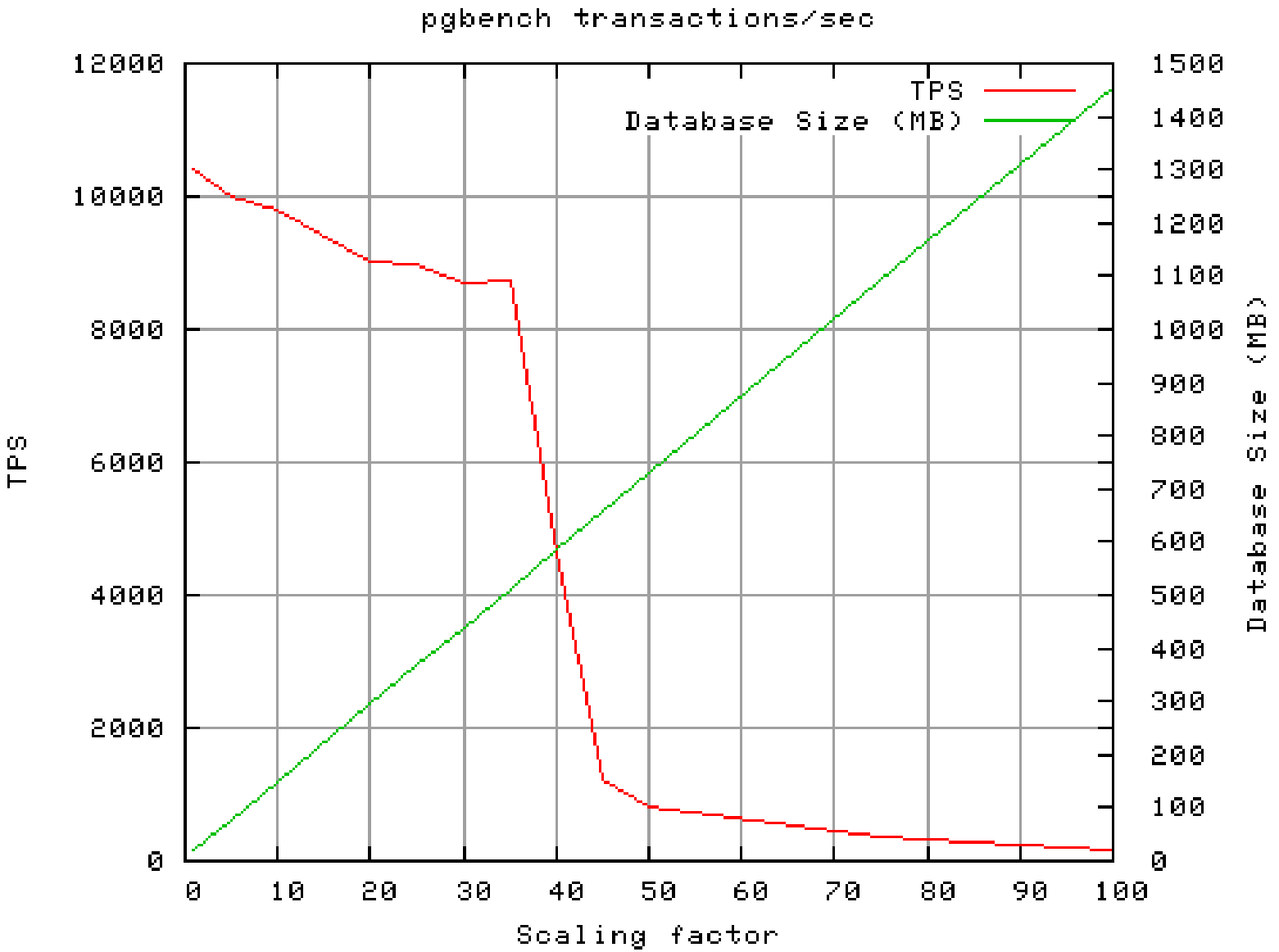
Select only test

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

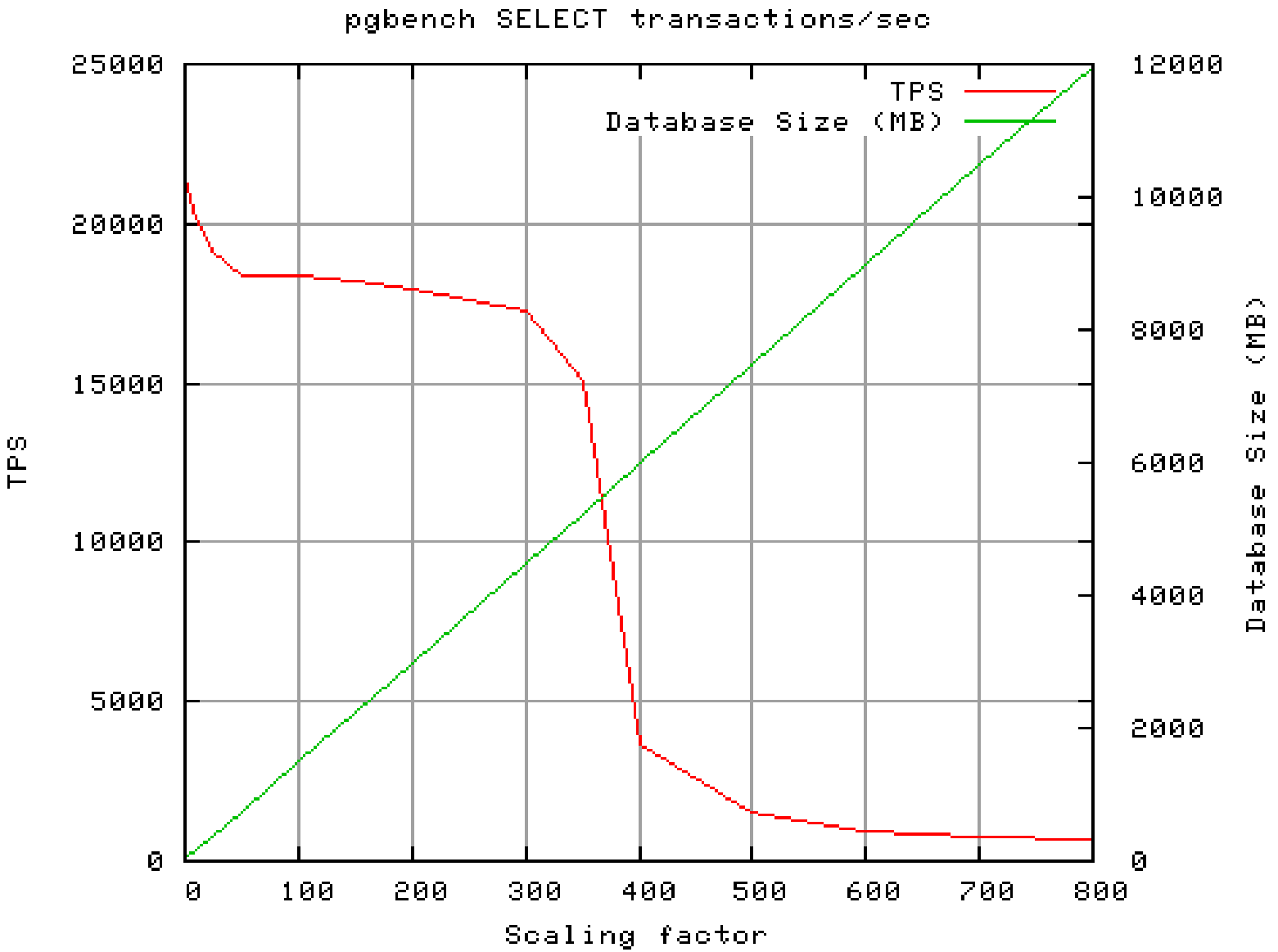
Creating a fairly large pgbench database

```
# pgbench -i -s 100 pgbench
select pg_size_pretty(pg_database_size('pgbench'));
  1416 MB
select pg_size_pretty(pg_relation_size('accounts'));
  1240 MB
select pg_size_pretty(pg_relation_size('accounts_pkey'));
  171 MB
```

pgbench Read-Only Scaling - 1GB server



pgbench Read-Only Scaling - 8GB server



postgresql.conf parameters that matter

- shared_buffers
- checkpoint_segments
- autovacuum
- synchronous_commit + wal_writer_delay
- wal_buffers
- checkpoint_completion_target
- wal_sync_method
- http://wiki.postgresql.org/wiki/Tuning_Your_PostgreSQL_Server

postgresql.conf parameters that don't matter

- effective_cache_size
- default_statistics_target
- work_mem
- random_page_cost

Gotchas

- Long enough runtime–1 minute or 100,000 transactions minimum
- Not paying attention to DB size relative to the various cache sizes
- Update contention warnings not as important
- pgbench client parsing and process switching limitations
- vacuum and bloating
- Checkpoints adding variation
- Custom scripts don't detect scale
- Developer PostgreSQL builds

pgbench-tools

- Set of shell scripts to automate running many pgbench tests
- Results are saved into a database for analysis
- Saves TPS, latency, background writer statistics
- Most of the cool features require PostgreSQL 8.3
- Inspired by the dbt2 benchmark framework

Server configuration

- Quad-Core Intel Q6600
- 8GB DDR2-800 RAM
- Areca ARC-1210 SATA II PCI-e x8 RAID controller, 256MB write cache
- DB: 3x160GB Maxtor SATA disks, Linux software RAID-0
- WAL: 160GB Maxtor SATA disk
- Linux Kernel 2.6.22 x86_64
- Untuned ext3 filesystems

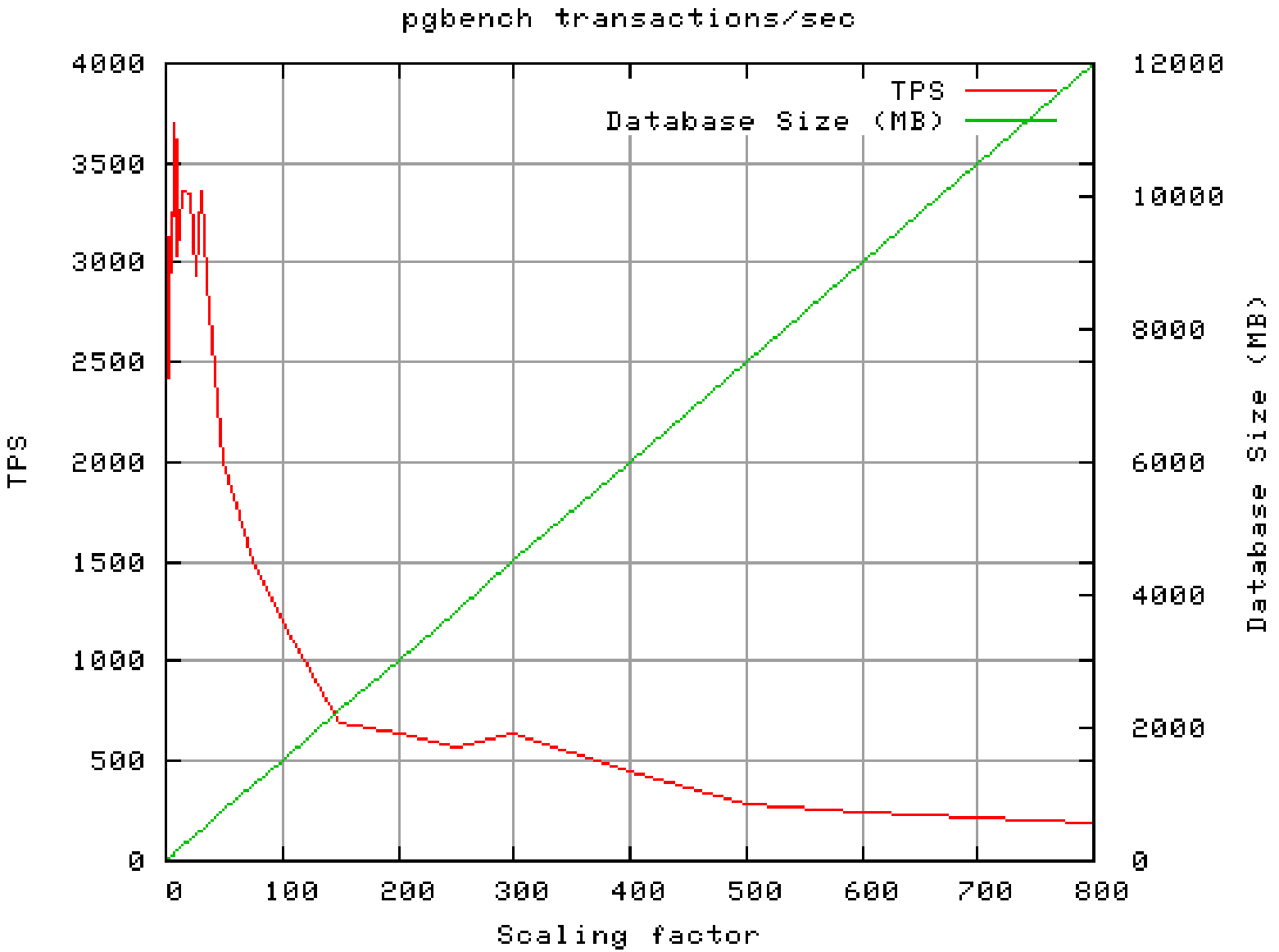
PostgreSQL Configuration

- PostgreSQL 8.4-devel
- shared_buffers = 2GB
- checkpoint_segments = 32
- checkpoint_completion_target = 0.9
- wal_buffers = 128KB
- max_connections = 100

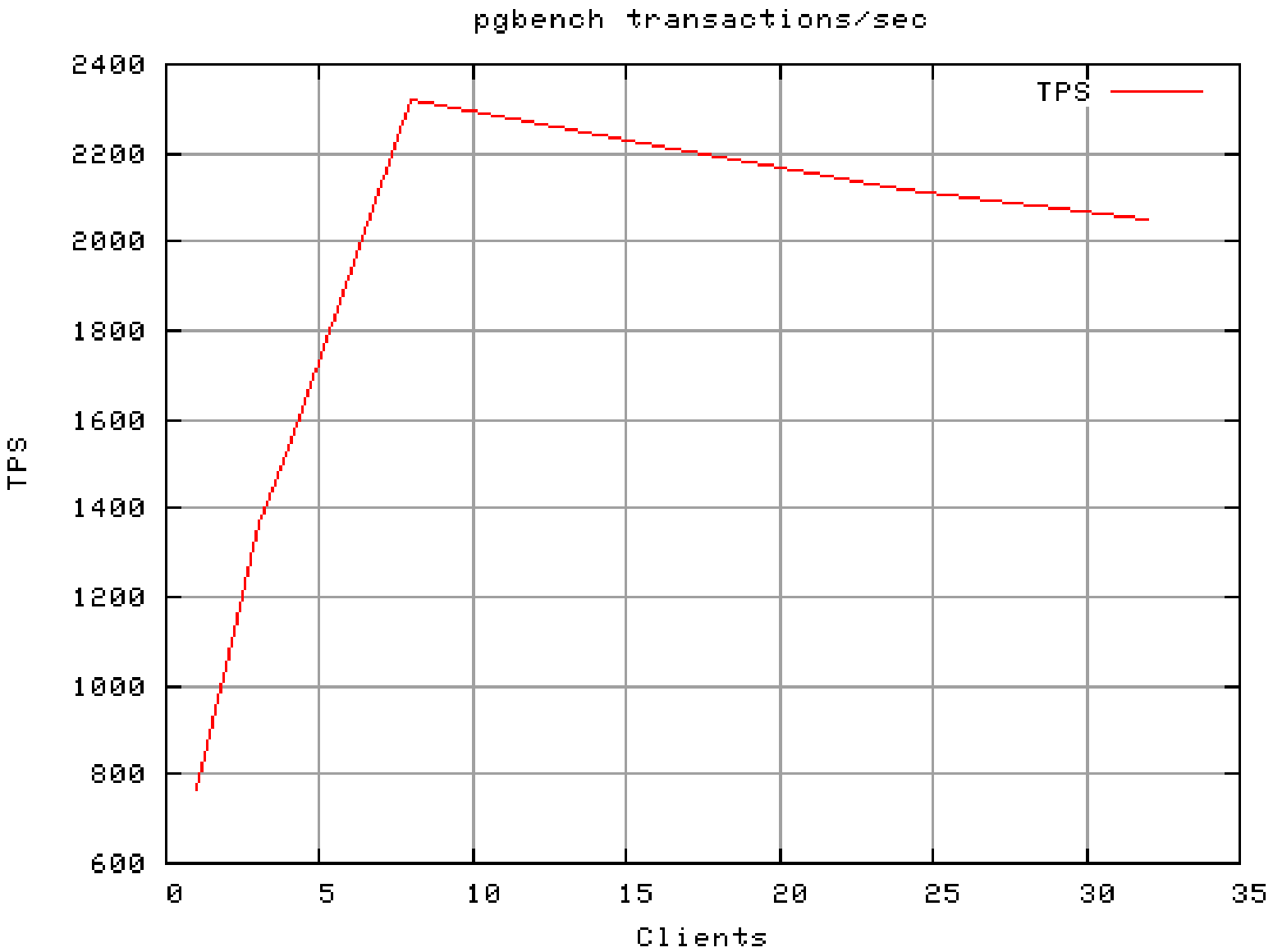
pgbench-tools Config

- SCALES="1 2 3 4 6 7 8 9 10 15 20 25 30 35 50 75 100 150 200 250 300 400 500 600 700 800"
- SCRIPT="tpc-b.sql"
- TOTTRANS=200000
- SETTIMES=3
- SETCLIENTS="1 2 3 4 8 16 24 32"

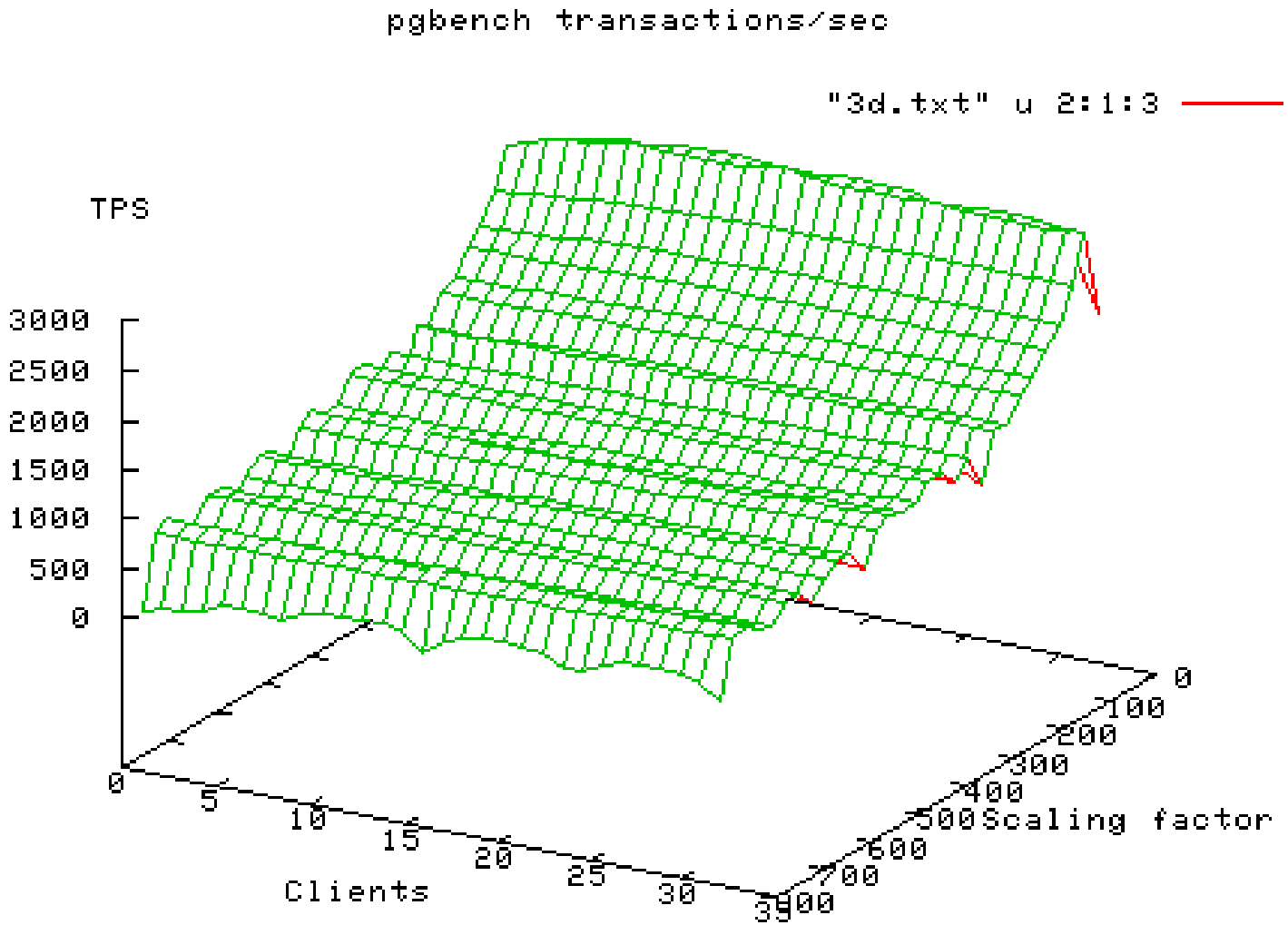
pgbench TPC-B Size Scaling



pgbench TPC-B Client Scaling



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

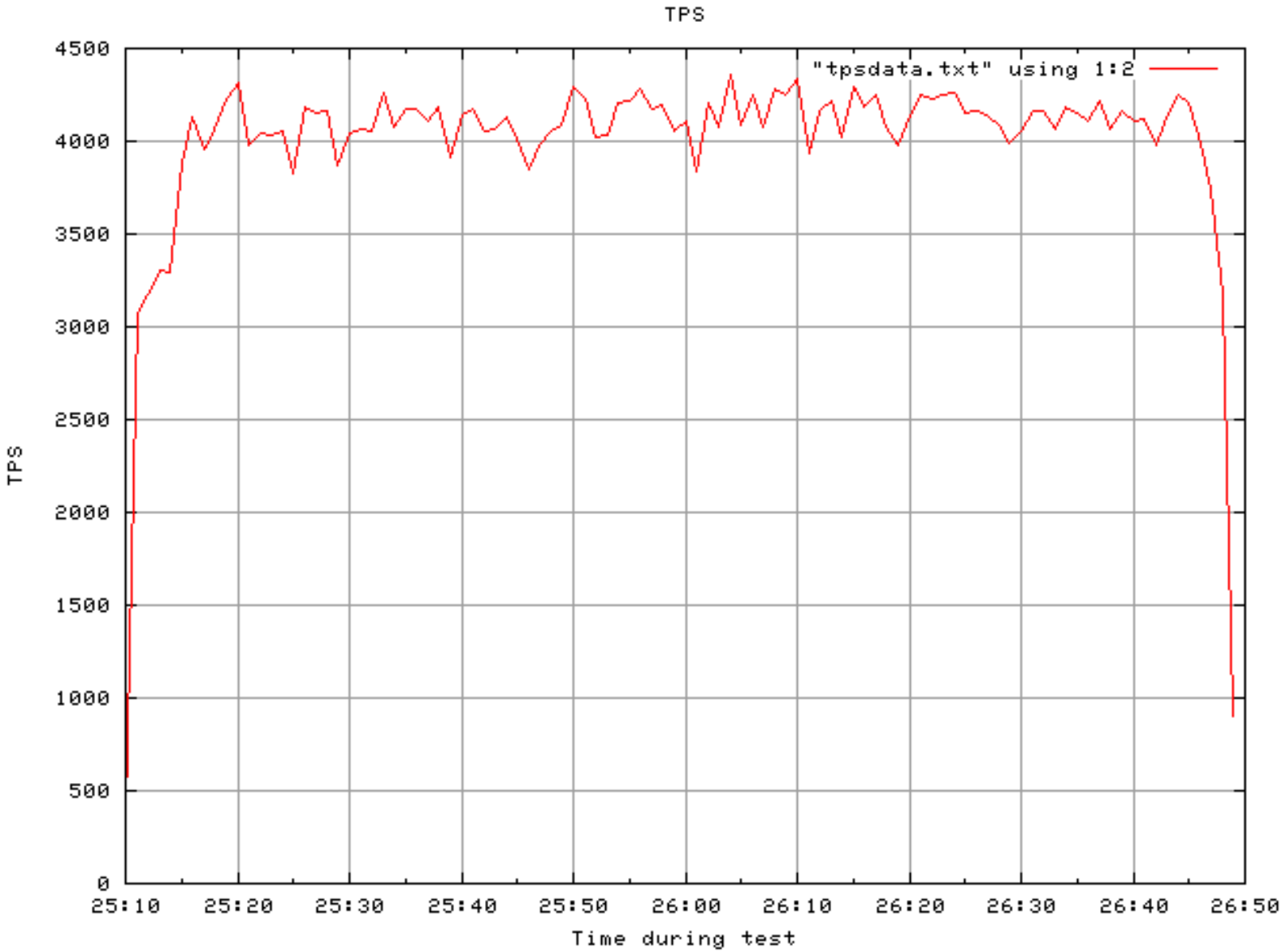


Don't be fooled by TPS

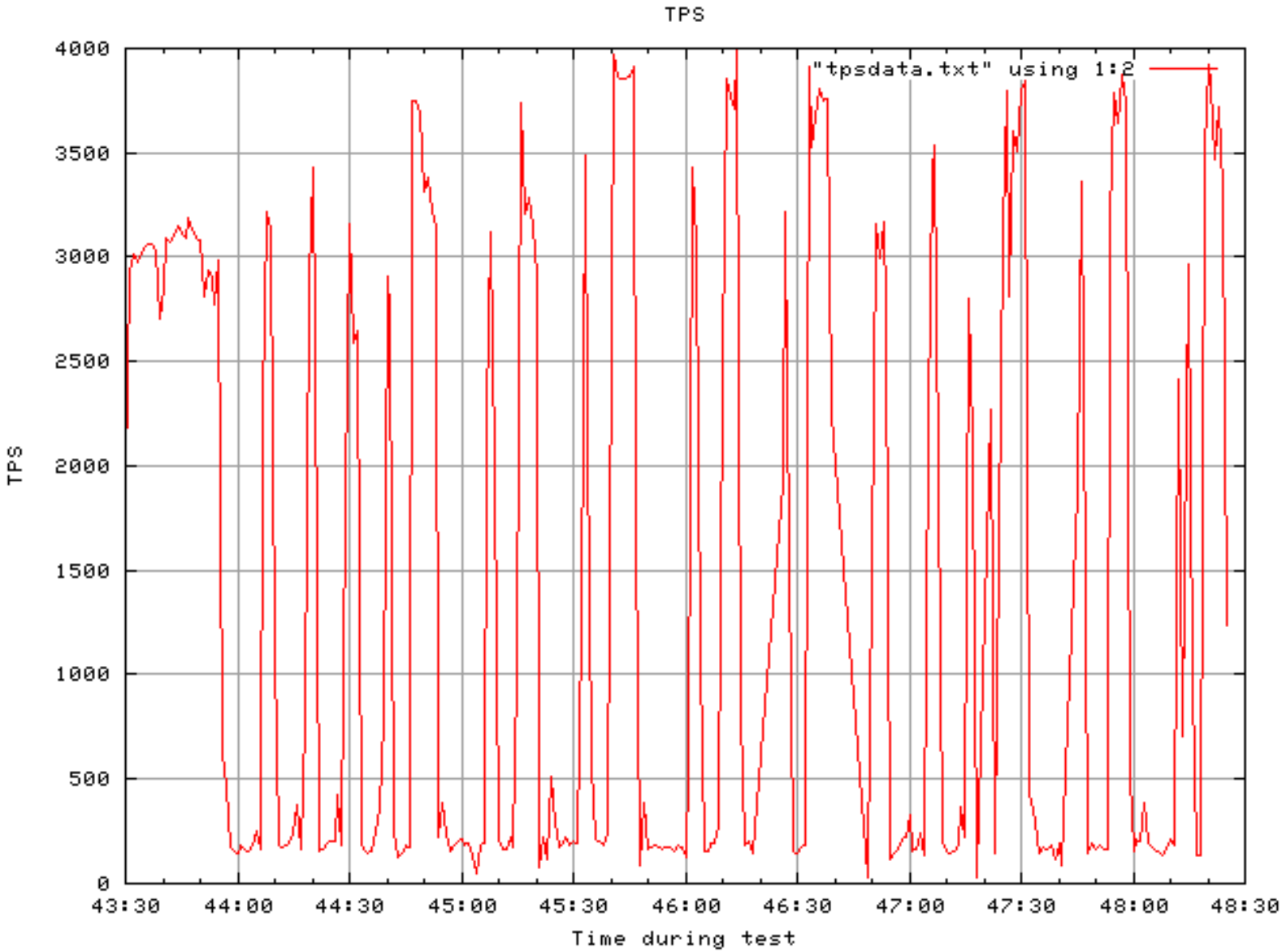
```
scaling factor: 600
number of clients: 8
number of transactions per client: 25000
number of transactions actually processed: 200000/200000
tps = 226.228995 (including connections establishing)

scaling factor: 600
number of clients: 32
number of transactions per client: 6250
number of transactions actually processed: 200000/200000
tps = 376.016694 (including connections establishing)
```

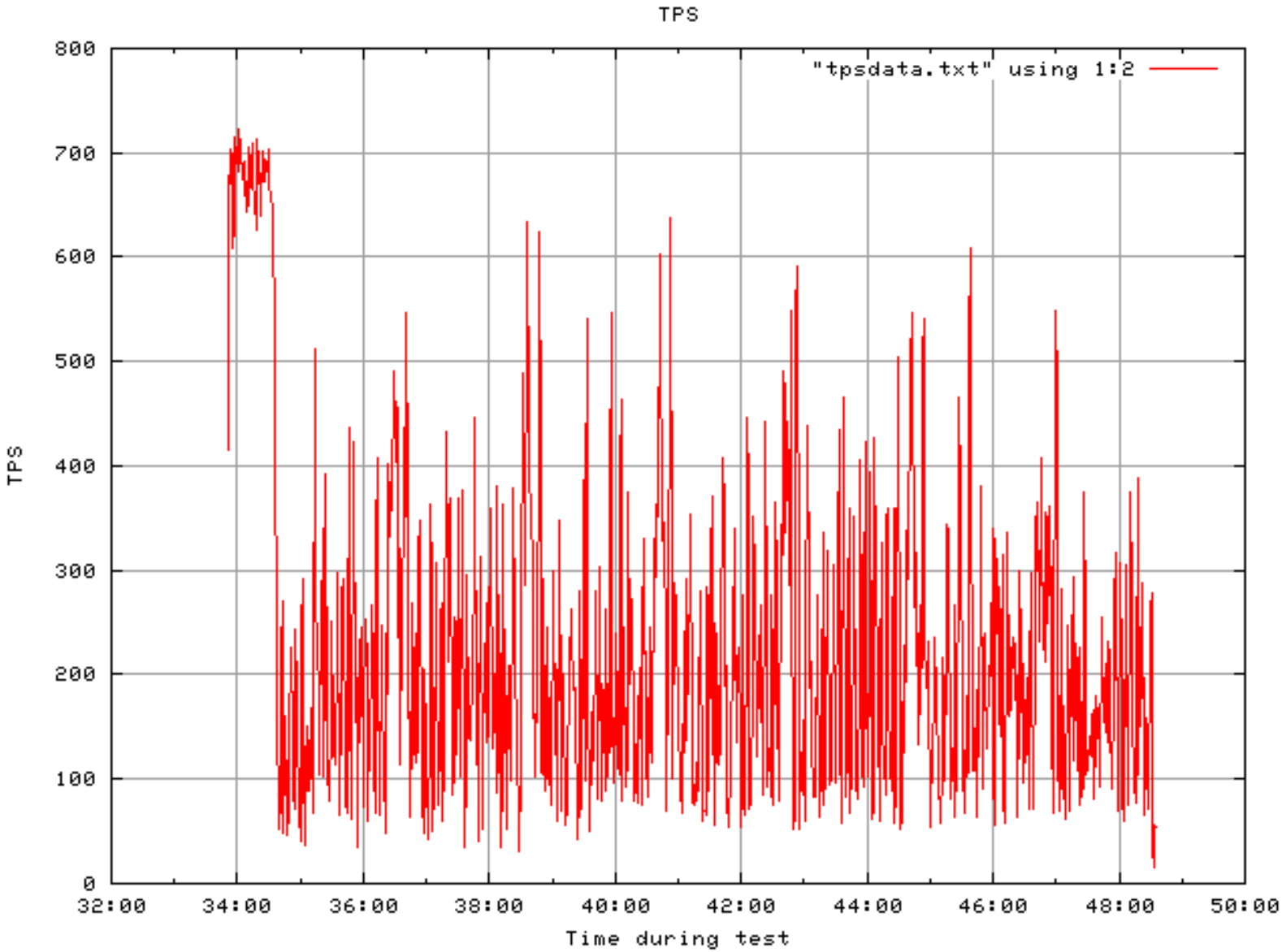
Great result: scale=10, clients=8, tps=4054



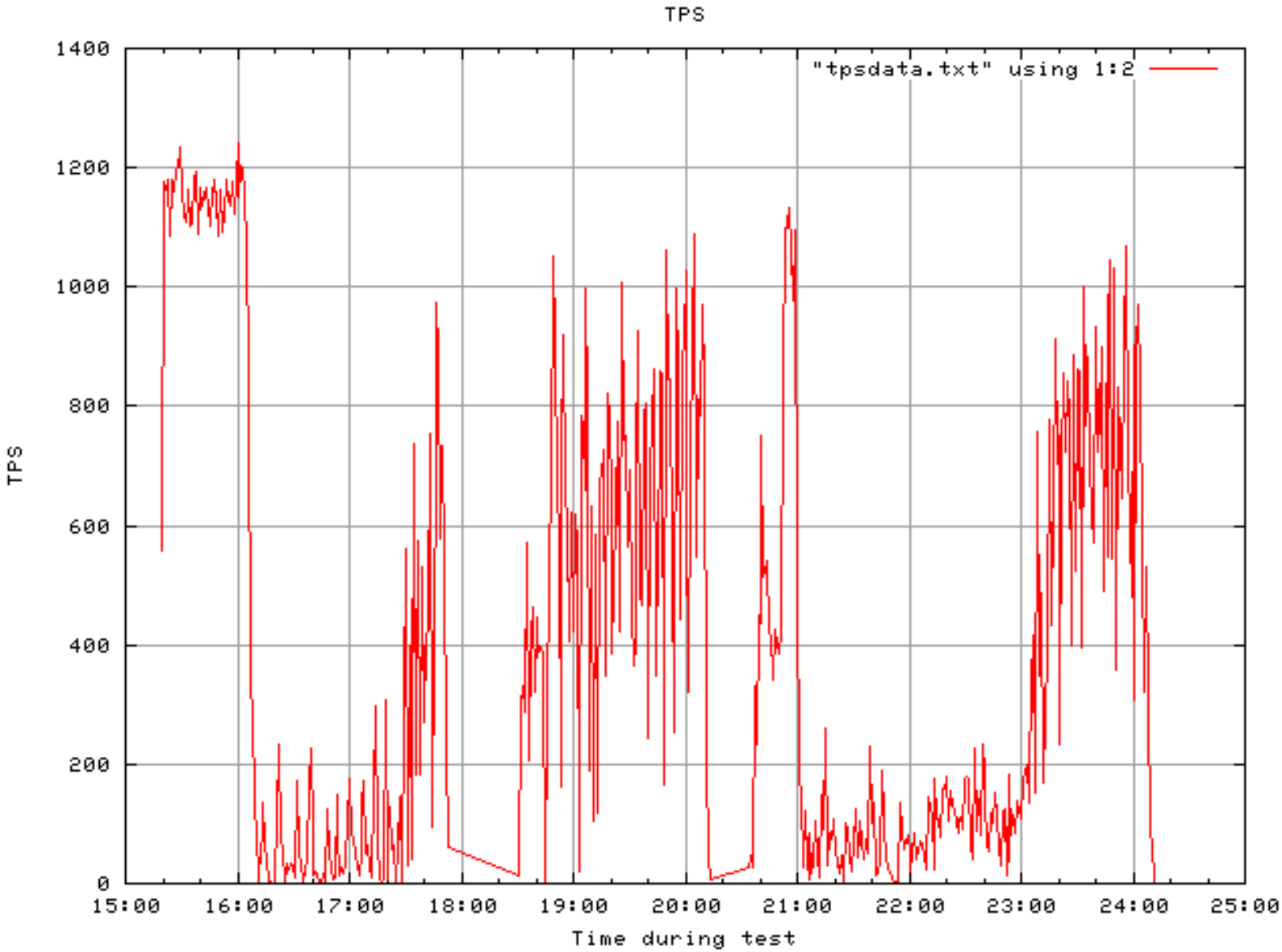
Mediocre result: scale=100, clients=24, tps=1354



Bad result: scale=600, clients=8, tps=226



Surprise! Awful result: scale=600, clients=32, tps=376



Thorough testing takes a long time

```
psql pgbench -c \
  "select sum(end_time - start_time) from tests"
sum
-----------------
53:45:32.383059
```

- ...and even then you may not get it right!
- Need to aim for an equal number of checkpoints

bgwriter stats - low scales

scale	tps	checkpoints	buf_check	buf_alloc
1	2179	0	46	2606
2	2510	0	0	1357
3	2744	0	0	1389
4	2891	0	0	1419
5	2958	0	179	2639
6	3062	0	0	1483
7	3170	0	0	1516
8	3154	0	0	1548
9	3565	0	0	1537

bgwriter stats - medium scales

scale	tps	checkpoints	buf_check	buf_alloc
10	3029	0	548	2701
15	3359	0	0	1704
20	3342	0	0	1841
25	2937	1	28295	2911
30	3385	1	11898	3113
35	2905	2	105911	3052
50	2001	4	178967	6672
75	1509	5	261757	8719
100	1203	6	325074	10742

bgwriter stats - high scales

scale	tps	checkpoints	buf_check	buf_alloc
150	692	5	264999	38239
200	633	5	309983	137743
250	575	6	333119	205253
300	643	11	634055	506451
400	450	5	313166	318942
500	285	6	392441	380465
600	246	6	404912	389897
700	217	6	433745	435527
800	197	7	456626	471071

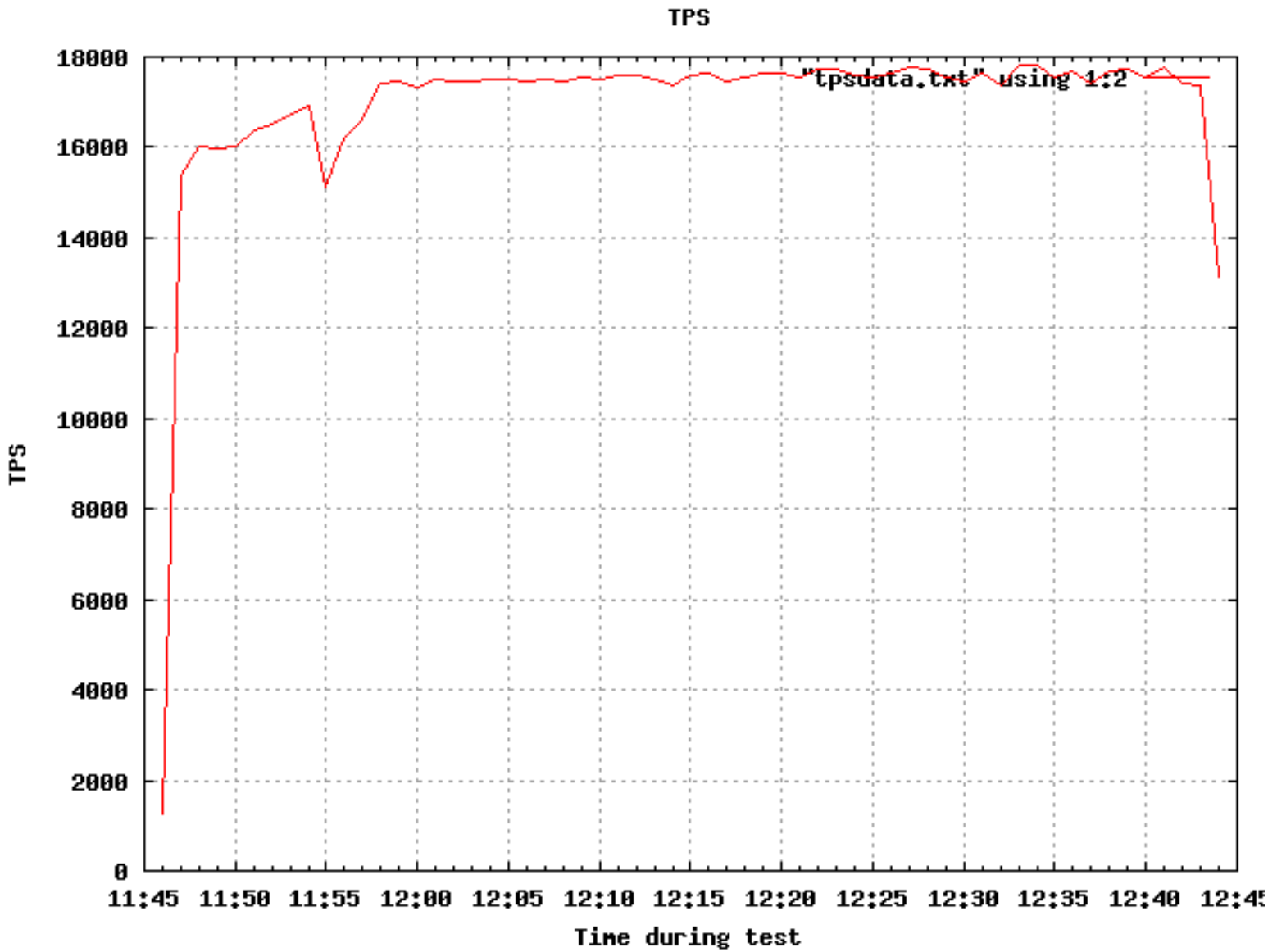
Cold vs. Warm Database Cache

- Results so far have all been warm cache
- Database initialization and benchmark run had no cache clearing in between
- This is fairly real-world once your app has been running for a while
- The situation just after a reboot will be far worse
- Both are interesting benchmark numbers, neither is 'right'

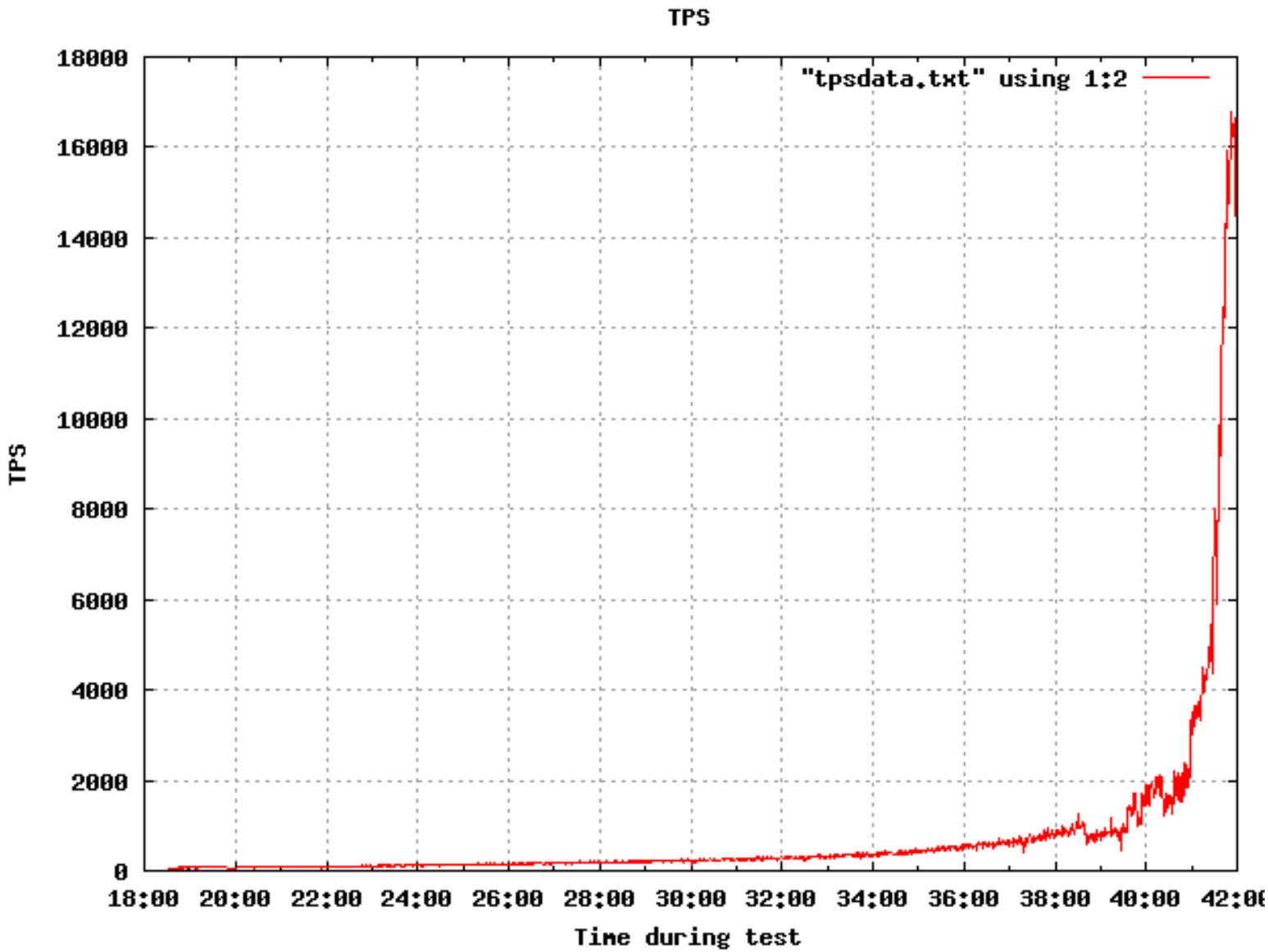
Cold Cache Comparison

- The select-only test can work like a simple test to measure real-world row+index seeks/second over some data size
- Create a pgbench database with the size you want to test normally
- Clear all caches: reboot, on Linux you can use pg_ctl stop plus drop_caches
- Run the select only test, disabling any init/VACUUM steps (SKIPINIT feature)
- Watch bi column in vmstat to see effective seek MB/s
- At the beginning of the test, you'll be fetching a lot of the index to the table along with the table
- That makes for two seeks per row until that's populated.
- Can look at pg_buffercache for more insight into when the index is all in RAM

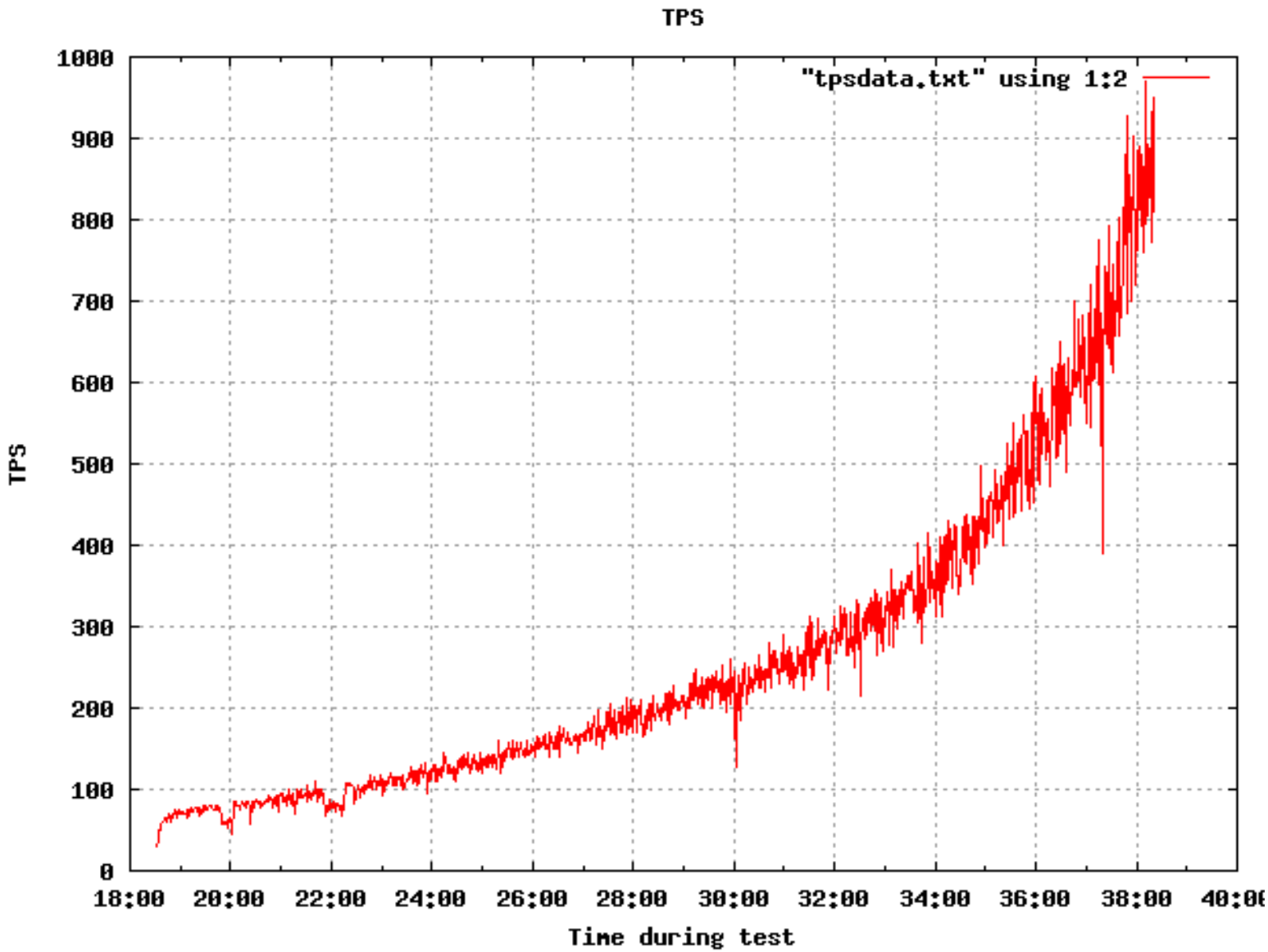
Warm cache: scale=100, clients=4, 1M transactions



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



Zoom on start: low of 37 TPS



Database typical row query

```
select relname,relpages,reltuples,
pg_size_pretty(pg_relation_size(relname::regclass))
 as relsize,
round(pg_relation_size(relname::regclass) / reltuples)
 as "bytes_per_row",
round(8192 / (pg_relation_size(relname::regclass)
 / reltuples)) as rows_per_page,
round(1024*1024 / (pg_relation_size(relname::regclass)
 / reltuples)) as rows_per_mb
from pg_class where relname='accounts';
```

Using seek rate to estimate fill time

reltuples	relsize	bytes_per_row	rows_per_mb
9.99977e+06	1240 MB	130	8064

```
Measured vmstat bo=1.1MB/s
1.1 MB/s * 8064 rows/MB = 8870 rows/second
Estimated cache fill time:
  10M rows / 8870 rows/sec = 1127 seconds = 18 minutes
```

- Database pages/MB are normally (1024*1024)/8192 = 128

Cold Cache Fill Time Measurement

- Had reached near full warm cache speed, 16799 TPS, by the end of the run

```
select end_time - start_time as run_time from tests
```

```
00:23:27.890015
```

- Didn't account for index overhead
- Would have made estimate above even closer to reality

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

Conclusions

- Pay attention to your database scale
- Client scaling may not work as you expect
- Automate your tests and be consistent how you run them
- Results in a database make life easier
- Generating pretty graphs isn't just for marketing
- pgbench can be used for running your specific tests, too