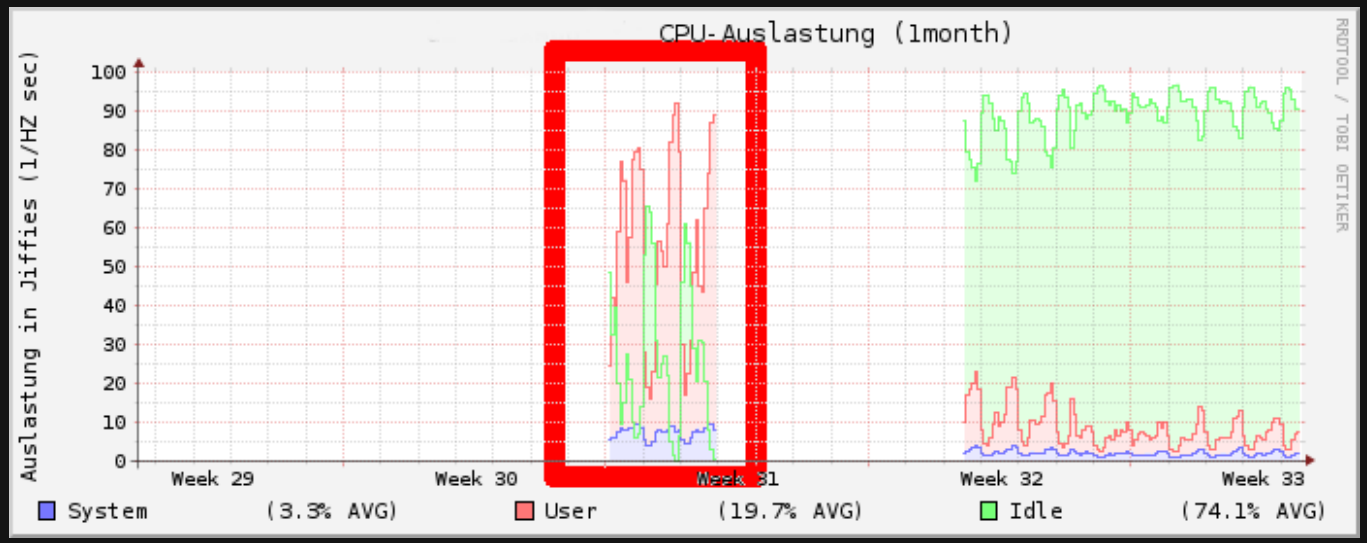
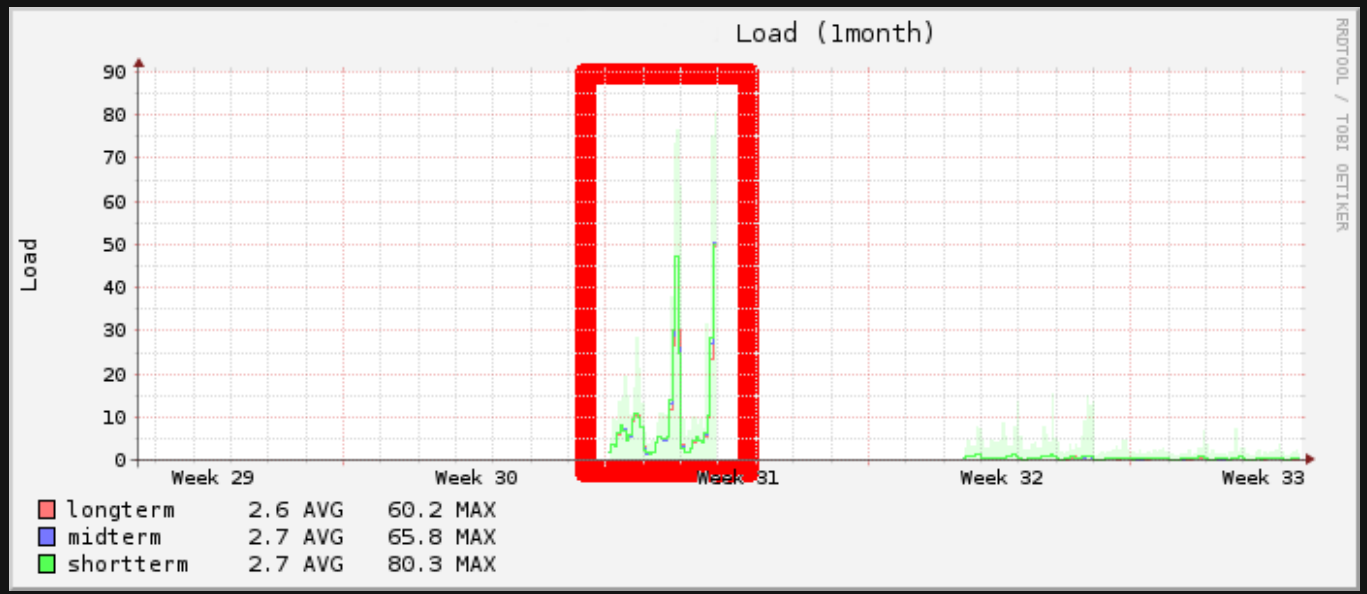
Scaling Apache/PHP/PostgreSQL

sECuRE, 2012-08-16

powered by LuaLaTeX

The issue: server can't cope with the load

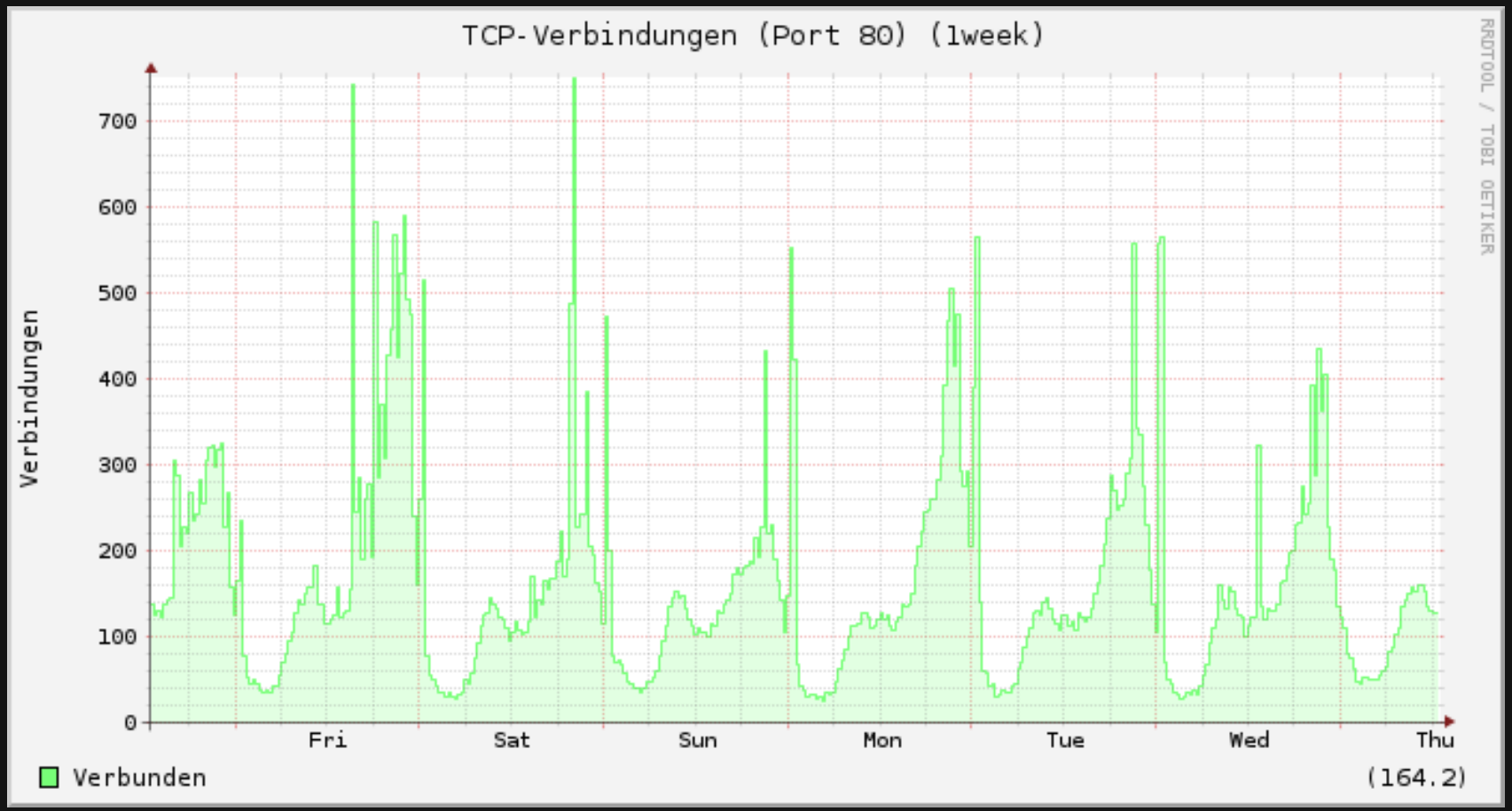




Server specs

- AMD Athlon™ 64 X2 Dual Core Processor 6000+
- 6 GB RAM
- 2 x SAMSUNG HD753LJ (SATA)
- MSI MS-7368
- (Hetzner dedicated)

HTTP load over one week



≈ 164 connections/s on average, peaks at 700 connections/s

Symptom

”Sometimes, loading the page takes a long time. But look, after loading it, it’s fast!”

Rule #1: Gather data!

Measure latency: httping

```
$ httping -c 2 http://www.google.com/
PING www.google.com:80 (http://www.google.com/):
connected to 173.194.69.104:80 (764 bytes), seq=0 time=91.20 ms
connected to 173.194.69.106:80 (764 bytes), seq=1 time=89.37 ms
--- http://www.google.com/ ping statistics ---
2 connects, 2 ok, 0.00% failed
round-trip min/avg/max = 89.4/90.3/91.2 ms
```

Data source: Apache's /server-status

```
Current Time: Thursday, 16-Aug-2012 13:00:22 CEST
Restart Time: Thursday, 16-Aug-2012 00:00:05 CEST
Parent Server Generation: 0
Server uptime: 13 hours 17 seconds
Total accesses: 1712126 - Total Traffic: 254.2 MB
CPU Usage: u40.34 s10.56 cu0 cs0 - .109% CPU load
36.6 requests/sec - 5.6 kB/second - 155 B/request
34 requests currently being processed, 14 idle workers

KK.W__W_CK_K.KKK.W__K.KKW._...KK.K._.._.KKW...W.._.K..W._.K.._..
..W...WK_....W...._...WK..W....KK..............................
............................................................
............................................................

Scoreboard Key:
"_" Waiting for Connection, "S" Starting up, "R" Reading Request,
"W" Sending Reply, "K" Keepalive (read), "D" DNS Lookup,
"C" Closing connection, "L" Logging, "G" Gracefully finishing,
"I" Idle cleanup of worker, "." Open slot with no current process
```

Apache worker configuration

```
KeepAliveTimeout 10

<IfModule mpm_prefork_module>
    StartServers          5
    MinSpareServers       5
    MaxSpareServers      10

    MaxClients          250
    MaxRequestsPerChild 150
</IfModule>
```

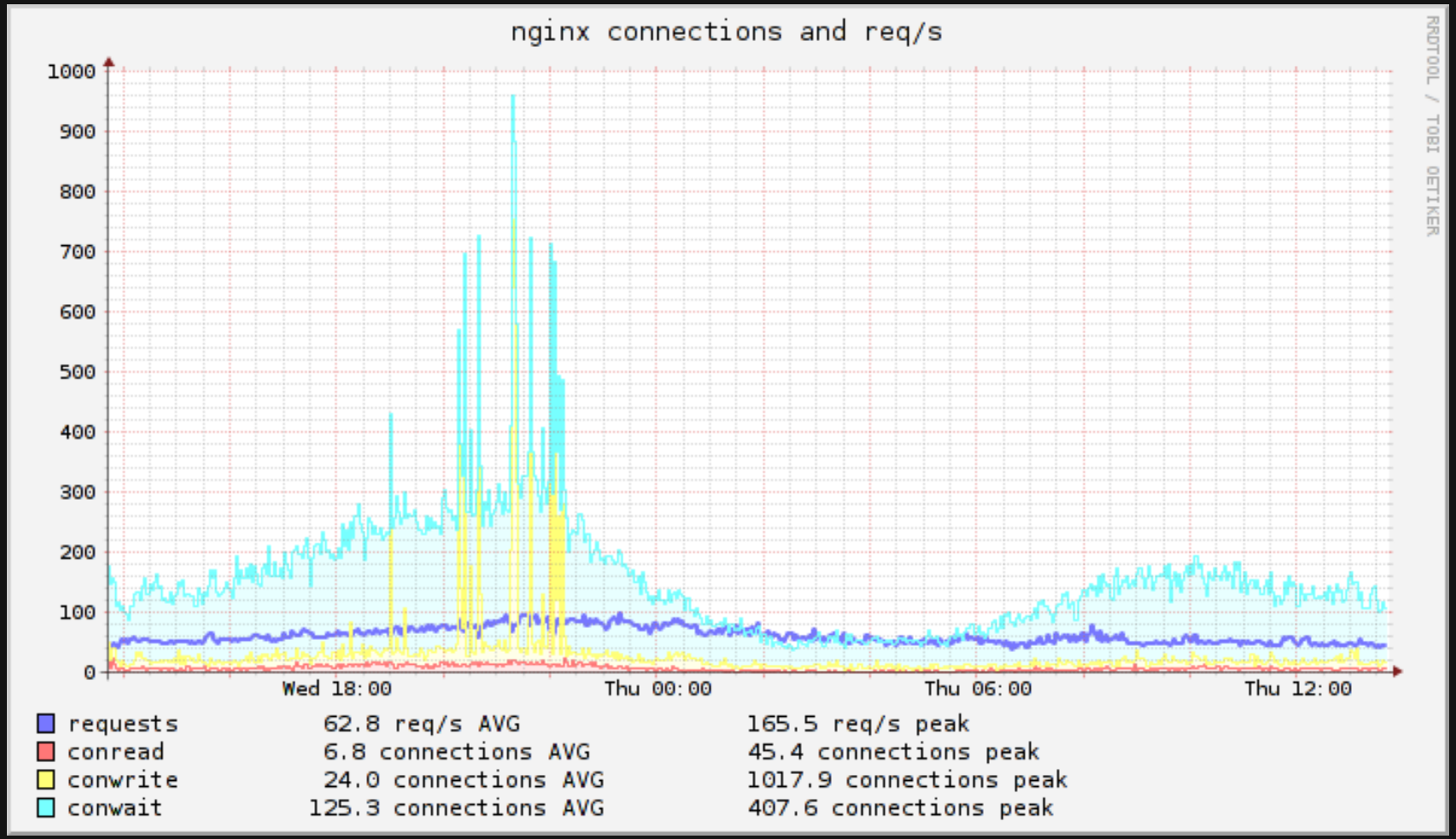
250 MaxClients not enough — now what?

- Try mpm_event? Still experimental in Apache 2.2.
- Use nginx to terminate HTTP connections.
- Cache the hell out of it.

nginx to the rescue

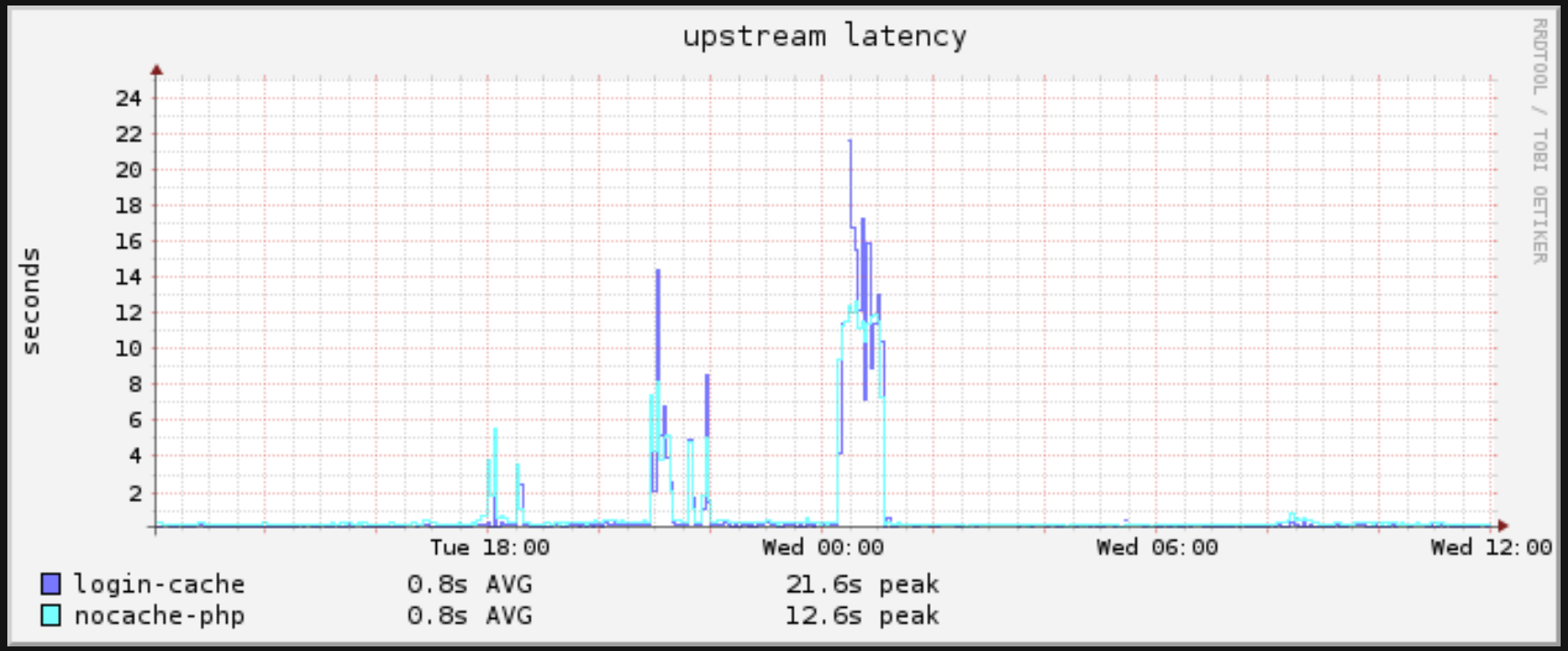
- `http://michael.stapelberg.de/Artikel/rescuing_webservers_with_nginx_as_cache`
- tl;dr: rewrite URLs to make them cachable
- force-cache dynamic replies for 10 minutes
- cache static assets (doh)
- test the setup with iptables
- monitor with collectd + nginx server-status

nginx connections and req/s

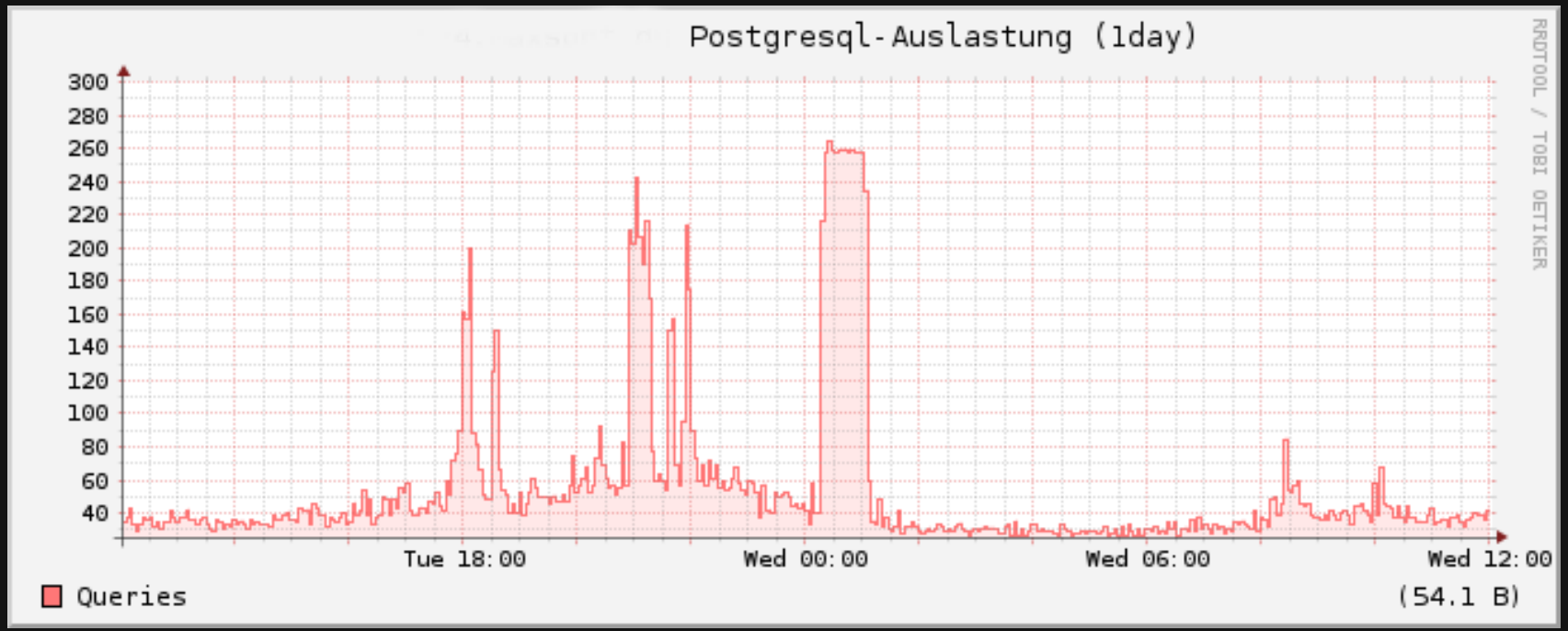


60-100 req/s, conwait implies keep-alive client

more interesting: nginx upstream latency



Still spikes in PostgreSQL active queries



postgresql.conf: `max_connections = 260`

Rule #2: Gather data!

postgresql.conf

```
# Log queries slower than 2s
log_min_duration_statement = 2000
log_duration = on

# Log all locks waiting longer than 1s
log_lock_waits = on
deadlock_timeout = 1s
```

postgresql-9.1-main.log

```
139707.459 ms  statement: UPDATE connections SET incoming='1923245811',outgoing='
     50351094',last_updated=NOW() WHERE conn_id='10795036'
146476.844 ms  statement: UPDATE connections SET incoming='33090205',outgoing='
     4312739',last_updated=NOW() WHERE conn_id='10798486'
195004.411 ms  statement: UPDATE connections SET token_status = 'USED',
     timestamp_out = last_updated WHERE token_status = 'INUSE' AND last_updated <
      '2012-08-15 00:20:01'::timestamp;
195949.991 ms  statement: UPDATE connections SET token_status = 'USED',
     timestamp_out = last_updated WHERE token_status = 'INUSE' AND timestamp_in <
      '2012-08-14 12:55:43'::timestamp;
235172.722 ms  statement: BEGIN TRANSACTION;
237125.822 ms  statement: UPDATE connections SET token_status = 'USED',
     timestamp_out = last_updated WHERE token_status = 'INUSE' AND last_updated <
      '2012-08-15 00:25:01'::timestamp;
242371.885 ms  statement: UPDATE connections SET token_status = 'USED',
     timestamp_out = last_updated WHERE token_status = 'INUSE' AND last_updated <
      '2012-08-15 00:45:01'::timestamp;
337313.517 ms  statement: UPDATE total SET online = 959
565424.327 ms  statement: DELETE FROM users WHERE account_status = 5 AND reg_date
      < '2012-08-13'::timestamp
1749510.255 ms  statement: DELETE FROM connections WHERE last_updated < '
     2012-08-08'::timestamp
```

Filtering and sorting

```
grep '2012-08-15 00:' \
  /var/log/postgresql/postgresql-9.1-main.log | \
  perl -nlE 's/.*  duration: //g; say $_' | \
  sort -n | \
  less
```

Query optimization (1)

```
DELETE FROM users WHERE account_status = 5 AND (
    now()-reg_date) > '2 days'::interval;
```

```
# EXPLAIN ANALYZE SELECT * FROM users WHERE account_status = 5 AND (now()-
      reg_date) > '2 days'::interval;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------

 Seq Scan on users  (cost=0.00..7401.34 rows=1246 width=208) (actual time
       =117.051..120.133 rows=3822 loops=1)
   Filter: ((account_status = 5) AND ((now() - (reg_date)::timestamp with time
         zone) > '2 days'::interval))
 Total runtime: 120.296 ms
(3 rows)
```

Query optimization (2)

```
DELETE FROM users WHERE account_status = 5 AND
    reg_date < '2012-08-16'::timestamp;
```

```
# EXPLAIN ANALYZE SELECT * FROM users WHERE account_status = 5 AND reg_date <
     '2012-08-16'::timestamp;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------

 Seq Scan on users  (cost=0.00..6201.89 rows=3737 width=208) (actual time
      =30.563..32.019 rows=3822 loops=1)
   Filter: ((reg_date < '2012-08-16 00:00:00'::timestamp without time zone) AND (
        account_status = 5))
 Total runtime: 32.154 ms
(3 rows)
```

Query optimization (3)

```
CREATE INDEX CONCURRENTLY idx_reg_date_status ON
    users (account_status, reg_date DESC);
```

```
# EXPLAIN ANALYZE SELECT * FROM users WHERE account_status = 5 AND reg_date <
     '2012-08-16'::timestamp;
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------
 Bitmap Heap Scan on users  (cost=98.58..4075.83 rows=3737 width=208) (actual
      time=0.634..1.599 rows=3822 loops=1)
   Recheck Cond: ((account_status = 5) AND (reg_date < '2012-08-16 00:00:00'::
        timestamp without time zone))
   ->  Bitmap Index Scan on idx_reg_date_status  (cost=0.00..97.64 rows=3737
        width=0) (actual time=0.596..0.596 rows=3822 loops=1)
         Index Cond: ((account_status = 5) AND (reg_date < '2012-08-16
              00:00:00'::timestamp without time zone))
 Total runtime: 1.862 ms
(5 rows)
```

Query optimization (4)

```
DELETE FROM users WHERE account_status = 5 AND
    reg_date < '2012-08-16 14:16:00'::timestamp;
```

$\Rightarrow$ Delete smaller amounts of data, but more often.

Lather, Rinse, Repeat

- 10 Watch for spikes
- 20 Identify slow quer{y,ies}
- 30 Optimize
- 40 `GOTO 10`

Lock optimization (1)

```
2012-08-12 21:00:18 CEST LOG:  duration: 7748.305 ms
    statement: UPDATE total SET online = 1277

2012-08-12 21:00:18 CEST LOG:  process 10968 acquired
    ShareLock on transaction 24671646 after 7169.261 ms

2012-08-12 21:00:18 CEST CONTEXT:  SQL statement "UPDATE
    total SET members = members+1"
        PL/pgSQL function "adj_totalusers" line 4 at SQL
            statement
```

Lock optimization (2)

```
postgres# \d users
                                  Table "public.users"
       Column         Type                 Modifiers
--------------------+----------------------------+-------------------------------------

 user_id             character varying(45)  not null
Triggers:
    adjtotalusers AFTER INSERT OR DELETE ON users FOR EACH ROW EXECUTE PROCEDURE
         adj_totalusers()


postgres# select prosrc from pg_proc where proname='adj_totalusers';
                 prosrc
-----------------------------------------
                                        +
 BEGIN                                  +
   IF TG_OP = 'INSERT' THEN             +
     UPDATE total SET members = members+1;+
     RETURN NEW;                        +
   END IF;                              +
   IF TG_OP = 'DELETE' THEN             +
     UPDATE total SET members = members-1;+
     RETURN OLD;                        +
   END IF;                              +
 END;                                   +
```

postgresql.conf: write buffers

Debian default config:

```
shared_buffers = 24MB
wal_buffers = -1
# -1 means shared_buffers/32 == 768 KB
```

postgresql.conf: write buffers

Debian default config:

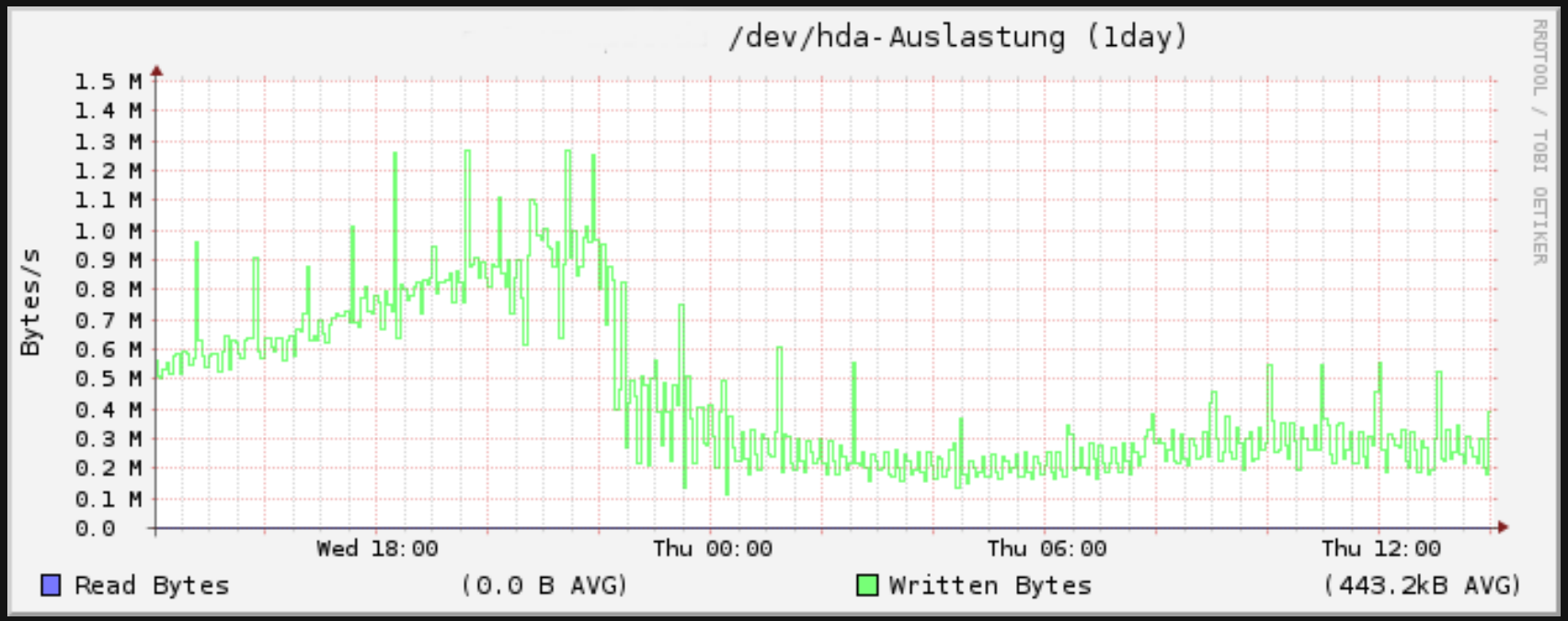
```
shared_buffers = 24MB
wal_buffers = -1
# -1 means shared_buffers/32 == 768 KB
```

Better:

```
# pg manual recommends 25% of system memory
shared_buffers = 2000MB
# one WAL segment is 16 MB
wal_buffers = 16MB
```

Warning: configure `kernel.shmmax` and `kernel.shmall`:
http://archives.postgresql.org/pgsql-admin/2010-05/msg00285.php

Disk utilization



after increasing `wal_buffers`

PHP: reduce database load

```
// With a 50% chance, nothing happens.
// Timeout for dead node detection is 5 min.
// Every node sends a heartbeat every 1 min.
// This is still good enough.
if (rand(0, 1) == 1) {
    $db->execSqlUpdate("UPDATE nodes SET
       last_heartbeat_timestamp=NOW() WHERE
       node_id='$node_id'");
}
```

TCP connections

Make the kernel reuse closed connections (`sysctl.conf`):

```
# Reuse TIME_WAIT connections
net.ipv4.tcp_tw_reuse=1
net.ipv4.tcp_fin_timeout=30
```

Fix collectd to use the netlink API, otherwise it takes 2.5s for measuring the tcp connections:
`https://github.com/collectd/collectd/pull/113`

URLs

- http://pgfouine.projects.postgresql.org/tutorial.html
- http://www.postgresql.org/docs/9.1/static/runtime-config-resource.html
- http://www.postgresql.org/docs/9.1/static/runtime-config-wal.html
- http://collectd.org/
- http://bethesignal.org/blog/2009/07/22/watching-nginx-upstreams-with-collectd/