
PostgreSQL @ TOMTOM – lessons learned

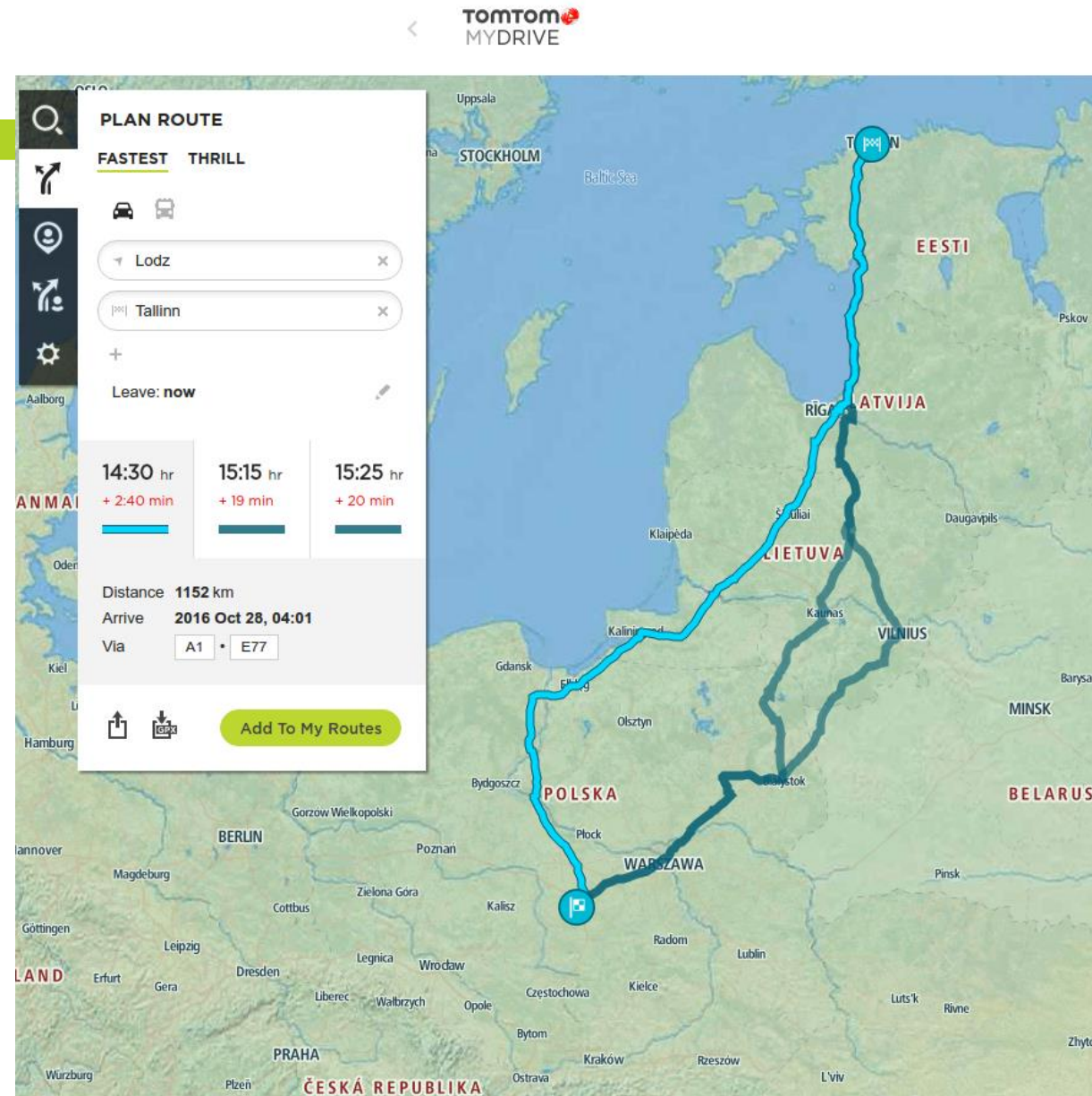
pgconf.eu 2016

About us

Rafał Hawrylak
rafal.hawrylak@tomtom.com
Software hacker and database expert

Michał Gutkowski
michal.gutkowski@tomtom.com
Software engineer
solving problems with Java, SQL, Python, Bash

We are from Łódź, Poland!



TomTom – What do we do?

The New TomTom Go



Motorcycle



Camper & Caravan

The New TomTom VIO



Mobile



Business to Business

TOMTOM SPORTS



TOUCH
FITNESS TRACKERS



SPARK 3
FITNESS WATCHES



GOLFER 2
GOLF WATCHES



RUNNER 3
RUNNING WATCHES



ADVENTURER
OUTDOOR WATCHES



BANDIT
ACTION CAMS

GET FIT

GET BETTER

GET OUT THERE

TomTom - What do we do?

- ✓ Database with spatial features
- ✓ Massive automated tools
- ✓ 2000+ of manual editors
- ✓ Billions of map objects



Map Making Platform

- ✓ Database Machines – 200+ machines (40 cores, 256GB RAM, RAID 10 ssd drives)
- ✓ Queries count – over 600k per second
- ✓ Inserted rows count 15k per second
- ✓ Storage – 30TB
- ✓ Daily db size increase – 200GB
- ✓ Reads : Writes ~ 100 : 1

PostgreSQL + PostGIS

- ✓ Out-of-the-box extension for geometry types
- ✓ Processing and analytic functions
- ✓ Spatial predicates: intersects, covers, covered by, inside
- ✓ Spatial GIST index on geometry (based on bounding boxes)
- ✓ Hint: spatial queries faster on simplified geometry



Query optimization: it is all about indexes

- ✓ Run analyze to update pg_statistics
 - “ALTER TABLE SET STATISTICS to 1000” for large tables
- ✓ Benchmark queries on production data using “explain analyze”
- ✓ Changing parameters may completely change query plan
- ✓ Indexes are not for free - increases disk size and row insert time
- ✓ Multicolumn Indexes - order of columns in B-tree index definition does matter for query

Customer table

name	order_date	type
John	2016-09-01	Paid
Alan	2016-09-02	Unpaid

idx_customer (name, order_date)

idx_type (type, order_date)

check out:

<https://explain.depesz.com>

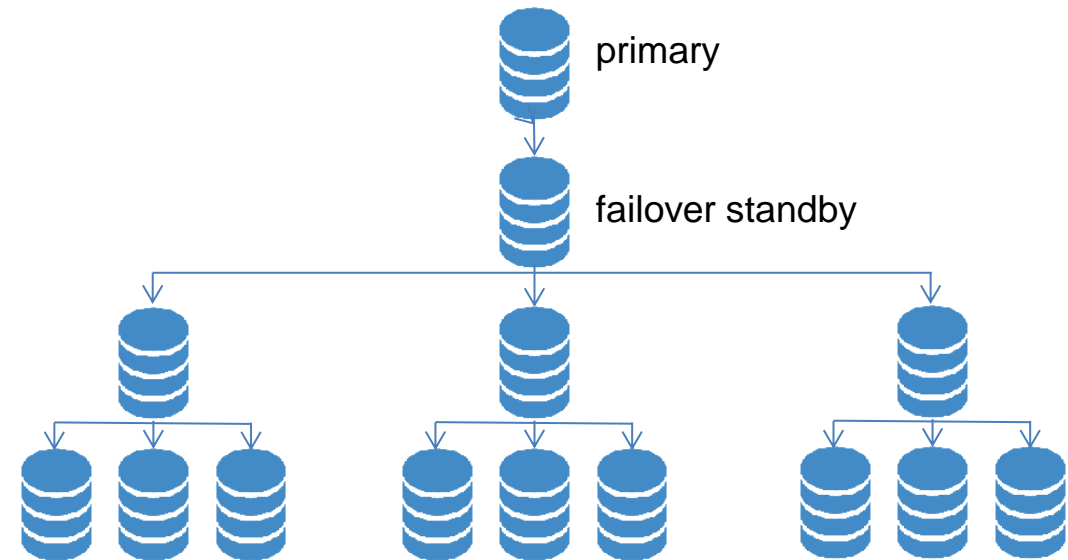
<http://use-the-index-luke.com/>

Query optimization: index bloat

- ✓ Check bloat on indexes: updates, inserts, deletes causes increased latency of query execution time and increases size of indexes
- ✓ Remove not used indexes (pg_stat_all_indexes view shows usage)
- ✓ Requires “reindex” (locks entire table for writes) or re-creating an index

Replication: streaming replication

- ✓ Streaming vs logical
- ✓ Replication tree – failover
- ✓ Application is prechecking if data is available on standby
- ✓ Standby in physical replication is read-only
- ✓ Initial copy: rsync vs pg_basebackup.



Check out:

https://wiki.postgresql.org/wiki/Streaming_Replication

Replication: replication lag

- ✓ High CPU/IO load
- ✓ Check with query: "SELECT now() - pg_last_xact_replay_timestamp()" or in pg_stat_replication view
- ✓ Graphite feed from nagios plugin
- ✓ Separate volumes for data, xlog, logs, pg_stats
- ✓ Use WALs compression for slow network – since 9.5 (increased CPU usage both on master and standby)
- ✓ Configuration tuning:

wal_keep_segments = 330000 # 5TB of WALs

max_wal_senders = 10

Replication: replication vs checkpoints

✓ Checkpoint configuration tuning:

checkpoint_completion_target = 0.9

checkpoint_timeout = 1h

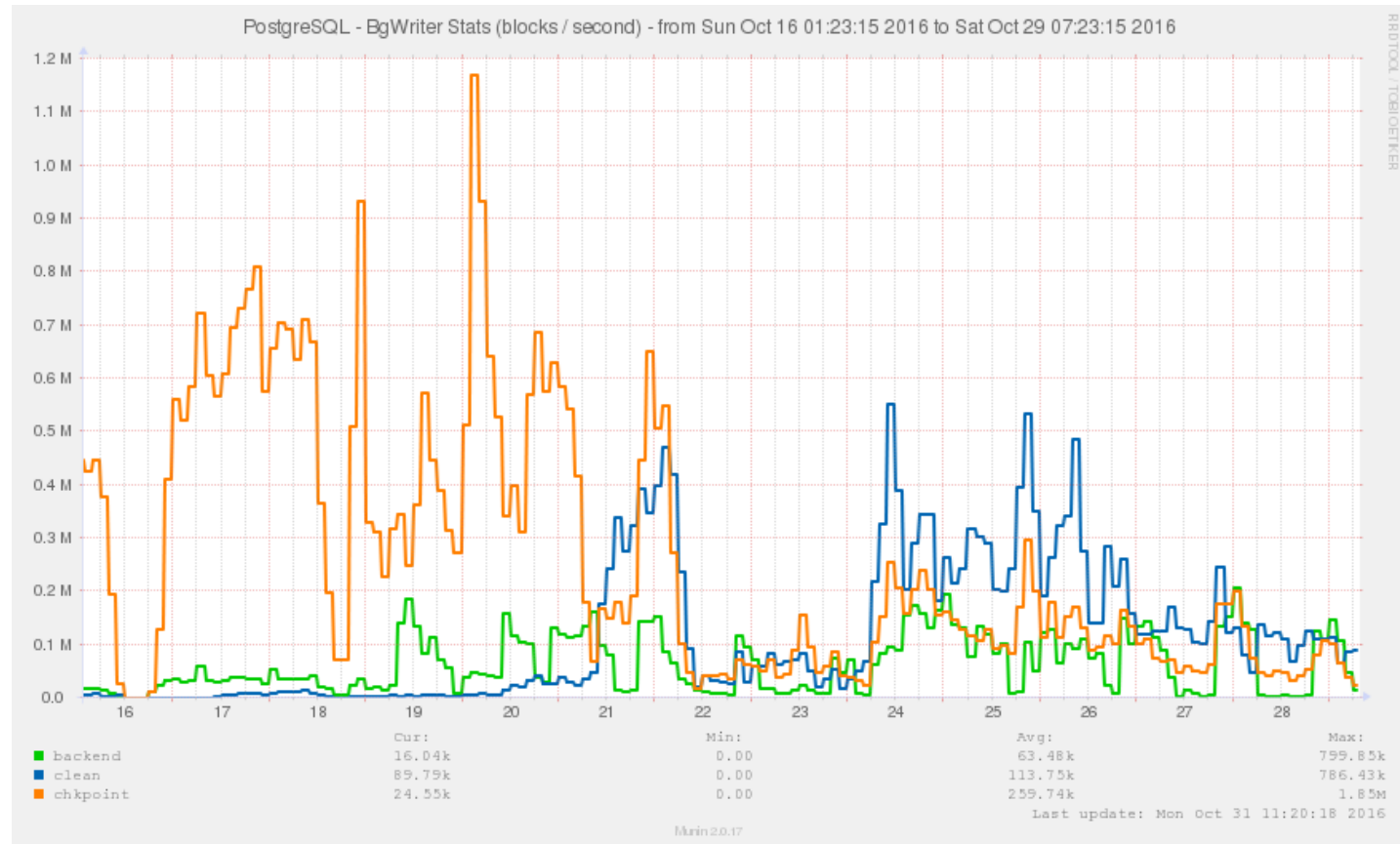
checkpoint_warning = 30s

max_wal_size = 100GB

min_wal_size = 1GB

bgwriter_delay = 50ms

bgwriter_lru_maxpages = 2000



Database High Availability

What we learned you cannot do in live production system:

- ✓ Vacuum full -> well tuned autovacuum instead (more WALs)
- ✓ Create index -> Create index concurrently (takes longer)
- ✓ Reindex -> Create copy of index and switch with original (needs extra disk space)

DDL under control

- ✓ Versioning and tracking database layout with Liquibase
- ✓ Changes tested on pre-production environments – it lowers risk of human error and inconsistencies
- ✓ Rollback logic

```
<changeSet id="journaltransactions_index" author="TomTom">
  <createIndex tableName="journaltransactions"
    indexName="journal_version_index" schemaName="${schemaName}">
    <column name="txn_version" />
  </createIndex>
  <sql>
    CREATE INDEX CONCURRENTLY journal_geometry_extremes_index
    ON
    ${schemaName}.journaltransactions
    USING gist(txn_geometry_extremes)
  </sql>
  <rollback>
    <sql>
      DROP INDEX CONCURRENTLY ${schemaName}.journal_geometry_extremes_index;
    </sql>
  </rollback>
</changeSet>
```

Query optimization - monitoring

- ✓ Use statistics collector

`pg_stat_activity`, `pg_stats_statements`

- ✓ Configuration in `postgresql.conf`

`log_min_duration_statement = 20000`

`shared_preload_libraries = 'pg_stat_statements,auto_explain'`

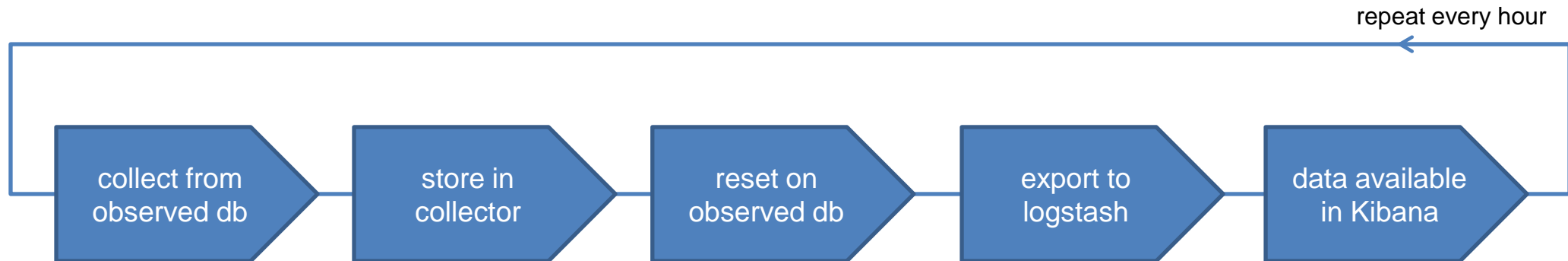
`auto_explain.log_min_duration = 20000`

`auto_explain.log_analyze = true;` <--- can be expensive to run query twice



Monitoring: pg_stat_statements

- ✓ gathers a bunch of useful statistics of query execution
- ✓ the best way to track lots of short queries
- ✓ one cumulative sack
- ✓ not usable if you need track query behavior changes



ElastAlert <https://github.com/Yelp/elastalert>

Monitoring: stat_statements in Kibana

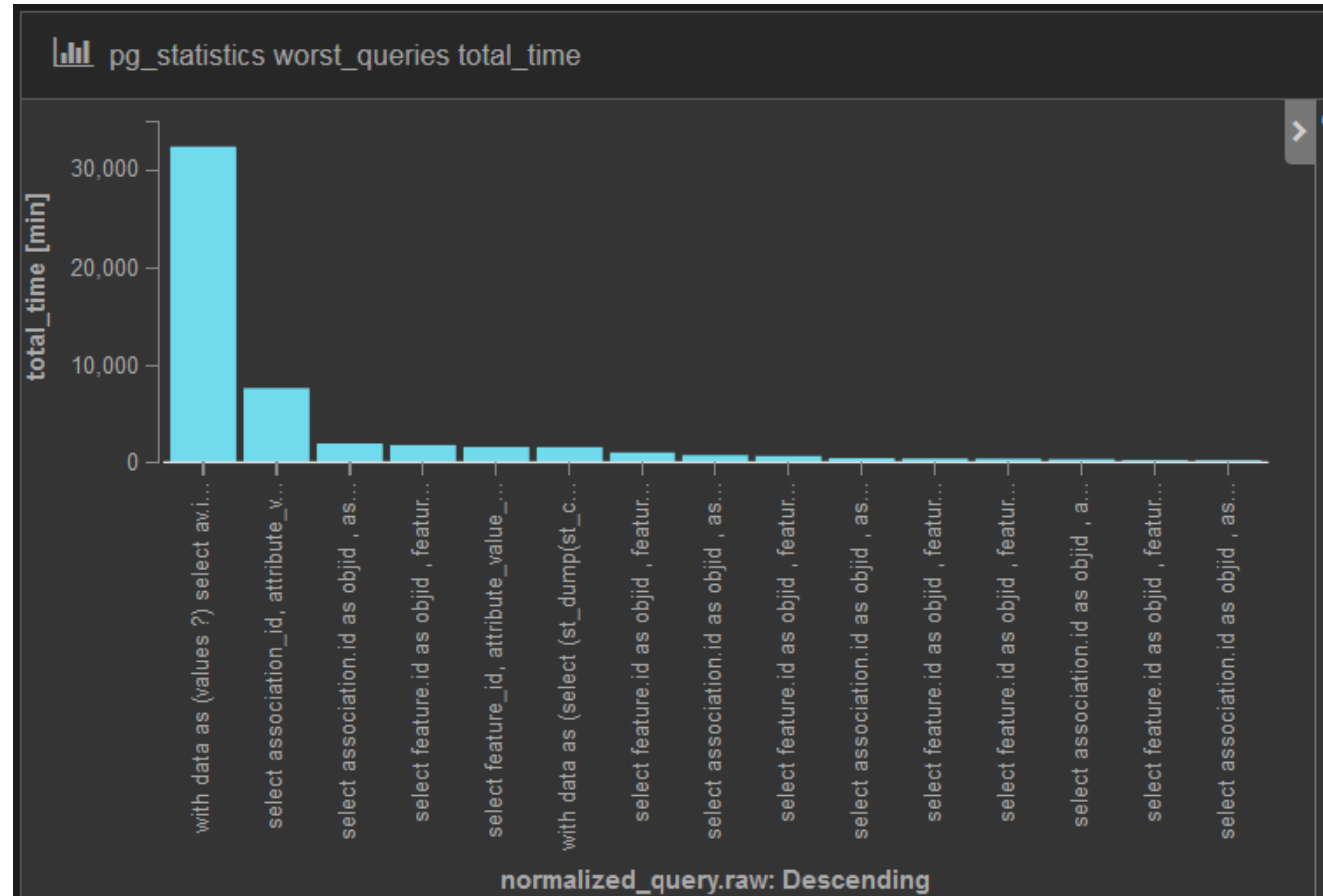
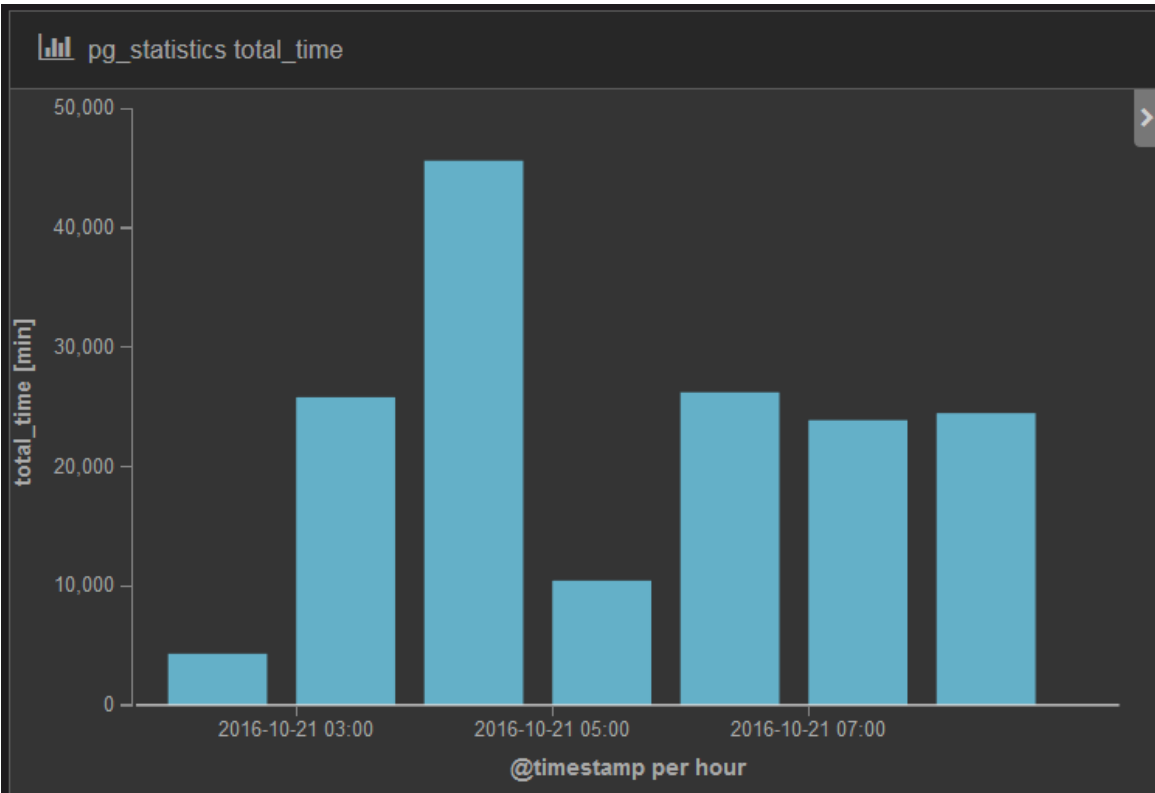
In Kibana, we can easily observe for each particular statement on each and every machine separately (if we want to):

- ✓ total execution time
- ✓ cpu execution time
- ✓ io execution time
- ✓ number of calls
- ✓ number of rows returned / affected
- ✓ average execution time
- ✓ average cpu execution time
- ✓ average io execution time
- ✓ average number of calls
- ✓ average number of rows returned / affected

In terms of:

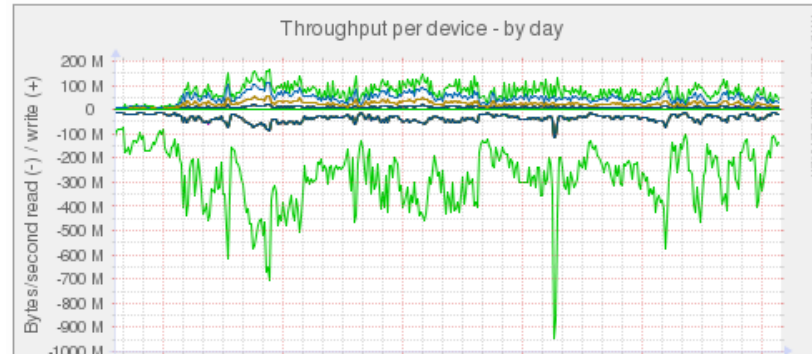
- historical data
- trends
- behaviour changes
- the heaviest query
- distribution

Monitoring: stat_statements in Kibana

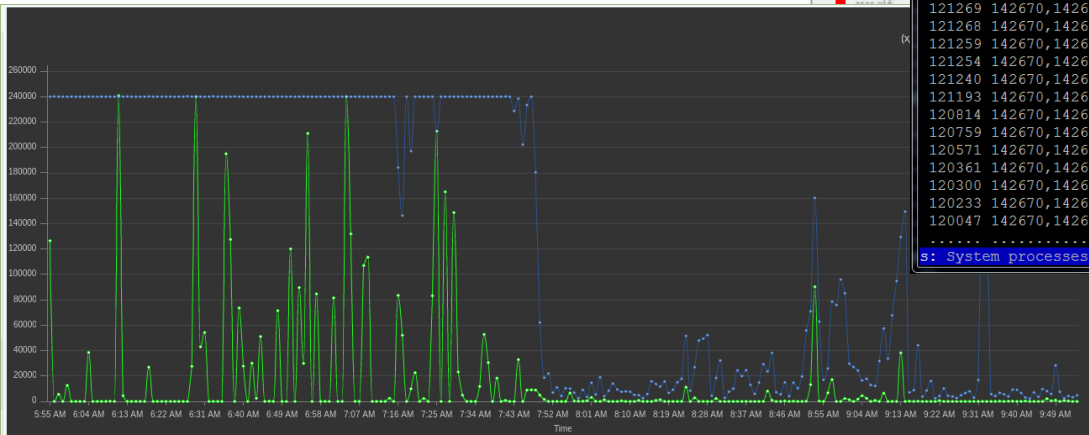


Monitoring: other tools we use

- ✓ Munin
- ✓ System/postgresql statistics
- ✓ AWS CloudWatch
- ✓ AppDynamics for performance
- ✓ Unix tools: htop / iotop
- ✓ pg_view



```
root@rprod-cpp-pgmdsproc-r1-001/nethomes/kaczmaew
rprod-cpp-pgmdsproc-r1-001.flatns.net up 318 days, 9:02:44 40 cores Linux 2.6.32-504.12.2.el6.x86_64 load average 7.01 9.27 13.92
sys: utime 5.0 stime 3.6 idle 90.8 iowait 0.6 cctx 13562 run 4 block 0
mem: total 125.9GB free 2.1GB buffers 21.0MB cached 74.7GB dirty 2.4MB limit 129.3GB as 62.3GB left 67.0GB
/var/lib/pgsql/9.4/data/mds_proc 9.4 master database connections: 746 of 5000 allocated, 107 active
type dev fill total left read write await path size path
data dm-5 1.0 4.8TB 42.4GB 23.8 0.0 467.8 4.5TB /var/lib/pgsql/9.4/data/mds_proc
xlog dm-5 0.0 4.8TB 42.4GB 23.8 0.0 467.8 9.7GB /var/lib/pgsql/9.4/data/mds_proc/pg_xlog
pid lock type s utime stime guest read write age db user query
142670 backend R 99.5 0.0 0.0 3.7 0.9 40:53 statistics postgres VACUUM FULL public.statio user tables;
121269 142670,142670 backend S 0.0 0.0 0.0 0.0 0.0 36:59 statistics stat_collector INSERT INTO public.stat_user_tables SELECT 'gacheckcore2-125.service.eu-west-...
121268 142670,142670 backend S 0.0 0.0 0.0 0.0 0.0 36:59 statistics stat_collector INSERT INTO public.stat_user_tables SELECT 'gacheckcore2-161.service.eu-west-...
121259 142670,142670 backend S 0.0 0.0 0.0 0.0 0.0 36:59 statistics stat_collector INSERT INTO public.stat_user_tables SELECT 'gacheckcore2-1512.service.eu-west-...
121254 142670,142670 backend S 0.0 0.0 0.0 0.0 0.0 36:59 statistics stat_collector INSERT INTO public.stat_user_tables SELECT 'gacheckcore2-133.service.eu-west-...
121240 142670,142670 backend S 0.0 0.0 0.0 0.0 0.0 36:59 statistics stat_collector INSERT INTO public.stat_user_tables SELECT 'gacheckcore2-134.service.eu-west-...
121193 142670,142670 backend S 0.0 0.0 0.0 0.0 0.0 36:59 statistics stat_collector INSERT INTO public.stat_user_tables SELECT 'gacheckcore2-115.service.eu-west-...
120814 142670,142670 backend S 0.0 0.0 0.0 0.0 0.0 36:59 statistics stat_collector INSERT INTO public.stat_user_tables SELECT 'gacheckcore2-1511.service.eu-west-...
120759 142670,142670 backend S 0.0 0.0 0.0 0.0 0.0 36:59 statistics stat_collector INSERT INTO public.stat_user_tables SELECT 'gacheckcore2-1513.service.eu-west-...
120571 142670,142670 backend S 0.0 0.0 0.0 0.0 0.0 36:59 statistics stat_collector INSERT INTO public.stat_user_tables SELECT 'gacheckcore2-1521.service.eu-west-...
120361 142670,142670 backend S 0.0 0.0 0.0 0.0 0.0 36:59 statistics stat_collector INSERT INTO public.stat_user_tables SELECT 'gacheckcore2-172.service.eu-west-...
120300 142670,142670 backend S 0.0 0.0 0.0 0.0 0.0 36:59 statistics stat_collector INSERT INTO public.stat_user_tables SELECT 'gacheckcore2-114.service.eu-west-...
120233 142670,142670 backend S 0.0 0.0 0.0 0.0 0.0 36:59 statistics stat_collector INSERT INTO public.stat_user_tables SELECT 'rprod-cpp-r2-coresup-slave', now(...
120047 142670,142670 backend S 0.0 0.0 0.0 0.0 0.0 36:59 statistics stat_collector INSERT INTO public.stat_user_tables SELECT 'gacheckcore2-124.service.eu-west-...
s: System processes f: Freeze output u: Measurement units a: Autohide fields t: No trim r: Realtime h: Help v.1.2.0
```



Questions?



We are hiring!