Benchmarking Databases

PGcon 2016

Jan Wieck

OpenSCG

Introduction

- UsedPostgres since version 4.2 (University Postgres)
- Joined PostgreSQL community around 1995.
- Contributed rewrite rule system fix, TOAST, procedural language handler, PL/Tcl, PL/pgSQL, foreign keys, background writer, lazy vacuum, statistics collector, NUMERIC and some other sources of headache.
- Designed and implemented Slony-I.
- Core team member from 2000 to 2010.

Why people mark benches

- Mirror mirror on the wall … ?
- Is the next version faster?
- How far does my current hardware scale?
- Rightsizing
- Tuning

The best thing to test with is YOUR application

- No “standard” benchmark will match the unique database access pattern of your application.
- Does your application have a “performance” test harness?
- Running your functional regression test harness a million times doesn’t make a performance test.

What are the criteria when looking for a benchmark?

- Database/table size and access distribution
- Database size to TPS ratio
- Complexity of database transactions
- Transaction type mix
- Similarity to your application

So far, so good

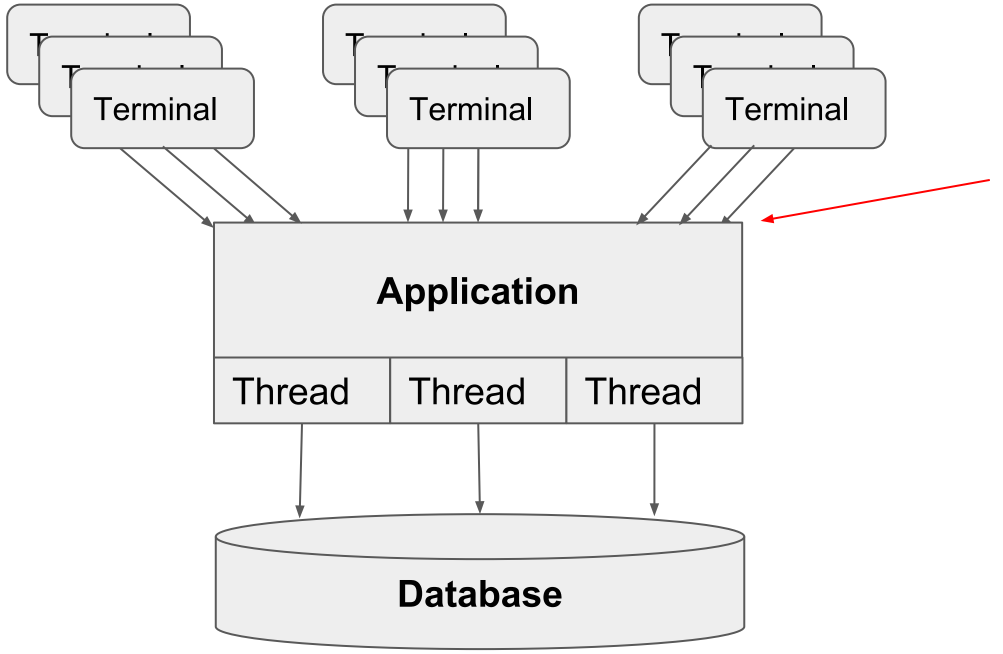
So let’s assume you decided that TPC/C is a good “enough” approximation of your workload.

Now you are shopping for the right TPC/C implementation. To find the right one it would be nice to know how they all differ from the standard.

The TPC/C implementations examined

- DBT2
  - Proper TPC/C implementation (AFAICT) but unfortunately the thread model doesn’t scale to modern machine sizes.
- HammerDB 2.16
  - Version 2.19 had a little surprise - the database size doubled since 2.16.
  - No rate limiting or proper timing.
- BenchmarkSQL 5.0rc2
  - No proper timing (yet) but at least rate limiting.

The model of the TPC/C



Terminals
10 * Warehouse

Response time measurement

Application with
Database connections

DBT2

- Is capable of proper timing and response time measurement.
- Unfortunately needs one pthread per Terminal.
- A pthread needs (by default) 10MB stack.
- At today's scaling factors the number of threads means excessive memory needs and overwhelms the scheduler.

http://osdldbt.sourceforge.net

HammerDB

- HammerDB is a Tcl/Tk GUI application
- Agglutinates the Terminal and an Application thread.
- The lack of actual terminals does not allow for the proper response time measurement as per TPC/C specs.
- HammerDB has no options to regulate the transaction mix or rate limiting.

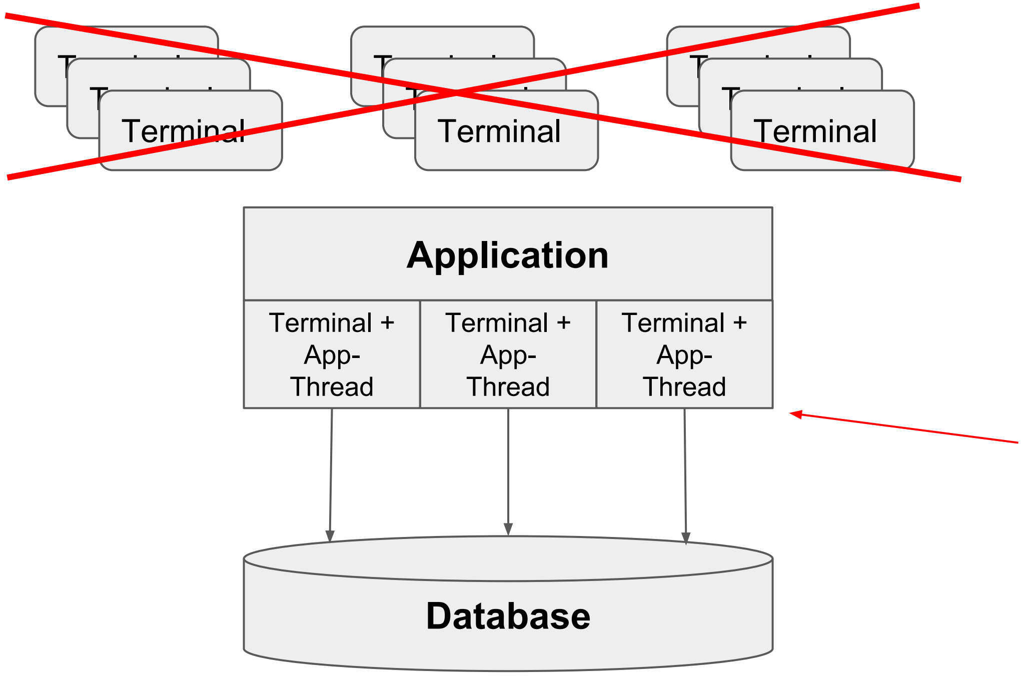
http://www.hammerdb.com

BenchmarkSQL

- Fork of jTPCC
- Got a major overhaul in version 5.0 (currently in RC)
- Command line interface
- Agglutinates the Terminal with the Application thread.
- Does allow controlling the transaction mix and rate limit.
- Disclamer: I am a maintainer of that project.

https://bitbucket.org/openscg/benchmarksql/overview

Not the model of the TPC/C



Terminals
10 * Warehouse

Application with
Database connections

Response time of what?

Scaling of the test

- The scaling should reflect your use case.
- TPC/C is scaled by the number of Warehouses
- One Warehouse is approximately 110MB of initial data (with one exception)
- In the standard the 10 Terminals per Warehouse can 12.86 NOPM max. (this is outdated)

The example test used

Sizes at 400 Warehouses

CUSTOMER	8 GB	Non Uniform Random Access
STOCK	14 GB	Non Uniform Random Access
ORDERS	1.7 GB	Only one delayed update per row
ORDER_LINE	17 GB	Only one delayed update per row
Database	40 GB	

Seems OK for a server with 32GB of RAM, 8GB shared.

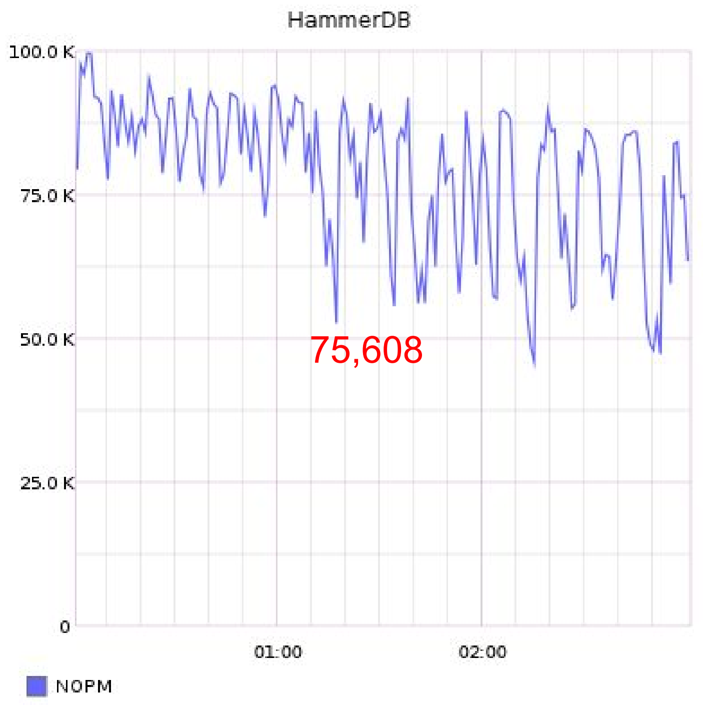
Transaction rate limiting and database scaling

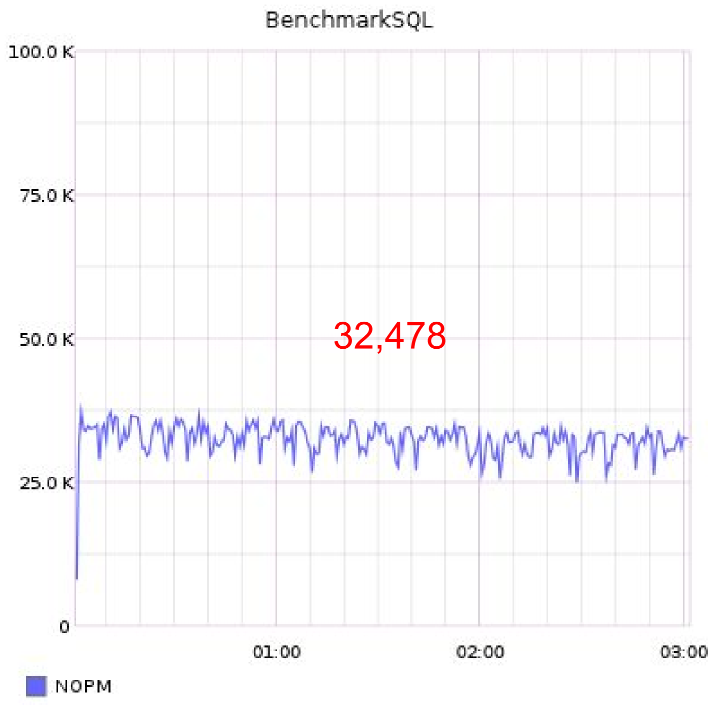
- OLTP is different from Batch.
- TPC/C has very detailed keying and think times per terminal that limit the NOPM per warehouse to 12.86
- 12.86 NOPM per Warehouse means about 28.5 Transactions per Minute maximum. At a 400 Warehouse scale that means 190 TPS. That was Block Terminal era.
- Simply racking up the rate changes more than just CPU usage, IO and TPS.

So where are the graphs?

Right ...

The NOPM (New Order Per Minute) no rate limit

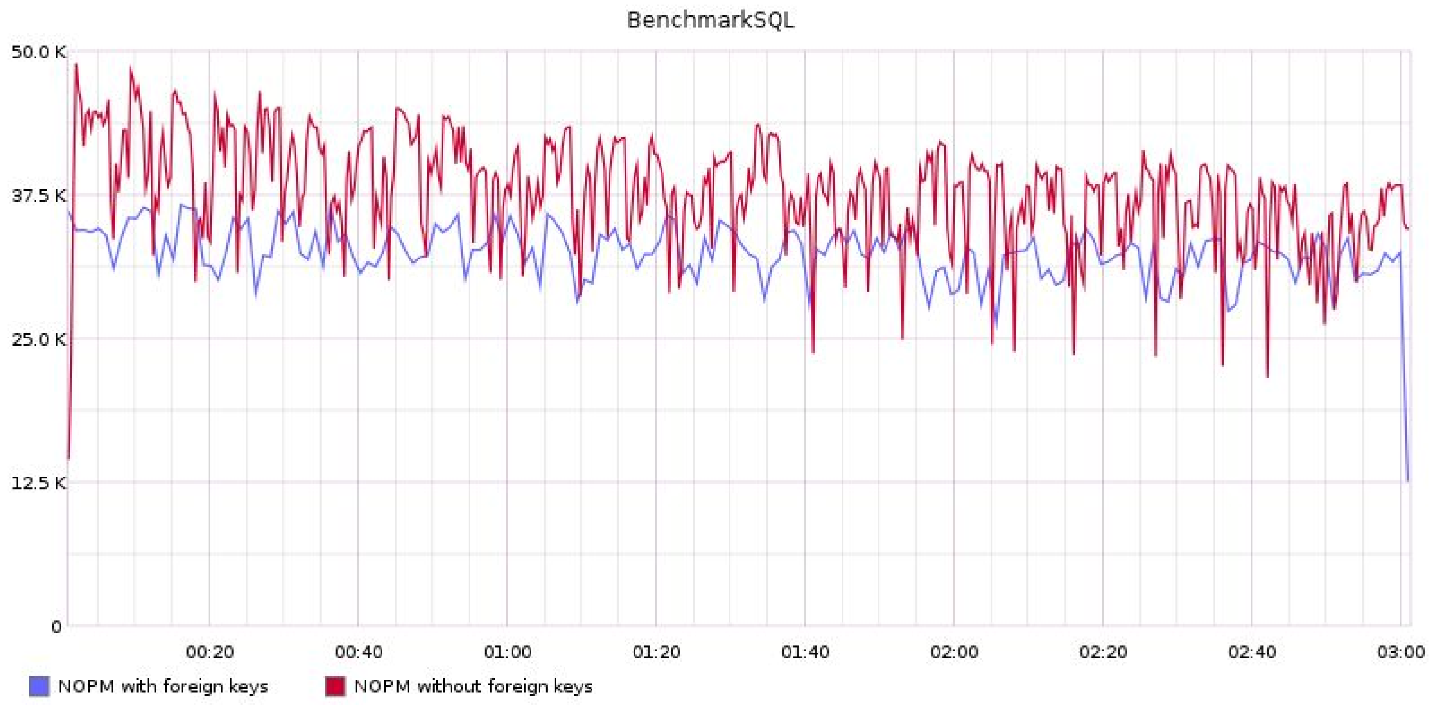




The difference in NOPM

- BenchmarkSQL defines the required FOREIGN KEYs.
- HammerDB omits the FOREIGN KEYs.
- BenchmarkSQL has all business logic implemented in the application (running remote).
- HammerDB has all business logic implemented in PL/pgSQL functions.

FOREIGN KEYs are expensive



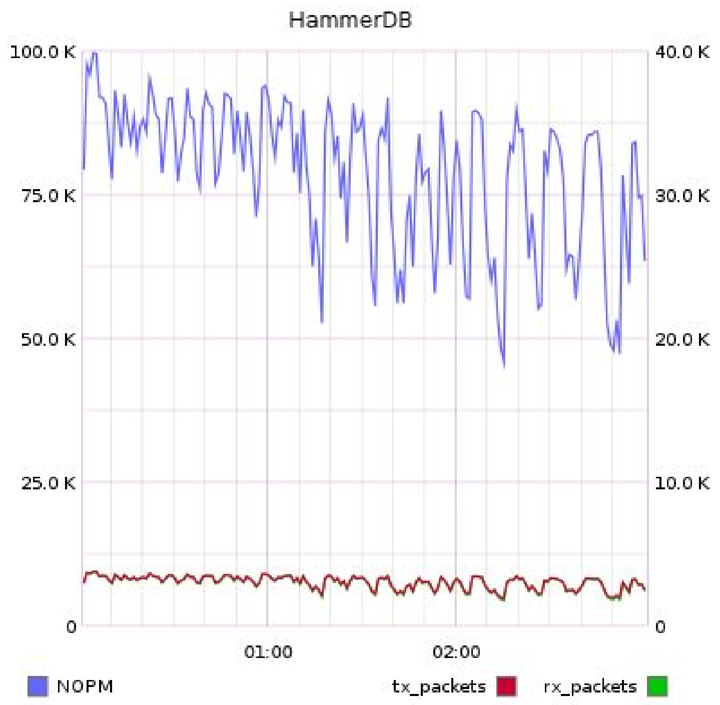
Adding foreign keys on HammerDB

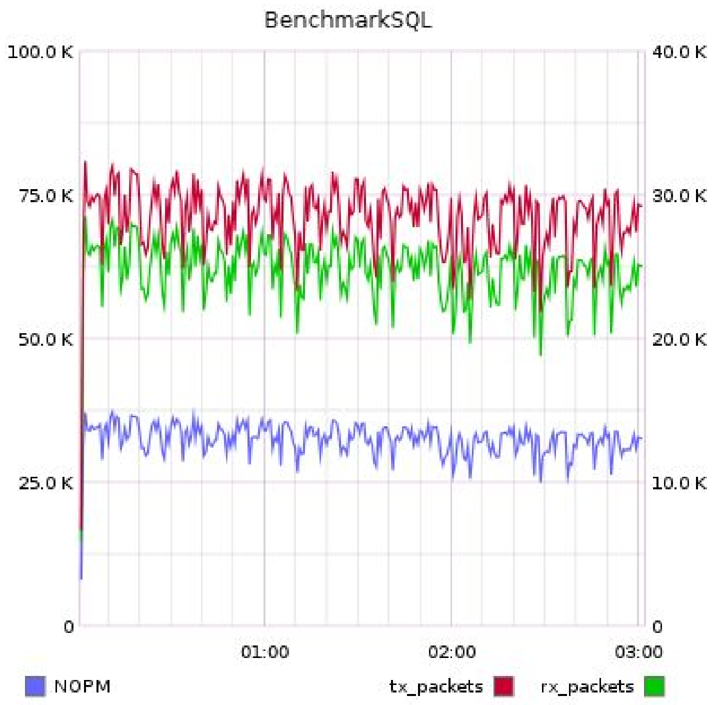
Creating the foreign keys on the HammerDB database leads to errors.

The NEW ORDER transaction tries to simulate the 1% user input error by using a non existing item ID. The INSERT attempt now fails and leads to leaking result handles in Tcl.

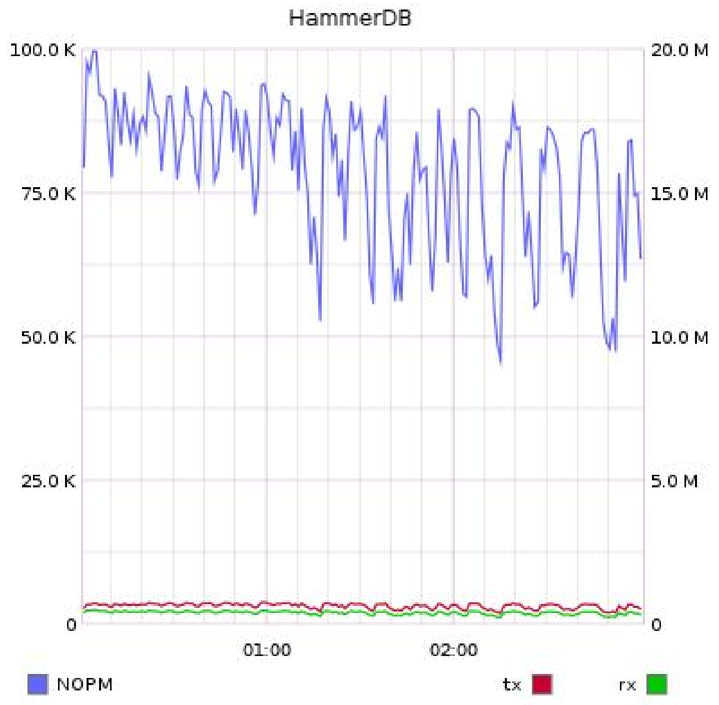
```
Error in Virtual User 17: hard limit on result handles
reached
```

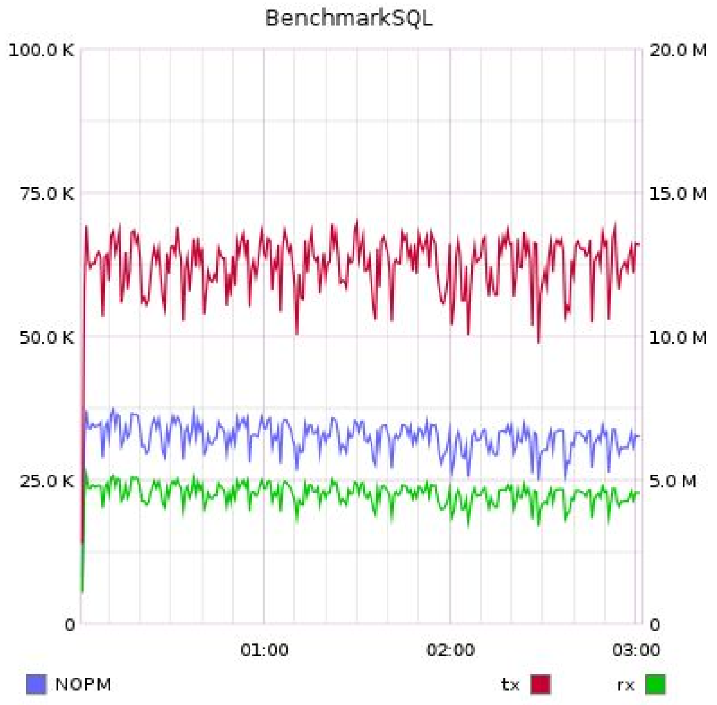
But there is more … how about network traffic?





More packets makes sense, but that much in bytes?





HammerDB went above and beyond

TPC/C defines Input/Output screens. Placing the business logic into a stored procedure/function is fine. But …

The New Order Transaction requires the USER to enter Item IDs, quantities and Warehouse IDs. The Output screen displays the Item name, price and other information. Logically this information must be transmitted between the Application and the Database.

This is the signature of the PL/pgSQL function:

```
CREATE FUNCTION NEWORD (
    no_w_id integer,
    no_max_w_id integer,
    no_d_id integer,
    no_c_id integer,
    no_o_ol_cnt integer,
    no_d_next_o_id integer)
RETURNS numeric AS …
```

All the communication of the customer address, the line data and so on over the network has been eliminated.

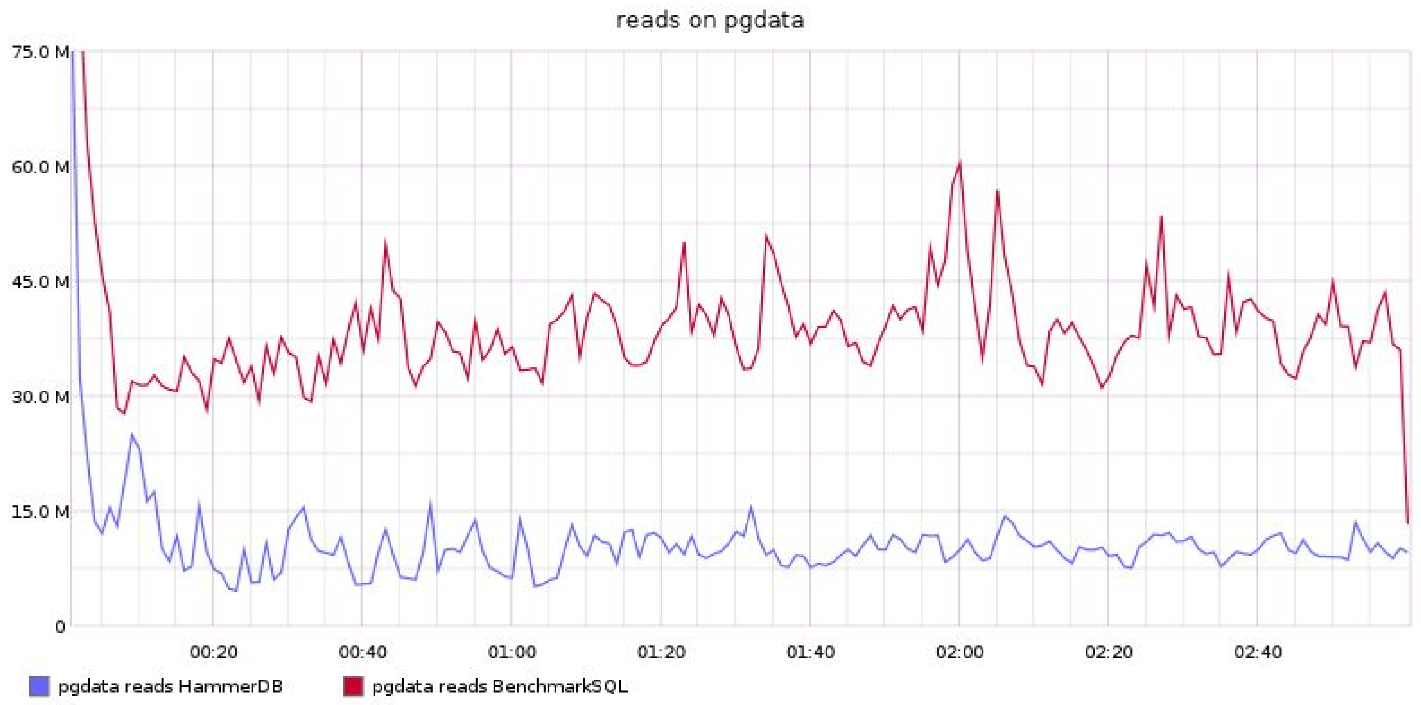
The PL/pgSQL function for PAYMENT is similar.

The functions still perform the queries internally.

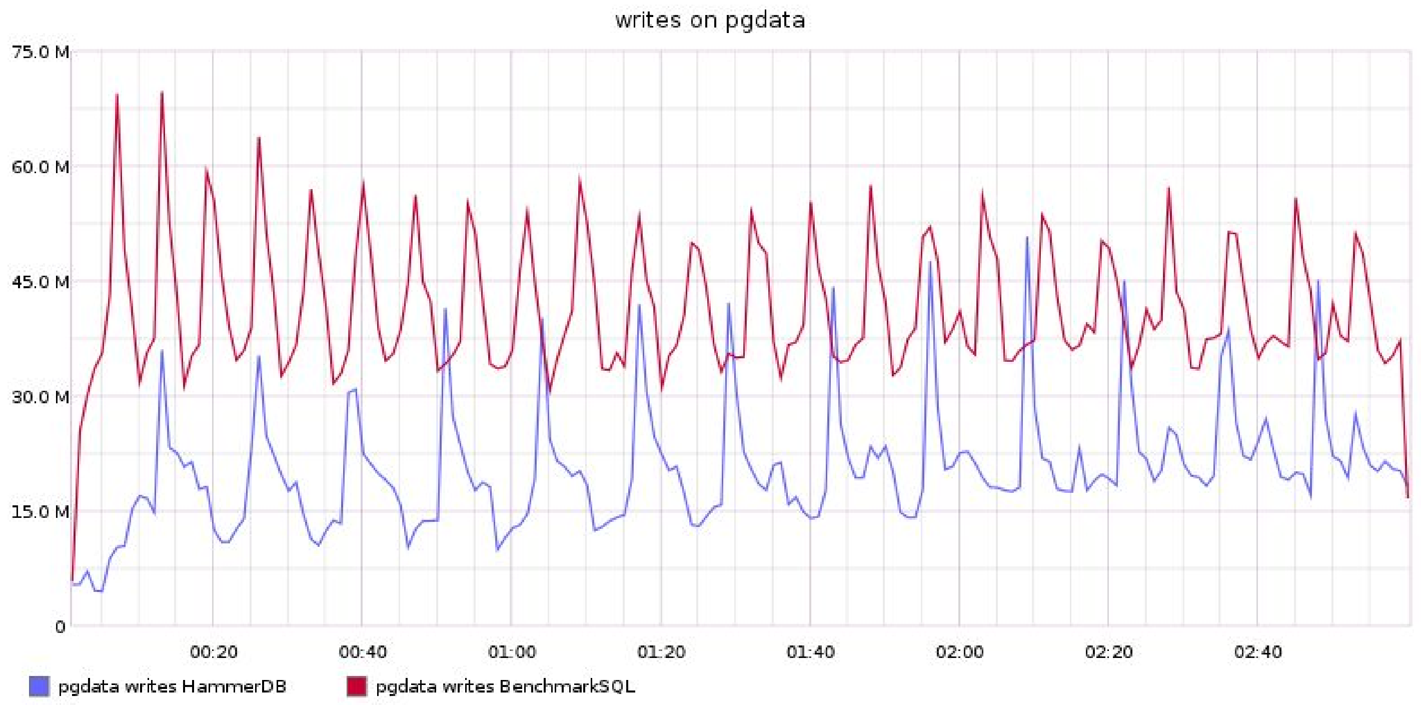
However, the New Order and Payment transactions make up 88% of the transaction mix in the TPC/C.

Eliminating most of the network round trips allows to keep fewer connections busy enough to reach saturation. Connections aren’t for free.

BenchmarkSQL needs more disk reads



… and writes


writes on pgdata
75.0 M
60.0 M
45.0 M
30.0 M
15.0 M
0
00:20
00:40
01:00
01:20
01:40
02:00
02:20
02:40
pgdata writes HammerDB
pgdata writes BenchmarkSQL

Why is that?

- 2.5 times the performance with less IO?
- HammerDB and BenchmarkSQL both implement TPC/C.
- Let me explain what happens here together with the next difference.

Let’s scale

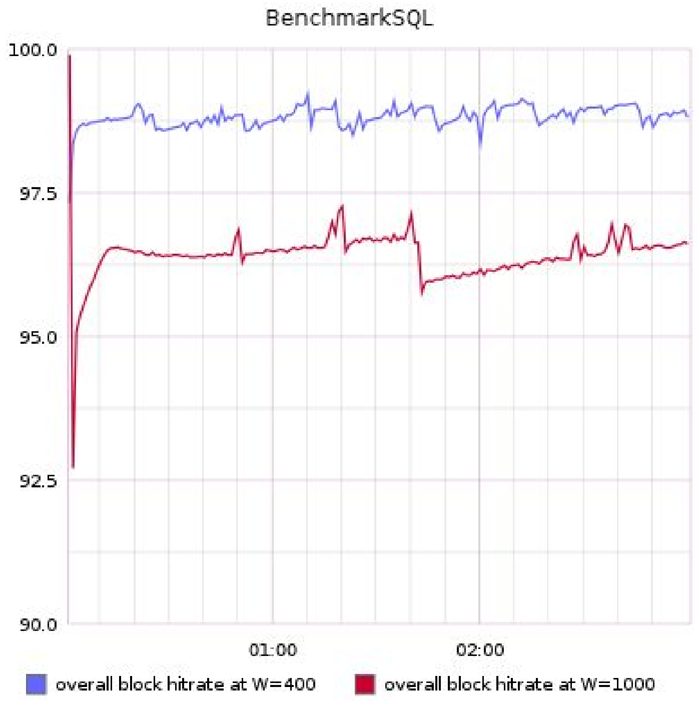
Knowing the TPC/C Benchmark, increasing the scaling factor (number of Warehouses) and thereby increasing the size of the database, must slow things down.

For the next tests the scale was increased from 400 to 1,000 Warehouses. The resulting test database has an initial size of over 100GB and the big tables for sure do no longer fit into the machine’s memory (32GB).

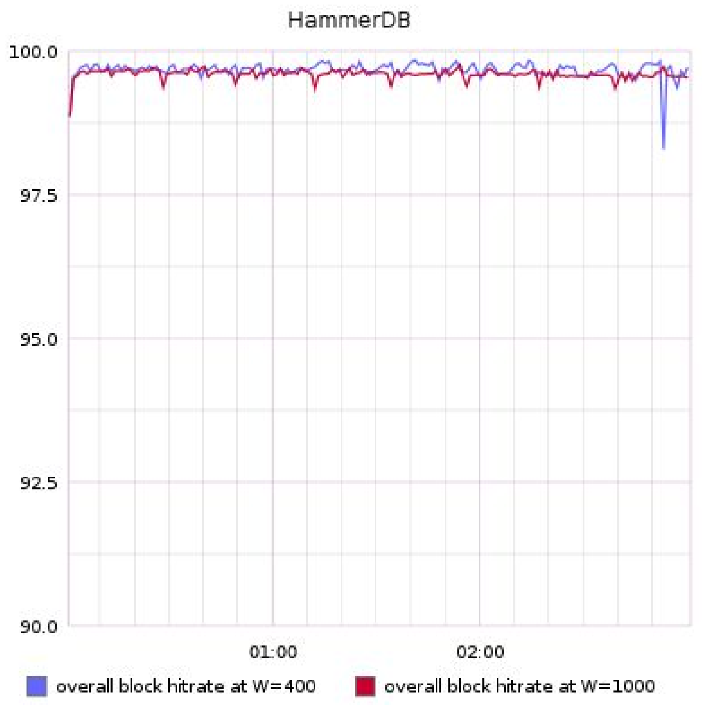
The resulting NOPM (are a little surprising)

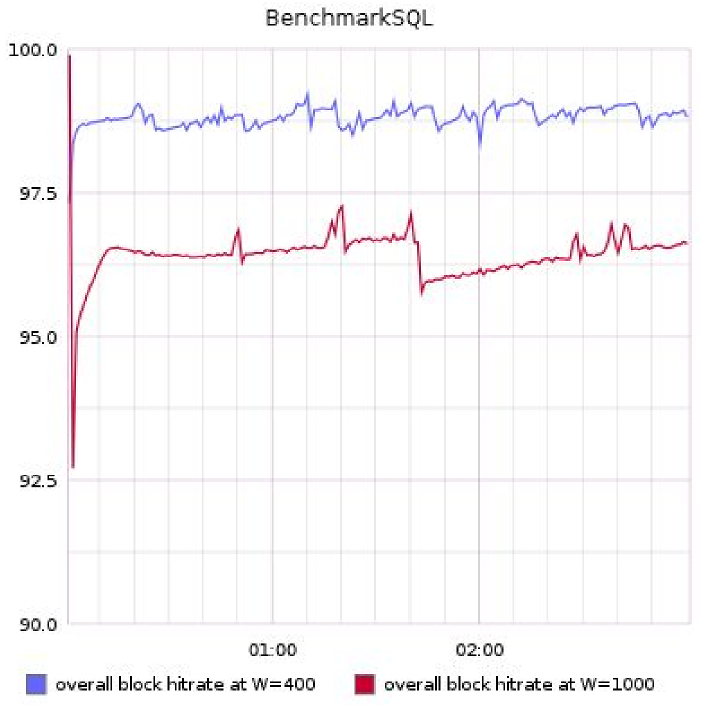
Test	**W=400**	**W=1000**	Change
HammerDB	75608	83913	+11%
BenchmarkSQL	32478	21747	-33%

The block hitrate with W=400 and W=1000



The block hitrate with W=400 and W=1000





The issue is the Home Warehouse ID

- Clauses 2.[4-8].1.1 specify that the W_ID used by a Terminal is constant for the duration of the test. For most of the data access of a transaction, this means that 85% of the data used belongs to that Home Warehouse ID.
- These benchmarks do not implement Terminals, but rather Application threads that should serve all Warehouses.
- BenchmarkSQL did this too (fixed in 5.0 and added a configuration option to get back the old behavior).

Other aspects

- Run long enough to cover checkpoints and autovacuum.
- Run even longer if there is a risk of txid wraparound.
- Consider replicating your benchmark database.
- Rebuild the environment, your database will run in.

Conclusion

- Make sure the test resembles you application (not necessarily the standard).
- Configure and tune properly.
- Verify that it produces the expected resource usage patterns.
- Run tests long enough.

Questions

???

Thank you

Jan Wieck

OpenSCG

TPC BENCHMARK^{TM} B

Standard Specification

Revision 2.0

7 June 1994

0.1 Introduction

…

This benchmark is not OLTP in that it does not require any terminals, networking, or think time.