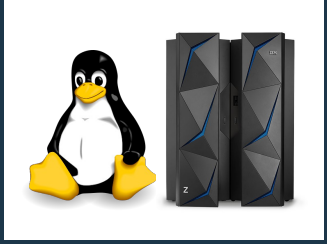
Open Source on IBM Z: Experiences with PostgreSQL on Linux on IBM Z

Marc Beyerle (marc.beyerle@de.ibm.com)

IBM Mainframe Specialist, Senior Java Performance Engineer



Linux on IBM Z and IBM LinuxONE Customer Webcast

28 February 2018

IBM

Trademarks

The following are trademarks of the International Business Machines Corporation in the United States, other countries, or both.

Not all common law marks used by IBM are listed on this page. Failure of a mark to appear does not mean that IBM does not use the mark nor does it mean that the product is not actively marketed or is not significant within its relevant market.

Those trademarks followed by ® are registered trademarks of IBM in the United States; all others are trademarks or common law marks of IBM in the United States.

For a more complete list of IBM Trademarks, see www.ibm.com/legal/copytrade.shtml:

*BladeCenter®, CICS®, DataPower®, DB2®, e business(logo)®, ESCON, eServer, FICON®, IBM®, IBM (logo)®, IMS, MVS, OS/390®, POWER6®, POWER6+, POWER7®, Power Architecture®, PowerVM®, PureFlex, PureSystems, S/390®, ServerProven®, Sysplex Timer®, System p®, System p5, System x®, System z®, System z9®, System z10®, WebSphere®, X-Architecture®, z13®, z13s, z Systems®, z9®, z10, z/Architecture®, z/OS®, z/VM®, z/VSE®, zEnterprise®, zSeries®, IBM Z®

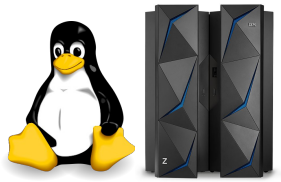
The following are trademarks or registered trademarks of other companies.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.
Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.
Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.
Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
UNIX is a registered trademark of The Open Group in the United States and other countries.
Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
ITIL is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.
IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency, which is now part of the Office of Government Commerce.

*** All other products may be trademarks or registered trademarks of their respective companies.**

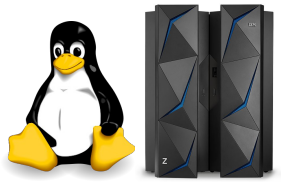
Notes:
Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.
IBM hardware products are manufactured Sync new parts, or new and serviceable used parts. Regardless, our warranty terms apply.
All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.
This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.
All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.
Information about non-IBM products is obtained Sync the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.
Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.

Agenda



- Some background information on PostgreSQL
- Performance measurement and tuning approach
- Observations and recommendations
  - Throughput
  - Response time
  - General
- Summary

PostgreSQL background information

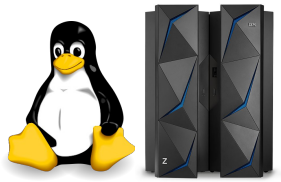


- 1st Google hit: *"The official site for PostgreSQL, the **world's most advanced** open source database."*
- More than ***15 years*** of active development
- Runs on ***all major*** operating systems
- Fully ***ACID compliant***
- See the next 2 pages: it seems like PostgreSQL® has quite a high (a) ***popularity rank*** and (b) ***market share***

PostgreSQL

Marc Beyerle (marc.beyerle@de.ibm.com) – Open Source on IBM Z: Experiences with PostgreSQL on Linux on IBM Z

Database popularity ranking

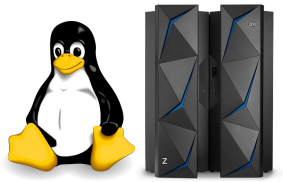


341 systems in ranking, February 2018

Rank Feb 2018	Rank Jan 2018	Rank Feb 2017	DBMS	Database Model	Score Feb 2018	Score Jan 2018	Score Feb 2017
1.	1.	1.	Oracle	Relational DBMS	1303.28	-38.66	-100.55
2.	2.	2.	MySQL	Relational DBMS	1252.47	-47.24	-127.83
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1122.04	-26.03	-81.42
4.	4.	4.	PostgreSQL	Relational DBMS	388.38	+2.19	+34.70
5.	5.	5.	MongoDB	Document store	336.42	+5.47	+0.92
6.	6.	6.	DB2	Relational DBMS	189.97	-0.30	+2.07
7.	7.	↑ 8.	Microsoft Access	Relational DBMS	130.07	+3.37	-3.32
8.	↑ 9.	↑ 10.	Redis	Key-value store	127.02	+3.88	+12.98
9.	↑ 10.	↑ 11.	Elasticsearch	Search engine	125.32	+2.76	+17.01
10.	↓ 8.	↓ 7.	Cassandra	Wide column store	122.78	-1.10	-11.60

Source: https://db-engines.com/en/ranking

Database market share



Market Share of Database Management System products

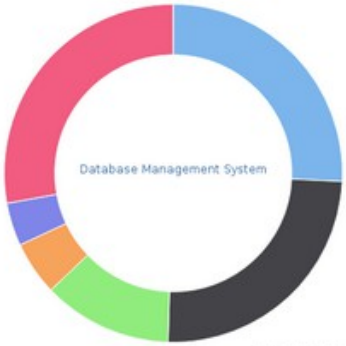
At iDatalabs, we use sophisticated, patent-pending algorithms to track the use of various Database Management System products and technologies. We track 69 products in the Database Management System category, and have found 541,338 companies using these products.

Database Management System

Share this report: Embed

541,338

Companies using Database Management System

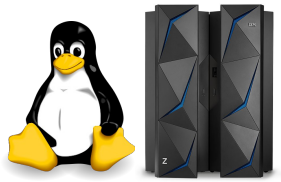


Database Management System

- Microsoft SQL Server (25.77%)
- MySQL (24.74%)
- Microsoft Access (12.37%)
- PostgreSQL (5.15%)
- MongoDB (4.12%)

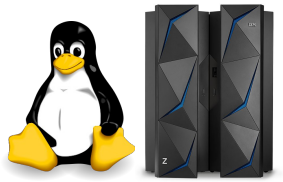
Source: https://idatalabs.com

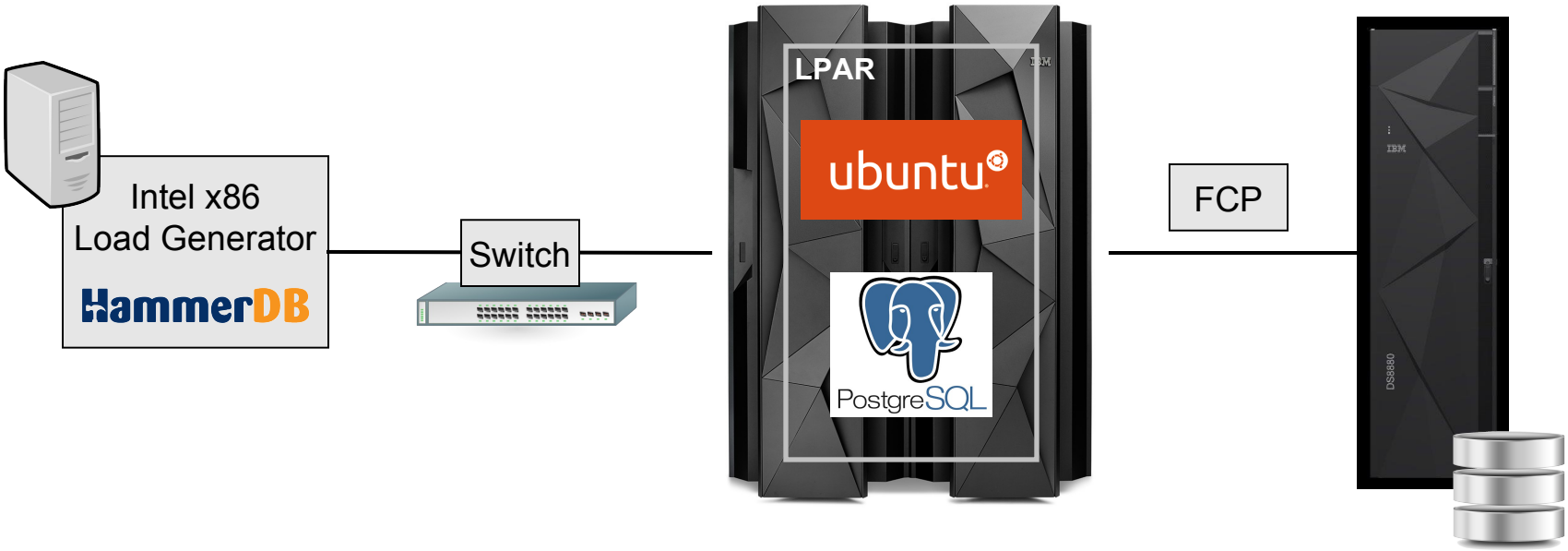
Agenda



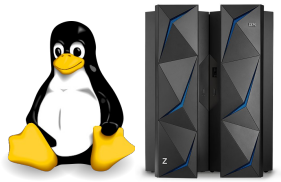
- Some background information on PostgreSQL
- Performance measurement and tuning approach
- Observations and recommendations
  - Throughput
  - Response time
  - General
- Summary

High-level environment setup



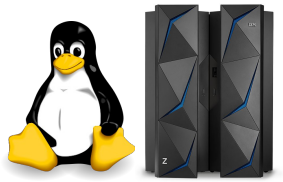


Approach



- First step was to install a ***sandbox environment***, in order to get familiar with Ubuntu®, PostgreSQL, and HammerDB
  - Goal: as simple as possible environment, start with out-of-the-box configuration
- Populated a ***small TPC-C™ test database*** with HammerDB built-in functionality
  - Most important configuration parameter: number of so-called *warehouses*
- Put the *System Under Test* (SUT) under ***light load*** (i.e. small number of virtual users), just to see how it behaves at run-time
  - Generally speaking, at a first glance PostgreSQL behaves like any other modern database: both CPU and I/O intensive
- Applied numerous ***tuning parameters*** at all levels (OS, I/O, database) in small test runs, just to see if they have any effect
  - Goal: decide if parameters should be evaluated in large test runs

Approach, *continued*



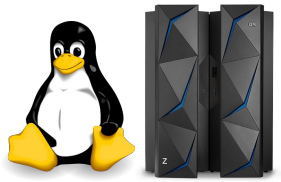
- Parameters were selected based on (a) ***Internet research*** (PostgreSQL documentation, Wikis, Blog entries, etc.), but also by (b) ***analyzing*** the run-time behavior of PostgreSQL and choosing appropriate well-known kernel parameters, *Logical Volume Manager* (LVM) settings, etc.

- After gaining experience with the sandbox environment, set up a ***large test environment*** in its own *Logical Partition* (LPAR)
  - *Large* means more memory (64GB) and more disk space (256GB database size)

- Applied ***additional tuning parameters***, where the expectation was that they would show some effects in the large test environment
  - See *observations and recommendations* section for more details

- Implemented ***automated test scripts*** based on an internal test framework and homegrown shell scripts, in order to be able to run identical benchmark runs
  - Results are stored in a database together with `sadc` / `sar` data, etc.

Approach, *continued*

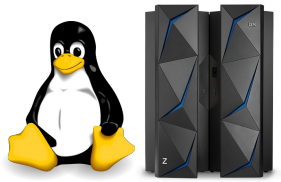


- Last but not least: ran so-called *night runs* with ***exclusive use*** of our IBM® Z® host (IBM z13®) and storage server (DS8000®)
  - Goal was to generate *reproducible* benchmark results

- ***Goal*** of the tuning experiments was to gain a ***pragmatic set*** of parameters / switches that lead to ***measurable*** performance improvements
  - Goal was ***not*** to execute performance test runs for the sake of ***scientific curiosity***

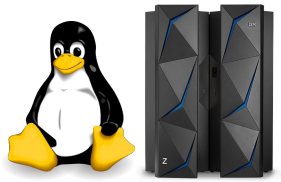
- Another goal of this exercise was ***not*** to performance-tune the HammerDB built-in TPC-C workload, but to give recommendations that – very probably – hold true for ***all transaction-oriented workloads***
  - No tuning of TPC-C related SQL statements, indexes, tablespace design, etc.

Agenda



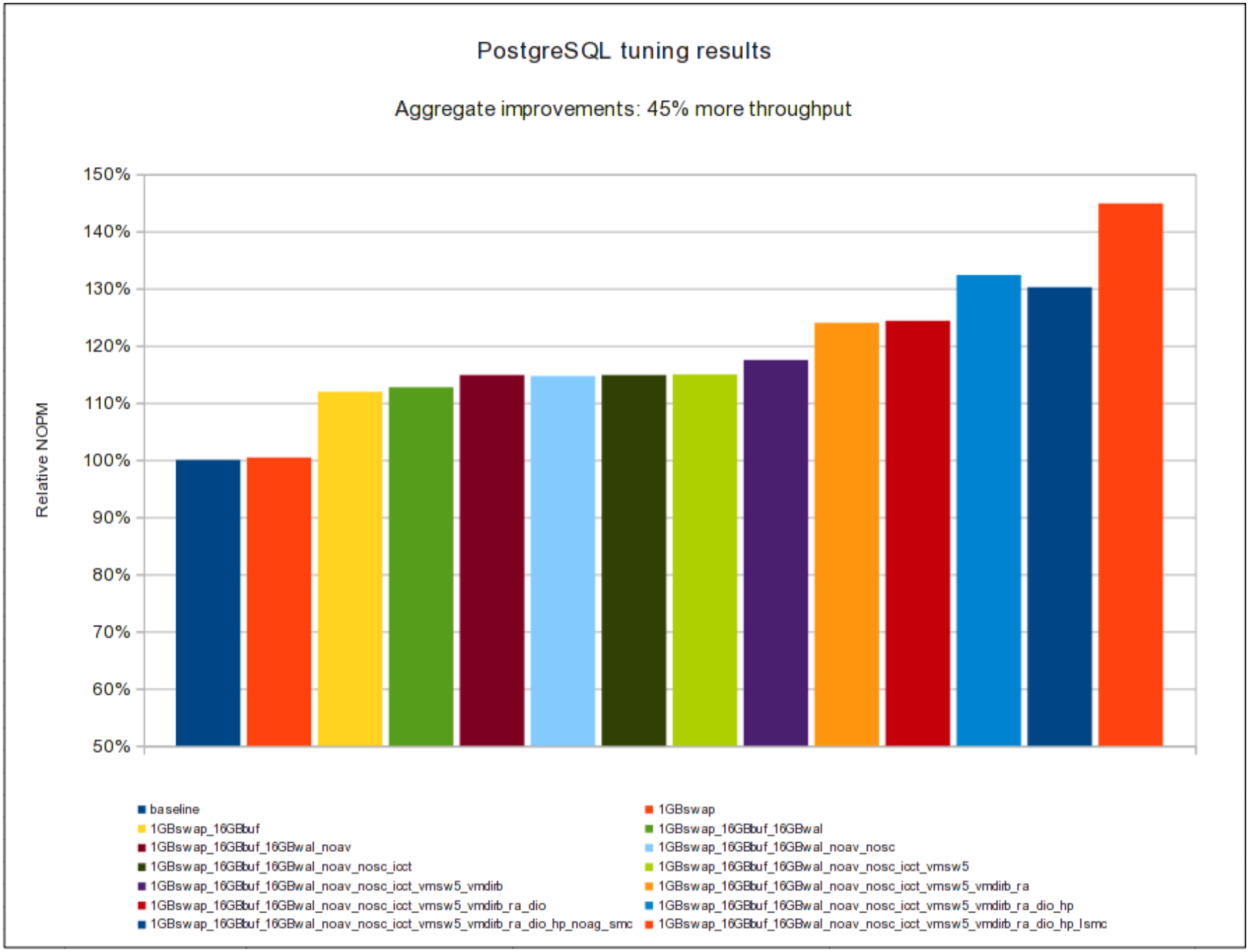
- Some background information on PostgreSQL
- Performance measurement and tuning approach
- Observations and recommendations
  - Throughput
  - Response time
  - General
- Summary

Disclaimers

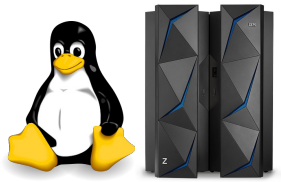


The following is ***important*** – please read it carefully

- The performance test results in the following charts were obtained in a ***controlled lab environment*** natively in LPAR. The measured differences in throughput might not be observed in real-life scenarios and environments other than native LPAR.
- All of the test runs were performed with Ubuntu 16.04.2, PostgreSQL 9.5.7, and HammerDB 2.23. ***Other*** product versions might produce ***different*** performance results.
- All of the tests were specifically executed for ***PostgreSQL***. The impact of the recommendations in this chart deck on ***other*** database management systems might be ***totally different***, including ***adverse*** performance effects.
- All of the tests were specifically executed for a heavy ***Online Transaction Processing*** (OLTP) workload. The impact of the recommendations in this chart deck on other types of workloads – *Online Analytical Processing* (OLAP), for example – might be ***totally different***, including ***adverse*** performance effects.


PostgreSQL tuning results
Aggregate improvements: 45% more throughput
Relative NOPM
150%
140%
130%
120%
110%
100%
90%
80%
70%
60%
50%
baseline
1GBswap
1GBswap_16GBbuf
1GBswap_16GBbuf_16GBwal
1GBswap_16GBbuf_16GBwal_noav
1GBswap_16GBbuf_16GBwal_noav_nosc
1GBswap_16GBbuf_16GBwal_noav_nosc_icct
1GBswap_16GBbuf_16GBwal_noav_nosc_icct_vmsw5
1GBswap_16GBbuf_16GBwal_noav_nosc_icct_vmsw5_vmdirb
1GBswap_16GBbuf_16GBwal_noav_nosc_icct_vmsw5_vmdirb_ra
1GBswap_16GBbuf_16GBwal_noav_nosc_icct_vmsw5_vmdirb_ra_dio
1GBswap_16GBbuf_16GBwal_noav_nosc_icct_vmsw5_vmdirb_ra_dio_hp
1GBswap_16GBbuf_16GBwal_noav_nosc_icct_vmsw5_vmdirb_ra_dio_hp_noag_smc
1GBswap_16GBbuf_16GBwal_noav_nosc_icct_vmsw5_vmdirb_ra_dio_hp_lsmc

Throughput recommendation #1

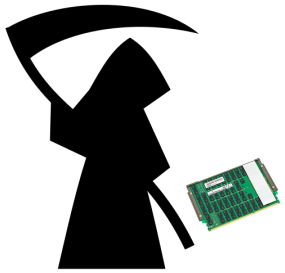


- Throughput recommendation #1: **increase the size of the PostgreSQL *shared buffers* to 1/4 of the total amount of physical memory and the size of the *effective cache* to 3/4**

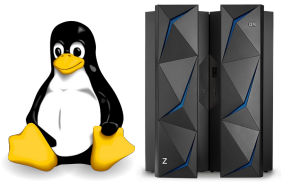
- Recommendation from the official PostgreSQL Tuning Wiki
  - https://wiki.postgresql.org/wiki/Tuning_Your_PostgreSQL_Server

- Turned out to be one of the major tuning knobs: ***almost 12% more throughput***
  - See also the tuning results chart

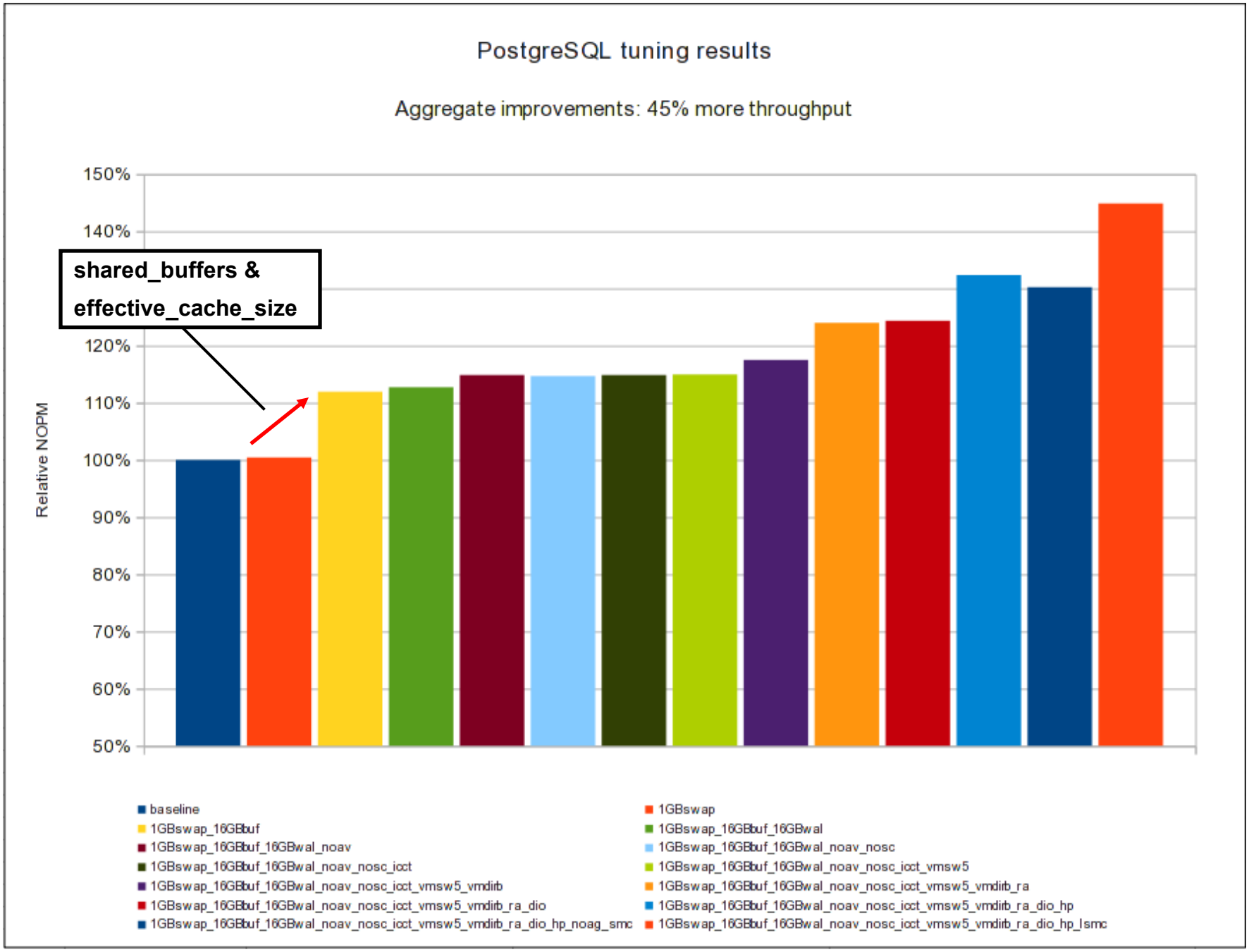
- ***Important***: do not increase the size of the PostgreSQL shared buffers too much – for example to 1/2 of the total amount of physical memory – otherwise, you will see activity from the ***Linux® out-of-memory (OOM) killer***



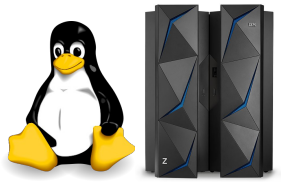
Throughput recommendation #1, *continued*



- ***Lowering*** the size of the shared buffers to 1/8 did not lead to a performance degradation for the HammerDB TPC-C workload
  - – ***Caution***: this might not hold true for other database workloads
- ***Question***: Why is the impact of larger shared buffers so marginal?
- ***Answer***: Very probably because PostgreSQL relies heavily on the effectiveness of the Linux ***page cache*** – this is mentioned several times in the PostgreSQL documentation
- The ***effective cache size*** setting is not an actual memory allocation, but only *"… an estimate of how much memory is available for disk caching by the operating system and within the database itself"* (quote from the Tuning Wiki)
  - – Used by the PostgreSQL query planner


PostgreSQL tuning results
Aggregate improvements: 45% more throughput
shared_buffers &
effective_cache_size
Relative NOPM
150%
140%
120%
110%
100%
90%
80%
70%
60%
50%
baseline
1GBswap
1GBswap_16GBbuf
1GBswap_16GBbuf_16GBwal
1GBswap_16GBbuf_16GBwal_noav
1GBswap_16GBbuf_16GBwal_noav_nosc
1GBswap_16GBbuf_16GBwal_noav_nosc_icct
1GBswap_16GBbuf_16GBwal_noav_nosc_icct_vmsw5
1GBswap_16GBbuf_16GBwal_noav_nosc_icct_vmsw5_vmdirb
1GBswap_16GBbuf_16GBwal_noav_nosc_icct_vmsw5_vmdirb_ra
1GBswap_16GBbuf_16GBwal_noav_nosc_icct_vmsw5_vmdirb_ra_dio
1GBswap_16GBbuf_16GBwal_noav_nosc_icct_vmsw5_vmdirb_ra_dio_hp
1GBswap_16GBbuf_16GBwal_noav_nosc_icct_vmsw5_vmdirb_ra_dio_hp_noag_smc
1GBswap_16GBbuf_16GBwal_noav_nosc_icct_vmsw5_vmdirb_ra_dio_hp_lsmc

Throughput recommendation #2



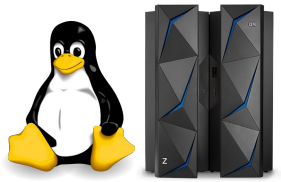
- Throughput recommendation #2: **for mostly read-only workloads, consider turning off the *autovacuum daemon* in PostgreSQL**

- In general, it is advisable to ***keep*** the autovacuum daemon ***turned on***
  - The autovacuum daemon is turned on by default

- The autovacuum daemon does ***very useful things***
  - Recovers or reuses disk space occupied by updated or deleted rows
  - Updates data statistics used by the PostgreSQL query planner
  - etc.

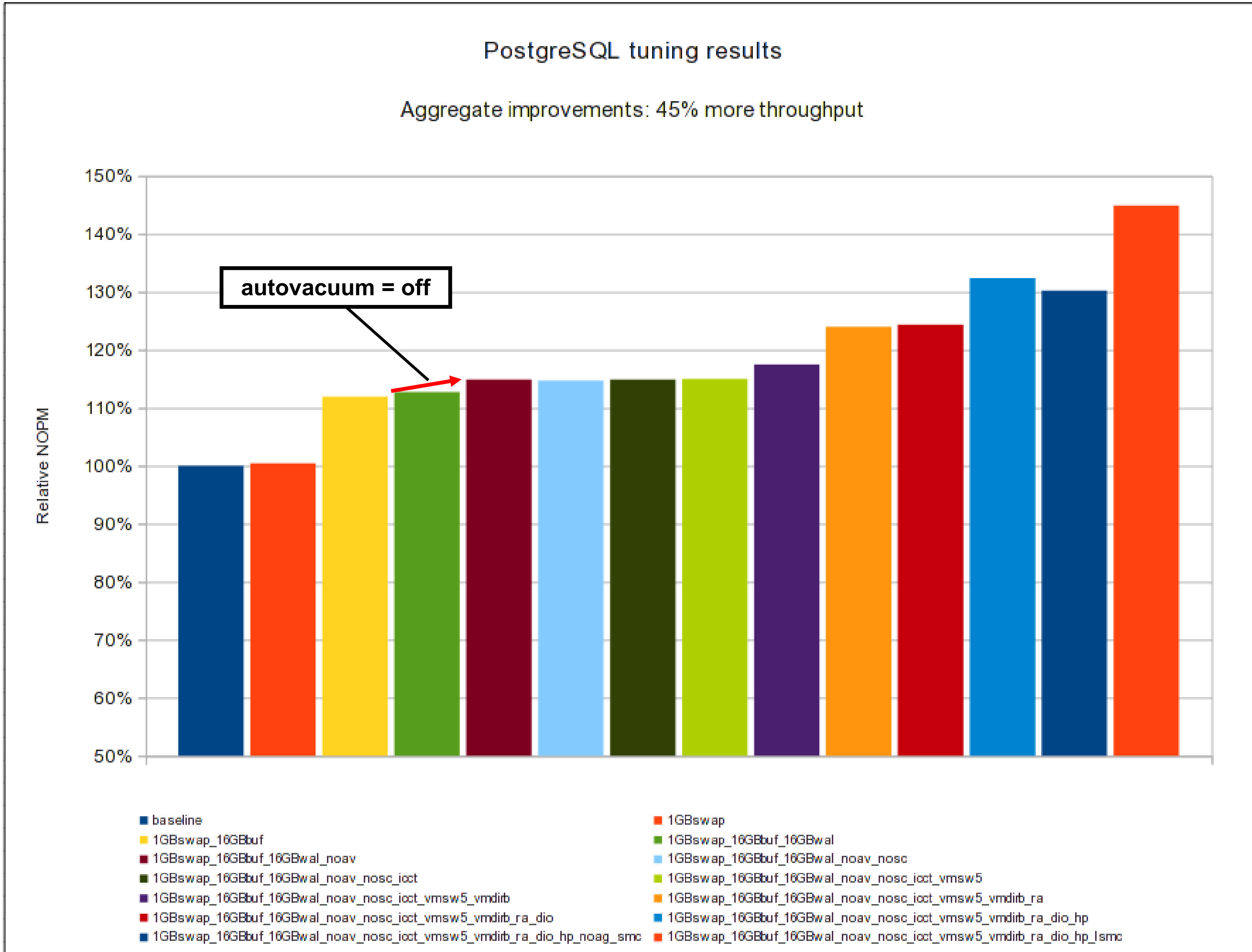
- However, this process costs some CPU cycles – impact of turning autovacuum off for the HammerDB TPC-C workload: ***almost 2% more throughput***
  - See also the tuning results chart

Throughput recommendation #2, *continued*

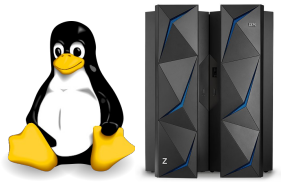


- For ***write-heavy*** production workloads – that is, lots of `INSERT`, `UPDATE`, and `DELETE` statements – the ***clear recommendation*** is to keep the autovacuum daemon ***turned on***, otherwise (a) a lot of disk space could be wasted and (b) the database statistics could be skewed / bad
  - Only exception to this rule might be situations where the inserted and / or updated data does not change the statistics information significantly, but this is very hard to figure out

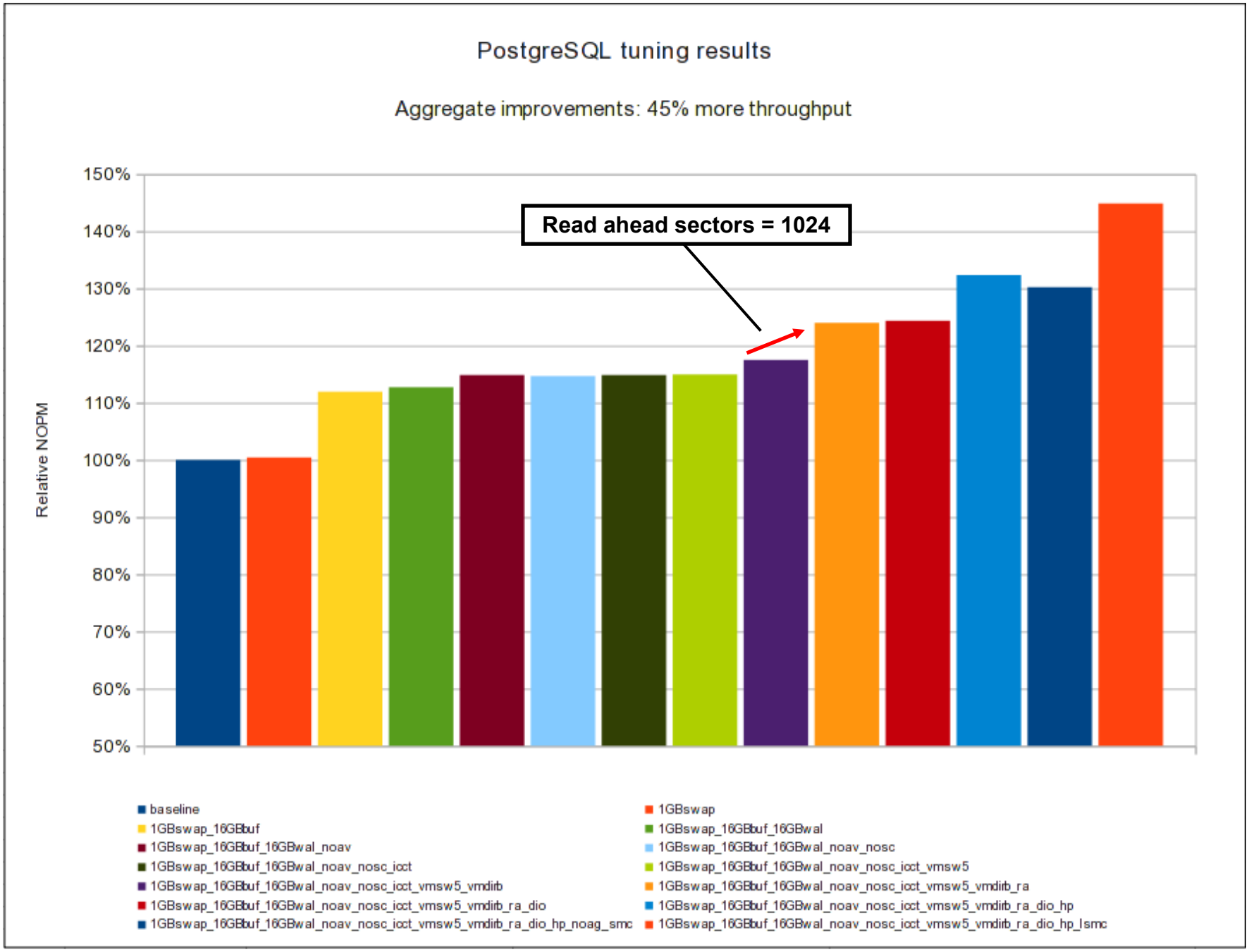
- For mostly ***read-only*** database workloads, the statistics information ***does not change significantly*** over time, so you might try running PostgreSQL with the autovacuum daemon turned off
  - Nevertheless, a manual `VACUUM` operation should be scheduled – via a `cron` job, for example – at times when the load on the system is low


PostgreSQL tuning results
Aggregate improvements: 45% more throughput
150%
140%
130%
120%
110%
100%
90%
80%
70%
60%
50%
Relative NOPM
autovacuum = off
baseline
1GBswap
1GBswap_16GBbuf
1GBswap_16GBbuf_16GBwal
1GBswap_16GBbuf_16GBwal_noav
1GBswap_16GBbuf_16GBwal_noav_nosc
1GBswap_16GBbuf_16GBwal_noav_nosc_icct
1GBswap_16GBbuf_16GBwal_noav_nosc_icct_vmsw5
1GBswap_16GBbuf_16GBwal_noav_nosc_icct_vmsw5_vmdirb
1GBswap_16GBbuf_16GBwal_noav_nosc_icct_vmsw5_vmdirb_ra
1GBswap_16GBbuf_16GBwal_noav_nosc_icct_vmsw5_vmdirb_ra_dio
1GBswap_16GBbuf_16GBwal_noav_nosc_icct_vmsw5_vmdirb_ra_dio_hp
1GBswap_16GBbuf_16GBwal_noav_nosc_icct_vmsw5_vmdirb_ra_dio_hp_noag_smc
1GBswap_16GBbuf_16GBwal_noav_nosc_icct_vmsw5_vmdirb_ra_dio_hp_lsmc

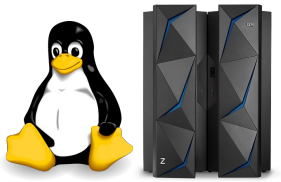
Throughput recommendation #3



- Throughput recommendation #3: **turn on *read ahead* for the logical volume containing the database data files**
- ***Question***: Why is this setting so important / so ***interesting***?
- ***Answer***: Because ***up to now***, the clear recommendation for databases was to turn read ahead ***off*** at the *Logical Volume* (LV) / block device level
  - DB2 e.g. performs better if read ahead is turned off at the LV / block device level
- Impact of turning read ahead on: ***almost 6% more throughput***
  - See also the tuning results chart
- Again, very probably, the reason for this is the fact that PostgreSQL relies heavily on the effectiveness of the Linux ***page cache***
  - Other databases ***decide themselves*** which pages should be read ahead and do not rely on read ahead functionality of the operating system

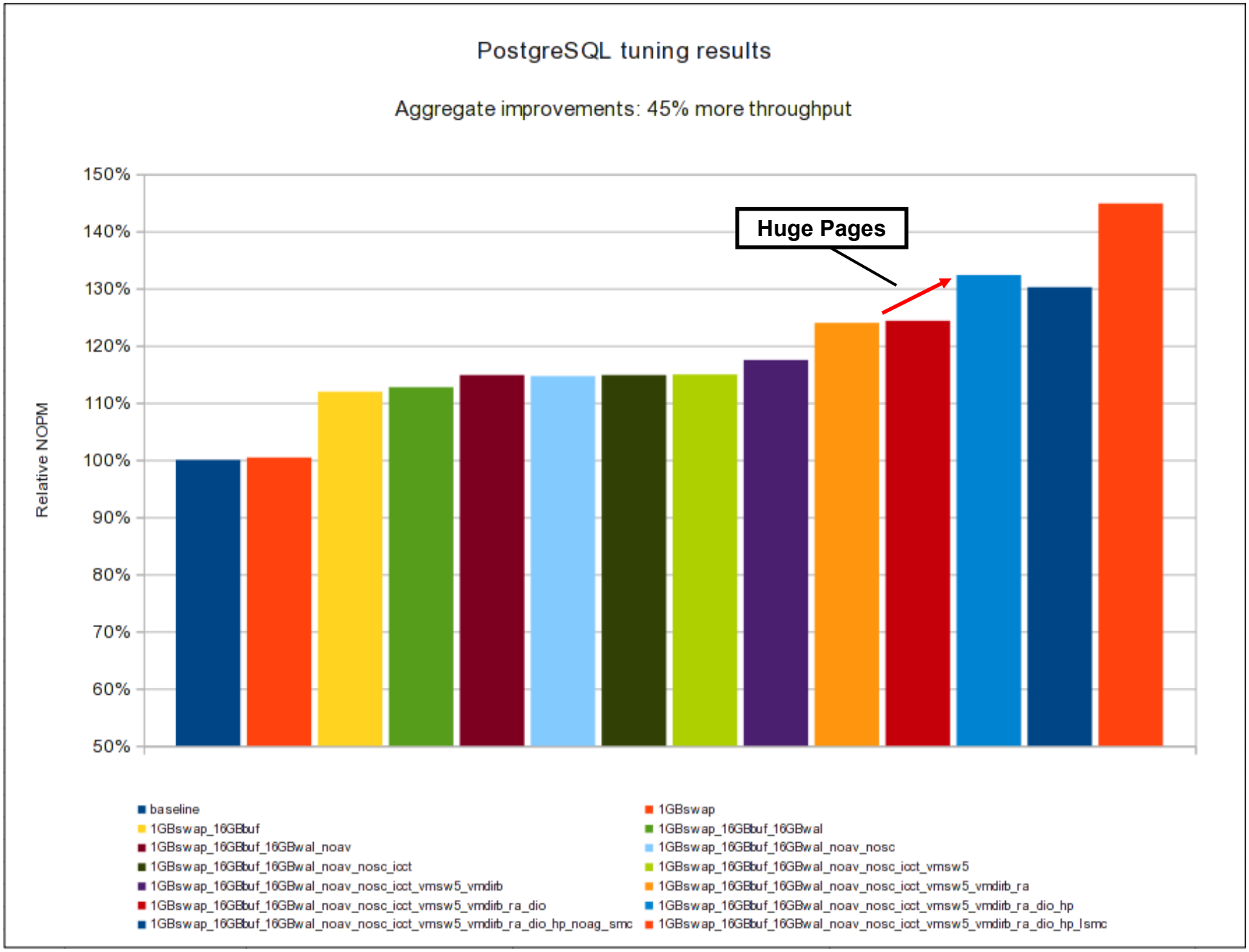

PostgreSQL tuning results
Aggregate improvements: 45% more throughput
Read ahead sectors = 1024
Relative NOPM
150%
140%
130%
120%
110%
100%
90%
80%
70%
60%
50%
baseline
1GBswap
1GBswap_16GBbuf
1GBswap_16GBbuf_16GBwal
1GBswap_16GBbuf_16GBwal_noav
1GBswap_16GBbuf_16GBwal_noav_nosc
1GBswap_16GBbuf_16GBwal_noav_nosc_icct
1GBswap_16GBbuf_16GBwal_noav_nosc_icct_vmsw5
1GBswap_16GBbuf_16GBwal_noav_nosc_icct_vmsw5_vmdirb
1GBswap_16GBbuf_16GBwal_noav_nosc_icct_vmsw5_vmdirb_ra
1GBswap_16GBbuf_16GBwal_noav_nosc_icct_vmsw5_vmdirb_ra_dio
1GBswap_16GBbuf_16GBwal_noav_nosc_icct_vmsw5_vmdirb_ra_dio_hp
1GBswap_16GBbuf_16GBwal_noav_nosc_icct_vmsw5_vmdirb_ra_dio_hp_noag_smc
1GBswap_16GBbuf_16GBwal_noav_nosc_icct_vmsw5_vmdirb_ra_dio_hp_lsmc

Throughput recommendation #4

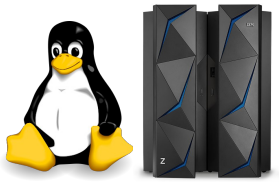


- Throughput recommendation #4: **turn on** *huge pages*
- *Huge pages* in this context means ***persistent*** huge pages configured via a setting in `/etc/sysctl.conf`: `vm.nr_hugepages=17408` (17GB)
  - Don't forget to turn off transparent huge pages
- Impact of turning huge pages on: ***ca. 7% more throughput*** in our tested workload
  - See also the tuning results chart

- ***Question***: Why 17GB?
- ***Answer***: shared buffers consume 16GB, plus some extra memory required by PostgreSQL for other purposes, and some extra "headroom" for safety reasons
- If you want to know exactly how much memory PostgreSQL uses, look at the `/proc/[PID]/task/[TID]/status` file and search for the `VmPeak` entry


PostgreSQL tuning results
Aggregate improvements: 45% more throughput
Relative NOPM
150%
140%
130%
120%
110%
100%
90%
80%
70%
60%
50%
Huge Pages
baseline
1GBswap
1GBswap_16GBbuf
1GBswap_16GBbuf_16GBwal
1GBswap_16GBbuf_16GBwal_noav
1GBswap_16GBbuf_16GBwal_noav_nosc
1GBswap_16GBbuf_16GBwal_noav_nosc_icct
1GBswap_16GBbuf_16GBwal_noav_nosc_icct_vmsw5
1GBswap_16GBbuf_16GBwal_noav_nosc_icct_vmsw5_vmdirb
1GBswap_16GBbuf_16GBwal_noav_nosc_icct_vmsw5_vmdirb_ra
1GBswap_16GBbuf_16GBwal_noav_nosc_icct_vmsw5_vmdirb_ra_dio
1GBswap_16GBbuf_16GBwal_noav_nosc_icct_vmsw5_vmdirb_ra_dio_hp
1GBswap_16GBbuf_16GBwal_noav_nosc_icct_vmsw5_vmdirb_ra_dio_hp_noag_smc
1GBswap_16GBbuf_16GBwal_noav_nosc_icct_vmsw5_vmdirb_ra_dio_hp_lsmc

Throughput recommendation #5



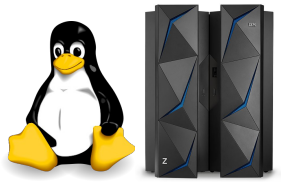
- Throughput recommendation #5: **for a smaller number of IFLs and many parallel users, consider lowering the kernel** *scheduler's migration cost*

- The test runs with the Intel® x86 related parameters lead us to perform some experiments with ***smaller migrations costs*** for the kernel scheduler
  - `kernel.sched_migration_cost_ns` – sets the number of nanoseconds the kernel will ***wait before considering moving*** a thread to another CPU

- The ***higher*** this migration cost is, the ***longer*** the scheduler will wait before considering moving a thread to another CPU
  - Makes a lot of sense for large Linux images and / or scattered Linux images that span multiple PU chips and / or multiple nodes or even multiple drawers in the z13 topology

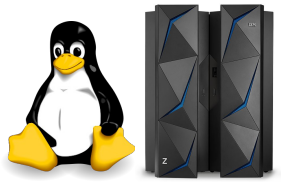
- However, our test runs were based on a 4 IFL configuration and therefore all CPUs did actually fit on the ***same PU chip***
  - Verified with `lscpu --extended`

Throughput recommendation #5, *continued*



- ***Important*** piece of information ***#1***: all cores on the same PU chip share the ***same L3 cache*** (and L4 cache, of course)
- ***Important*** piece of information ***#2***: PostgreSQL spawns a new process for each virtual user – this means we had ***lots of Linux processes***
- ***Important*** piece of information ***#3***: although the system was heavily loaded, there was still quite some ***idle time***
- Lowering the scheduler's migration cost by one order of magnitude ***increases the throughput by almost 9%***
  - Add the following to `/etc/sysctl.conf`:
    `kernel.sched_migration_cost_ns=50000`

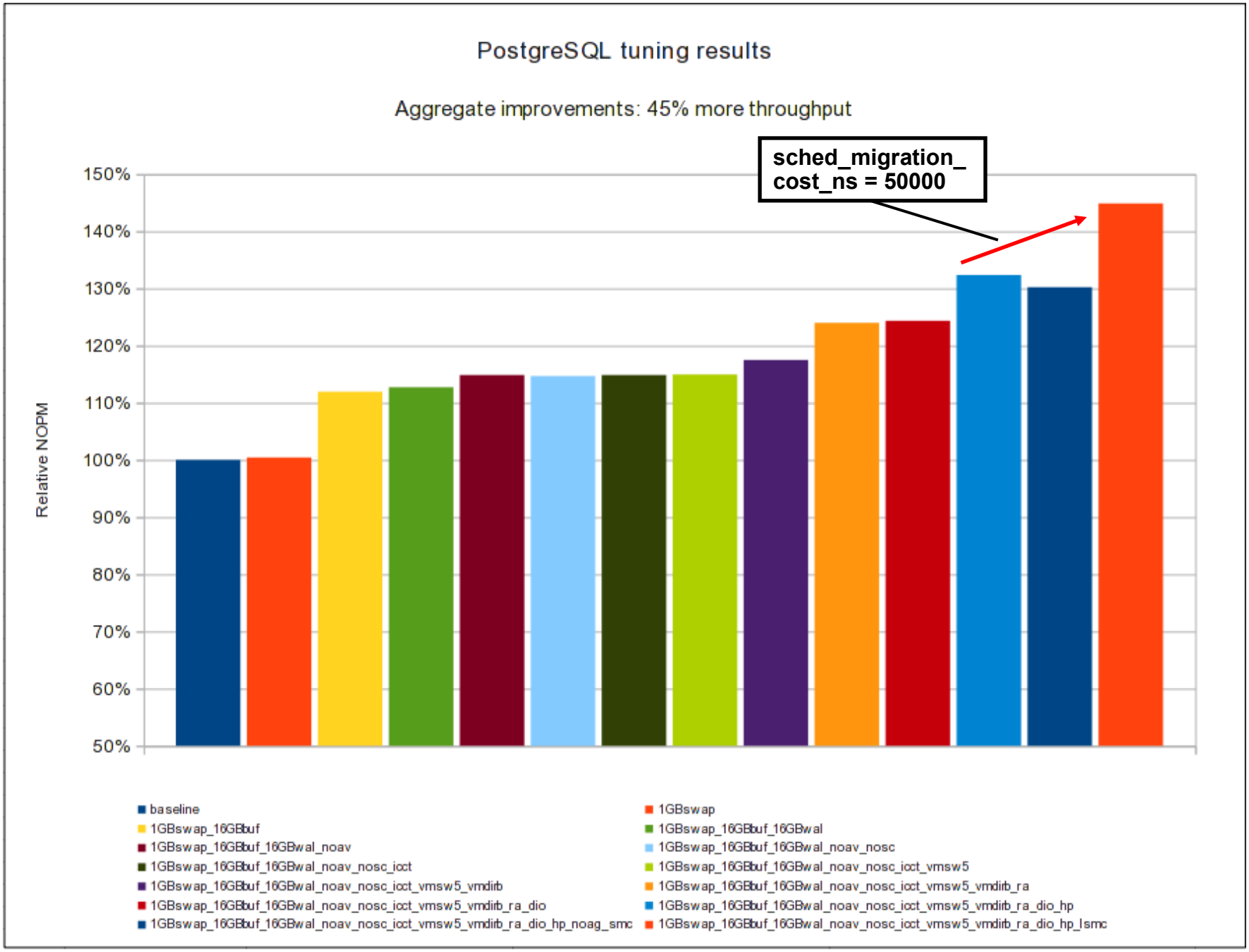
Throughput recommendation #5, *continued*



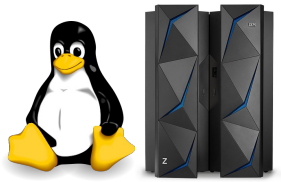
- This in turn means that in our particular configuration, it is ***not beneficial for performance*** trying to let the individual PostgreSQL processes run as long as possible on the same CPU / core
  - Even if they are dispatched on another core, they are still on the same PU chip, meaning either the L3 cache still contains a good amount of relevant information for the process or the L3 cache is totally flooded anyway

- One ***hint*** that the Linux image is performing ***more useful work*** with this setting applied is that the amount of ***user time increases*** significantly
  - Can be verified with `top` and / or `sar` data

- ***Important***: don't simply apply this setting without testing for large Linux images and / or Linux images that have their CPUs scattered all over the z13 topology and / or environments where you only have a very small amount of parallel users
  - Verify the current topology with `lscpu --extended`
  - *Small amount* of users in this context means #users = #cores

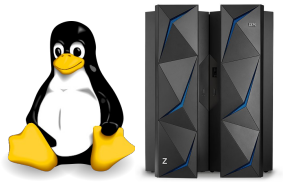

PostgreSQL tuning results
Aggregate improvements: 45% more throughput
sched_migration_
cost_ns = 50000
Relative NOPM
150%
140%
130%
120%
110%
100%
90%
80%
70%
60%
50%
baseline
1GBswap
1GBswap_16GBbuf
1GBswap_16GBbuf_16GBwal
1GBswap_16GBbuf_16GBwal_noav
1GBswap_16GBbuf_16GBwal_noav_nosc
1GBswap_16GBbuf_16GBwal_noav_nosc_icct
1GBswap_16GBbuf_16GBwal_noav_nosc_icct_vmsw5
1GBswap_16GBbuf_16GBwal_noav_nosc_icct_vmsw5_vmdirb
1GBswap_16GBbuf_16GBwal_noav_nosc_icct_vmsw5_vmdirb_ra
1GBswap_16GBbuf_16GBwal_noav_nosc_icct_vmsw5_vmdirb_ra_dio
1GBswap_16GBbuf_16GBwal_noav_nosc_icct_vmsw5_vmdirb_ra_dio_hp
1GBswap_16GBbuf_16GBwal_noav_nosc_icct_vmsw5_vmdirb_ra_dio_hp_noag_smc
1GBswap_16GBbuf_16GBwal_noav_nosc_icct_vmsw5_vmdirb_ra_dio_hp_lsmc

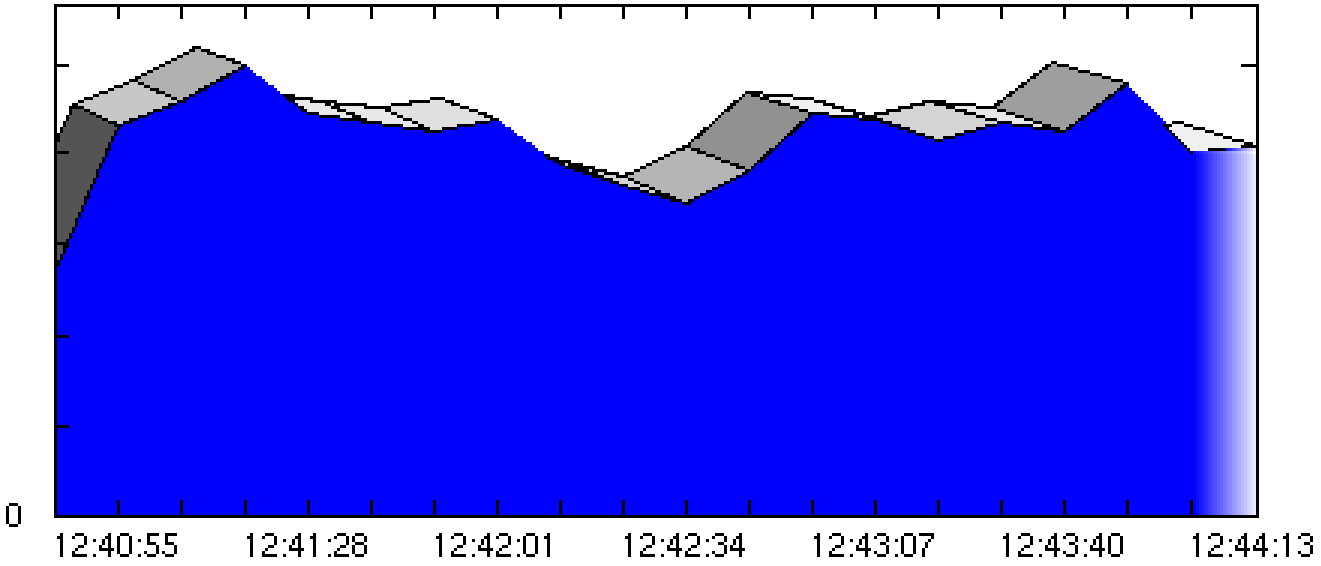
Agenda



- Some background information on PostgreSQL
- Performance measurement and tuning approach
- Observations and recommendations
  - Throughput
  - Response time
  - General
- Summary

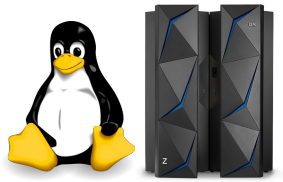
Throughput graph(*) *before* configuration changes

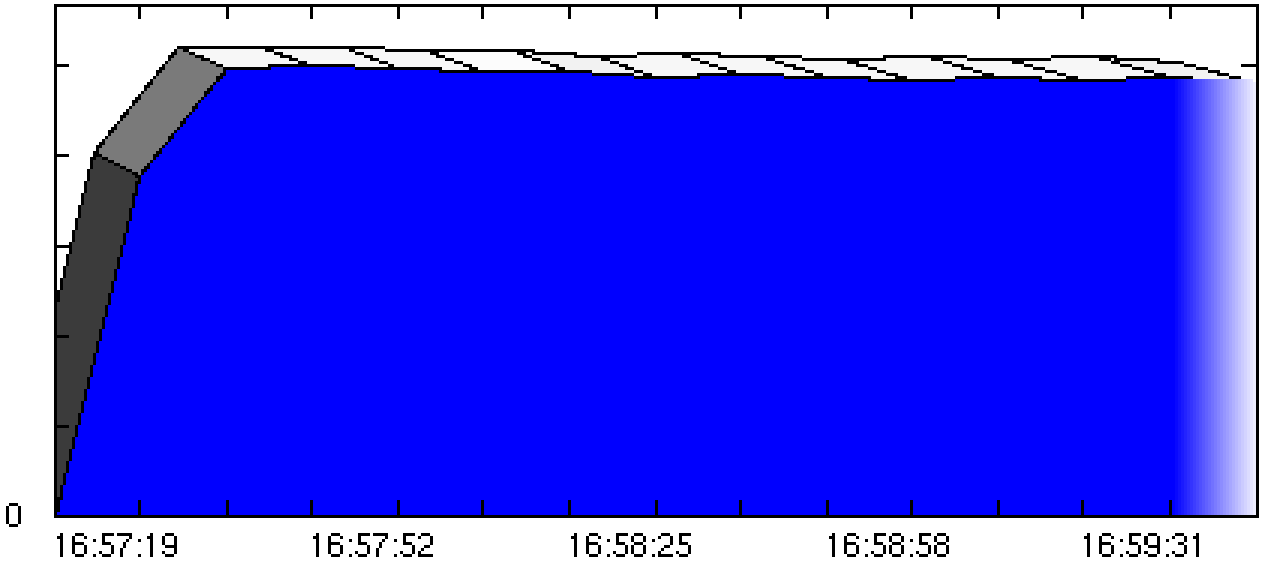




Note: Of course, response time is not equal to throughput. However, from the throughput graph, one can conclude that the response times must have been very shaky, because the amount of virtual users didn't change after the ramp-up phase.

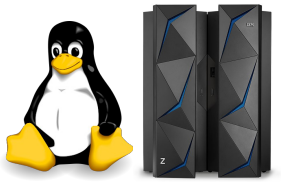
Throughput graph(*) *after* configuration changes





Note: A stable overall throughput graph is not an absolute guarantee that the response times of the individual virtual users were 100% consistent during the entire test run. It is, however, a very strong indicator.

Response time recommendation #1



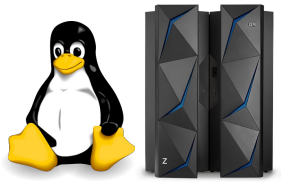
- Response time recommendation #1: **adjust the kernel's settings regarding the *writeback* of dirty pages**

- Default values of these settings are ***very high***:
  - `vm.dirty_background_ratio=10` (10% of 64GB = 6.4GB)
  - `vm.dirty_ratio=20` (20% of 64GB = 12.8GB)

- Our recommendation is to ***lower*** those 2 values in `/etc/sysctl.conf`:
  - `vm.dirty_background_bytes=67108864` (64MB)
  - `vm.dirty_bytes=536870912` (512MB)

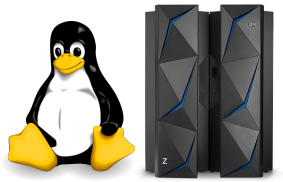
- This recommendation is not only ***improving overall throughput by ca. 2%***, it also helps greatly to ***avoid burst situations*** in the disk I/O subsystem
  - Without adjusting these settings, you will see spikes in the amount of pages / kilobytes written to disk per second

Response time recommendation #1, *continued*



- Applying the mentioned values leads to ***more consistent response times*** for end users, since I/O bursts are smoothed out
  - Throughput is not the only important metric when it comes to performance
- Important side-effect: these settings ***speed up the restore*** of a 256GB database snapshot from ca. ***20 minutes*** to ca. ***13 minutes***
  - Great for recovery situations
- Exactly ***how much*** you lower those values is not particularly important, but it is important to lower them ***significantly*** compared to the defaults
  - For example, experiments with 32MB, 64MB, and 128MB for `vm.dirty_background_bytes` did not lead to significant changes for the duration of the database restore, but 512MB turned out to increase the duration again
  - 64MB seems to be a reasonable value for `vm.dirty_background_bytes` and is also recommended in other PostgreSQL-related publications

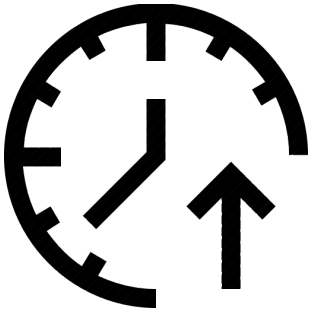
Response time recommendation #2



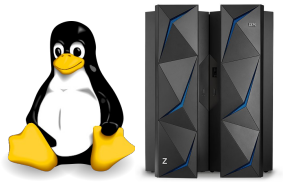
- Response time recommendation #2: **consider applying** ***a number of PostgreSQL settings*** **that smooth out end user response times**

- There are a number of PostgreSQL settings that do not actually increase throughput, but help to ***smooth out*** end user response times
  - Remember: throughput is not the only important metric when it comes to performance

- Parameter #1: `checkpoint_completion_target`
  - In order to change this setting, you have to edit `/etc/postgresql/9.5/main/postgresql.conf`
  - Default value for this parameter is `0.5`
  - Value recommended by many PostgreSQL-related Internet sources: `0.9`
  - ***Pro***: reduces the I/O load from checkpoints by spreading the checkpoint over a larger period of time
  - ***Con***: prolonging checkpoints affects recovery time in case of a failure



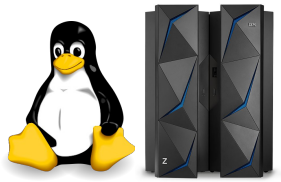
Response time recommendation #2, *continued*



- Parameter #2: `max_wal_size`
  - In order to change this setting, you have to edit `/etc/postgresql/9.5/main/postgresql.conf`
  - Default value for this parameter is **`1GB`**
  - Value recommended by many PostgreSQL-related Internet sources: **`16GB`**
  - ***Pro***: checkpoints occur less frequently (check `/var/log/postgresql/postgresql-9.5-main.log`), because more *Write Ahead Log* (WAL) can be written per checkpoint
  - ***Con***: less frequent checkpoints affect recovery time in case of a failure

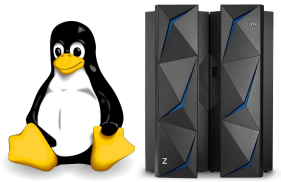
- Parameter #3: `wal_buffers`
  - In order to change this setting, you have to edit `/etc/postgresql/9.5/main/postgresql.conf`
  - Default value for this parameter is **`-1`**
  - Value recommended by many PostgreSQL-related Internet sources: **`16MB`**
  - ***Pro***: less physical writes to the disks due to increased buffering of WAL data
  - ***Con***: higher amount of lost transactions in case of a failure

Response time recommendation #2, *continued*



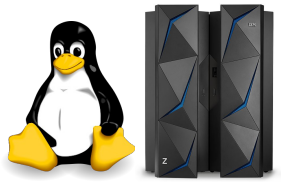
- Parameter #4: `synchronous_commit`
  - In order to change this setting, you have to edit `/etc/postgresql/9.5/main/postgresql.conf`
  - Default value for this parameter is **`on`**
  - Value recommended by many PostgreSQL-related Internet sources: **`off`**
  - ***Pro***: slightly improves response times, since success is reported to the client before the transaction is guaranteed to be written to disk
  - ***Con***: can lead to loss of transactions
  - We do ***not recommend*** turning `synchronous_commit` off, since (a) it does not increase performance in our test cases and (b) there is a risk of loss of end user transactions

Agenda



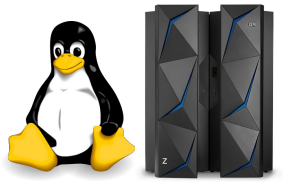
- Some background information on PostgreSQL
- Performance measurement and tuning approach
- Observations and recommendations
  - Throughput
  - Response time
  - General
- Summary

General recommendation #1



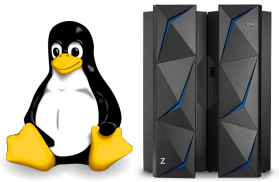
- General recommendation #1: **add some** *swap space*
- In general, it is advisable to have ***at least a small amount*** of swap space available, in order to be prepared when there are ***spikes*** in memory consumption
  - Otherwise, you will see activity from the Linux OOM killer
- In our series of test runs, there were combinations of parameters / settings when turning swap space on ***increased throughput***, and there were combinations of parameters / settings where having swap space turned on or off ***did not make any difference at all*** regarding database throughput
  - At least turning swap space on did not hurt performance
- Even if it doesn't have a positive impact on performance, having swap space turned on has ***more benefits*** than disadvantages

General recommendation #2



- General recommendation #2: **use *Direct I/O* for the transaction logs**
- ***Question***: Why should you use Direct I/O for your database workloads?
- ***Answer***: Because it greatly ***improves overall reliability*** of your database workload's operations
  - By using Direct I/O, the database *bypasses* the Linux page cache and writes directly to the disks, thus avoiding loss of transactions in case of a failure
- The only question that remains is ***how much of an impact*** does the usage of Direct I/O have on performance
- In our test cases, the impact was ***way less than one percent***
  - This is usually considered *white noise* in benchmarking

General recommendation #3



- General recommendation #3: **be *careful* with tips for Intel x86 Linux and / or different kernel versions**

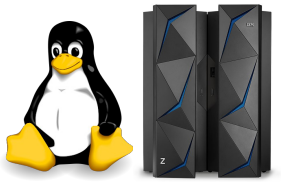
- Example: on the PostgreSQL ***performance mailing list***, there is a popular post called *Two Necessary Kernel Tweaks for Linux Systems*
  - https://www.postgresql.org/message-id/50E4AAB1.9040902@optionshouse.com

- Users reported throughput increases of ***up to 30%*** with the mentioned settings
  - However, those tips were (a) for Linux on Intel x86 and (b) very probably for a kernel version different from the one in our test runs

- In our environment, the mentioned settings ***decreased*** the throughput by ***ca. 2%***
  - Settings were related to the Linux scheduler: `kernel.sched_migration_cost` and `kernel.sched_autogroup_enabled`

- Therefore: ***be careful*** with tips for Intel x86 and / or different kernel versions, they might have ***opposite effects*** when applied to your environment

General recommendation #4



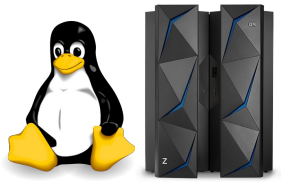
- General recommendation #4: *separate* **the database data files from the database transaction log files**

- ***General recommendation*** for database workloads, not specific for PostgreSQL
  - Holds true for at least all relational databases

- Performance gain in the sandbox environment: ***ca. 10% more throughput***
  - Reason: (a) separate the different I/O behavior of the data files (random r/w) from the I/O behavior of the transaction log files (sequential writes) and (b) better overall usage of the storage server infrastructure due to more disks (1 disk for data + logs vs. 1 disk for data + 1 disk for logs)

- Performance gain in the large environment: ***close to 0%***
  - One possible reason for the 0% gain in performance in the large environment is that we used ***many different*** LUNs for our LVMs and ***each LUN by itself is spread*** across many physical disks by means of DS8K storage pool striping

General recommendation #4, *continued*

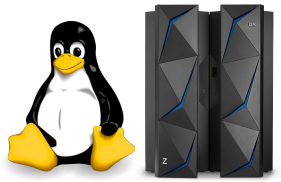


- ***Nevertheless***, the recommendation is to separate the data files from the transaction log files due to the ***unpredictable nature*** of your to-be-deployed database workload
  - Random reads vs. sequential reads, large hit ratio vs. small hit ratio, etc.
  - And: separating the data files from the transaction log files does not hurt performance

- ***Important***: make sure that the disks for the log files are ***physically different*** from the disks for the data files, otherwise you will not see any benefit at all
  - DS8K: if you cannot guarantee physically different disks, use at least disks from (a) different storage servers and (b) different extent pools

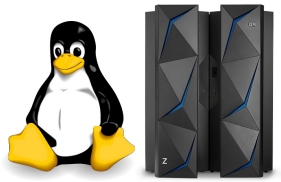
- Easy to implement in PostgreSQL: just ***create a symbolic link*** from the original transaction log file location to the new location
  - Blog post: *How To Move a PostgreSQL Data Directory to a New Location on Ubuntu 16.04*
    https://www.digitalocean.com/community/tutorials/how-to-move-a-postgresql-data-directory-to-a-new-location-on-ubuntu-16-04

Agenda



- Some background information on PostgreSQL
- Performance measurement and tuning approach
- Observations and recommendations
  - Throughput
  - Response time
  - General
- Summary

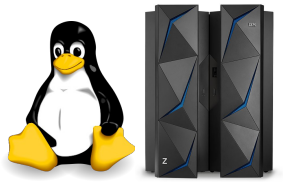
Summary



- PostgreSQL is a ***pretty mature*** Open Source database
  - In all of our test runs, we did not experience a single failure / crash or anything similar
- If you follow the recommendations in this presentation, you can ***greatly improve performance*** of your PostgreSQL database on Linux on Z
  - Examples: shared buffers, read ahead, huge pages, scheduler settings
- There are some ***additional recommendations*** that you should also follow – even if they don't have a direct impact on performance
  - Examples: swap space, usage of Direct I/O, writeback of dirty pages
- ***Outlook***: we will perform ***even more tests*** with PostgreSQL and with other databases
  - Inside a Docker® container
  - Inside an IBM *Secure Service Container* (SSC)

Thank you

Resources



- Linux on IBM Z
  https://www.ibm.com/systems/z/os/linux
- Linux on Z and LinuxONE documentation
  https://www.ibm.com/developerworks/linux/linux390/documentation_dev.html
- Linux on Z and LinuxONE tuning hints & tips
  https://www.ibm.com/developerworks/linux/linux390/perf/
- PostgreSQL homepage
  https://www.postgresql.org
- PostgreSQL documentation
  https://www.postgresql.org/docs
- PostgreSQL Tuning Wiki
  https://wiki.postgresql.org/wiki/Tuning_Your_PostgreSQL_Server