Benchmark Report

Splendid Data PostgresPURE
on IBM Power System S822L

splendid data
GRIP ON POSTGRES

IBM
Power Systems
Open Innovation to Put Data to Work

Tectrade™

Executive summary

Open Source is becoming more and more a "de facto" standard in the IT landscape of enterprises and large institutions. These enterprises and large institutions more and more are looking for valuable solutions to replace traditional software vendors, which cause high TCO's due to license policies and lock-in, by proven Open Source alternatives.

IBM France and Splendid Data performed a benchmark to demonstrate that a complete (100%) Open Source stack can deliver an excellent price-performance ratio.

The Open Source stack is built on an IBM POWER8 server using PostgreSQL as part of Splendid Data's PostgresPURE, RedHat Enterprise Linux operating system and PowerKVM hypervisor.

Finally these results of the benchmark are compared with the performance of EnterpriceDB's Postgres Plus Advance Server (commercial license, NOT Open Source) on a X86 Xeon Intel server (Benchmark reference: EDB Blog, Jason Davis, June 19, 2015).

Conclusions

- The high scale factor, this means a large data set of 45 GB, results show a 2x better performance of PostgresPURE (based on PostgreSQL 9.4.4) in combination with Power S822L and compared to Postgres Plus Advanced Server on an x86 Intel Xeon.
- The high scale factor, this means a large data set of 45 GB, results show a 1.5x better performance of PostgresPURE (based on PostgreSQL 9.5 Beta 1) in combination with Power S822L and compared to Postgres Plus Advanced Server on an x86 Intel Xeon.
- The price per transaction on an IBM Power S822L server with PostgresPURE based on PostgreSQL 9.4.4 is approximately 12x cheaper as compared to an x86 Intel Xeon server with Postgres Plus Advanced Server.
- The price per transaction on an IBM Power S822L server with PostgresPURE based on PostgreSQL 9.5 Beta 1 is approximately 8x cheaper as compared to an x86 Intel Xeon server with Postgres Plus Advanced Server.

The combination IBM POWER8 server in combination with Splendid Data's PostgresPURE proved to be an excellent combination not only performance wise but also cost wise. This combined with the ability to support enterprises and large institutions to migrate from expensive alternatives like Intel servers and/or Oracle databases and other Oracle software components like Weblogic.

Executive summary

1. Introduction

IBM France and Splendid Data performed a benchmark to demonstrate that a complete Open Source stack can deliver an excellent price-performance ratio.

The Open Source stack is built on an IBM POWER8 server using PostgreSQL as part of Splendid Data's PostgresPURE, RedHat Enterprise Linux operating system and PowerKVM hypervisor.

In close cooperation both parties proved that with less hardware resources, the key benefit of the POWER8 processor, in combination with the proper Open Source products you can create an extremely powerful combination for enterprises with an excellent price-performance ratio, compared to an x86 Intel Xeon server in combination with EnterpriseDB's Postgres Plus Advanced Sever which is a commercial product (not Open Source).

Special thanks go to Patrick Briant (IBM Innovation Center Paris), Sébastien Chabrolles (IBM Client Center Montpellier) and Henk Waanders (IBM Innovation Center Benelux) for their great help during all the tests we did in close cooperation.

2. Purposes of the benchmark

1. Get various performance results for PostgreSQL, as part of PostgresPURE, running on IBM POWER8.
2. Demonstrate the Power of Open Source on IBM POWER8 compared to x86 Xeon Intel.
3. Demonstrate commercial efficiency of IBM POWER8 server as PostgreSQL DB node.

3. Environment

3.1 Hardware

IBM Power System S822L including

- 2 x POWER8 CPU (10 cores, 3.42 GHz).
- 256 GB RAM.
- 4 x 387 GB SAS Disk.

On IBM POWER architecture, the focus of processor design is not only to increase core performance, but also to deliver more cores per processor chip and to deliver more hardware threads in each core, using Simultaneous Multi-Threading. SMT improves system performance by increasing the throughput of workloads with large or frequently changing working sets, such as database servers and Web servers by:

- Ability to invoke up to 8 concurrent threads per core.
- Ability to schedule threads across core.
- Ability to designate/change primary thread.

3.2 Hypervisor

PowerKVM

PowerKVM is the open virtualization choice for Power scale-out Linux Systems.

- Optimize Linux workload consolidation at a lower cost of ownership.
- Exploit the advantage of performance, scalability and security built into Linux and the Kernel-Based Virtual Machine (KVM) hypervisor.
- Avoids high cost proprietary x86 virtualization.
- Managed just like any other KVM host – OpenStack, libvirt and open Linux tools help you avoid vendor lock-in.

KVM enables single cross platform virtualization, which simplifies management.

Kimchi

Kimchi is an open source HTML5 based management tool for PowerKVM. It is designed to make it as easy as possible to get started with KVM and create your first guest.
Kimchi runs as a daemon on the hypervisor host. It manages PowerKVM guests through libvirt. The management interface is accessed over the web using a browser that supports HTML5.

3.3 Operating system

Red Hat Enterprise Linux (RHEL) is an Open Source Linux distribution developed by Red Hat and targeted toward the commercial market. RHEL 7.1 Little Endian for IBM POWER architecture was used during this benchmark.

3.4 Software

PostgresPURE is an Open Source Postgres distribution developed by Splendid Data and targeted toward the commercial market. The foundation is the Postgres database version, which is released by the PostgreSQL.org community. To which Splendid Data adds a large variety of essential tools, including tools for the purpose of the administration and the monitoring of the database – and all of this on Open Source basis. PostgresPURE is intended to act as a fundamental basis for organizations, identical to Oracle, but entirely Open Source based.

100% Open Source

Making use of PostgresPURE offers all the benefits of Open Source Software. The following applies to Splendid Data: Open Source = 100% Open Source combined with 7x24 technical support and release management.

PostgreSQL (as part of PostgresPURE) is a powerful, open source object-relational database system. It has more than 15 years of active development and a proven architecture that has earned it a strong reputation for reliability, data integrity, and correctness. It runs on all major operating systems. It is fully ACID compliant,

has full support for foreign keys, joins, views, triggers, and stored procedures (in multiple languages). It includes most SQL:2008 data types, including INTEGER, NUMERIC, BOOLEAN, CHAR, VARCHAR, DATE, INTERVAL, and TIMESTAMP. It also supports storage of binary large objects, including pictures, sounds, or video. It has native programming interfaces for C/C++, Java, .Net, Perl, Python, Ruby, Tcl, ODBC, among others.

An enterprise class database, PostgreSQL boasts sophisticated features such as Multi-Version Concurrency Control (MVCC), point in time recovery, table spaces, asynchronous replication, nested transactions (save points), online/hot backups, a sophisticated query planner/optimizer, and write ahead logging for fault tolerance. It supports international character sets, multi-byte character encodings, Unicode, and it is local-aware for sorting, case-sensitivity, and formatting. It is highly scalable both in the sheer quantity of data it can manage and in the number of concurrent users it can accommodate. There are active PostgreSQL systems in production environments that manage in excess of 4 terabytes of data.

4. Details of technical environment

4.1 PostgreSQL versions

During benchmark we have focused on the versions 9.4.4 and 9.5 Beta 1 of PostreSQL as part of PostgresPURE. We have chosen for version 9.4.4 because this version is used in other benchmarks on x86 Intel Xeon technology, which significantly differs from the IBM POWER8 technology.

PostgreSQL 9.4

Version 9.4.4 is officially released on June 12th, 2015. It is a mature, stable and proven release on different operating systems and CPU technologies.

Regarding the benchmark the regular version of PostgreSQL is used without any patches and the usual parameters for the POWER8.

PostgreSQL 9.4.4 (pre built configuration):

$ ./configure --prefix=/usr/pgpure/postgres-9.4.4/ --enable-nls --with-libxml -with-libxslt --with-CC=gcc CFLAGS=-O3 -g -pipe -Wall -D_FORTIFY_SOURCE=2 -fexceptions -fstack-protector-strong --param=ssp-buffer-size=4 -m64 -mcpu=power8 -mtune=power8 -DLINUX_OOM_SCORE_ADJ=0

PostgreSQL 9.5 Beta 1

Version 9.5 Beta 1 is officially released on October 8th, 2015. Although it is still a first beta release it is already more than stable enough for using it in a benchmark. The PostgreSQL community officially released the second beta version on November 12th, 2015, but at that time the benchmark was already finalized. It is the aim of the PostreSQL community to release the first production version of 9.5 somewhere mid January 2016.

Regarding the benchmark the regular version of PostgreSQL is used without any patches and the usual parameters for the POWER8.

PostgreSQL 9.5 Beta 1 (pre built configuration):

./configure --prefix=/usr/pgpure/postgres-9.5beta/ --enable-nls --with-libxml -with-libxslt --with-CC=gcc CFLAGS=-O3 -g -pipe -Wall -D_FORTIFY_SOURCE=2 -fexceptions -fstack-protector-strong --param=ssp-buffer-size=4 -m64 -mcpu=power8 -mtune=power8 -DLINUX_OOM_SCORE_ADJ=0

4.2 Operating system tuning

RHEL Profile: a newly performance control subsystem "Tuned" is used that monitors the use of system components and dynamically tunes system settings based on the monitoring information. Dynamic tuning accounts for the way that various system components are used differently throughout the uptime for any given system. For example, the hard drive is used heavily during start-up and login, but is barely used later. Similarly, the CPU and network devices are used differently at different times. The subsystem "Tuned" monitors the activity of these components and reacts to changes in their use.

The best results during PostgreSQL testing were obtained with profile "throughput-performance":

```
vm.dirty_background_bytes = 0
vm.dirty_ratio = 40
vm.dirty_background_ratio = 10
vm.hugetlb_shm_group = 0
vm.dirty_bytes = 0
vm.swappiness = 10
vm.hugepages_treat_as_movable = 0
vm.zone_reclaim_mode = 0
vm.drop_caches = 0
kernel.sched_autogroup_enabled = 0
```

4.3 Virtualization

The benchmark is ran on a single VM and a double VM to show the effect of how PowerKVM shares physical resources (see Appendix for the PowerKVM configuration).

Single VM

Linux iccparis.localdomain 3.10.0-229.ael7b.ppc64le #1 SMP Fri Jan 30 12:03:50 EST 2015 ppc64le ppc64le ppc64le GNU/Linux.

- Architecture: ppc64le
- Byte Order: Little Endian
- CPU(s): 160
- On-line CPU(s) list: 0-159
- Thread(s) per core: 8
- Core(s) per socket: 5
- Socket(s): 4
- NUMA node(s): 1
- Model: IBM pSeries (emulated by qemu)
- L1d cache: 64K

- L1i cache: 32K
- NUMA nodeo CPU(s): 0-159

4.4 PostgreSQL tuning parameters

The following parameters were added to default PostgreSQL server configuration:

max_connections = 300

The max_connections sets the maximum number of client connections allowed. This is very important to some of the parameters listed below (particularly work_mem) because there are some memory resources that are or can be allocated on a per-client basis, so the maximum number of clients suggests the maximum possible memory use. Generally, PostgreSQL on good hardware can support a few hundred connections.

shared_buffers = 50GB

The shared_buffers configuration parameter determines how much memory is dedicated to PostgreSQL to use for caching data. A reason the defaults are low is because on some platforms (like older Solaris versions and SGI), having large values requires invasive action like recompiling the kernel.

effective_cache_size = 150GB

The effective_cache_size should be set to an estimate of how much memory is available for disk caching by the operating system and within the database itself, after taking into account what is used by the Operating System (OS) itself and other applications. This is a guideline for how much memory you expect to be available in the OS and PostgreSQL buffer caches, not an allocation! This value is used only by the PostgreSQL query planner to figure out whether the plans it is considering would be expected to fit in RAM or not. If it is set too low, indexes may not be used for executing queries the way you'd expect.

work_mem = 87381kB

If you do a lot of complex sorts, and have a lot of memory, then increasing the work_mem parameter allows PostgreSQL to do larger in-memory sorts that, unsurprisingly, will be faster than disk-based equivalents. This size is applied to each and every sort done by each user, and complex queries can use multiple working memory sort buffers. Set it to 50MB, and have 30 users submitting queries, and you are soon using 1.5GB of real memory.

maintenance_work_mem = 2GB

Specifies the maximum amount of memory to be used by maintenance operations, such as VACUUM, CREATE INDEX, and ALTER TABLE ADD FOREIGN KEY. It defaults to 16 megabytes (16Mb). Since only one of these operations can be executed at a time by a database session and an installation normally does not have many of them running concurrently, it is safe to set this value significantly larger than work_mem. Larger settings might improve performance for vacuuming and for restoring database dumps.

checkpoint_completion_target = 0.9

PostgreSQL writes new transactions to the database in files called WAL segments that are 16MB in size. Every time checkpoint_segments worth of these files have been written, by default 3, a checkpoint occurs. Checkpoints can be resource intensive, and on a modern system doing one every 48Mb will be a serious performance bottleneck. Setting checkpoint_segments to a much larger value improves that.

wal_buffers = 16MB

Increasing wal_buffers from its tiny default of a small number of kilobytes is helpful for write-heavy systems. Benchmarking generally suggests that just increasing to 1MB is enough for some large systems, and given the amount of RAM in modern servers allocating a full WAL segment (16MB, the useful upper-limit here) is reasonable. Changing wal_buffers requires a database restart. Starting with PostgreSQL 9.1 wal_buffers defaults to being 1/32 of the size of shared_buffers, with an upper limit of 16MB (reached when shared_buffers=512MB).

default_statistics_target = 100

The database software collects statistics about each of the tables in your database to decide how to execute queries against it. If you're not getting good execution query plans particularly on larger (or more varied) tables you should increase default_statistics_target then ANALYZE the database again (or wait for autovacuum to do it for you).

The starting default_statistics_target value was raised from 10 to 100 in PostgreSQL 8.4. Increases beyond 100 may still be useful, but this increase makes for greatly improved statistics estimation in the default configuration. The maximum value for the parameter was also increased from 1,000 to 10,000 in 9.4.

5. pgbench benchmarking tool

5.1 Description

pgbench is a program for running benchmark tests on PostgreSQL. It runs the same sequence of SQL commands over and over, possibly in multiple concurrent database sessions, and then calculates the average transaction rate (transactions per second). By default, pgbench tests a scenario that is loosely based on TPC-B, involving five SELECT, UPDATE, and INSERT commands per transaction. However, it is easy to test other cases by writing your own transaction script files.

Different options are used during database initialization and while running benchmarks; some options are useful in both cases.

5.2 pgbench scaling factor

The default TPC-B-like transaction test requires specific tables to be set up beforehand. pgbench should be invoked with the -i (initialize) option to create and populate these tables. pgbench -i creates four tables pgbench_accounts, pgbench_branches, pgbench_history, and pgbench_tellers. At the default "scale factor" of 1, the tables initially contain this many rows:

```
table                    # of rows
---------------------------------
pgbench_branches         1
pgbench_tellers          10
pgbench_accounts         100000
pgbench_history          0
```

The option:

```
-s scale_factor
```

Multiply the number of rows generated by the scale factor, creating a larger data set. For example, -s 100 will create 10,000,000 rows in the pgbench_accounts table, instead of the default 100,000 rows for scale factor =1. Default is 1. During the benchmark , the following values where used:

```
-s scale_factor [100, 300, 1000, 3000]
```

Scale factor	Number of rows in pgbench_accounts table	Size of Data set
100	10,000,000	1.5 GB
300	30,000,000	4.5 GB
1,000	100,000,000	15.0 GB
3,000	300,000,000	45.0 GB

5.3 Benchmarking Options

pgbench accepts the following command-line benchmarking arguments:

Number of clients simulated

```
-c clients [2…160]
```

Number of clients simulated, that is, number of concurrent database sessions. Default is 1.

Number of worker threads

```
 -j threads [1..160]
```

Number of worker threads within pgbench. Using more than one thread can be helpful on multi-CPU machines. The number of clients must be a multiple of the number of threads, since each thread is given the same number of client sessions to manage. Default is 1.

Perform select-only transactions

```
-S
```

This is done instead of a TPC-B-like test, just to measure CPU/ memory architecture and avoid influence of disk I/O.

Duration of test

```
-T seconds [300]
```

Run the test for this many seconds, rather than a fixed number of transactions per client. -t and -T are mutually exclusive.

Vacuum all four standard tables

Before running the test. With neither -n nor -v, pgbench will vacuum the pgbench_tellers and pgbench_branches tables, and will truncate pgbench_history: -v

6. Benchmark runs

6.1 Bash scripts

To automate benchmark, a bash shell script was developed to:

To automate benchmark, a bash shell script was developed to:

- Create tmpfs mountpoint size of 50GB.
- Initialize PostgreSQL database (2 different builds).
- Copy configuration options into PostgreSQL configuration file.
- Create pgbench database.
- Initialize pgbench (4 different scale factors).
- Start PostgreSQL.
- Run pgbench:
  - Each test run 5 minutes.
  - 2 different builds.
  - 4 different scale factors.
  - Client amount (-c) from 2 till N (where N=cpucores).
  - 2 variants of threads (-j) 2:1 and 1:1.
- Save results to csv.

6.2 Benchmark preparations

VM tuning

IIC Paris team tuned PowerKVM settings in order to provide maximum performance for a Single VM.

RHEL tuning

RHEL 7.1 kernel parameters were tuned in order to get optimal performance. Performance probes were executed for each performance profile and best profile "throughput-performance" was chosen.

PostgreSQL tuning

Configuration options were adjusted to get maximum performance:

```
max_connections = 300
shared_buffers = 50GB
effective_cache_size = 150GB
work_mem = 87381kB
maintenance_work_mem = 2GB
checkpoint_completion_target = 0.9
wal_buffers = 16MB
default_statistics_target = 100
```

Performance probing

To determine optimal amount of test steps and representative results some preflight probing were done. Each PostgreSQL build was first manually compiled and configured, and then batch script was executed in order to check if results are correctly produced and saved to csv log files.

6.3 Benchmark time

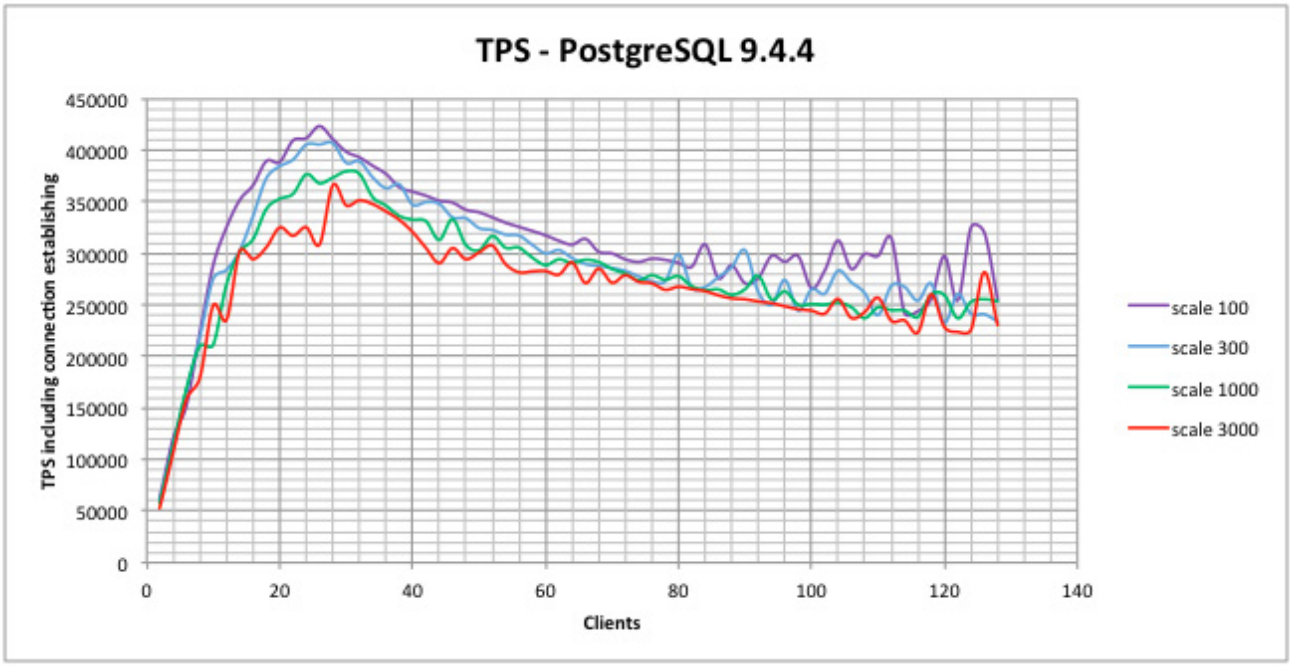
Single VM (6,400 minutes = 107 hours):

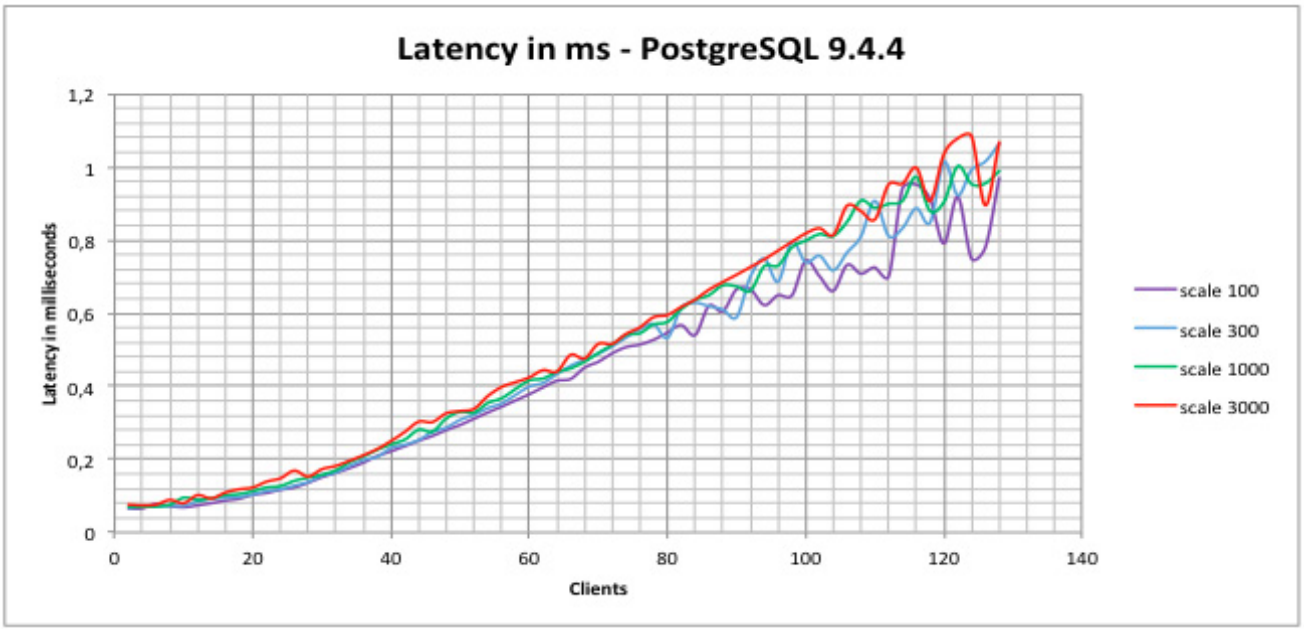
- 2 builds (PostgreSQL 9.4.4 / 9.5 Beta 1).
- 4 scale factors (100, 300, 1,000, 3,000).
- 2 ratios (client:thread ratio 2:1 and 1:1).
- 80 steps (client from 2 till 160 step 2).
- 5 minutes per test.

7. Results

7.1 PostgreSQL 9.4.4 results

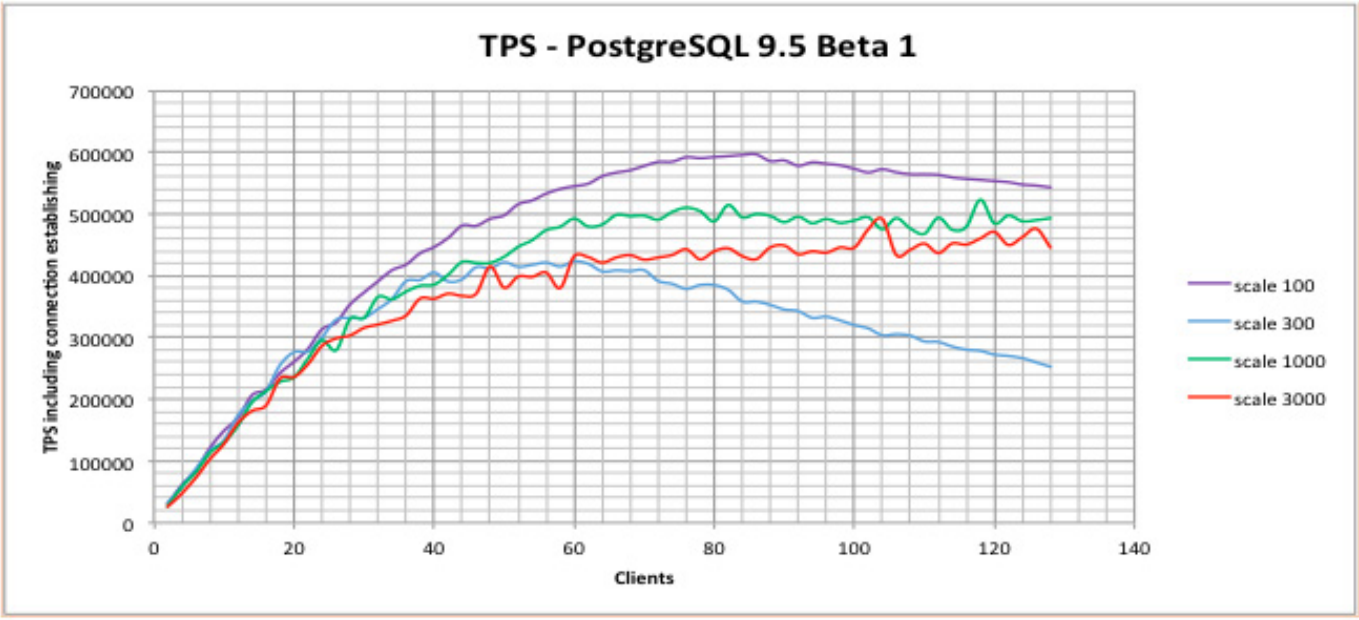
PostgreSQL 9.4.4		Scale 100		Scale 300		Scale 1,000		Scale 3,000	
		1:1	2:1	1:1	2:1	1:1	2:1	1:1	2:1
incl. conn	Clients/Threads	26/26	48/24	28/28	52/26	30/30	52/26	28/28	48/24
	TPS /Latency	423,148/0.123	379,866/0.126	406,589/0.138	377,233/0.138	379,414/0.158	368,224/0.141	365,974/0.153	355,882/0.135
excl. conn	Clients /Threads	124/124	260/130	90/90	292/146	30/30	244/122	28/28	48/24
	TPS /Latency	726,022/0.749	709,271/0.819	435,311/0.589	566,129/1.017	380,335/0.158	446,477/0.859	366,383/0.153	356,229/0.135

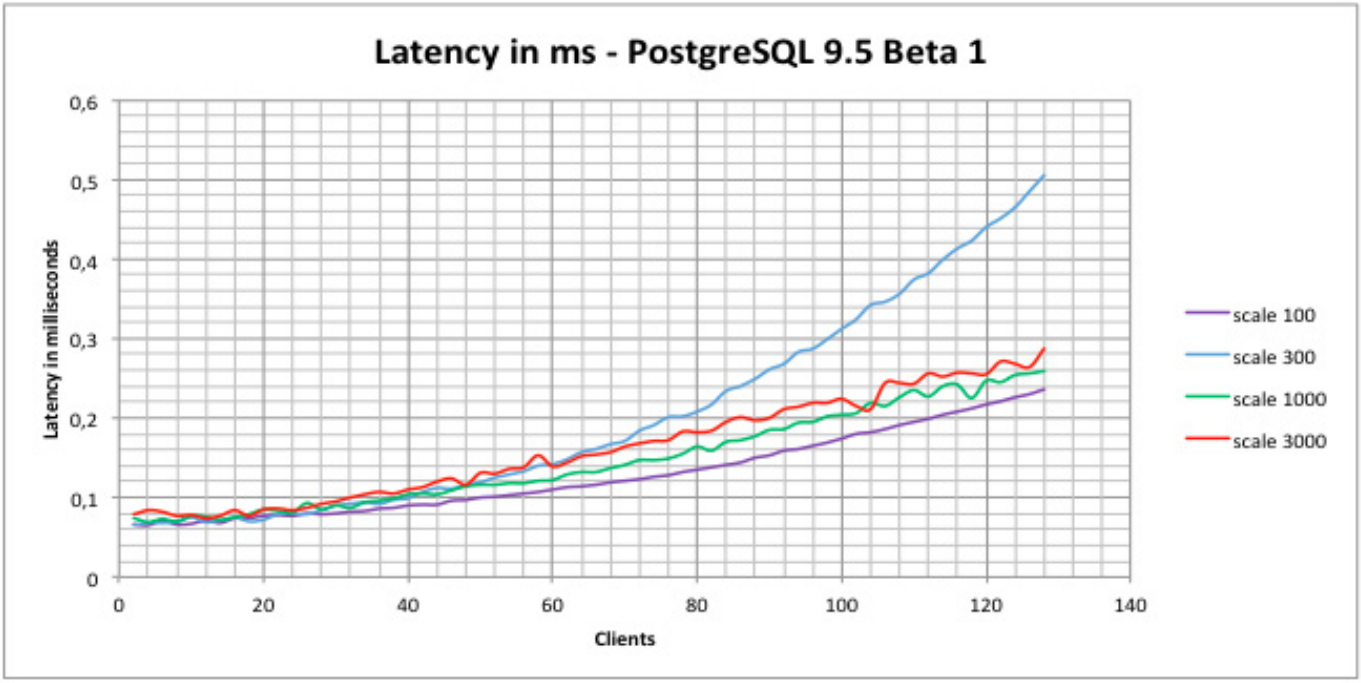




7.2 PostgreSQL 9.5 Beta 1 results

PostgreSQL 9.5 Beta 1		Scale 100		Scale 300		Scale 1,000		Scale 3,000	
		1:1	2:1	1:1	2:1	1:1	2:1	1:1	2:1
incl. conn	Clients/Threads	86/86	80/40	60/60	80/40	118/118	80/40	104/104	144/72
	TPS/Latency	596,590/0.144	614,637/0.135	423,356/0.142	587,628/0.137	523,103/0.225	535,254/0.151	492,072/0.211	460,610/0.316
excl. conn	Clients/Threads	86/86	84/42	60/60	80/40	118/118	80/40	104/104	144/72
	TPS/Latency	596,879/0.144	617,996/0.138	423,606/0.142	590,474/0.137	523,755/0.225	529,554/0.151	493,270/0.211	462,404/0.316





8. Price per transaction ratio's

8.1 Benchmark PostgresPURE (100% Open Source) on a POWER8 server

Hardware specifications:

- IBM Power S822L
- 2 x POWER8 CPU (10 cores, 3.42GHz)
- 256 GB RAM
- 4 x 300 GB SAS Disk
- RHEL 7.1 Little Endian OS
- PowerKVM

List price hardware

€ 38,364.00 excl. VAT and incl. 3 year maintenance (quote IBM France).

Results Single VM:

- 365,974 TPS on PostgesSQL 9.4.4 at scale factor 3,000.
- 492,072 TPS on PostgreSQL 9.5-Beta 1 at scale factor 3,000.

Price per transaction based on Single VM:

- € 0.11 (€ 38,364 / 365,974 TPS) on PostgesSQL 9.4.4 at scale factor 3,000.
- € 0.08 (€ 38,364 / 492,072 TPS) on PostgreSQL 9.5-Beta 1 at scale factor 3,000.

8.2 Benchmark EnterpriseDB Postgres Plus Advanced Server on x86 Intel Xeon server

EnterpriseDB Postgres Plus Advanced Server (PPAS) is a proprietary/commercial license. Benchmark reference: EDB Blog, Jason Davis, June 19, 2015.

Hardware specifications (as used during the benchmark mentioned above):

- 4 x Intel Xeon CPU (15 cores, 2.80 GHz)
- 256 GB RAM
- 4 x 300 GB Disk
- RHEL 7.1 Little Endian OS

List price hardware € 61,179.00 excl. VAT and incl. 3 year maintenance (quote IBM France).

Results:

- 190,000 TPS on PostgreSQL 9.4.4 at scale factor 3,000
- 360,000 TPS on PostgreSQL 9.5 develop at scale factor 3,000

Price per transaction:

- € 0.32 (€ 61,719 / 190,000 TPS) on PostgreSQL 9.4.4 at scale factor 3,000
- € 0.17 (€ 61,719 / 360,000 TPS) on PostgreSQL 9.5 develop at scale factor 3,000

8.3 Comparison Splendid Data PostgresPURE/IBM Power S822L server versus EnterpriseDB PPAS/x86 Intel Xeon server

PostgresPURE based on PostgreSQL 9.4.4 and 9.5 Beta 1.

List price PostgresPURE per CPU € 4,200 excl. VAT (yearly subscription fee including 7x24 technical support and release management).

Splendid Data PostgresPURE/IBM Power8

Price per transaction based on Single VM including PostgresPURE (3 years basis) on a IBM S822L composed of 2 CPU sockets:

- € 0.17 ((€ 38,364 + 3 x 2 x € 4,200) / 365,974 TPS)) on PostgesSQL 9.4.4 at scale factor 3,000.
- € 0.13 ((€ 38,364 + 3 x 2 x € 4,200) / 492,072 TPS)) on PostgreSQL 9.5-Beta 1 at scale factor 3,000.

EnterpriseDB PPAS on PostgreSQL 9.4.4. and PostgreSQL 9.5-A.

List price PPAS per core € 1,800 excl. VAT (yearly subscription fee).

EnterpriseDB PPAS /x86 Intel Xeon server

Price per transaction including EnterpriseDB PPAS (3 years basis) on Lenovo Intel based system composed of 4 sockets for a total of 60 cores:

- € 2.03 ((€ 61,719 + (3 x 60 x € 1,800)) / 190,000 TPS) PostgesSQL 9.4.4 at scale factor 3,000.
- € 1.07 ((€ 61,719 + (3 x 60 x € 1,800)) / 360,000 TPS) PostgesSQL 9.5 develop at scale factor 3,000.

9. Conclusions

In this conclusive paragraph the test results are summarized and a comparison is provided using database technology running on IBM's Power S822L server with a x86 Intel Xeon server.

The best way to compare results is to end up with an overview of price per transaction in both environments (IBM's Power technology and x86 Intel Xeon technology). In order to make a conclusive, encompassing picture, the pricing of the databases running at both platforms is added.
The tests have been performed for a period of 160 hours in total at the POWER8 platform. The toolset used for testing has been "pgbench".

The pricing of PostgresPURE on the POWER8 S822L server (2 CPU's, 10 cores each, 20 cores in total), and of EnterpriseDB's Postgres Plus Advanced Server on an x86 Intel Xeon server (4 CPU's, 15 cores each, 60 cores in total) is added. Please note the following:

- Pricing of EnterpriseDB's Postgres Plus Advanced Server is per core, namely € 1,800 per core.
- Pricing of Splendid Data's PostgresPURE is per CPU: € 4,200 per CPU and independent of the number of cores per CPU.
- Please note that EnterpriseDB's Postgres Plus Advanced Server does NOT provide as an Open Source license, and Splendid Data's PostgresPURE is totally, 100%, Open Source.

Testing is carried out at different scale factors, though the focus centered on the high scale factor tests.

Including the database pricing on the servers, the price per transaction is calculated using different versions of PostgreSQL namely the latest releases of 9.4 and 9.5 Beta 1. Each test compares scale factor 3,000, to target a large data set.

Both server environments were equipped with exactly the same RAM and disk space.

The list prices of the IBM Power S822L server (€ 38,364) and the x86 Xeon Intel server (€ 61,179 based on the specifications as mentioned in the benchmark EDB Blog) servers have been provided by IBM, the prices of databases by Splendid Data.

2x Performance with PostgresPURE based on PostgreSQL 9.4.4

The high scale factor results show a 2x better performance of PostgresPURE (based on PostgreSQL 9.4.4) in combination with Power S822L and compared to Postgres Plus Advanced Server on an x86 Intel Xeon.

1.5x Performance with PostgresPURE based on PostgreSQL 9.5

The high scale factor results show a 1.5x better performance of PostgresPURE (based on PostgreSQL 9.5) in combination with Power S822L and compared to Postgres Plus Advanced Server on an x86 Intel Xeon.

12x Cheaper price per transaction with IBM POWER S822L and PostgreSQL 9.4.4

The price per transaction on an IBM Power S822L server with PostgresPURE based on PostgreSQL 9.4.4 is approximately 12x cheaper as compared to an x86 Intel Xeon server with Postgres Plus Advanced Server.

8x Cheaper price per transaction with IBM POWER S822L and PostgreSQL 9.5

The price per transaction on an IBM Power S822L server with PostgresPURE based on PostgreSQL 9.5 is approximately 8x cheaper as compared to an x86 Intel Xeon server with Postgres Plus Advanced Server.

10. Appendix 1 - PowerKVM configuration

10.1 VM xml configuration

```
<domain type='kvm' id='41'>
  <name>SplendidData1-RHEL7u1LE</name>
  <uuid>04ffb22f-4613-4bf7-8919-4245b4e95302</uuid>
  <metadata xmlns:ns0="https://github.com/kimchi-project/kimchi">
    <ns0:kimchi>
      <ns0:os distro="unknown" version="unknown"/>
    </ns0:kimchi>
  </metadata>
  <memory unit='KiB'>268435456</memory>
  <currentMemory unit='KiB'>268435456</currentMemory>
  <vcpu placement='static'>160</vcpu>
  <cputune>
    <vcpupin vcpu='0' cpuset='0,8'/>
    <vcpupin vcpu='1' cpuset='0,8'/>
…
    <vcpupin vcpu='79' cpuset='40,72'/>
  </cputune>
  <numatune>
    <memory mode='strict' nodeset='0-1'/>
  </numatune>
  <resource>
    <partition>/machine</partition>
  </resource>
  <os>
    <type arch='ppc64' machine='pseries-2.2'>hvm</type>
    <boot dev='hd'/>
    <boot dev='cdrom'/>
  </os>
  <features>
    <acpi/>
    <apic/>
    <pae/>
  </features>
  <cpu mode='host-passthrough'>
    <topology sockets='4' cores='5' threads='8'/>
  </cpu>
  <clock offset='utc'/>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>restart</on_crash>
  <devices>
    <emulator>/usr/bin/qemu-kvm</emulator>
    <disk type='file' device='disk'>
      <driver name='qemu' type='raw' cache='none'/>
      <source file='/dev/local-storage_S822L/04ffb22f-4613-4bf7-8919-4245b4e95302-0.img'/>
      <backingStore/>
      <target dev='sda' bus='scsi'/>
      <alias name='scsi0-0-0-0'/>
```

```
    <address type='drive' controller='0' bus='0' target='0' unit='0'/>
  </disk>
  <disk type='network' device='cdrom'>
    <driver name='qemu' type='raw'/>
   <source protocol='http' name='/isoiic/ppc/RHEL-LE-7.1-20150109.0-Server-ppc64le-dvd1.iso'>
      <host name='172.16.180.10' port='80'/>
    </source>
    <backingStore/>
    <target dev='sdc' bus='scsi'/>
    <readonly/>
    <alias name='scsi0-0-0-2'/>
    <address type='drive' controller='0' bus='0' target='0' unit='2'/>
  </disk>
  <controller type='usb' index='0'>
    <alias name='usb0'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x0'/>
  </controller>
  <controller type='pci' index='0' model='pci-root'>
    <alias name='pci.0'/>
  </controller>
  <controller type='scsi' index='0'>
    <alias name='scsi0'/>
    <address type='spapr-vio' reg='0x2000'/>
  </controller>
  <interface type='bridge'>
    <mac address='52:54:00:94:d5:a5'/>
    <source bridge='brenp1s0f2'/>
    <target dev='vnet0'/>
    <model type='spapr-vlan'/>
    <alias name='net0'/>
    <address type='spapr-vio' reg='0x1000'/>
  </interface>
  <serial type='pty'>
    <source path='/dev/pts/1'/>
    <target type='isa-serial' port='0'/>
    <alias name='serial0'/>
    <address type='spapr-vio' reg='0x30001000'/>
  </serial>
  <console type='pty' tty='/dev/pts/1'>
    <source path='/dev/pts/1'/>
    <target type='serial' port='0'/>
    <alias name='serial0'/>
    <address type='spapr-vio' reg='0x30001000'/>
  </console>
  <input type='mouse' bus='usb'>
    <alias name='input0'/>
  </input>
  <input type='keyboard' bus='usb'>
    <alias name='input1'/>
  </input>
  <input type='tablet' bus='usb'>
    <alias name='input2'/>
  </input>
```

```
    <graphics type='vnc' port='5900' autoport='yes' listen='0.0.0.0'>
      <listen type='address' address='0.0.0.0'/>
    </graphics>
    <video>
      <model type='vga' vram='9216' heads='1'/>
      <alias name='video0'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
    </video>
    <memballoon model='virtio'>
      <alias name='balloon0'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x02' function='0x0'/>
    </memballoon>
  </devices>
  <seclabel type='dynamic' model='selinux' relabel='yes'>
    <label>system_u:system_r:svirt_t:s0:c920,c949</label>
    <imagelabel>system_u:object_r:svirt_image_t:s0:c920,c949</imagelabel>
  </seclabel>
</domain>
```