Advantages of running EDB PostgreSQL on IBM Power

An overview of performance aspects of running an open source database such as PostgreSQL within a modernized banking application

Table of contents

Introduction

EDB PostgreSQL, or simply, Postgres, is an open source relational database management system (RDBMS) known for its reliability and robustness. As an open source database, PostgresSQL delivers a level of trust important to the banking industry. It has an active community that continuously contributes to its development, ensuring regular updates and improvements. PostgreSQL remains a top choice for businesses and organizations looking for a reliable and scalable database solution. PostgreSQL also adheres to SQL standards and offers a wide range of advanced features, such as JSON support and geospatial data processing, and it enables easier migration from commercial databases such as Oracle.

Red Hat OpenShift Container Platform (OCP) automates the deployment and management of containerized applications which enables agility, portability, scalability, security, and more for hybrid cloud.

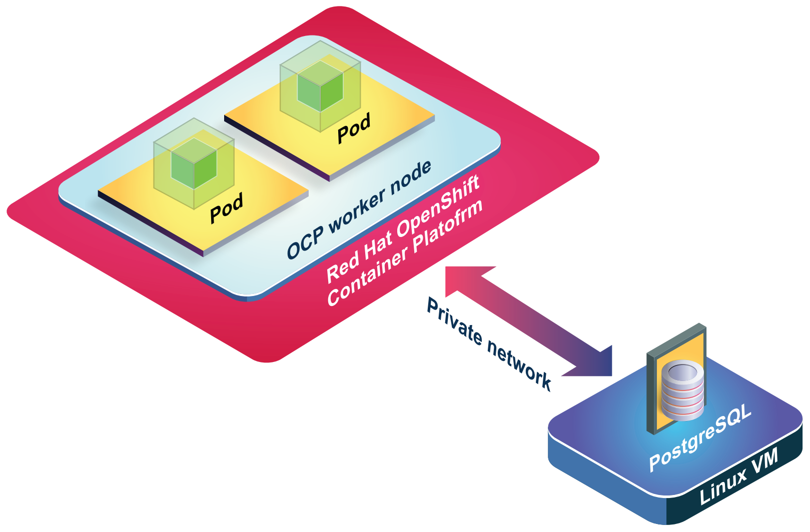
In this paper, we examine the performance characteristics of Postgres with applications running with OCP on two different hardware platforms: IBM Power and x86. We also share the techniques we used to tune the operating environment and the database engine.

Database deployment

The underlying infrastructure for a database is an important factor for performance, especially when the database is used within a hybrid cloud environment where the database may be on a different system than the application. To measure Postgres performance in a simulated hybrid cloud environment, we ran a containerized application in an OCP cluster accessing the database system which was configured outside of the OCP cluster. Specifically, we ran a single instance of Postgres running in a Linux virtual machine (VM) accessed by multiple pods running the pgbench workload on the OCP cluster with a private 25 G-based SRIOV network.

Infrastructure

Two different hardware platforms were used to run the performance tests. However, the configuration of the database and workload was done on the same server. The diagram below depicts the high-level view of the infrastructure configuration applicable to both platforms.



A single system was used for both platforms: The workload to stress the database was deployed on the OCP worker node and the database was installed on a stand-alone Linux VM. The two VMs were co-located on the same system. The Linux VM is not part of the OCP cluster, it just runs within the same physical system where the OCP cluster is also running. The OCP worker node connects to the database through a private 25 G SRIOV based network. The database on the Linux VM was based on a dedicated NVMe storage device. Note that the Power VMs used half the number of cores as the x86 VMs.

Following are descriptions of the systems used for the tests:

IBM Power10

Model: IBM Power S1022 (9105-22A)

Number of physical cores: 32

Number of sockets: 2

Total system memory: 1,024 GB

The first socket of thePower10 system had a total of 16 physical cores, where all 16 cores were equally used to run the OCP worker node and the database Linux VM.

Intel x86 (Ice Lake)

Model: Intel® Xeon® Platinum 8380 CPU @ 2.30GHz

Number of physical cores: 80

Number of sockets: 2

Total system memory: 1,024 GB

The first socket of the x86 system had a total of 40 physical cores, where 32 cores were equally used to run the OCP worker node and the database Linux VM.

Software stack

Database OS: Red Hat Enterprise Linux (RHEL) 8.6

Database Kernel: 4.18.0-425.el8.ppc64le / 4.18.0-372.9.1.el8.x86_64

Database storage: Local NVMe

Database file system: Ext4

OpenShift Container Platform (OCP): 4.11.25

Pgbench Version: PostgreSQL 14.4 (pgbench (14.4.0, server 14.4.0))

Workload

pgbench is a benchmarking tool used for PostgreSQL which can evaluate performance by simulating workloads with multiple concurrent database clients executing sets of transactions against the database. The default workload in pgbench consists of four types of transactions: *selects*, *updates*, *inserts*, and *deletes*. These transactions are performed on a predefined set of tables with synthetic data generated by pgbench.

The workload distribution can be adjusted by specifying the *scaling factor*, which determines the number of rows in the tables. We used the scaling factor of 1000 which set the database size to approximately 16 GB. The workload can also be customized by modifying the transaction mix, transaction weight, and the number of concurrent clients.

We used the default transaction mix and simple update to measure the performance. The following database commands shows the database tables and sizes tuned for the performance tests:

```
$/usr/edb/as14/bin/psql  -d mydb -c "SELECT pg_size_pretty(
pg_database_size('mydb') );"
pg_size_pretty
----------------
16 GB
(1 row)

$/usr/edb/as14/bin/psql mydb
psql (14.4.0, server 14.4.0)
Type "help" for help.

mydb-# \d+
List of relations
 Schema |       Name       | Type  |    Owner     | Persistence | Access
method |  Size   | Description
--------+------------------+-------+--------------+-------------+------------
---+---------+-------------
 public | pgbench_accounts | table | enterprisedb | permanent   | heap
| 13 GB   |
 public | pgbench_branches | table | enterprisedb | permanent   | heap
| 12 MB   |
 public | pgbench_history  | table | enterprisedb | permanent   | heap
| 1394 MB |
 public | pgbench_tellers  | table | enterprisedb | permanent   | heap
| 14 MB   |
(4 rows)
```

The default transaction mix test using -b option:

```
BEGIN;
UPDATE pgbench_accounts SET abalance = abalance + :delta WHERE aid = :aid;
SELECT abalance FROM pgbench_accounts WHERE aid = :aid;
UPDATE pgbench_tellers SET tbalance = tbalance + :delta WHERE tid = :tid;
UPDATE pgbench_branches SET bbalance = bbalance + :delta WHERE bid = :bid;
INSERT INTO pgbench_history (tid, bid, aid, delta, mtime) VALUES (:tid, :bid,
:aid, :delta, CURRENT_TIMESTAMP);
END;
```

The simple update built-in mix is achieved by using the -N option, where the following two SQL statements are not included:

```
UPDATE pgbench_branches SET bbalance = bbalance + :delta WHERE bid = :bid;
INSERT INTO pgbench_history (tid, bid, aid, delta, mtime) VALUES (:tid, :bid,
:aid, :delta,
```

The Read only test can be executed with the -S option where the only the following statement is used:

```
SELECT abalance FROM pgbench_accounts WHERE aid = :aid;
```

The database size can be adjusted by increasing or decreasing the number of rows using the -s (scale factor) option. For the performance tests, the *scale factor* value of 1000 resulted in a 16 GB database size. With the scale factor set to 2000, the size will be approximately 30 GB.

On the pgbench workload side, the number of clients and threads were configured using the -C and -J options. The number of clients and threads per each instance of pgbench was set at 7.

Performance tests and results

Every application that accesses the database will have its own requirements and criteria. Some might only read data; however, most workloads will require read, write, and update transactions.

The pgbench workload scripts using the -b option provides a mixture of read, write, and update transactions which is what we primarily used in our performance tests.

As noted, the infrastructure used to run the tests were based on two different hardware platforms, however the workload configurations were identical: The pgbench workload running in the OCP pods accessing the database running in the Linux VM outside of OCP. Various load tests were done to measure the performance with different resource configurations. All the performance measurements are representative of the Linux VM's database. Following are the load types for both infrastructures:

Power platform

The pgbench workload on the OCP worker node had three different test configurations:

- 16 pods with 4 vCPU and 14 G memory
- 32 pods with 2 vCPU and 7 G memory
- 64 pods with 1 vCPU and 3.5 G memory

The database on the Linux VM had a single instance of PostgreSQL database using the 8 physical cores at SMT8 based on Ext4 file system with the following characteristics:

- Block size of file system: 64 KB (stat -f /dev/nvme1n1)
- Sector size of file system: 4 KB (fdisk -l)
- Operating System page size: 64 KB (getconf PAGE_SIZE)

x86 platform

The pgbench workload on the OCP worker node:

- 16 pods with 2 vCPU and 14 G memory
- 32 pods with 1 VCPU vCPU and 7 G memory

The database on the Linux VM had a single instance of PostgreSQL database using the 16 physical cores at hyperthreading based on Ext4 file system and with the following characteristics:

- Block size of file system: 4 KB (stat -f /dev/nvme1n1)
- Sector size of file system: 512 KB (fdisk -l)
- Operating System page size: 4 KB (getconf PAGE_SIZE)

For all tests, a baseline was captured using the default PostgreSQL database configuration

as outlined at the following link:
https://github.com/postgres/postgres/blob/master/src/backend/utils/misc/postgresql.conf.sample.
There are several PostgreSQL databases tunables that impact the performance, however the following two variables had the most impact in our testing:

- The max_wal_size refers to the maximum size allowed for the Write-Ahead Log in a PostgreSQL database management system. The Write-Ahead Log is a crucial component for ensuring data integrity and durability. It records changes to the database before they are applied, providing a recovery mechanism in case of system failures. Setting an appropriate max WAL size can have performance benefits. When the max WAL size is too small, frequent log rotations occur, which can introduce overhead due to disk writes and other related operations. On the other hand, if the max WAL size is too large, it might result in longer recovery times during a crash or failover scenario.
  Having max_wal_size too small can cause checkpoints to happen very frequently where during the checkpoints all the dirty buffers in shared buffers need to be written out. The first time a page is changed after a checkpoint, the entire page is written to the WAL rather than just the change. On a busy system, this can be a very significant burst of WAL activity.

- The checkpoint_timeout parameter determines the maximum time interval between automatic checkpoints, which are points in time when the system flushes dirty data from memory to disk to ensure data durability and prevent data loss in case of a crash. A longer checkpoint timeout, intervals can lead to improved write throughput. Frequent checkpoints can introduce overhead and contention, affecting the overall write performance, especially in scenarios with high write loads. By increasing the timeout, you allow the system to perform more writes before initiating a checkpoint, potentially reducing the checkpoint-related overhead. Power results

SMT8 TPC-B mixed results with default and tuned configurations

The following table shows the database performance measurements using default and tuned configurations with SMT8 and database size of 16 GB. To read the content of this table, the far-left column, indicates weather whether the test was done using the default or tuned configuration. The next two columns show the number of pgbench workload pods and number of clients within each pod. For example, the first row, with 16 pods and 7 clients per pod translates into 112 total number of database connections.

The reminder of the columns in this table are the performance measurements of the PostgreSQL database running on the Linux VM, where the optimal performance is achieved with 32 pods in tuned configuration. The other important observation is the effect of the workload on the database locking, where the amount of database locking is increased with the increase of the load.

Test	PODs	C/J	TPS	LAT	CPU %	SMT	Row Exclusive	Access Share	Exclusive Lock
Default	16	7	56470.1	1.98	67.34	8	622	203	219
Default	32	7	55241.3	4.05	72.05	8	842	271	353
Default	64	7	51978	8.61	75.2	8	2993	897	995
Tune	16	7	60642.3	1.84	70.12	8	427	141	187
Tune	32	7	61963.9	3.61	79.49	8	915	254	361
Tune	64	7	57459.9	7.79	83.05	8	2095	623	722

SMT4 and SMT2 TPC-B mixed results with default and tuned configurations

The number of available CPUs can have an impact on the number of database transactions per second, and generally the higher the number of CPUs, the higher the transactions per seconds throughput. However, as shown in above table, with 64 pods accessing the database, the transactions per second drops due to the amount of database locking. With more database tuning, database locking can be reduced, however as noted before, we tried several other database tunable knobs, and could not get benefits, except for the two tunables mentioned previously. We believe the locking could be due to the implementation of PostgreSQL's internal interaction with the higher number of CPUs. We plan to engage with PostgreSQL development team to explore opportunities to enhance the performance.

The unique SMT setting in Power systems allows you to adjust the available CPUs within the operating system environment. To test the effect of the number of CPUs in database locking, we changed the SMT setting for the Linux VM to 4 and 2, to reduce the number of available CPUs. Note that because of this change, we did not run the 64 pods test scenario.

The following table shows the database performance measurements using default and tuned configurations with SMT4 and SMT2 and database size of 16 GB. Note the increase in the transactions per seconds with tuned configuration was similar to the SMT8 results. The amount of database locking does not improve with lower SMT, as we hoped, however that could be due to the much higher CPU utilization with lower SMT levels.

Test	PODs	C/J	TPS	LAT	CPU %	SMT	Row Exclusive	Access Share	Exclusive Lock
Default	16	7	50499.4	2.21	84.04	4	339	115	185
Default	32	7	48678.2	4.6	87.52	4	971	311	381
Tune	16	7	51555.7	2.17	85.5	4	646	207	223
Tune	32	7	49861.2	4.49	89.03	4	920	281	393

x86 results

Hyperthreaded TPC-B mixed results with default and tuned configurations

As outlined before, the configuration of the workload and test methodologies between the two platform were identical. The following table shows the database performance measurements for x86 using default and tuned configurations with hyperthreading and database size of 16 GB.

Test	Core	vCPU	PODS	C/J	TPS	Lat	CPU %	SMT	Row Exclusive	Access Share	Exclusive Lock
No Tune	16	32	16	7	46978.66	2.38	91.73	2	454	146	189
No Tune	16	32	32	7	44477.46	5.03	93	2	985	334	404
Tune	16	32	16	7	47678.65	2.34	92.36	2	571	191	219
Tune	16	32	32	7	45570.24	4.91	93.63	2	1152	375	431

Competitive advantage of IBM Power

SMT8 and hyperthreaded TPC-B mixed results with tuned configuration comparison

The Power platform is known for its high availability, and security aspects making it an ideal database server and the performance comparison of the two platform with respect to the number of transactions in our test does show the advantage of the Power platform as a database server. The following table shows the results we achieved using the tuned configuration that we previously discussed.

To compare the load with 32 pods, the 16 physical cores of the x86 platform produced 45570.24 transaction per second, whereas the 8 physical cores of Power produced 61963.90 transactions per second. Considering the difference in TPS with respect to the used physical cores, there is a 2.72 X per core advantage for the Power platform to run PostgreSQL database transactions.

System	PODs	C/J	TPS	LAT	CPU %	SMT	Row Exclusive	Access Share	Exclusive Lock
Power	16	7	60642.3	1.84	70.12	8	427	141	187
Power	32	7	61963.9	3.61	79.49	8	915	254	361
x86	16	7	47678.65	2.34	92.36	2	571	191	219
x86	32	7	45570.24	4.91	93.63	2	1152	375	431

Impact of workload and database options

pgbench -N option

As described earlier, the pgbench workload allows different SQL operations to be used when running the load. Using the *-N* option, there is only a single UPDATE SQL operation used. Note that using -N option makes the test case less like TPC-B, however we tested this option to measure the amount of contention on the tables, and its impact on the amount of database locking. Running 64 pods against the SMT8 based database with default configuration, there is almost 20% increase in performance and decrease in number of database lock. The results highlight the effect of database locking and its impact on the overall performance.

EXT4 file System	PODs Clients	C/J	TPS	LAT	CPU %	SMT	Row Exclusive	Access Share	Exclusive Lock
Without -N	64	7	51978	8.61	75.2	8	2993	897	995
With -N	64	7	62874.62	7.12	59.2	8	1093	638	782

fsync on/off

Another commonly used option for the PostgreSQL database is the *fsync* setting which ensures that all updates are physically written to disk so that database cluster can recover to a consistent state after an operating system or hardware crash. Setting this option to off can benefit performance. Running 64 pods against the SMT8 based database with default configuration resulted in 15% performance benefit.

LTC patch	PODs Clients	C/J	TPS	LAT(ms)	CPU %	SMT	Row Exclusive	Access Share	Exclusive Lock
fsync Off	64	7	66106.3	6.77	88.08	8	2073	674	829
fsync On	64	7	57459.9	7.79	83.05	8	2095	623	722

SRIOV based network adapter

There are several variables in the underlying infrastructure that could impact the performance of applications. The network configuration is one of those variables that we examined. With reference to the infrastructure description noted earlier in this paper, we focused on the network connection between the OCP cluster where the pgbench workload runs and the Linux VM where the PostgreSQL database runs.

On both platforms, the SRIOV based network communication between the OCP worker node and Linux VM were based on a single 25 G adapter with 2 physical ports, where the Virtual Functions (VF) used for both OCP worker nodes and Linux VMs were based on a single physical port. We used the VF from two different 25 G network adapters to measure the impact of the network configuration. The table below shows around 4% performance improvement when the VFs were from a single port of a single 25 G adapter.

Network Type	PODs Clients	C/J	TPS	LAT(ms)	CPU %	SMT	Row Exclusive	Access Share	Exclusive Lock
Diff N/W Adapter	64	7	50080	8.9	71.62	8	1691	503	806
Same N/W adapter	64	7	51978	8.61	75.2	8	2993	897	995

Summary

The Power platform's architecture and unique hardware capabilities, along with its processing power and memory capacity, deliver superior performance for database-intensive workloads with PostgreSQL, as we've demonstrated with our testing. Couple this performance with the Power platform's reliability and PostgreSQL on Power should be recognized as a preferred choice for businesses seeking a dependable and high-performing solution to meet their evolving data needs.

About the authors

Mel Bakhshi is a senior Power system performance analyst in IBM infrastructure Organization. He has experience in systems architecture design, distributed database, capacity planning and sizing of the hybrid cloud environment. You can reach Mel at melb@ca.ibm.com.

Vijay K Puliyala is a Power system performance analyst in IBM infrastructure Organization. He has more than 10 years' experience working iwht the IBM Power System platform. You can reach Vijay Kumar at vpuliya@in.ibm.com.