DB-Project3- Benchmarking openGauss Against PostgreSQL: A Comparative Analysis

---

12311153 廖子韬

1 引言

openGauss 是华为公司开发的开源关系型数据库管理系统 （RDBMS）。它专为企业级应用程序而设计，基于PostgreSQL，具有专为高性能、可靠性和安全性量身定制的增强功能。据称，openGauss 特别适用于复杂的数据密集型场景，例如金融系统、电信和大型电子商务平台。

本实验旨在探讨和评估 openGauss 的实际应用性能。根据要求，我想从以下两个使用场景和几个关键测试指标出发，检测其性能。

2 实验设计

2.1 实验设计主要指标

1. **查询性能**：根据 openGauss 官方描述，openGauss 数据库在统计信息、行数估算、代价估算、路径搜索等方面进行了优化，同时引入了行列混合引擎以根据不同的场景选择不同的存储类型。可以通过比较两者在处理复杂查询时的**运行时间**来反映该性能。

2. **并发处理能力**：根据 openGauss 官方描述，openGauss 数据库在客户端建立了连接池，能够避免连接的频繁创建和销毁；线程池在数据库服务器上配置，控制数据库服务器活动线程数目，线程复用，对系统的业务起到流控作用，防止出现雪崩。据称，在高并发场景下，将连接池和线程池结合起来使用，可以有效解决大并发问题。评估**每秒事务数（TPS）和事务延时**，测试在高并发情况下的性能表现。

3. **连接能力：**根据以上一点关于并发处理能力的讲述，注意到 openGauss 中连接池的设计应当对客户端与数据库的连接效率有一定的影响。可以通过比较调用 SQL 时候的**初始连接时间**及多次调时的**时间变化**，测试连接能力。

4. **数据导入速度**：考虑到 openGauss 所适应的场景下往往也关联着大规模数据点的导入，openGauss 官方描述执行引擎中的扫描算子和连接算子等算子模块能够对来自数据系统或者网络的数据进行扫描与处理。通过测量**大规模数据导入的速度**，检验其数据导入性能。

5. **效率与资源使用率**：根据 openGauss 官方描述，openGauss 应用 NUMA 内核数据结构，能够通过线程绑核、数据分区实现减少线程的核间偏移、访问冲突和跨核访问，来优化 CPU 架构。通过测量运行事务时的 **CPU、内存、磁盘读写效率**，比较 CPU 和内存的使用效率。

6. **安全性**：根据 openGauss 官方描述，openGauss 运用了全密态等值查询，将数据转化为以密文形式存在，无法解密，同时引进来防篡改账本数据库来加强安全性。通过比较两者**数据加密、访问控制**等方面的表现验证其安全特性。

2.2 实验主要方法

主要通过在本地编辑 SQL 脚本，并通过 PostgreSQL 所带的 **pgbench 测试工具**分别对 openGauss 数据库和 PostgreSQL 数据库进行相应测试。

Pgbench 介绍：

该实验有对多用户、高并发的测试需求，而 Datagrip 作为一个数据库管理工具，并不能够直接支持高并发测试，因此我们需要利用其它工具完成实验。我们通过查找资料发现，pgbench 是 PostgreSQL 自带的一个基准测试工具，勇于评估数据库的性能。它通过模拟多个并发的数据库会话，反复执行一系列预定义或自定义的 SQL 命令，从而计算事务执行的平均速率（每秒执行的事务个数）。在终端调用 pgbench 利用多客户端、多线程执行 SQL 脚本后，pgbench 将会输出**每个客户端的事务数、实际处理的事务数、平均延迟时间和每秒的事务数**等，能够帮助我恩了解数据库的性能表现。由于 openGauss 也是基于 PostgreSQL 开发并同时兼容 pgbench，因此 pgbench 非常支持该实验用来测试 PostgreSQL 和 openGauss 数据库的并发量、吞吐量、延迟等性能指标。

pgbench 输入样例：

```
C:\Users\48946\DataGripProjects\Project3>pgbench -r -T 60 -h localhost -p 5433 -U postgres -d postgres -f query_test.sql

Password:
```

pgbench 输出样例：

```
transaction type: query_test.sql
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
maximum number of tries: 1
duration: 60 s
number of transactions actually processed: 115
number of failed transactions: 0 (0.000%)
latency average = 521.369 ms
initial connection time = 133.946 ms
tps = 1.918027 (without initial connection time)
statement latencies in milliseconds and failures:
         5.766           0  DROP TABLE IF EXISTS query_test_table;
         2.227           0  CREATE TABLE query_test_table (
       496.337           0  INSERT INTO query_test_table (id,name, num)
         0.097           0  BEGIN;
         0.228           0  SELECT name FROM query_test_table WHERE query_test_table.id = 12345;
        10.922           0  SELECT id, name, num FROM query_test_table WHERE query_test_table.id > 75000;
         5.682           0  SELECT id, name, num FROM query_test_table WHERE query_test_table.id > 75000 AND query_test_
table.num BETWEEN 500 AND 5000;
         0.107           0  COMMIT;
```

3 实验内容

3.0 环境配置

Database	Version
PostgreSQL	PostgreSQL **17.2** on x86_64-windows, compiled by msvc-19.42.34435, 64-bit

openGauss	(openGauss **3.0.0** build 02c14696) compiled at 2022-04-01 18:12:34 commit 0 last mr on x86_64-unknown-linux-gnu, compiled by g++ (GCC) 7.3.0, 64-bit

3.1 查询性能测试

首先，我自己编辑了一个 SQL 脚本文件，先清空原有表，再创建一个表，通过随机生成行数据进行插入。我想控制客户端数目为 1，线程数为 1，运行时间为 60 秒，并通过比较运行过程中的吞吐量（tps)、平均延迟市场（latency average）以及单条语句执行的平均延迟来检验 openGauss 和 PostgreSQL 的性能。我设计了三组实验，分别对简单查询、聚合查询和 order by 查询进行检测，每组实验进行 5 次并取平均值后得到以下结果：

测试脚本如下：

```
DROP TABLE IF EXISTS query_test_table;
CREATE TABLE query_test_table (
   id int primary key ,
   name varchar(60),
   num int
);
INSERT INTO query_test_table (id,name, num)
SELECT  i, 'Test User'|| i , i * 10
FROM generate_series(1, 100000) AS s(i);
BEGIN;
SELECT name FROM query_test_table WHERE query_test_table.id = 12345;
SELECT id, name, num FROM query_test_table WHERE query_test_table.id > 75000;
SELECT id, name, num FROM query_test_table WHERE query_test_table.id > 75000 AND
query_test_table.num BETWEEN 500 AND 5000;
SELECT COUNT(*) FROM query_test_table WHERE query_test_table.num > 75000;
SELECT AVG(num) FROM query_test_table WHERE query_test_table.num BETWEEN 500 AND
5000;
SELECT id, name, num FROM query_test_table ORDER BY query_test_table.num DESC LIMIT 300;
COMMIT;
```

Pgbench 命令如下：

```
C:\Users\48946\DataGripProjects\Project3>pgbench -r -T 60 -h localhost -p 5433 -U postgres -d postgres -f query_test.sql
Password:
```

3.1.1 简单查询

	吞吐量	平均延迟/ms	等于查询/ms	不等查询/ms	多列查询/ms
PostgreSQL	2.46782	405.87175	0.215	10.5385	5.59075
openGauss	1.7576872	570.4644	0.8672	32.8572	13.8892

3.1.2 聚合查询

	吞吐量	平均延迟/ms	count()/ms	avg()/ms
PostgreSQL	2.832161	353.5222	7.7602	6.0626
openGauss	1.3829998	767.0942	22.4714	16.4272

3.1.3 order by 查询

	吞吐量	平均延迟/ms	orderby/ms
PostgreSQL	2.5677165	390.1475	16.912
openGauss	1.7192726	582.5934	24.6854

由实验数据可知，**在单用户端、单线程的情况下，PostgreSQL 的吞吐量表现优于 openGauss，平均延迟和各类查询时长也均短于 openGauss**；同时也注意到，不等查询的时长远远长于等于查询的时长，这可能是由于不等查询涉及到整表的扫描，而我们设计的等于查询是针对 primary key 的查询，也许并不需要整表扫描，这对我们后续的脚本设计有一定的指导意义。

显然，openGauss 的“优化”并没有在本次实践中得到体现，考虑到其对“**估算优化**”的介绍，我想运用“EXPLAIN ANALYSE”语句对此进行对比检测。

3.1.4 查询代价估算

针对同一个内含 62777 行数据的表“Spotify_tracks”，我们分别在 Postgres 和 openGauss 中进行简单查询、聚合查询和 orderby 查询：

输入与输出：

```
explain (analyse,buffers)select spotify_tracks.track_name,album_name from spotify_tracks
                                where artist_name = 'Taylor Swift' and spotify_tracks.album_name = 'folklore';
explain (analyse ,buffers)select count(*) from spotify_tracks
where year > 2022 and language = 'English';
explain (analyse ,buffers)select count(*) from spotify_tracks
where artist_name = 'Taylor Swift';
```

```
QUERY PLAN
Aggregate  (cost=3662.00..3662.01 rows=1 width=8) (actual time=13.382..13.383 rows=1 loops=1)
  Buffers: shared hit=2716
  ->  Seq Scan on spotify_tracks  (cost=0.00..3650.76 rows=4499 width=0) (actual time=0.028..13.105 rows=5953 loops=1)
        Filter: ((year > 2022) AND (language = 'English'::text))
        Rows Removed by Filter: 56364
        Buffers: shared hit=2716
Planning Time: 0.117 ms
Execution Time: 13.410 ms
```

其中 cost 表示节点的预估代价，rows 是预估行数，width 是预估结果的宽度（字节）。

Execution Time： 执行的时间； Buffers：所使用的磁盘页数。

int查询	估算行	实际行	误差	不等查询	估算行	实际行	误差
Postgres	6498	6427	1.10%	**Postgres**	11971	11898	**0.61%**
openGauss	6414	6427	**0.20%**	openGauss	11792	11898	0.89%
聚合查询	**估算行**	**实际行**	**误差**	**多列查询**	**估算行**	**实际行**	**误差**
Postgres	906	903	**0.33%**	**Postgres**	4499	5953	**24.42%**
openGauss	860	903	4.76%	openGauss	4418	5953	25.78%

（误差= | 估算行数 - 实际行数 | / 实际行数 * 100%）

可以发现，在不同查询条件的执行估算中，**PostgreSQL 在整体上更胜一筹**；且发现重复执行同一 SQL 语句时，估算行数并**不会发生优化，这在两个数据库中的表现相同。**

在本实验条件下，openGauss 也并不能展现出很好的估算性能，可能是由于本实验的查询规模有限，对代价估算和优化路径的需求较小，无法模拟实际场景下多客户端、多线程、复杂查询条件下的查询场景。

3.2 并发事务处理能力测试

针对该项测试，我编辑的 SQL 脚本内同时包含了 CREATE、INSERT、SELECT、UPDATE、DELETE、DROP 等操作。我想通过记录并比较 tps 的大小及其变化来反映 openGauss 和 PostgreSQL 的并发事务处理能力

测试脚本如下：

```
DROP TABLE IF EXISTS transaction_test_table;
CREATE TABLE transaction_test_table (
   id int primary key ,
   name varchar(60),
   num int
);
INSERT INTO transaction_test_table (id,name, num)
SELECT  i, 'Test User'|| i , i * 10
FROM generate_series(1, 100000) AS s(i);
SELECT COUNT(*) FROM transaction_test_table WHERE transaction_test_table.num = 75000;
SELECT id,name,num FROM transaction_test_table WHERE transaction_test_table.num
BETWEEN 500 AND 5000;
INSERT INTO transaction_test_table (id,name, num)
VALUES (100001,'New User', (random() * 1000)::INT);
INSERT INTO transaction_test_table (id,name, num)
VALUES (100002,'New User', (random() * 1000)::INT);
UPDATE transaction_test_table SET num = num + 100 WHERE num = 12130;
DELETE FROM transaction_test_table
WHERE id IN (SELECT id FROM transaction_test_table WHERE num < 100 LIMIT 100);
```

3.2.1 限制总运行时长的高并发情形

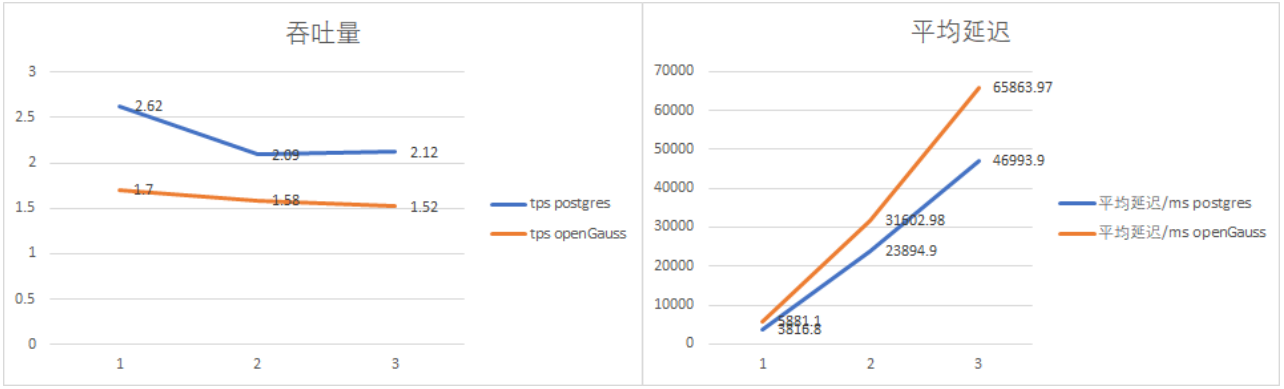
Pgbench 命令：　（-c 参数为客户端数目，由于设备限制最大值为 100；-j 参数为线程数目；-T 参数为总运行时长）

```
pgbench  -c xx -j xx -r -T 60 -h localhost -p 5433 -U postgres -d postgres -f transaction_test.sql
pgbench -c xx -j xx r -T 60 -h localhost -p 15433 -U gaussdb -d postgres -f transaction_test.sql
```

该设计可以用来模拟在同一集中时间段内对事务的处理能力，结果如下：

（1）　控制线程数不变；改变客户端数目

表-3.2.1(1)	客户端数目	线程数目	运行时长	吞吐量	平均延迟/ms	总事务数
PostgreSQL	10	10	60s	**2.62**	**3816.8**	**156**
openGauss				1.7	5881.1	102
PostgreSQL	50			**2.09**	**23894.9**	**116**
openGauss				1.58	31602.98	95
PostgreSQL	100			**2.12**	**46993.9**	**108**
openGauss				1.52	65863.97	91

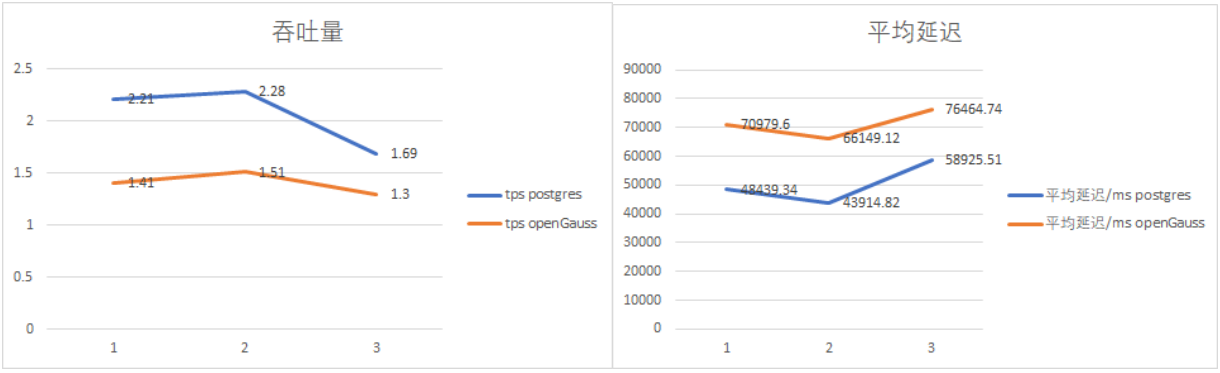


由实验结果及表格可知，在该实验条件情况下，**PostgreSQL 的吞吐量和平均延迟均优于 openGauss**；同时注意到，当线程数目不变时，随着客户端数目增加，**PostgreSQL 的吞吐量和平均延迟的波动均较大**，其中 tps 先减后增，平均延迟持续增加；而 **openGauss 的吞吐量和平均延迟波动相对稳定**，其中 tps 呈单调递减，平均延迟单调递增。

(2) 控制客户端数目不变；改变线程数目

在第一组实验后我注意到，增大总运行时长可以增大样本量，提高实验结果的准确性，因此将总运行时长从 1 分钟调整至 10 分钟。

表-3.2.1(2)	客户端数目	线程数目	运行时长	吞吐量	平均延迟/ms	总事务数
PostgreSQL	100	32	600s	**2.21**	**48439.34**	**1305**
openGauss				1.41	70979.6	851
PostgreSQL		64		**2.28**	**43914.82**	**1349**
openGauss				1.51	66149.12	907
PostgreSQL		100		**1.69**	**58925.51**	**992**
openGauss				1.3	76464.74	784



由实验数据及图表可知，在该实验条件情况下，**PostgreSQL 的吞吐量和平均延迟仍均优于 openGauss**；同时注意到，当客户端数目不变时，随着客户端数目增加，PostgreSQL 和 openGauss 的 tps 均先增后减，平均延迟均先减后增。

3.2.2 限制单客户线程数目的高并发测试

Pgbench 命令： （-c 参数为客户端数目，由于设备限制最大值为 100；-j 参数为线程数目；-t 参数为单个客户端发出的事务数）

```
pgbench  -c xx -j xx -r -t 10 -h localhost -p 5433 -U postgres -d postgres -f transaction_test.sql
pgbench -c xx -j xx r -t 10 -h localhost -p 15433 -U gaussdb -d postgres -f transaction_test.sql
```

实验数据如下：

表-3.2.2	客户端数目	线程数目	事务数（单客户端/总数）	吞吐量	平均延迟/ms	初始连接时长/ms	总运行时长/s
PostgreSQL	100	100	10/1000	**1.45**	**54957.968**	15907.1436	24
openGauss				1.3680368	73248.488	**364.8544**	**8.5**
PostgreSQL			100/10000	**2.2186954**	**45092.2472**	9185.0672	**51**
openGauss				1.560471167	64531.563	**260.4255**	60

由表格可知，在客户端数目和线程数目一定的条件下，

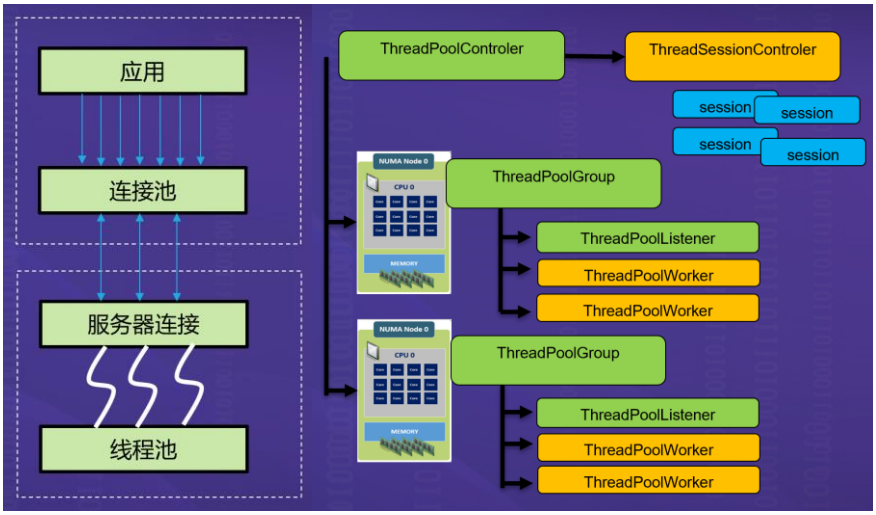
(1) PostgreSQL 的吞吐量和平均延迟两项指标均优于 openGauss

(2) openGauss 的初始连接时长均远远小于 PostgreSQL

(3) 在单事务数较小时，openGauss 的运行总时长小于 PostgreSQL，但是在单事务数较大时，openGauss 的运行总时长仍输于 PostgreSQL

(4) 对比实验 3.2.1 中的总事务数发现，无论是 openGauss 还是 PostgreSQL，在限制单个客户端事务数时，完成相同总事务数所需要的时间远远少于不限制单客户端事务数。

在本实验条件下，openGauss 并未在高并发条件下体现出效率方面的优势。可能是由于 openGauss 的优化更多地面向电商、企业中更复杂的应用场景，相比于 PostgreSQL 做了更多关于稳定性和安全性的优化。例如，连接池的加入也许能够在“双十一”等类似高并发场景下，在数据库服务器端崩溃时，能够保持客户端的连接与保存客户端的事务请求，如下图所示。在稳定性方面的优化也许牺牲了 openGauss 的效率，导致了该结果。

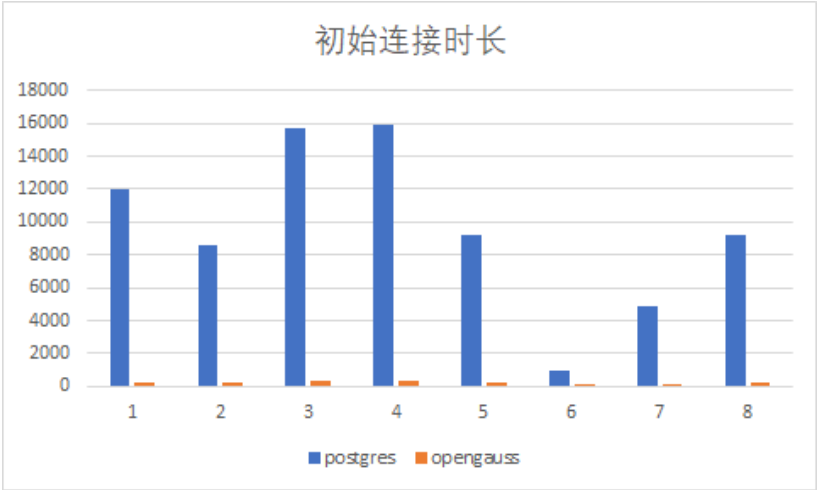


同时，观察到（4）的特殊结果，经过资料搜索发现，这可能是由于限制单个客户端的事务数时，可以减少多个事务对数据库资源（如锁、连接、内存等）的竞争。这样可以避免资源争用导致的性能瓶颈，提高单个事务的执行效率，有助于提高数据库的并发处理能力，同时也能够可以减少事务 ID 的分配和回收频率，降低事务管理的复杂性，从而提高性能。

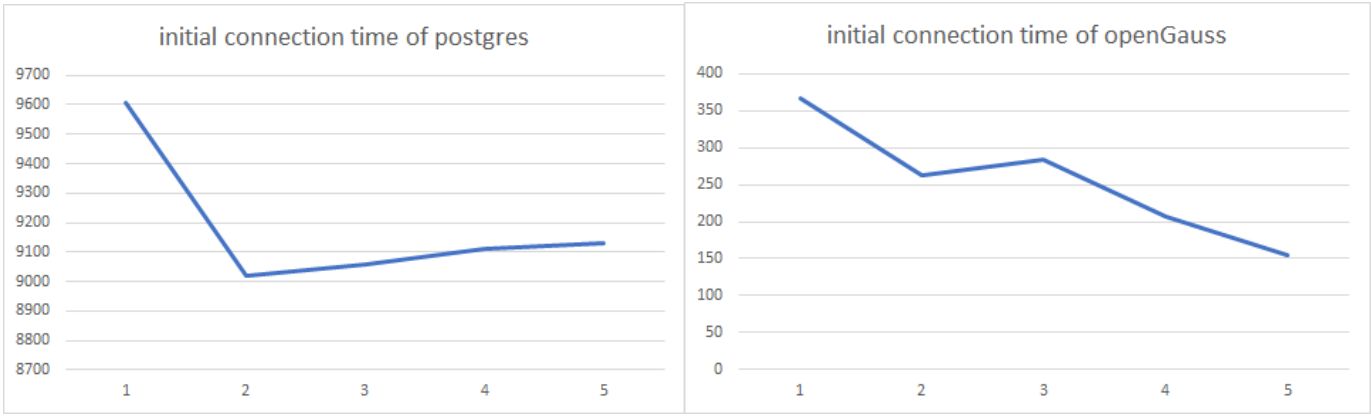
但观察到（2）-（3）的特殊现象，我认为期间有所关联。结合前面几项实验观察到的“初始连接时长”数据，接下来我将对这一指标进行实验对比。

3.3 连接能力

调用 3.2 实验中每次的初始连接时长数据，得到以下图表：



选取 3.2.2 的第一组中的初始连接时长数据，得到以下图表：



由图表可知，**openGauss 的初始连接时长遥远远小于 PostgreSQL，初始连接时长随实验次数有明显的减小趋势**，而 PostgreSQL 的初始连接时长并无明显稳定的变化趋势。

在初始连接时长角度，openGauss 体现出巨大优势。同时，openGauss 的连接时长优化表现也优于 PostgreSQL。这可能是由于线程连接池发挥了作用，查询资料后，发现 openGauss 官方描述的该运行模型优势能够得到实验的验证。

关键差异化因素		openGauss	PostgreSQL
运行时模型	执行模型	线程池模型，高并发连接切换代价小、内存损耗小，执行效率高，一万并发连接比最优性能损耗<5%	进程模型，数据库进程通过共享内存实现通讯和数据共享。每个进程对应一个并发连接，存在切换性能损耗，导致多核扩展性问题。

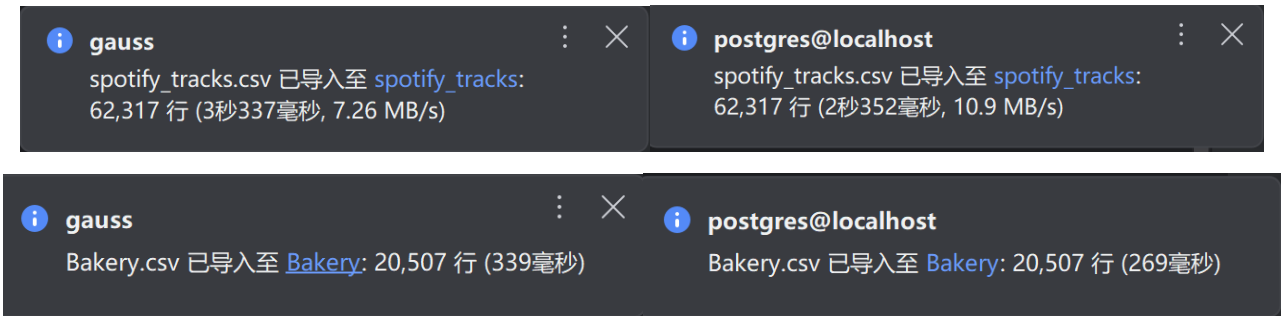
回归到 3.2 的结论（3）的可能原因是，在总事务数较少，总延迟较小的情况下，初始连接时长对总运行时长的影响较大，初始连接时长短的 openGauss 有显著优势；而在总事务数和总延迟变大时，初始连接时长的影响被稀释，因此在平均延迟性能方面有优势的 PostgreSQL 又有了更短的总运行时长。

3.4 数据导入速度测试

由于数据对设备内存有所需求，受到设备内存限制，在本实验中，我想从单个数据集的导入入手，回归 Datagrip 来对该性能进行测试。主要分为 csv 数据的导入、sql 脚本的数据导入和大规模随机数据的导入，来模仿各种场景下的数据导入情形。

3.4.1 csv 数据的导入

我从 kaggle 网站上搜索下载了一个内含 62317 行，22 列的“Spotify_tracks”数据集和一个内含 20507 行，5 列的”backery”数据集分别导入到 PostgreSQL 和 openGauss 数据库，结果如下：



3.4.2 sql 脚本的导入

分别在 PostgreSQL 和 openGauss 中执行“filmdb.sql”，得到以下输出：

Database	Execution Time
PostgreSQL	20731ms
openGauss	41685ms

3.4.3 大规模随机输入

分别在 PostgreSQL 和 openGauss 数据库终端执行以下语句得到输出：

```
CREATE TABLE insert_test_table (
   id int primary key ,
   name varchar(60),
   num int
);

INSERT INTO insert_test_table (id,name, num)
SELECT  i, to_char(i,'0999999') , i * 10
FROM generate_series(1, 10000000) AS s(i);
```

Database	Size of 1,000,000	Size of 10,000,000
PostgreSQL	8692ms	113038ms
openGauss	11325ms	155329ms

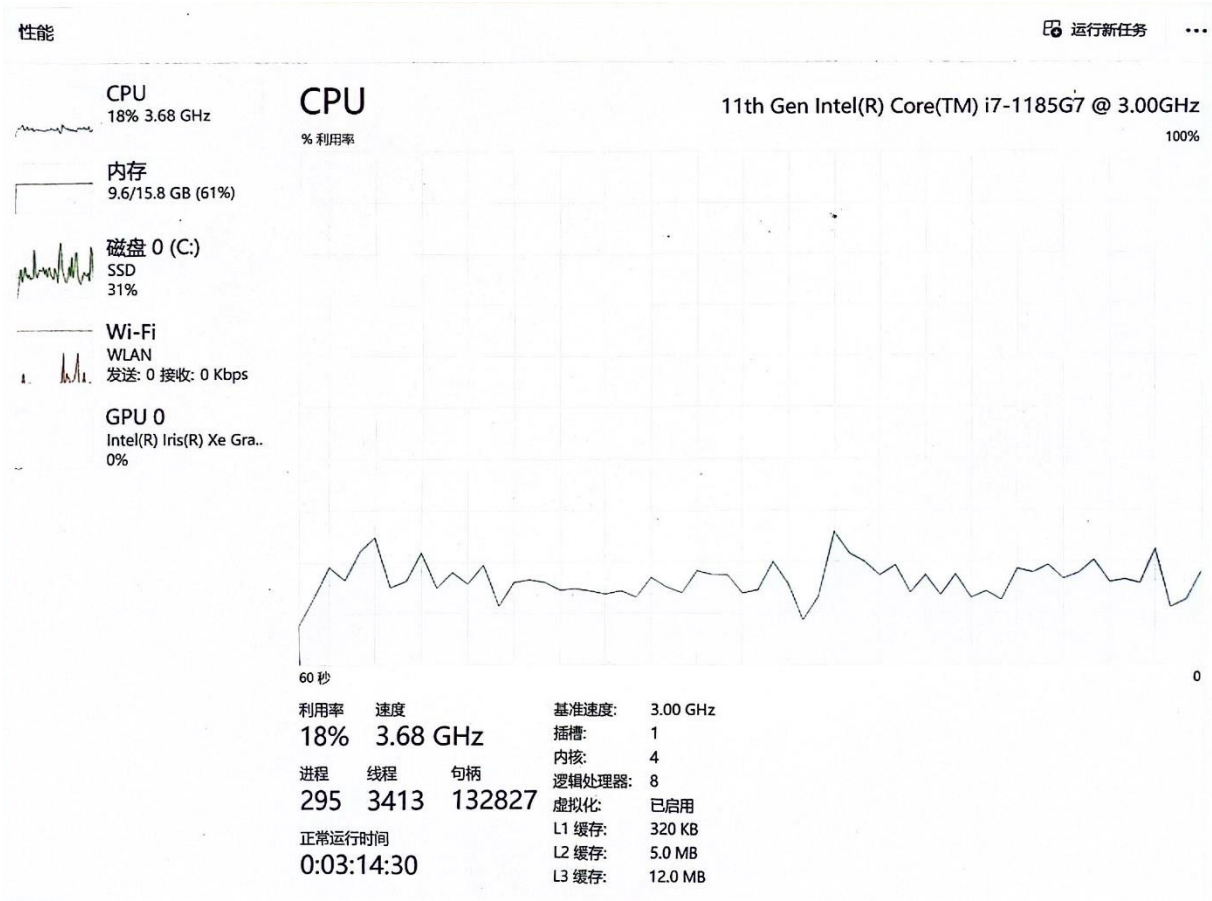
由以上数据可知，openGauss 在大规模数据导入方面也无明显优势。但是考虑到设备限制，无法检测多客户端、多主机的数据导入效果。

3.5 效率与资源使用率

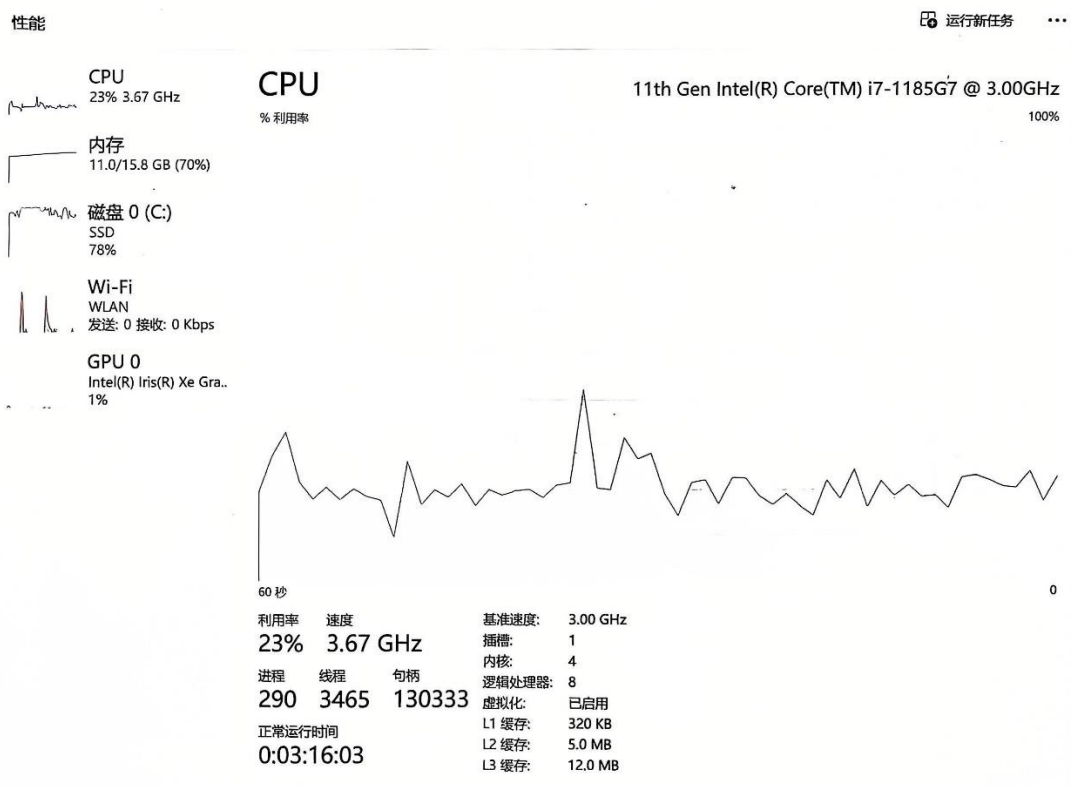
重复利用 3.2 中的 SQL 测试脚本，运用 pgbench 分别对 PostgreSQL 和 openGauss 进行测试，同时打开系统的资源管理器监控设备 CPU、内存和磁盘使用情况的变化，得到实验结果如下：

3.5.1 对 PostgreSQL 和 openGauss 运行相同脚本时的 CPU 监测

PostgreSQL 表现：

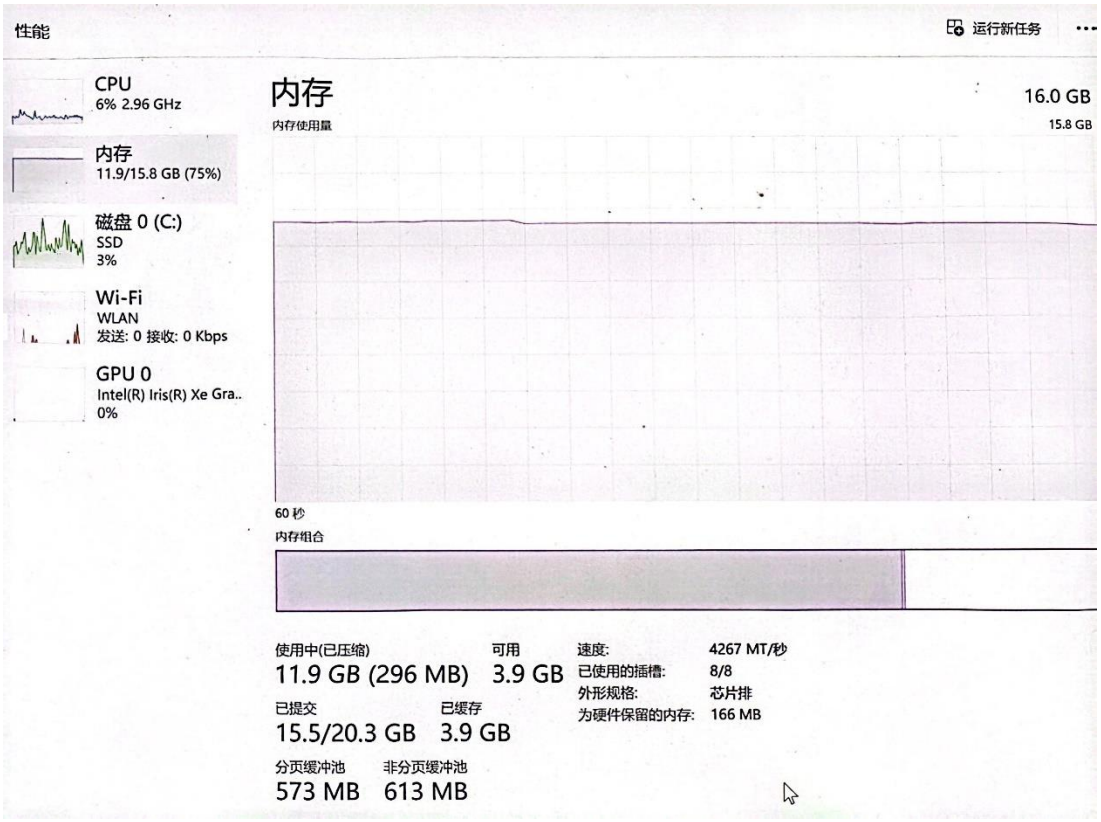


openGauss 表现：

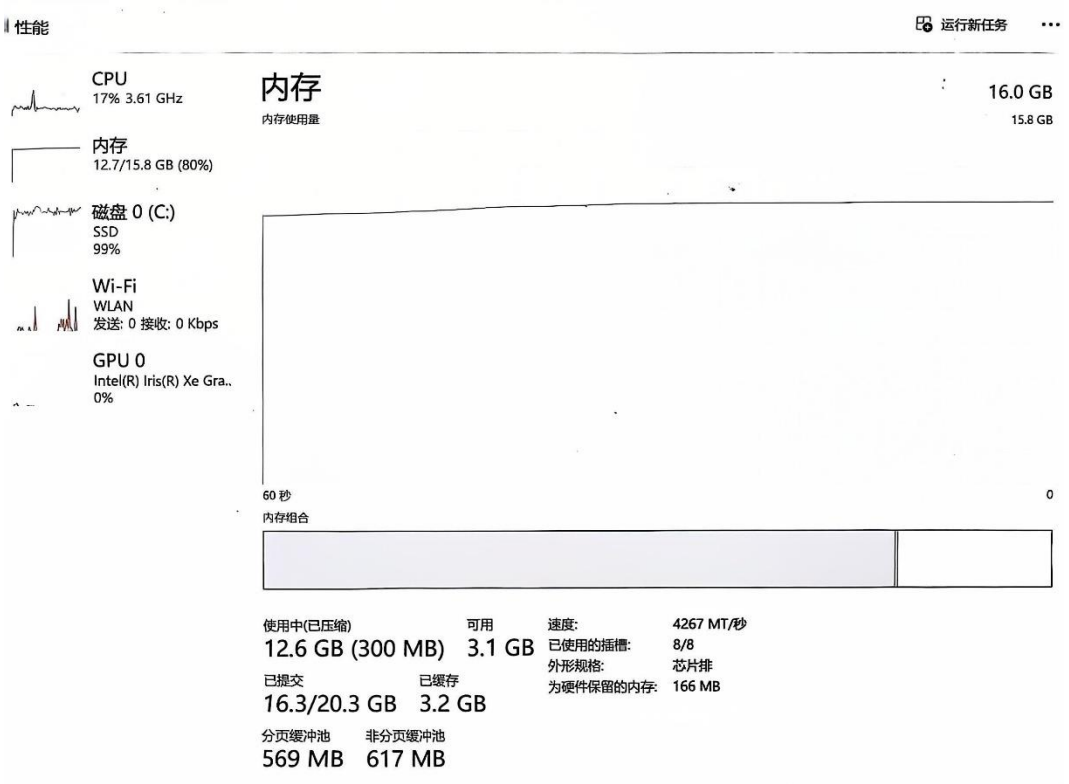


3.5.2 对 PostgreSQL 和 openGauss 运行相同脚本时的内存检测

PostgreSQL 表现：

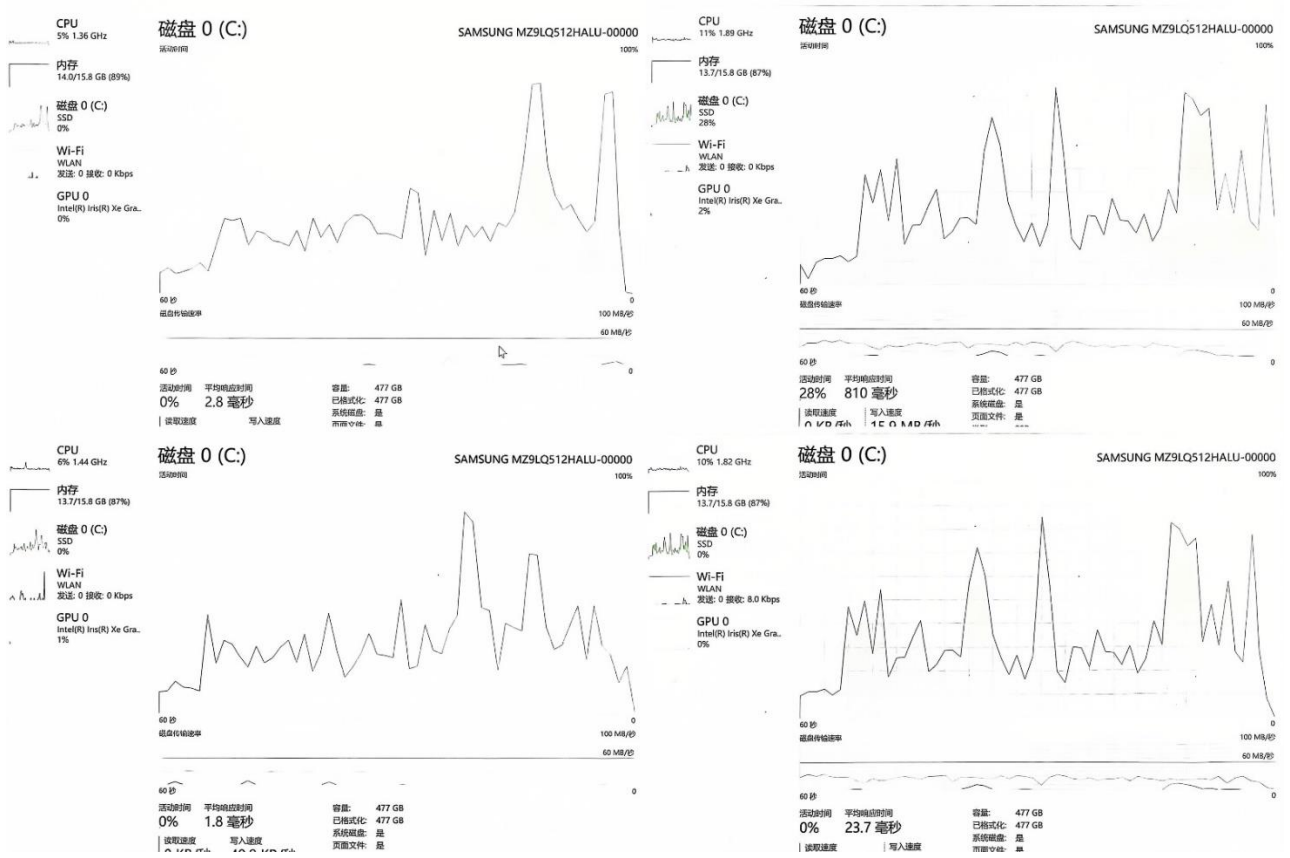


openGauss 表现：

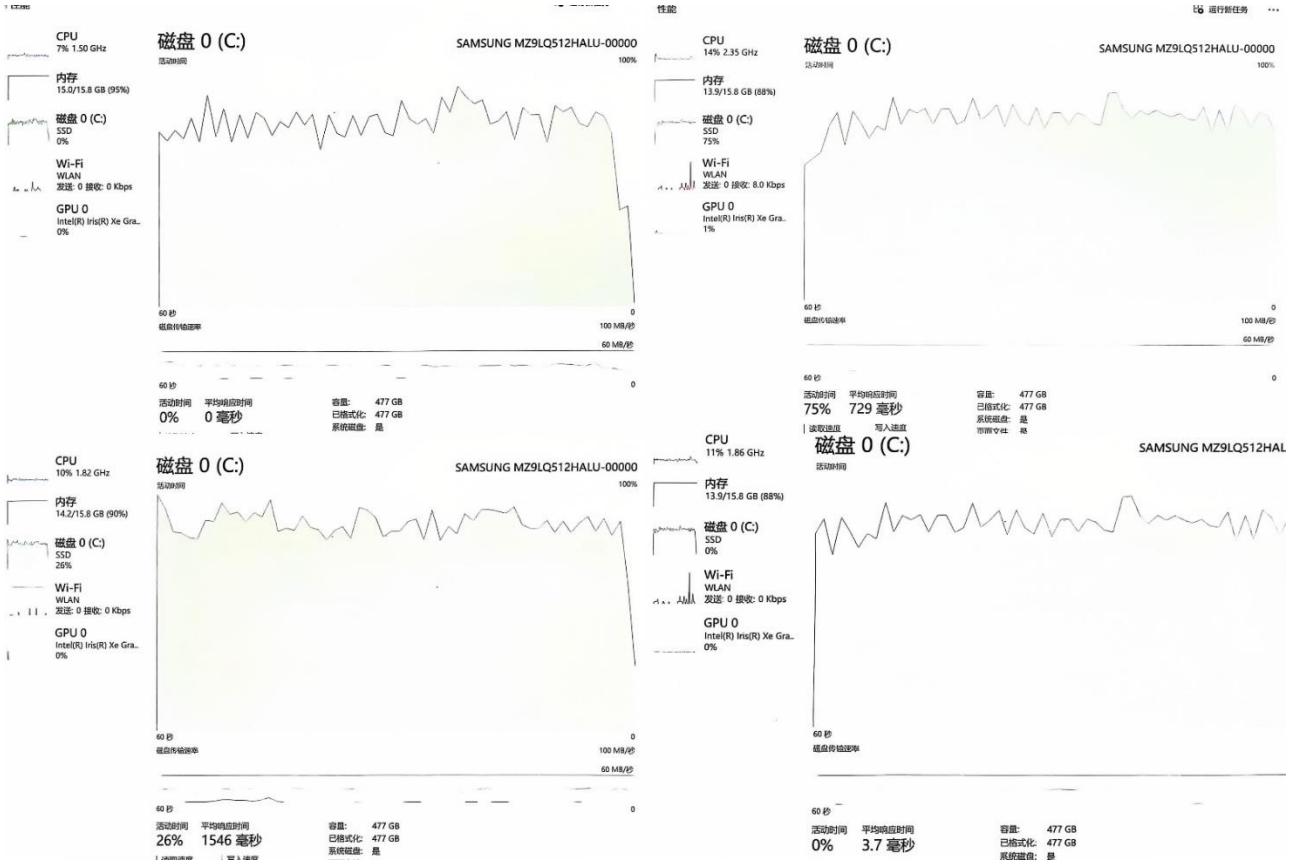


3.5.3 对 PostgreSQL 分别进行 SELECT、INSERT、UPDATE、DELETE 操作测试的 CPU 和磁盘活动情况

PostgreSQL 表现：



openGauss 表现：



由图像可知，在运行时，openGauss 在 CPU 占用和内存占用方面的情况与 PostgreSQL 大体一致；而在磁盘读写方面，openGauss 的磁盘占用率的平均值较高，在 80%左右，而方差较小，波动性较弱，PostgreSQL 的磁盘占用率的平均值较低，在 50%左右，但是方差较大，常会出现许多尖锐波峰。

在该实验条件下，openGauss 在磁盘读写方面的稳定性更强，但读写活动的活跃性较高。

调查资料发现可能的原因是，openGauss 的存储引擎中应用了 NUMA 内核数据结构，同时应用单 Query 索引机制，使得在相同负载下，磁盘 I/O 操作更为频繁，从而提高了磁盘占用率。这也与 pgbench 测试 openGauss 运行命令时的 NOTICE 输出相对应。如下图，每一次执行 CREATE TABLE 操作时，openGauss 都会自动建立一个隐式索引（“implicit index”）。

```
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "transaction_test_table_pkey" for table "transaction_test_table"
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "transaction_test_table_pkey" for table "transaction_test_table"
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "transaction_test_table_pkey" for table "transaction_test_table"
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "transaction_test_table_pkey" for table "transaction_test_table"
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "transaction_test_table_pkey" for table "transaction_test_table"
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "transaction_test_table_pkey" for table "transaction_test_table"
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "transaction_test_table_pkey" for table "transaction_test_table"
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "transaction_test_table_pkey" for table "transaction_test_table"
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "transaction_test_table_pkey" for table "transaction_test_table"
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "transaction_test_table_pkey" for table "transaction_test_table"
```

3.6 安全性能

在 Lab 课上，我们曾对 PostgreSQL 的 Privilege 性能做过测试。我想讲这些测试语句也在 openGauss 内进行运行，观察结果输出的差距，从而反应两者安全性能的差别。

在访问控制方面，发现 openGauss 的用户或角色创建必须要设置相应的密码，否则无法创建；PostgreSQL 则无此限制。

在成功创建新用户并执行“select * from pg_user;”后发现，在 PostgreSQL 中创建的新用户“admin”拥有“usecreatedb”权限，而 openGauss 内创建的新用户“usrtest3”无此权限。

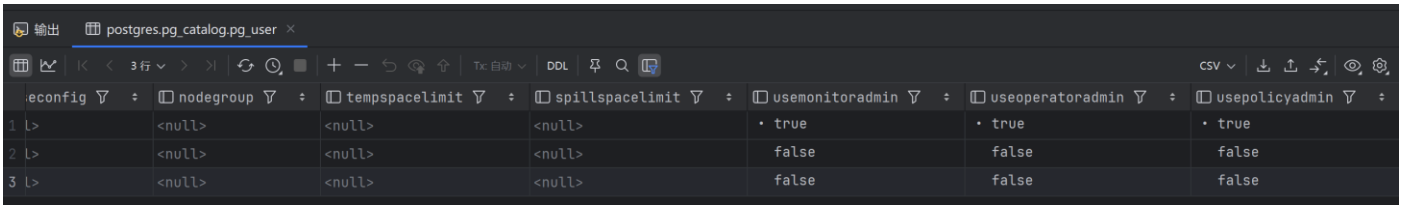
PostgreSQL：

	usename	usesysid	usecreatedb	usesuper	userepl	usebypassrls
1	postgres	10	true	true	true	true
2	usrtest3	113929	false	false	false	false
3	admin	113930	true	false	false	false

openGauss：

	usename	usesysid	usecreatedb	usesuper	usecatupd	userepl
1	omm	10	true	true	true	true
2	gaussdb	16385	false	false	false	false
3	usrtest3	107855	false	false	false	false

另外发现，在 pg_user 里 openGauss 对“usemonitoradmin”、“useoperatoradmin”等更多权限进行了细分。



	econfig	nodegroup	tempspacelimit	spillspacelimit	usemonitoradmin	useoperatoradmin	usepolicyadmin
1	l>	<null>	<null>	<null>	true	true	true
2	l>	<null>	<null>	<null>	false	false	false
3	l>	<null>	<null>	<null>	false	false	false

除了以上几项以外，其他几项实验项目中 openGauss 与 PostgreSQL 的表现一致。**该实验从粗浅的访问权限控制方面，表现了 openGauss 拥有更完善更高的安全性能。**

4 实验结论

从本实验结果来看，openGauss 作为一个基于 PostgreSQL 的企业级数据库管理系统，在查询性能和事务处理能力方面相对于 PostgreSQL 并无明显优化，反而出现了明显差距，但在连接能力和磁盘稳定性方面表现出色。在安全性方面，openGauss 展现出了更高的安全性能，这可能使其更适合对安全性要求较高的企业级应用场景。

openGauss 的优化可能更侧重于稳定性和安全性，而非单纯的性能提升。因此，在实际应用中，openGauss 可能更适合那些对数据安全和系统稳定性有较高要求的场景，如金融系统、电信和大型电子商务平台。

可以认为，openGauss 官方在其“高并发”、“高性能”优化介绍方面有一定的夸大成分，但它的稳定性与安全性确实得到了一定程度的保障和优化，这一点与其原本的自我定位相符合。

5 结语

虽然通过本结论均从实验得来，但是由于设备、配置、环境、时间等方面的限制，我仍旧无法很好地将试验条件最大化模拟各类高并发等现实场景，所以实验结果与结论与实际应当存在一定偏差。另外，openGauss 的拓展性、可靠性等方面也可惜没有机会展开深入探究。这也说明了，数据库的开发和优化是对于软件和硬件都非常复杂且困难的系统工程。

6 附录

1. Github 仓库：https://github.com/Tsurumalu/DB-Project3

2. 相关配置

计算机型号	Surface Pro 8 Model 1983 i7
CPU（处理器）	11th Gen Intel® Core™ i7-1185G7 @3.00GHz
已安装的 RAM（内存）	16GB
磁盘	SAMSUNG MZ9LQ512HALU-00000 (SSD)