

Benchmarking openGauss Against PostgreSQL: A Comparative Analysis

吴鑫亿 12310220

一、引言

我们希望通过罗列数据库的评价指标，依次测试 `OpenGauss` 和 `PostgreSQL`，比较性能优劣势，同时可以帮助我们理解这两种数据库系统在不同场景下的表现，还能为选择适合特定应用需求的数据库提供参考。

1.1 数据库性能评价指标

数据库的性能评价指标包括了：

1. 响应时间：请求完成所需要的时间，通常使用平均响应时间、中位数、90% 等等来衡量；
2. 吞吐量：衡量网络上成功运输的数量，单位是 Byte/s
3. TPS(**Transactions Per Second**)：系统在一定时间内能够处理的事务，每秒事务数TPS
4. QPS：每秒的查询率
5. RPS：系统内能够同时使用的接口
6. 并发用户数：系统能同时处理的最大用户请求数量
7. 资源利用率：包括CPU使用率、内存使用情况、磁盘I/O 和网络I/O
8. 错误率：测试过程中失败的请求所占的比例
9. 稳定性：在长时间运行下系统性能的波动情况
10. 安全性：衡量数据库系统能否在面对恶意攻击和非授权访问时保护其数据不被泄露、篡改或丢失的能力

对于上面的评价指标，我曾经了解较多的就只有响应时间、安全性，而对于这些不熟悉、不直观的数据测量，需要寻找专门的测量工具。其中 `Pgbeach` 无法连接 `OpenGauss`，其他测试工具大多都是在 `Linux` 环境下，最终选择了 `JMeter`，不过逐渐发现这个更多用在网站性能的测试上。

1.2 实验变量设置

经过研究，我认为影响数据库性能的因素包括了数据规模、用户（接口）数量、`SELECT` 查询的复杂程度、index 的使用、事务间隔级别等方面，接下来我将尝试设计实验，对这些有关的因素进行对照测试。

测试工具：`JMeter`，`DataGrip`

使用数据规模生成器 `insert_generator.cpp` 和 `query_generator.cpp` 进行数据的生成。

使用的表的结构为：包含两列数字，三列的字符串，模拟日常中可能会使用的数据。

```
D: > Desktop > Project3 > test_case > test_tables.sql
1 DROP TABLE IF EXISTS test_1;
2 CREATE TABLE test_1 (
3     col1 INT NOT NULL,
4     col2 INT NOT NULL,
5     col3 VARCHAR(20) NOT NULL,
6     col4 VARCHAR(20) NOT NULL,
7     col5 VARCHAR(20) NOT NULL,
8     PRIMARY KEY (col1)
9 );
```

二、基本测试

2.1 数据规模测试

我首先测试不同数据规模分别对 OpenGauss 和 PostgreSQL 进行了对比，测试背景是 单用户、多操作，包括了 insert, update, select 等等

2.1.1 对于一个用户，200个表，每个表的行数为1000的数据：

为了排除偶然因素的影响，我选择了多次测量取平均值，分别记录：响应时间平均数、响应时间中位数、响应时间90% 百分位数、发送速度、吞吐量等等。

INSERT 比较：

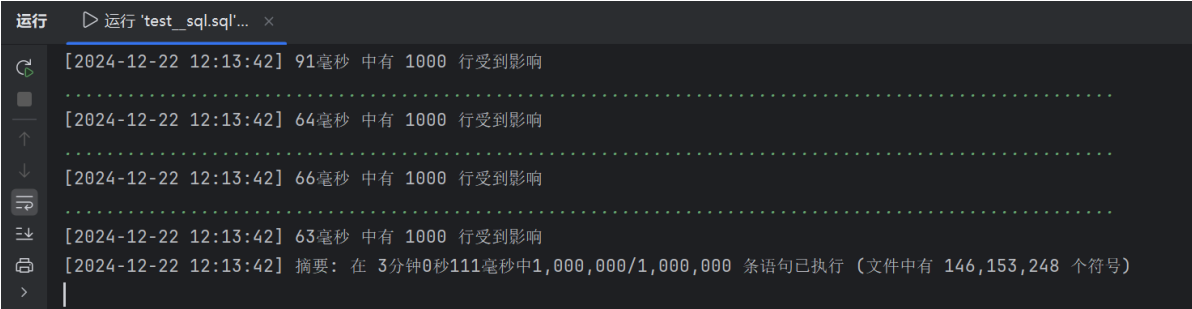
测试1		postgreSQL	Jmeter	test1	test2	test3	test4	test5	平均值
数据规模				响应时间平均	1	1	1	1	1
	用户数量	1		响应时间中位	1	1	1	1	1
	表的数量	20		90%百分位(ms)	2	2	2	2	2
	每个表的行数	1000		接受速度(KB/s)	27.8	27.44	28.13	27.9	27.85
	测试语句	insert		发送速度(KB/s)	0	0	0	0	0
	Transaction isolation default			吞吐量(Bytes/s)	694.3	685.4	702.5	696.7	694.8
				平均单次测试	32.13s				694.74
			DataGrip	每1000条sql	32ms				
				20,000总用时	2.996s				
		OpenGauss	Jmeter	test1	test2	test3	test4	test5	平均值
数据规模				响应时间平均	1	1	1	1	1
	用户数量	1		响应时间中位	2	2	2	2	2
	表的数量	20		90%百分位(ms)	2	2	2	2	2
	每个表的行数	1000		接受速度(KB/s)	23.09	23.06	23.04	22.92	23.08
	测试语句	insert		发送速度(KB/s)	0	0	0	0	0
	Transaction isolation default			吞吐量(Bytes/s)	576.7	577.3	575.5	572.5	576.4
				平均单次测试	34.54s				575.68
			DataGrip	每1000条sql	81ms				
				20,000总用时	3.841ms				

简单的 UPDATE、SELECT 比较：

测试2		postgreSQL	Jmeter	test1	test2	test3	test4	test5	平均值
数据规模				响应时间平均	0	0	0	0	0
	用户数量	1		响应时间中位	1	1	1	1	1
	表的数量	20		90%百分位(ms)	1	1	1	1	1
	每个表的行数	1000		接受速度(KB/s)	59.57	57.19	56.91	57.82	56.98
	测试语句	select&update		发送速度(KB/s)	0	0	0	0	0
	Transaction isolation default			吞吐量(Bytes/s)	1584.1	1521.3	1513.8	1525.7	1515.6
				平均单次测试	15.76s				1532.1
			DataGrip	每1000条sql	7.485s				
				总用时	3m43s652ms				
		OpenGauss	Jmeter	test1	test2	test3	test4	test5	平均值
数据规模				响应时间平均	0	0	0	0	0
	用户数量	1		响应时间中位	1	1	1	1	1
	表的数量	20		90%百分位(ms)	1	1	1	1	1
	每个表的行数	1000		接受速度(KB/s)	49.84	51.23	51.74	49.96	50.63
	测试语句	select&update		发送速度(KB/s)	0	0	0	0	0
	Transaction isolation default			吞吐量(Bytes/s)	1325.6	1362.9	1376.4	1331.1	1351.3
				平均单次测试	14.54s				1349.46
			DataGrip	每1000条sql	7.153ms				
				20,000总用时	3m22s205ms				

2.1.2 对于一个用户，50个表，每个表行数为20000，总共 1,000,000 的数据规模（耗时过久，不方便测量更大的规模）

使用 JMeter 进行百万级的 insert、select，需要的时间会超过30min；而使用 DataGrip 则分别需要 2,3 min即可。（下图中显示 1,000,000 条数据的总用时，在 OpenGauss 中）



以下为测试结果，横向比较下，OpenGauss 在百万级的数据规模中，在两个工具中响应速度、接受速度、吞吐量都要慢于 PostgreSQL，但是平均慢 5%-20%；同时 Datagrip 的执行 sql 的响应速度、吞吐量明显要优于 JMeter，可能是 Jmeter 可视化 UI 的影响，之后具体分析。

测试3							
		postgreSQL	Jmeter		test1	test2	平均值
数据规模				响应时间平均值(ms)	1	1	1
	用户数量		1	响应时间中位数(ms)	1	1	1
	表的数量		50	90%百分位(ms)	3	3	3
	每个表的行数		20000	接受速度(KB/sec)	23.32	24.15	23.735
	测试语句	insert		发送速度(KB/sec)	0	0	0
	Transaction isolation	default		吞吐量(Bytes/sec)	576.8	603.3	590.05
				平均单次测试用时	28m12s		
			DataGrip	每1000条sql	27.56ms		
				1,000,000总用时	2m24s326ms		
		OpenGauss	Jmeter		test1	test2	平均值
数据规模				响应时间平均值(ms)	0	0	0
	用户数量		1	响应时间中位数(ms)	1	1	1
	表的数量		50	90%百分位(ms)	1	1	1
	每个表的行数		20000	接受速度(KB/sec)	21.42	20.95	21.185
	测试语句	select&update		发送速度(KB/sec)	0	0	0
	Transaction isolation	default		吞吐量(Bytes/sec)	534.9	523.2	529.05
				平均单次测试用时	31m12s		
			DataGrip	每1000条sql	67.04ms		
				1,000,000总用时	3m0s111ms		

2.1.3 结论

通过上面两个规模不同的数据，分别进行 INSERT, SELECT, UPDATE 操作，并且同时使用了两种测试工具，最终的总体趋势就是 OpenGauss 在单用户下，在多种数据规模的情况下多个评价指标（吞吐量、响应速度）等都是不如 PostgreSQL 的。

2.2 复杂 SELECT 语句 优化测试

接下来，我选择测试不同数据库对于较复杂的 SELECT 语句 响应时间的表现。

在50个表，每个表有20,000行的数据集中，依次进行 100,000 条 SELECT 测试，比较响应时间时间、错误率、吞吐量等。

本次实验中基本变量为：表的数量为50，每个表的行数是20,000，事务隔离类型为 Default，用户数量为 20。

2.2.1 对照组测试：使用简单的SELECT sql 命令

发现 OpenGauss 仍然慢于 PostgreSQL 一些。

数据结果是：

测试4								
		postgreSQL	Jmeter		test1	test2	test3	平均值
数据规模				响应时间平均值(ms)	0	0	0	0
用户数量	20			响应时间中位数(ms)	1	1	1	1
命令数量	100000			90%百分位(ms)	1	1	1	1
表的数量	50			接受速度(KB/sec)	1318.85	1403.19	1414.96	1379
每个表的行数	20000			发送速度(KB/sec)	0	0	0	0
测试语句	select			吞吐量(Bytes/sec)	16289.3	17331	17476.4	17032.23
Transaction isolation	default			平均单次测试用时	8.57s			
		OpenGauss	Jmeter		test1	test2	test3	平均值
数据规模				响应时间平均值(ms)	1	1	1	1
用户数量	20			响应时间中位数(ms)	1	1	1	1
命令数量	100000			90%百分位(ms)	2	2	2	2
表的数量	50			接受速度(KB/sec)	1100.61	1141.99	1143.95	1128.85
每个表的行数	20000			发送速度(KB/sec)	0	0	0	0
测试语句	select			吞吐量(Bytes/sec)	11013.2	11427.3	11446.9	11295.8
Transaction isolation	default			平均单次测试用时	12.08s			

2.2.2 对照单用户的测试：

用户数量设置为 1，进行对照试验。

发现用时是 20 用户的近 10 倍，吞吐量、接受速度都是 20 个用户的 $\frac{1}{10}$ ，说明此时限制数据库响应速度的变量是线程的数量，20 个线程的性能显著比 1 个线程要快得多。

测试5								
		postgreSQL	Jmeter		test1	test2	test3	平均值
数据规模				响应时间平均值(ms)	0	1	0	0.333333
用户数量	20			响应时间中位数(ms)	1	1	1	1
命令数量	100000			90%百分位(ms)	1	3	1	1.666667
表的数量	50			接受速度(KB/sec)	159.61	158.77	158.91	159.0967
每个表的行数	20000			发送速度(KB/sec)	0	0	0	0
测试语句	select			吞吐量(Bytes/sec)	1597.1	1588.8	1590.1	1592
Transaction isolation	default			平均单次测试用时	1m7.35s			
		OpenGauss	Jmeter		test1	test2	test3	平均值
数据规模				响应时间平均值(ms)	0	0	0	0
用户数量	20			响应时间中位数(ms)	1	1	1	1
命令数量	100000			90%百分位(ms)	1	1	1	1
表的数量	50			接受速度(KB/sec)	92.71	103.43	95.28	97.14
每个表的行数	20000			发送速度(KB/sec)	0	0	0	0
测试语句	select			吞吐量(Bytes/sec)	927.7	1035	953.4	972.0333
Transaction isolation	default			平均单次测试用时	1m36.46s			

2.2.3 包含聚合函数的测试：

sql 语句如下图：

```
SELECT col2 , SUM(col2) AS sum_col2 FROM test_6 GROUP BY col2;
SELECT col2 , AVG(col1) AS avg_col1 FROM test_16 GROUP BY col2;
SELECT col2 , AVG(col1) AS avg_col1 FROM test_30 GROUP BY col2;
SELECT col2 , MAX(col1) AS max_col1 FROM test_47 GROUP BY col2;
SELECT col2 , AVG(col1) AS avg_col1 FROM test_32 GROUP BY col2;
SELECT col2 , AVG(col1) AS avg_col1 FROM test_1 GROUP BY col2;
```

我选择测试了聚合函数中的 SUM、AVG、MAX、MIN，随机在不同表中对于 col2 列聚合。为了体现聚合的效果，我将表中的 col2 数据设置在 1-2000 的范围内，总行数则是 20,000，意味着每条 col2 平均对应 10 行数据，这样也更加符合聚合函数的实际使用场景。

测试6									
		postgreSQL	Jmeter		test1	test2	test3	平均值	
数据规模				响应时间平均值(ms)	59	1	0	20	
用户数量	20			响应时间中位数(ms)	8	1	1	3.333333	
命令数量	100000			90%百分位(ms)	184	3	1	62.66667	
表的数量	50			接受速度(KB/sec)	9647.95	9677.87	9659.23	9661.683	
每个表的行数	20000			发送速度(KB/sec)	0	0	0	0	
测试语句	select			吞吐量(Bytes/sec)	321.5	322.6	322.3	322.1333	
Transaction isolation	default			平均单次测试用时	5m14.44s				
		OpenGauss	Jmeter		test1	test2	test3	平均值	
数据规模				响应时间平均值(ms)	237	208	214	219.6667	
用户数量	20			响应时间中位数(ms)	241	212	219	224	
命令数量	100000			90%百分位(ms)	317	280	286	294.3333	
表的数量	50			接受速度(KB/sec)	2470.08	2820.24	2699.65	2663.323	
每个表的行数	20000			发送速度(KB/sec)	0	0	0	0	
测试语句	select			吞吐量(Bytes/sec)	82.5	93.9	90.1	88.83333	
Transaction isolation	default			平均单次测试用时	17m26s				

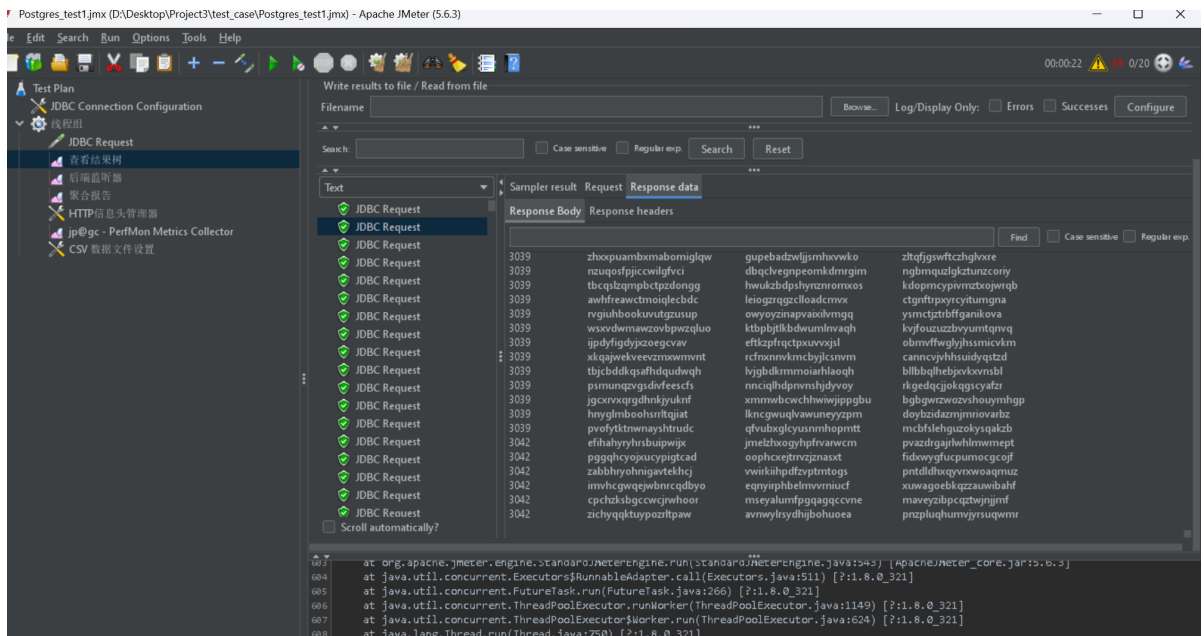
即使是多线程，100,000 条聚合查询的时间也远远超过了前面的简单查询。同时，在这次测试中，OpenGauss 的响应时间、发送速度、吞吐量多个维度都远远比不上 PostgreSQL。

2.2.4 包含join操作的测试：

选择的sql 语句如下，但是发现占用内存过多，每次运行到 450 左右个样本时就会自动停止。

```
SELECT t2.col1, t3.col3, t3.col4, t3.col5 FROM test_44 t2 LEFT JOIN test_16 t3 ON t2.col2 = t3.col2 WHERE t2.col2 < 922 AND t2.col1 < 3024 ;
SELECT t2.col1, t3.col3, t3.col4, t3.col5 FROM test_4 t2 LEFT JOIN test_15 t3 ON t2.col2 = t3.col2 WHERE t2.col2 < 1004 AND t2.col1 < 3176 ;
SELECT t2.col1, t3.col3, t3.col4, t3.col5 FROM test_24 t2 LEFT JOIN test_40 t3 ON t2.col2 = t3.col2 WHERE t2.col2 < 848 AND t2.col1 < 1692 ;
SELECT t2.col1, t3.col3, t3.col4, t3.col5 FROM test_28 t2 LEFT JOIN test_36 t3 ON t2.col2 = t3.col2 WHERE t2.col2 < 1298 AND t2.col1 < 9122 ;
SELECT t2.col1, t3.col3, t3.col4, t3.col5 FROM test_11 t2 LEFT JOIN test_4 t3 ON t2.col2 = t3.col2 WHERE t2.col2 < 683 AND t2.col1 < 8475 ;
SELECT t2.col1, t3.col3, t3.col4, t3.col5 FROM test_24 t2 LEFT JOIN test_40 t3 ON t2.col2 = t3.col2 WHERE t2.col2 < 1049 AND t2.col1 < 7984 ;
SELECT t2.col1, t3.col3, t3.col4, t3.col5 FROM test_50 t2 LEFT JOIN test_5 t3 ON t2.col2 = t3.col2 WHERE t2.col2 < 554 AND t2.col1 < 9090 ;
SELECT t2.col1, t3.col3, t3.col4, t3.col5 FROM test_45 t2 LEFT JOIN test_18 t3 ON t2.col2 = t3.col2 WHERE t2.col2 < 1059 AND t2.col1 < 6186 ;
SELECT t2.col1, t3.col3, t3.col4, t3.col5 FROM test_48 t2 LEFT JOIN test_18 t3 ON t2.col2 = t3.col2 WHERE t2.col2 < 1105 AND t2.col1 < 1649 ;
```

查看样本中的结果集发现，每一次询问都会返回 数百行的结果集，导致 JMeter 脚本自动停止。因此我需要优化 SQL 查询语句，将结果集大小限制后进行测试，否则会因过度占用内存强制停止。



调整过后的结果集大小在 5-50 行不等，最终结果如下，：

4	测试7								
5		postgreSQL	Jmeter		test1	test2	test3	平均值	
6	数据规模			响应时间平均值(ms)	65	64	65	64.66667	
7	用户数量	20		响应时间中位数(ms)	20	20	21	20.33333	
8	命令数量	100000		90%百分位(ms)	182	181	182	181.6667	
9	表的数量	50		接受速度(KB/sec)	534.2	537.3	538.4	536.6333	
0	每个表的行数	20000		发送速度(KB/sec)	0	0	0	0	
1	测试语句	select		吞吐量(Bytes/sec)	298.9	300.1	301.5	300.1667	
2	Transaction isolation	default		平均单次测试用时	5m34.77s				
3									
4									
5		OpenGauss	Jmeter		test1	test2	test3	平均值	
6	数据规模			响应时间平均值(ms)	253	231	240	241.3333	
7	用户数量	20		响应时间中位数(ms)	267	256	255	259.3333	
8	命令数量	100000		90%百分位(ms)	382	384	384	383.3333	
9	表的数量	50		接受速度(KB/sec)	137.55	134.53	137.98	136.6867	
0	每个表的行数	20000		发送速度(KB/sec)	0	0	0	0	
1	测试语句	select		吞吐量(Bytes/sec)	77.1	84.9	85.2	82.4	
2	Transaction isolation	default		平均单次测试用时	19m41.22s				
3									

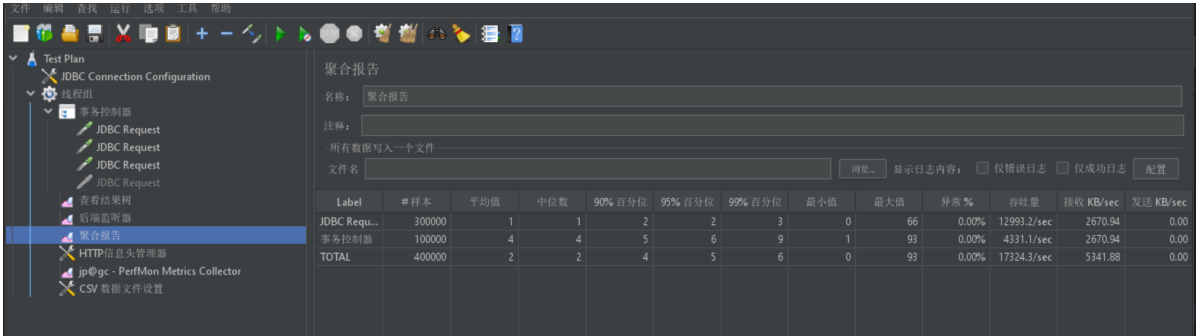
这也说明了结果集大小、数据传输速度也是限制数据库响应时间的关键因素，结果集尽量小则数据库处理速度更快，系统资源占用更低，从而可以提高整体的性能和稳定性。因此，在设计和优化SQL查询时，应尽可能减少不必要的数据返回，以确保查询效率和稳定性。

2.2.6 结论

在进行了一系列测试后，我们发现在没有考虑索引影响的情况下，OpenGauss 在执行简单和复杂查询时的性能普遍不如 PostgreSQL。然而，这一结论并不意味着 OpenGauss 在所有情况下都会表现较差。但是我当时设计实验的时候就没有考虑 INDEX 的影响，如果加上index，或许就能测试对比出哪种数据库的查询计划更加优异，因而得到更少的响应时间。

三、压力测试

我首先将参数设置为：100并发用户，5张表，每张表20000行，操作类型包括了 50% SELECT，30% UPDATE，20% INSERT，同时进行了 400,000 条查询，但是发现 PostgreSQL、OpenGauss 运行速度都很快，在 10s左右的时间内跑完，这样的测试显然没有



Label	# 样本	平均值	中位数	90% 百分位	95% 百分位	99% 百分位	最小值	最大值	异常 %	吞吐量	接收 KB/sec	发送 KB/sec
JDBC Requ...	300000	1	1	2	2	3	0	66	0.00%	12993.2/sec	2670.94	0.00
事务控制...	100000	4	4	5	6	9	1	93	0.00%	4331.1/sec	2670.94	0.00
TOTAL	400000	2	2	4	5	6	0	93	0.00%	17324.3/sec	5341.88	0.00

我逐渐转换思路，希望得到在相同时间内能完成的最多事务，以及两个数据库的极限吞吐量、TPS、并发用户数量。因此我设置了 60s的运行时间，逐渐改变 用户(线程)数量，尝试得到所能够运行的最大样本数量、吞吐量、传输速度等因素。

具体的测试数据如下。可以发现，PostgreSQL 的吞吐量、样本总量峰值出现在 40-60个线程之间，而 OpenGauss 虽然总体各指标更少一点，但是吞吐量等峰值却是出现在 100-200 线程之间，远远大于 Postgre 的线程峰值。

测试9			用户数量	20	50	100	150	75	60	40
	postgreSQL	Jmeter	测试时间	60s	60s	60s	60s	60s	60s	60s
数据规模			响应时间平均值(ms)	4	8	19	38	13	10	6
	命令数量	200000	响应时间中位数(ms)	4	8	17	22	11	9	6
	表的数量	20	90%百分位(ms)	6	12	31	56	21	16	9
	每个表的行数	20000	接受速度(KB/sec)	2463.5	3464.79	3046.62	2127.69	3302.16	3363.35	3363.49
	测试语句	select	发送速度(KB/sec)	0	0	0	0	0	0	0
	Transaction isolation	default	吞吐量(Bytes/sec)	3994.7	5618.4	4958.4	3527.6	5354.7	5454.1	5454.2
			样本	239657	337128	297577	211757	321438	327274	327257
	OpenGauss	Jmeter	用户数量	20	50	100	150	200	300	250
数据规模			测试时间	60s	60s	60s	60s	60s	60s	60s
	命令数量	200000	响应时间平均值(ms)	9	22	23	34	42	85	62
	表的数量	20	响应时间中位数(ms)	6	12	21	29	35	50	43
	每个表的行数	20000	90%百分位(ms)	21	37	40	60	78	114	103
	测试语句	select	接受速度(KB/sec)	1239.78	1280.84	2376.41	2506.64	2640.89	1969.93	2225
	Transaction isolation	default	发送速度(KB/sec)	0	0	0	0	0	0	0
			吞吐量(Bytes/sec)	2010.5	2089.7	3853.6	4065	4282.7	3204.2	3618.5
			样本	123651	125451	231287	244003	257001	192940	217522

根据网络上调查的资料，OpenGauss 最大的变化就是把 PostgreSQL 的进程模式改成了线程模式，当然这两个模式其实各有优缺点。线程模式对短连接有优势，比进程模式的数据库可以承担更大的并发短请求，但线程模式也有明显的缺点，所有的线程共享内存，如果一个线程的野指针把别人的内存改了，不会报错，一时半会可能还发现不了，极端情况下会导致数据损坏而不被发现。所以说这个改变不能说有什么明显的好处，某些情况下可能还是一个退步。为了改成线程模式，openGauss 的把 C 语言的源代码改成了 C++。C++的好处是容易封装，坏处是移植性降低了。

注：测试到最后，我才发现还有一个docker生成的两个数据库容器默认基本参数不一样，比如 共享缓存区、工作内存、有效缓存等等。可以通过以下 SQL 语言进行查看

```

show shared_buffers ;
show work_mem ;
show maintenance_work_mem ;
show effective_cache_size ;
show random_page_cost ;
show seq_page_cost ;
show temp_buffers ;
show query_dop;

```

四、 OpenGauss 和 PostgreSQL 其他区别

1. 事务隔离等级

在上面的压力测试中，使用了事务间的 BEGIN 和 COMMIT，我原本希望通过改变事务隔离级别，来比较两个数据库之间的安全性能。甚至，当我在测试中出现了错误率，我感觉我就会找到安全性的测试比较。然而，我意识到出现了 可重复读 或 读已提交 的情况，这些情况并不会导致错误，而是会展示出事务提交前或事务开始前的数据状态。显然，测试中出现的错误并不是由于数据库的安全性问题，而是由于测试命令本身存在问题，比如违反了主键的唯一性约束，或者表中的数据需要频繁更新等。

在事务隔离等级方面，OpenGauss支持两种标准的事务隔离级别：READ COMMITTED（读已提交）和REPEATABLE READ（可重复读），而不支持最严格的 SERIALIZABLE隔离级别。

2. X_{id} 的扩展

openGauss 将 Xid 由 int_32 改成了 int_64。原本 int_32 的值大约是 43亿($2^{32} - 1 = 4294967295$)。在 PostgreSQL 中，在一些高并发的系统中，可能会耗尽所有 Xid，导致出现事务 ID 回卷问题。

OpenGauss将Xid扩展为64位，显著增加了Xid的范围，从而减少了Xid耗尽的风险。但是显然这也会占用大量内存，因此 OpenGauss对每一个写事务都会分配一个唯一标识。当事务插入时，会将事务信息写到元组头部的xmin，代表插入该元组的xid；当事务进行更新和删除时，会将当前事务信息写入。使用 txid_current()函数用于获取当前事务ID，而gs_txid_oldestxmin()函数用于获取当前最小事务id的值 oldestxmin，这应该可以使用 二叉树之类的结构获取。

我原本希望通过下面的代码，测试 Xid 能不能溢出 int32，但是受限于硬件条件，每次运行 5min DataGrip 就会自动闪退！

```
50 DO $$
51 DECLARE
52     i INT := 1;
53 BEGIN
54     WHILE i <= 2147483647 LOOP
55         BEGIN
56             update test_2 set col2=1 where col3='%sad%';
57             COMMIT;
58             EXCEPTION WHEN others THEN
59                 -- 处理异常，可能是Xid耗尽
60                 RAISE NOTICE 'Error: %', SQLERRM;
61             END;
62             i := i + 1;
63         END LOOP;
64 END $$;
```

根据报错信息，闪退原因是 Java 虚拟机占用过高内存。



3. Checkpoint

PostgreSQL 使用的是全量检查点，执行全量检查点时，会将buffer中的所有脏页（Dirty Pages）刷新到磁盘，这个过程需要在一定时间内完成，可能会导致数据库性能波动较大，有时会高达 15% 的数据波动，严重影响性能。

而 OpenGauss 引入了增量检查点功能，这允许小批量、分阶段地进行脏页刷盘，减少了对业务的影响。

下图是在 PostgreSQL 中查看的脏页数量，经过检查点后数量从 93 变成了 8，脏页数量明显下降。不过在 OpenGauss 中需要使用其他拓展查看脏页数量，暂时无法检测。


```
64 END $$;
65
66 ✓ CREATE TABLE chkpt AS SELECT * FROM generate_series(1,10000) AS g(n);
67 ✓ CREATE EXTENSION IF NOT EXISTS pg_buffercache;
68 ✓ SELECT count(*) FROM pg_buffercache WHERE isdirty;
69 ✓ CHECKPOINT;|
```

输出 count(*):bigint x

count
93

4. 主备库模式差异

在PostgreSQL 中，备库模式是拉的模式，即备库主动到主库上拉WAL日志。而OpenGauss 则改成了推的模式，即主库主动将WAL日志推送到备库。这种模式可以有效避免因网络抖动、进程闪断等因素导致主机在最大保护模式和最大可用模式之间频繁来回切换导致的性能损失。

我也尝试对主备库状态进行测试，但是发现对应的 gs_om 时一个服务端工具，而不是一个 SQL 函数。

```
49 SELECT * FROM pg_xact_commit_timestamp();
50 SELECT gs_om('status --detail');|
```

[42883] ERROR: function gs_om(unknown) does not exist
建议: No function matches the given name and argument types. You might need to add explicit type casts.
位置: 8
在位置: referenced column: gs_om

输出 pg_reload_conf():boolean

```
postgres: gaussdb, public> SELECT gs_om('status --detail')
[2024-12-22 23:13:54] [42883] ERROR: function gs_om(unknown) does not exist
[2024-12-22 23:13:54] 建议: No function matches the given name and argument types. You might need to add explicit type casts.
[2024-12-22 23:13:54] 位置: 8
[2024-12-22 23:13:54] 在位置: referenced column: gs_om
```

根据调查资料，可以发现 OpenGauss 在 PostgreSQL 的基础上做出了很多改进，但是其中的绝大部分，就我现在的知识水平和硬件条件不能测试出来。

五、优化和改进

5.1 误差分析

1. JMeter 的初始化

在测试过程中，我发现在刚开始运行 JMeter 时会存在卡顿，调查后发现在 Jmeter 运行前，有初始化过程，包括设置线程组、添加配置元件、数据库连接等等。

为了减少初始化时间，我们可以预先配置好测试计划并保存为模板，下次测试时直接加载模板进行微调。此外，可以考虑使用 JMeter 的命令行模式运行测试，以减少图形界面的初始化时间。

2. 数据库基本参数不一致

测试到最后，我才发现还有一个 docker 生成的两个数据库容器默认基本参数不一样，比如 共享缓存区、工作内存、有效缓存等等。可以通过以下 SQL 语言进行查看

```
show shared_buffers ;
show work_mem ;
show maintenance_work_mem ;
show effective_cache_size ;
show random_page_cost ;
show seq_page_cost ;
show temp_buffers ;
show query_dop;

SELECT pg_reload_conf(); //一定要执行重新加载配置，否则没有用
```

5.2 取消UI界面的JMeter

不适用 JMeter 的UI 图形界面，需要提前设置好线程、样本数量和运行时间， csv 文件会自动导入，脚本自动生成对应的 SQL 语句，然后将 查看结果树 中的内容导出到输出文件上即可，在命令行中也显示对应的平均数、中位数、最大值、最小值、错误率等结果。

```
PS D:\Desktop\Project3\test_case> jmeter -n -t Opengauss_test.jmx -l results.jtl
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future release
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future release
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future release
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future release
Creating summariser <summary>
Created the tree successfully using Opengauss_test.jmx
Starting standalone test @ December 22, 2024 11:20:58 PM CST (1734880858927)
Waiting for possible Shutdown/StopTestNow/HeapDump/ThreadDump message on port 4445
summary + 1 in 00:01:01 = 0.0/s Avg: 45 Min: 45 Max: 45 Err: 0 (0.00%) Active: 249 Started: 250 Finished: 1
summary + 249 in 00:01:01 = 4.1/s Avg: 454 Min: 1 Max: 61355 Err: 48 (19.28%) Active: 1 Started: 250 Finished: 249
summary = 250 in 00:02:02 = 2.0/s Avg: 452 Min: 1 Max: 61355 Err: 48 (19.20%)
summary = 250 in 00:02:04 = 2.0/s Avg: 452 Min: 1 Max: 61355 Err: 48 (19.20%)
Tidying up ... @ December 22, 2024 11:23:03 PM CST (1734880983402)
... end of run
PS D:\Desktop\Project3\test_case> jmeter -n -t Opengauss_test.jmx -l results.jtl
```

5.3 观测CPU、内存占用，比较资源利用率

通过比较，可以发现运行过程中，OpenGauss 可以使用更多资源，且采用单进程多线程架构，这种架构有助于提高资源利用率和处理高并发请求的能力。



5.4 其他可以优化的部分

1. 在表的设置这部分，没有考虑外键约束等常见因素，考虑不够全面。
2. 缺少 INDEX 索引

在进行 select 查询时，没有有目的的创建 INDEX，导致不能很好的体现 SQL 优化计划的不同。因为总是处理上万的数据，甚至忘记了使用 EXPLAIN 或 EXPLAIN ANALYZE 命令分析查询计划，并根据分析结果调整索引策略。

六、结论

6.1 OpenGauss 的优劣势

优势：

- 1. **资源利用率**：OpenGauss的单进程多线程架构在高并发场景下表现出色，能够更有效地利用CPU和内存资源，尤其是在多核CPU环境下。
- 2. **扩展性**：OpenGauss将Xid从int32扩展到int64，显著增加了事务ID的范围，减少了Xid耗尽的风险，适合高并发事务处理。
- 3. **增量检查点**：引入增量检查点机制，减少了对业务的影响，平滑了IO，降低了数据库性能波动。

劣势：

- 1. **性能表现**：在一些测试场景中，OpenGauss的性能表现不如PostgreSQL，尤其是在复杂查询和大数据量处理方面。（也可能是测试软硬件问题）
- 2. **功能支持**：OpenGauss在某些高级功能，如逻辑解码和并行处理方面，可能不如PostgreSQL成熟，同时使用 C++ 后，移植性降低了。

6.2 结语

通过本次对 openGauss 和 PostgreSQL 的比较分析，我们得以管中窥豹，了解了这两种数据库系统在不同场景下的表现。虽然 openGauss 可能在安全、高并发等领域存在优势，而这部分我不能很好地测试出来，但在性能和生态系统方面仍有提升空间。PostgreSQL 作为一个成熟且广泛使用的数据库系统，其稳定性和丰富的功能集得到了市场的验证，在多项比较中都表现出色。

本次测试也是我个人学习和成长的过程。通过亲手设计实验、执行测试、分析数据，我对数据库的工作原理和性能优化有了更深刻的理解。我相信，随着技术的不断进步和我个人能力的提高，我们能够更好地利用数据库技术，解决实际问题，创造价值。

Reference

- 1. [openGauss与PostgreSQL对比 - 墨天轮](#)
- 2. [openGauss数据与PostgreSQL的差异对比 | openGauss社区](#)
- 3. [openGauss和PostgreSQL深度差异对比_opengauss postgresql-CSDN博客](#)

附录

测试环境

测试环境	型号或者版本	备注
PostgreSQL	PostgreSQL 14.12 (Ubuntu 14.12-0ubuntu0.22.04.1)	Docker 镜像
OpenGauss	openGauss 3.0.0 build 02c14696	Docker 镜像
JMeter	JMeter 5.6.3	
DataGrip	2024.2.2	

测试环境	型号或者版本	备注
Java	1.8.0	适配 JMeter
笔记本处理器	AMD Ryzen 9 7940H w/ Radeon 780M Graphics 4.00 GHz	
机带 RAM	16.0 GB （15.2 GB可用）	
笔记本系统	Windows 11 家庭中文版	

代码、数据文件说明：

git仓库地址：<https://github.com/wly561wly/Project3-of-database.git>

文件名	功能	备注
Opengauss_test.jmx	JMeter中 openGauss 配置文件	
Postgres_test1.jmx	JMeter中 PostgreSQL 配置文件	
query_dataGenerator.cpp	随机生成select/update对应的 csv/sql文件	
insert_dataGenerator.cpp	随机生成 insert 对应的 csv/sql文件	
data.xlsx	记录了所有的测试数据	
picture(folder)	存储了本文档中的对应图片	
test_tables.sql	包含 更新测试table 的sql 命令	
query_sql.sql	包含 SELECT 测试中 sql 命令	