Project3: Benchmarking openGauss Against PostgreSQL

12312507 林政宇

https://github.com/lzy2005/Project3-Benchmarking-openGauss-Against-PostgreSQL

一、 安装 PostgreSql-14.11

1.1. 进入 omm 账户，切换到根目录，并配置文件目录

```
su - omm
cd ../..
mkdir -p /dbs/pg14/data
mkdir -p /pg14/soft
chown -R postgres:postgres /dbs
chown -R postgres:postgres /pg14
chmod -R 775 /pg14
chmod -R 775 /dbs
```

1.2. 安装 PSql 依赖

```
dnf install -y perl-ExtUtils-Embed readline-devel python3-devel pam-devel libxml2-devel libxslt-devel openldap-devel lz4-devel llvm-devel systemd-devel container-selinux selinux-policy-devel openssl-devel clang-devel flex-devel bison-devel glibc-devel gcc-c++ gcc cmake lsof net-tools tar zlib-devel --allowerasing --skip-broken
```

1.3. 在/pg14/soft 目录下获取 PostgreSql-14.11 安装包并解压安装

```
cd pg14/soft
wget https://ftp.postgresql.org/pub/source/v14.11/postgresql-14.11.tar.gz
tar -zxvf postgresql-14.11.tar.gz
cd postgresql-14.11
./configure --prefix=/pg14/soft --without-readline
make && make install
```

1.4. 配置环境变量

```
vim ~/.bash_profile
```

在文件末尾添加

```
export PGPORT=5666
export PGDATA=/dbs/pg14/data
export PGHOME=/pg14/soft
export PATH=$PGHOME/bin:$PATH:.
```

使环境变量生效

```
source ~/.bash_profile
```

1.5. 初始化 PSql 并启动

```
initdb
pg_ctl start -D $PGDATA
```

二、 评价的标准确定

评价数据库应有以下**标准**:

1. 数据库执行各种操作的效率，如 insert, select, index, join 等。
2. 数据库执行各种操作的稳定性，一是效率上的稳定性，二是执行正确语句的成功率。
3. 数据库面对外部攻击的安全性，例如对抗注入攻击的能力。

对于具体的评价:

创建三个表 table1(id int, name char(5), property int), table2(id int, name char(5), region_code char(10)) 和 table3(id int, property int, salary int)，每个表均加入 $n=10^6$ 组数据，并以同样的格式创建三个表 s_table1, s_table2 和 s_talbe3，每个表均加入 $n=10^4$ 组数据，以重点测试 OpenGauss 在复杂的，高数据强度的场景下的表现，并通过对照确定 OpenGauss 是否在该场景下有更好的表现。代码如下:

```
create table table1 (id int, name char(5), property int);
create table s_table1 (id int, name char(5), property int);
create table table2(id int, name char(5), region_code char(10));
create table s_table2(id int, name char(5), region_code char(10));
create table table3(id int, property int, salary int);
create table s_table3(id int, property int, salary int);
```

对于测试，分析**关键指标： insert, select, update, delete, join, index 等操作在 OpenGauss 和 PostgreSql-14.11 下的运行时间(运行效率)**并进行对比，以得出 OpenGauss 相对于 PostgreSql-14.11 的优劣之处与其总体的特性。

2.0. table1, table2, table3 及其 small 版本的构建

对于 table1，使用如下代码进行构建，使得每个出现的 name 和 property 恰有 10 个 id 与其对应，以使 join 时的大小分布合理地增长。

```
do $$
begin
    for i in 0..999999 loop
        insert into table1(id,name,property) values (
            i,
            chr(65+i/100000)||
            chr(97+i%100000/10000)||
            chr(97+i%10000/1000)||
            chr(97+i%1000/100)||
            chr(97+i%100/10),
            (i%100000)+1
        );
    end loop;
end
$$;
```

对于 s_table1，将表名改为 s_table1，for 循环的 i 上界改为 9999，property 中的 i 取模改为 1000，再做执行。

对于 table2，使用如下代码进行构建，使得 table2 和 table1 的 name 相对应，region_code 为随机生成的不定长字符串，以测试对不定长数据进行各种操作的情况下 OpenGauss 和 PSql 的表现差异。

```
create or replace function rnd_char9() returns char(9) as $$
declare
    o integer;
begin
    o = floor(random() * 9)+1;
    if o=1 then
        return to_char(floor(random() * 10),'0');
    elsif o=2 then
        return to_char(floor(random() * 100),'00');
    elsif o=3 then
        return to_char(floor(random() * 1000),'000');
    elsif o=4 then
        return to_char(floor(random() * 10000),'0000');
    elsif o=5 then
        return to_char(floor(random() * 100000),'00000');
    elsif o=6 then
        return to_char(floor(random() * 1000000),'000000');
    elsif o=7 then
        return to_char(floor(random() * 10000000),'0000000');
    elsif o=8 then
        return to_char(floor(random() * 100000000),'00000000');
    else
        return to_char(floor(random() * 1000000000),'000000000');
    end if;
end;
$$language plpgsql;

select setseed(0);
do $$
begin
    for i in 0..999999 loop
        insert into table2(id,name,region_code) values (
            i,
            chr(65+i/100000)||
            chr(97+i%100000/10000)||
            chr(97+i%10000/1000)||
            chr(97+i%1000/100)||
            chr(97+i%100/10),
            rnd_char9()
        );
    end loop;
```

```
end
$$;
```

对于 s_table2，更改表名并将 for 循环的 i 上界改为 9999，再做执行。

对于 table3，使用如下代码进行构建，使得 table3 和 table1 的 property 相对应，salary 为在 $[0, 10^8)$ 范围内生成的随机正整数。

```
create or replace function rnd_int8() returns integer as $$
declare
begin
    return floor(random()*100000000);
end;
$$language plpgsql;

select setseed(0);
do $$
begin
    for i in 0..999999 loop
        insert into table3(id,property,salary) values (
            i,
            (i%100000)+1,
            rnd_int8()
        );
    end loop;
end
$$;
```

对于 s_table3，更改表名并将 for 循环的 i 上界改为 9999，property 中的 i 取模改为 1000，再做执行。

2.1. insert 评价方式

在执行表的构建之前运行代码

```
\timing on
```

打开时间记录，以在执行的同时测试 OpenGauss 与 PSql 在三种类型上的 insert 效率。

2.2. select 评价方式

对于 select，分别从“全表 select 关键量”，“全表 select 全部内容”，“全表排序 select”，“简单 where select”，“where like select”和“select 嵌套 select”六个方面评价效率，具体评价方式如下：

(1) 全表 select 关键量(count)

对于 OpenGauss 和 PSql，执行代码

```
begin;
explain analyze select count(*) from table1;
rollback;
```

并在更改表名后对 s_table1 执行该代码，以评估两者的效率。

(2) 全表 select 全部内容

对于 OpenGauss 和 PSql，执行代码

```
begin;
explain analyze update table2 set region_code=name||to_char(id/1000,'000') where id<1000;
rollback;
```

并在更改表名后对 s_table2 执行该代码，以评估两者的效率。

(3) 全表排序 select

对于 OpenGauss 和 PSql，执行代码

```
begin;
explain analyze select * from table3 order by salary;
rollback;
```

并在更改表名后对 s_table3 执行该代码，以评估两者的效率。

(4) 简单 where select

对于 OpenGauss 和 PSql，执行代码

```
begin;
explain analyze select * from table1 where property<10000;
rollback;
```

并在更改表名后对 s_table1 执行该代码，以评估两者的效率。

(5) where like select

对于 OpenGauss 和 PSql，执行代码

```
begin;
explain analyze select * from table2 where name like '%ac%';
rollback;
```

并在更改表名后对 s_table2 执行该代码，以评估两者的效率。

(6) select 嵌套 select

对于 OpenGauss 和 PSql，执行代码

```
begin;
explain analyze select * from
(select * from table3 where property>30000) t1
where salary<1000000;
rollback;
```

并在更改表名，将“property>30000”改为“property>300”后对“s_table3”执行该代码，以评估两者的效率。

2.3. update 评价方式

对于 update，分别从“全表 update”，“简单 where update”，“where like update”，“update 嵌套 select”四个方面评价效率，具体评价方式如下：

(1) 全表 update

对于 OpenGauss 和 PSql，执行代码

```
begin;
explain analyze update table1 set property=id*10;
rollback;
```

并在更改表名后对 s_table1 执行该代码，以评估两者的效率。

(2) 简单 where update

对于 OpenGauss 和 PSql，执行代码

```
begin;
explain analyze update table2 set region_code=name||to_char(id/1000,'000') where id<1000;
rollback;
```

并在更改表名后对 s_table2 执行该代码，以评估两者的效率。

(3) where like update

对于 OpenGauss 和 PSql，执行代码

```
begin;
explain analyze update table1 set property=property+id where name like '%b%e%';
rollback;
```

并在更改表名后对 s_table1 执行该代码，以评估两者的效率。

(4) update 嵌套 select

对于 OpenGauss 和 PSql，执行代码

```
begin;
explain analyze update table2 set propery=property+id where name like '%b%e%';
rollback;
```

并在更改表名后对 s_table2 执行该代码，以评估两者的效率。

2.4. delete 评价方式

对于 delete，分别从"全表 delete"，"简单 where delete"，"where like delete"三个方面评价效率，具体评价方式如下：

(1) 全表 delete

对于 OpenGauss 和 PSql，执行代码

```
begin;
explain analyze delete from table1;
rollback;
```

并在更改表名后对 s_table1 执行该代码，以评估两者的效率。

(2) 简单 where delete

对于 OpenGauss 和 PSql，执行代码

```
begin;
explain analyze delete from table2 where id%2=0;
rollback;
```

并在更改表名后对 s_table2 执行该代码，以评估两者的效率。

(3) where like delete

对于 OpenGauss 和 PSql，执行代码

```
begin;
explain analyze delete from table2 where region_code like '%3%7%';
rollback;
```

并在更改表名后对 s_table2 执行该代码，以评估两者的效率。

2.5. join 评价方式

对于 join，分别从“table1&table2 inner join on select”，“table1&table3 left outer join on select”，“table1&table2&table3 inner join on select”，“table1&table2&table3 cross join select where”四个方面评价效率，具体评价方式如下：

(1) table1&table2 inner join on select

对于 OpenGauss 和 PSql，执行代码

```
begin;
explain analyze select count(*) from(
    table1 join table2 on table1.name=table2.name
);
rollback;
```

并在更改表名后对 s_table 执行该代码，以评估两者的效率。

(2) table1&table3 left outer join on select

对于 OpenGauss 和 PSql，执行代码

```
begin;
explain analyze select count(*) from(
    table1 left outer join table3 on table1.property=table3.property
);
rollback;
```

并在更改表名后对 s_table 执行该代码，以评估两者的效率。

(3) table1&table2&table3 inner join on select

对于 OpenGauss 和 PSql，执行代码

```
begin;
explain analyze select count(*) from(
    table1 join table2 on table1.name=table2.name join table3 on table1.property=table3.property
);
rollback;
```

并在更改表名后对 s_table 执行该代码，以评估两者的效率。

(4) table1&table2&table3 cross join select where

对于 OpenGauss 和 PSql，执行代码

```
begin;
explain analyze select count(*) from table1 cross join table2 cross join table3
where table1.name=table2.name and table1.property=table3.property;
```

```
rollback;
```

并在更改表名后对 s_table 执行该代码，以评估两者的效率。

2.6. index 评价方式

对于 index，分别从“create index”，“index select”，“index select(upper_case function)”，“create multi index”，“multi index select(full match)”和“multi index select(partial match)”六个方面评价效率，具体评价方式如下：

(1) create index

首先输入代码

```
\timing on
```

打开计时。

对于 OpenGauss 和 PSql，执行代码

```
create index idx1 on table1(id);
```

并在更改表名和 index 名后对 s_table1 执行该代码，以评估两者的效率。

(2) index select

对于 OpenGauss 和 PSql，执行代码

```
begin;
explain analyze select count(*) from table1 where name>'Ac' and name <'Bd';
rollback;
```

并在更改表名后对 s_table1 执行该代码，以评估两者的效率。

(3) index select(upper_case function)

对于 OpenGauss 和 PSql，执行代码

```
begin;
explain analyze select count(*) from table1 where upper(name)>'AC' and upper(name) <'BD';
rollback;
```

并在更改表名后对 s_table1 执行该代码，以评估两者的效率。

(4) create multi index

对于 OpenGauss 和 PSql，执行代码

```
create index idx3 on table2(name,region_code);
```

并在更改表名和 index 名后对 s_table2 执行该代码，以评估两者的效率。

(5) multi index select(full match)

对于 OpenGauss 和 PSql，执行代码

```
begin;
explain analyze select count(*) from table2 where name>'BC' and region_code<'79';
rollback;
```

并在更改表名后对 s_table2 执行该代码，以评估两者的效率。

(6) multi index select(partial match)

对于 OpenGauss 和 PSql，执行代码

```
begin;
explain analyze select count(*) from table2 where name<'CF' and id%2=0;
rollback;
```

并在更改表名后对 s_table2 执行该代码，以评估两者的效率。

三、 评价实验与结论

3.1. insert 实验数据与结论

Runtime(ms)	table1	table2	table3		s_table1	s_table2	s_table3
OpenGauss	71642.226	136866.403	73073.105		649.576	1262.361	737.747
PSql	8435.347	13813.677	6007.169		91.734	160.904	64.201

insert 结论：Psql 在小数据和大数据上的 insert 表现均远好于 OpenGauss，效率差距大致为 8~10 倍，且随着数据量的增大，Psql 与 OpenGauss 的差距逐渐增大。

3.2. select 实验数据与结论

(1) 全表 select 关键量(count)：

Runtime(ms)	table	s_table
OpenGauss	252.696	5.597
PSql	77.459	1.477

结论：PSql 处理小数据和大数据的效率均约为 OpenGauss 的 3.7 倍。随着数据量的增大，Psql 与 OpenGauss 的差距略微缩小。

(2) 全表 select 全部内容：

Runtime(ms)	table	s_table
OpenGauss	2241.813	3.480
PSql	117.339	1.109

结论：PSql 处理小数据的效率约为 OpenGauss 的 3 倍，处理大数据的效率约为 OpenGauss 的 20 倍，随着数据量的增大，Psql 与 OpenGauss 的差距相当显著地增大。

(3) 全表排序 select：

Runtime(ms)	table	s_table
OpenGauss	2878.200	7.029
PSql	594.657	4.009

结论: PSql 处理小数据的效率约为 OpenGauss 的 1.75 倍, 处理大数据的效率约为 OpenGauss 的 4.8 倍，随着数据量的增大，Psql 与 OpenGauss 的差距显著增大。

(4) 简单 where select：

Runtime(ms)	table	s_table
OpenGauss	223.405	3.468
PSql	83.241	1.762

结论：PSql 处理小数据的效率约为 OpenGauss 的 2 倍，处理大数据的效率约为 OpenGauss 的 2.7 倍，随着数据量的增大，Psql 与 OpenGauss 的差距逐渐增大。

(5) where like select:

Runtime(ms)	table	s_table
OpenGauss	270.353	3.174
PSql	137.415	1.606

结论：PSql 处理小数据和大数据的效率均约为 OpenGauss 的 1.97 倍，随着数据量的增大，Psql 与 OpenGauss 的差距非常不明显地缩小。

(6) select 嵌套 select:

Runtime(ms)	table	s_table
OpenGauss	218.793	2.530
PSql	49.120	0.879

结论：PSql 处理小数据的效率约为 OpenGauss 的 3 倍，处理大数据的效率约为 OpenGauss 的 4.5 倍，随着数据量的增大，Psql 与 OpenGauss 的差距逐渐增大。

select 结论：PSql 处理小数据的效率普遍为 OpenGauss 的 2~3 倍，处理大数据的效率普遍为 OpenGauss 的 2~5 倍，随着数据量的增大，Psql 与 OpenGauss 的差距普遍较明显地增大，但在“select 全局关键量”，“where like select”两个任务上差距略微缩小，在“全表 select 全部内容”任务上差距非常显著地增大，对比认为 OpenGauss 在输出上的优化与 PSql 有很大差距。

3.3. update 实验数据与结论

(1) 全表 update:

Runtime(ms)	table	s_table
OpenGauss	5200.875	33.375
PSql	1855.264	17.832

结论：PSql 处理小数据的效率约为 OpenGauss 的 2 倍，处理大数据的效率约为 OpenGauss 的 3 倍，随着数据量的增大，Psql 与 OpenGauss 的差距逐渐增大。

(2) 简单 where update:

Runtime(ms)	table	s_table
OpenGauss	208.575	8.313
PSql	88.916	7.582

结论：PSql 处理小数据的效率约为 OpenGauss 的 1.1 倍，处理大数据的效率约为 OpenGauss 的 2.3 倍，随着数据量的增大，Psql 与 OpenGauss 的差距逐渐增大。

(3) where like update:

Runtime(ms)	table	s_table
OpenGauss	6717.642	6.990
PSql	254.895	2.104

结论：PSql 处理小数据的效率约为 OpenGauss 的 3 倍，处理大数据的效率约为 OpenGauss 的 26.34 倍，随着数据量的增大，Psql 与 OpenGauss 的差距相当显著地增大。

(4) update 嵌套 select:

Runtime(ms)	table	s_table

OpenGauss	52815.937	272.683
PSql	36114.431	193.452

结论：PSql 处理小数据的效率约为 OpenGauss 的 1.4 倍，处理大数据的效率约为 OpenGauss 的 1.46 倍，随着数据量的增大，Psql 与 OpenGauss 的差距略微增大。

update 结论：PSql 处理小数据的效率普遍为 OpenGauss 的 1.1~3 倍，处理大数据的效率普遍为 OpenGauss 的 1.5~3 倍，随着数据量的增大，Psql 与 OpenGauss 的差距普遍逐渐增大，但在“where like update”任务中相当显著地增大，说明 OpenGauss 在处理 update 结合 where like 的任务时优化相当不足。

3.4. delete 实验数据与结论

(1) 全表 delete：

Runtime(ms)	table	s_table
OpenGauss	2638.709	17.210
PSql	691.506	6.203

结论：PSql 处理小数据的效率约为 OpenGauss 的 3 倍，处理大数据的效率约为 OpenGauss 的 3.8 倍，随着数据量的增大，Psql 与 OpenGauss 的差距逐渐增大。

(2) 简单 where delete：

Runtime(ms)	table	s_table
OpenGauss	2679.456	12.142
PSql	454.113	3.976

结论：PSql 处理小数据的效率约为 OpenGauss 的 3 倍，处理大数据的效率约为 OpenGauss 的 5.9 倍，随着数据量的增大，Psql 与 OpenGauss 的差距逐渐增大。

(3) where like delete：

Runtime(ms)	table	s_table
OpenGauss	2316.474	5.543
PSql	340.601	2.748

结论：PSql 处理小数据的效率约为 OpenGauss 的 2 倍，处理大数据的效率约为 OpenGauss 的 6.8 倍，随着数据量的增大，Psql 与 OpenGauss 的差距逐渐增大。

delete 结论：PSql 处理小数据的效率普遍为 OpenGauss 的 2~3 倍，处理大数据的效率普遍为 OpenGauss 的 4~6 倍，随着数据量的增大，Psql 与 OpenGauss 的差距普遍逐渐增大，说明 OpenGauss 在处理 delete 任务时优化相当不足。

3.5. join 实验数据与结论

(1) table1&table2 inner join on select：

Runtime(ms)	table	s_table
OpenGauss	7001.256	40.989
PSql	1495.828	26.492

结论：PSql 处理小数据的效率约为 OpenGauss 的 1.5 倍，处理大数据的效率约为 OpenGauss 的 5 倍，随着数据量的增大，Psql 与 OpenGauss 的差距显著增大。

(2) table1&table3 left outer join on select：

Runtime(ms)	table	s_table
OpenGauss	4050.180	37.914
PSql	1360.176	22.433

结论：PSql 处理小数据的效率约为 OpenGauss 的 1.7 倍，处理大数据的效率约为 OpenGauss 的 3 倍，随着数据量的增大，Psql 与 OpenGauss 的差距逐渐增大。

(3) table1&table2&table3 inner join on select：

Runtime(ms)	table	s_table
OpenGauss	34573.372	334.461
PSql	13427.709	229.477

结论：PSql 处理小数据的效率约为 OpenGauss 的 1.5 倍，处理大数据的效率约为 OpenGauss 的 2.6 倍，随着数据量的增大，Psql 与 OpenGauss 的差距逐渐增大。

(4) table1&table2&table3 cross join select where：

Runtime(ms)	table	s_table
OpenGauss	34765.139	332.916
PSql	13154.805	229.749

结论：PSql 处理小数据的效率约为 OpenGauss 的 1.5 倍，处理大数据的效率约为 OpenGauss 的 2.6 倍，随着数据量的增大，Psql 与 OpenGauss 的差距逐渐增大。

join 结论：PSql 处理小数据的效率普遍为 OpenGauss 的 1.5~1.7 倍，处理大数据的效率普遍为 OpenGauss 的 2.6~5 倍，随着数据量的增大，Psql 与 OpenGauss 的差距普遍逐渐增大，但在“table1&table2 inner join on select”任务中显著地增大，对比任务(2)说明 OpenGauss 在处理以 char 为匹配内容的任务时优化相当不足；对比(3)(4)发现 OpenGauss 和 PSql 一样对 cross join 叠加 select where 有优化处理，效果与“inner join on+条件”基本相同。

3.6. index 实验数据与结论

(1) create index：

Runtime(ms)	table	s_table
OpenGauss	3663.337	25.566
PSql	399.126	9.481

结论：PSql 处理小数据的效率约为 OpenGauss 的 2.6 倍，处理大数据的效率约为 OpenGauss 的 9 倍，随着数据量的增大，Psql 与 OpenGauss 的差距显著增大。

(2) index select：

Runtime(ms)	table	s_table
OpenGauss	45.522	0.076
PSql	34.780	0.029

结论：PSql 处理小数据的效率约为 OpenGauss 的 2.6 倍，处理大数据的效率约为 OpenGauss 的 1.3 倍，随着数据量的增大，Psql 与 OpenGauss 的差距显著减小。

(3) index select (upper_case function)：

Runtime(ms)	table	s_table

OpenGauss	1911.305	10.241
PSql	559.301	6.141

结论：PSql 处理小数据的效率约为 OpenGauss 的 1.6 倍，处理大数据的效率约为 OpenGauss 的 3.4 倍，随着数据量的增大，Psql 与 OpenGauss 的差距逐渐增大。

(4) create multi index：

Runtime(ms)	table	s_table
OpenGauss	4142.315	38.283
PSql	1420.561	25.716

结论：PSql 处理小数据的效率约为 OpenGauss 的 1.5 倍，处理大数据的效率约为 OpenGauss 的 2.9 倍，随着数据量的增大，Psql 与 OpenGauss 的差距逐渐增大。

(5) multi index select(full match)（重复实验三次）：

Runtime(ms)	table	s_table
OpenGauss	632.266	0.077
PSql	176.507	0.030

结论：PSql 处理小数据的效率约为 OpenGauss 的 2.5 倍，处理大数据的效率约为 OpenGauss 的 3.5 倍，随着数据量的增大，Psql 与 OpenGauss 的差距逐渐增大。

(6) multi index select(partial match)（重复实验三次）：

Runtime(ms)	table	s_table
OpenGauss	119.341	5.125
PSql	106.769	2.649

结论：PSql 处理小数据的效率约为 OpenGauss 的 2 倍，处理大数据的效率约为 OpenGauss 的 1.1 倍，随着数据量的增大，Psql 与 OpenGauss 的差距逐渐减小。

index 结论：PSql 处理小数据的效率普遍为 OpenGauss 的 1.5~2.6 倍，处理大数据的效率普遍为 OpenGauss 的 1.1~3 倍，随着数据量的增大，Psql 与 OpenGauss 的差距普遍逐渐增大，但在“create index”任务中显著地增大，在“index select”，“multi index select(partial match)”任务中逐渐减小，猜测 OpenGauss 在建立 index 时进行了更多的处理，以使大数据下的 select 和 multi index 部分匹配的 select 更加快速； 两数据库在“index select (upper_case function)”任务中的表现均远劣于其它同类型任务，认为 OpenGauss 与 PSql 一样，没有解决函数破坏索引顺序的问题。

四、 OpenGauss 特性综述与使用建议

OpenGauss 在小数据与大数据上，执行各类型任务的性能相比于 PostgreSql-14.11 均表现出劣势，且往往呈现出性能劣势随数据量的增大而更加明显的特点，在直接进行“全表 select 全部内容”，“全表排序 select”，“where like update”，“table1&table2 inner join on select”，“create index”，“create multi index”这些任务上劣势相当显著地增大，说明 OpenGauss 不适合在大数据下处理这些任务，使用 OpenGauss 且在高数据强度环境下时，调用它们需要更加慎重；但在“index select”，“multi index select(partial match)”这些任务上劣势显著地缩小，说明 OpenGauss 擅长在大数据下处理这两个任务，遇到数据规模巨大（应显著大于 10^6），需要多次调用此类任务的环境时，可以考虑采用 OpenGauss。