GaussDB

Performance White Paper

Issue	01
Date	2024-10-12

HUAWEI

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Trademarks and Permissions


and other Huawei trademarks are the property of Huawei Technologies Co., Ltd.
All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei Cloud and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Huawei Cloud Computing Technologies Co., Ltd.

Address: Huawei Cloud Data Center Jiaoxinggong Road
Qianzhong Avenue
Gui'an New District
Gui Zhou 550029
People's Republic of China

Website: https://www.huaweicloud.com/intl/en-us/

Contents

1 Test Method

This section describes BenchmarkSQL performance tests and test reports for GaussDB.

BenchmarkSQL is a JDBC benchmark test tool and runs a TPC-C. It can be used for many databases, such as PostgreSQL, Oracle, and MySQL.

TPC-C is a specification for online transaction processing (OLTP) systems. Generally, such systems are called service processing systems. Almost all mainstream international vendors that provide software and hardware platforms in the OLTP market have released TPC-C test results. With the development of computer technologies, these test results are continuously updated.

Test Environments

- JDK: JDK 1.8 or later is recommended.
- Apache Ant: apache-ant-1.10 or later is recommended.
- BenchmarkSQL: BenchmarkSQL 5.0 is recommended.
- ECS client: 32 vCPUs | 64 GB or larger specifications are recommended.
- Database: Prepare a GaussDB instance before the test and create a **$db_name** database and **$db_user** user for load testing.

Installing BenchmarkSQL

1. Download the installation package.
   ```
   wget https://sourceforge.net/projects/benchmarksql/files/latest/download
   ```
2. Decompress the installation package.
   ```
   unzip download -d /home
   ```
3. Go to the directory created after decompressing the BenchmarkSQL installation package, and use Apache Ant to compile BenchmarkSQL.
   ```
   cd /home/benchmarksql-5.0
   ant
   ```
   If the following information is displayed, the compilation is successful.


```
[root@test benchmarksql-5.0]# ant
Buildfile: /tools/benchmarksql-5.0/build.xml

init:
    [mkdir] Created dir: /tools/benchmarksql-5.0/build

compile:
    [javac] Compiling 11 source files to /tools/benchmarksql-5.0/build

dist:
    [mkdir] Created dir: /tools/benchmarksql-5.0/dist
      [jar] Building jar: /tools/benchmarksql-5.0/dist/BenchmarkSQL-5.0.jar

BUILD SUCCESSFUL
Total time: 1 second
```

Adapting GaussDB to BenchmarkSQL

1. Replace the driver package.

   Obtain the **driver package** based on the DB engine version, instance type, and OS of your instance. In this example, the driver package for distributed instances of version V2.0-8.*x* running on EulerOS 2.5 is used.

   ```
   cd /home
   wget https://dbs-download.obs.cn-north-1.myhuaweicloud.com/GaussDB/1716897684140/GaussDB_driver.zip
   unzip GaussDB_driver.zip
   cd GaussDB_driver/Distributed/Euler2.5_X86_64
   tar -zxvf GaussDB-Kernel_505.1.0_Euler_64bit_Jdbc.tar.gz
   rm -rf /home/benchmarksql-5.0/lib/postgres/postgresql-9.3-1102.jdbc41.jar
   cp -rp gsjdbc4.jar /home/benchmarksql-5.0/lib/postgres/
   ```

2. Modify the configuration file.

   Back up and rewrite the **/home/benchmarksql-5.0/run/props.pg** file.

   ```
   db=postgres
   driver=org.postgresql.Driver
   // Centralized instances
   // conn=jdbc:postgresql://$host_ip:$host_port/$db_name?targetServerType=master&loggerLevel=OFF
   // Distributed instances
   conn=jdbc:postgresql://$host_ip1:$host_port1,host_ip2:$host_port2,host_ip3:$host_port3/$db_name?autoBalance=true&loggerLevel=OFF
   user=$db_user
   password=$db_user_passwd
   warehouses=10
   loadWorkers=10
   terminals=5
   // To run specified transactions per terminal- runMins must equal zero
   runTxnsPerTerminal=0
   // To run for specified minutes- runTxnsPerTerminal must equal zero
   runMins=3
   // Number of total transactions per minute
   limitTxnsPerMin=0
   // Set to true to run in 4.x compatible mode. Set to false to use the
   // entire configured database evenly.
   terminalWarehouseFixed=true
   // The following five values must add up to 100.
   // The default percentages of 45, 43, 4, 4 & 4 match the TPC-C spec.
   newOrderWeight=45
   paymentWeight=43
   orderStatusWeight=4
   deliveryWeight=4
   stockLevelWeight=4
   // Directory name to create for collecting detailed result data.
   // Comment this out to suppress.
   resultDirectory=my_result_%tY-%tm-%td_%tH%tM%tS
   // osCollectorScript=./misc/os_collector_linux.py
   // osCollectorInterval=1
   // osCollectorSSHAddr=user@dbhost
   // osCollectorDevices=net_eth0 blk_sda
   ```


> **NOTE**
>
> Key parameters in **prop.pg**:
>
> - **conn**: JDBC connection string. Use either of the following connection strings based on the type of your instance:
>   - For centralized instances: **conn=jdbc:postgresql://$host_ip:$host_port/$db_name?targetServerType=master&loggerLevel=OFF**
>   - For distributed instances: **conn=jdbc:postgresql://$host_ip1:$host_port1,host_ip2:$host_port2,host_ip3:$host_port3/$db_name?autoBalance=true&loggerLevel=OFF**
> - **warehouses/loadWorkers**: the amount of data to be imported. Set this parameter as required.
> - **terminals/runMins**: the number of concurrent connections for load testing and the testing duration. Set this parameter as required.
>
>   The value of **terminals** must meet the following requirement: 0 < **terminals** ≤ 10 x **warehouses**. Otherwise, there will be an error.
> - Comment out **osCollector**-related parameters if necessary.

3. Modify the SQL statements in BenchmarkSQL 5.0.
   a. Create the **/home/benchmarksql-5.0/run/sql.postgres/tableCreates.sql** file.
      - Statement for creating a table in a centralized instance:

```
create table bmsql_config (
        cfg_name    varchar(30),
        cfg_value   varchar(50)
) ;

create table bmsql_warehouse (
        w_id        integer   not null,
        w_ytd       decimal(12,2),
        w_tax       decimal(4,4),
        w_name      varchar(10),
        w_street_1  varchar(20),
        w_street_2  varchar(20),
        w_city      varchar(20),
        w_state     char(2),
        w_zip       char(9)
)WITH (FILLFACTOR=80);

create table bmsql_district (
  d_w_id       integer       not null,
  d_id         integer       not null,
  d_ytd        decimal(12,2),
  d_tax        decimal(4,4),
  d_next_o_id  integer,
  d_name       varchar(10),
  d_street_1   varchar(20),
  d_street_2   varchar(20),
  d_city       varchar(20),
  d_state      char(2),
  d_zip        char(9)
 )WITH (FILLFACTOR=80);

create table bmsql_customer (
  c_w_id         integer        not null,
  c_d_id         integer        not null,
  c_id           integer        not null,
  c_discount     decimal(4,4),
  c_credit       char(2),
  c_last         varchar(16),
  c_first        varchar(16),
  c_credit_lim   decimal(12,2),
  c_balance      decimal(12,2),
```



```
  c_ytd_payment  decimal(12,2),
  c_payment_cnt  integer,
  c_delivery_cnt integer,
  c_street_1     varchar(20),
  c_street_2     varchar(20),
  c_city         varchar(20),
  c_state        char(2),
  c_zip          char(9),
  c_phone        char(16),
  c_since        timestamp,
  c_middle       char(2),
  c_data         varchar(500)
)WITH (FILLFACTOR=80) ;

create sequence bmsql_hist_id_seq cache 1000;

create table bmsql_history (
  hist_id  integer,
  h_c_id   integer,
  h_c_d_id integer,
  h_c_w_id integer,
  h_d_id   integer,
  h_w_id   integer,
  h_date   timestamp,
  h_amount decimal(6,2),
  h_data   varchar(24)
)WITH (FILLFACTOR=80);

create table bmsql_new_order (
  no_w_id  integer   not null,
  no_d_id  integer   not null,
  no_o_id  integer   not null
)WITH (FILLFACTOR=80) ;

create table bmsql_oorder (
  o_w_id       integer      not null,
  o_d_id       integer      not null,
  o_id         integer      not null,
  o_c_id       integer,
  o_carrier_id integer,
  o_ol_cnt     integer,
  o_all_local  integer,
  o_entry_d    timestamp
)WITH (FILLFACTOR=80) ;

create table bmsql_order_line (
  ol_w_id         integer   not null,
  ol_d_id         integer   not null,
  ol_o_id         integer   not null,
  ol_number       integer   not null,
  ol_i_id         integer   not null,
  ol_delivery_d   timestamp,
  ol_amount       decimal(6,2),
  ol_supply_w_id  integer,
  ol_quantity     integer,
  ol_dist_info    char(24)
)WITH (FILLFACTOR=80) ;

create table bmsql_item (
  i_id     integer      not null,
  i_name   varchar(24),
  i_price  decimal(5,2),
  i_data   varchar(50),
  i_im_id  integer
);

create table bmsql_stock (
  s_w_id       integer       not null,
  s_i_id       integer       not null,
```



```
 s_quantity   integer,
 s_ytd        integer,
 s_order_cnt  integer,
 s_remote_cnt integer,
 s_data       varchar(50),
 s_dist_01    char(24),
 s_dist_02    char(24),
 s_dist_03    char(24),
 s_dist_04    char(24),
 s_dist_05    char(24),
 s_dist_06    char(24),
 s_dist_07    char(24),
 s_dist_08    char(24),
 s_dist_09    char(24),
 s_dist_10    char(24)
)WITH (FILLFACTOR=80) ;
```

- Statement for creating a table in a distributed instance:

```
create table bmsql_config (
        cfg_name    varchar(30),
        cfg_value   varchar(50)
) DISTRIBUTE BY REPLICATION;

create table bmsql_warehouse (
        w_id        integer   not null,
        w_ytd       decimal(12,2),
        w_tax       decimal(4,4),
        w_name      varchar(10),
        w_street_1  varchar(20),
        w_street_2  varchar(20),
        w_city      varchar(20),
        w_state     char(2),
        w_zip       char(9)
)WITH (FILLFACTOR=80) DISTRIBUTE BY hash(w_id);

create table bmsql_district (
 d_w_id       integer       not null,
 d_id         integer       not null,
 d_ytd        decimal(12,2),
 d_tax        decimal(4,4),
 d_next_o_id  integer,
 d_name       varchar(10),
 d_street_1   varchar(20),
 d_street_2   varchar(20),
 d_city       varchar(20),
 d_state      char(2),
 d_zip        char(9)
 )WITH (FILLFACTOR=80) DISTRIBUTE BY hash(d_w_id);

create table bmsql_customer (
 c_w_id         integer        not null,
 c_d_id         integer        not null,
 c_id           integer        not null,
 c_discount     decimal(4,4),
 c_credit       char(2),
 c_last         varchar(16),
 c_first        varchar(16),
 c_credit_lim   decimal(12,2),
 c_balance      decimal(12,2),
 c_ytd_payment  decimal(12,2),
 c_payment_cnt  integer,
 c_delivery_cnt integer,
 c_street_1     varchar(20),
 c_street_2     varchar(20),
 c_city         varchar(20),
 c_state        char(2),
 c_zip          char(9),
 c_phone        char(16),
 c_since        timestamp,
 c_middle       char(2),
```



```
  c_data         varchar(500)
)WITH (FILLFACTOR=80) DISTRIBUTE BY hash(c_w_id);

create sequence bmsql_hist_id_seq cache 1000;

create table bmsql_history (
  hist_id  integer,
  h_c_id   integer,
  h_c_d_id integer,
  h_c_w_id integer,
  h_d_id   integer,
  h_w_id   integer,
  h_date   timestamp,
  h_amount decimal(6,2),
  h_data   varchar(24)
)WITH (FILLFACTOR=80) DISTRIBUTE BY hash(h_w_id);

create table bmsql_new_order (
  no_w_id  integer   not null,
  no_d_id  integer   not null,
  no_o_id  integer   not null
)WITH (FILLFACTOR=80) DISTRIBUTE BY hash(no_w_id);

create table bmsql_oorder (
  o_w_id       integer      not null,
  o_d_id       integer      not null,
  o_id         integer      not null,
  o_c_id       integer,
  o_carrier_id integer,
  o_ol_cnt     integer,
  o_all_local  integer,
  o_entry_d    timestamp
)WITH (FILLFACTOR=80) DISTRIBUTE BY hash(o_w_id);

create table bmsql_order_line (
  ol_w_id         integer   not null,
  ol_d_id         integer   not null,
  ol_o_id         integer   not null,
  ol_number       integer   not null,
  ol_i_id         integer   not null,
  ol_delivery_d   timestamp,
  ol_amount       decimal(6,2),
  ol_supply_w_id  integer,
  ol_quantity     integer,
  ol_dist_info    char(24)
)WITH (FILLFACTOR=80) DISTRIBUTE BY hash(ol_w_id);

create table bmsql_item (
  i_id     integer      not null,
  i_name   varchar(24),
  i_price  decimal(5,2),
  i_data   varchar(50),
  i_im_id  integer
) DISTRIBUTE BY REPLICATION;

create table bmsql_stock (
  s_w_id       integer       not null,
  s_i_id       integer       not null,
  s_quantity   integer,
  s_ytd        integer,
  s_order_cnt  integer,
  s_remote_cnt integer,
  s_data       varchar(50),
  s_dist_01    char(24),
  s_dist_02    char(24),
  s_dist_03    char(24),
  s_dist_04    char(24),
  s_dist_05    char(24),
  s_dist_06    char(24),
```



```
  s_dist_07    char(24),
  s_dist_08    char(24),
  s_dist_09    char(24),
  s_dist_10    char(24)
)WITH (FILLFACTOR=80) DISTRIBUTE BY hash(s_w_id);
```

b. Create the **/home/benchmarksql-5.0/run/sql.postgres/tableDrops.sql** file.

```
drop table bmsql_config;
drop table bmsql_new_order;
drop table bmsql_order_line;
drop table bmsql_oorder;
drop table bmsql_history;
drop table bmsql_customer;
drop table bmsql_stock;
drop table bmsql_item;
drop table bmsql_district;
drop table bmsql_warehouse;
drop sequence bmsql_hist_id_seq;
```

c. Create the **/home/benchmarksql-5.0/run/sql.postgres/indexCreates.sql** file.

```
set maintenance_work_mem='4GB';
alter table  bmsql_warehouse add constraint bmsql_warehouse_pkey
  primary key (w_id);

alter table  bmsql_district add constraint bmsql_district_pkey
  primary key (d_w_id, d_id);

alter table  bmsql_customer add constraint bmsql_customer_pkey
  primary key (c_w_id, c_d_id, c_id);

alter table  bmsql_oorder add constraint bmsql_oorder_pkey
  primary key (o_w_id, o_d_id, o_id);

alter table  bmsql_new_order add constraint bmsql_new_order_pkey
  primary key (no_w_id, no_d_id, no_o_id);

alter table  bmsql_order_line add constraint bmsql_order_line_pkey
  primary key (ol_w_id, ol_d_id, ol_o_id, ol_number);

alter table  bmsql_stock add constraint bmsql_stock_pkey
  primary key (s_w_id, s_i_id);

alter table  bmsql_item add constraint bmsql_item_pkey
  primary key (i_id);

create index bmsql_oorder_idx1 on  bmsql_oorder(o_w_id, o_d_id, o_c_id, o_id);

create index bmsql_customer_idx1
  on   bmsql_customer (c_w_id, c_d_id, c_last, c_first);
```

d. Create the **/home/benchmarksql-5.0/run/sql.postgres/indexDrops.sql** file.

```
set statement_timeout=0;
set maintenance_work_mem='4GB';
alter table  bmsql_warehouse drop constraint bmsql_warehouse_pkey;
alter table  bmsql_district drop constraint bmsql_district_pkey;
alter table  bmsql_customer drop constraint bmsql_customer_pkey;
drop index bmsql_customer_idx1;
alter table  bmsql_oorder drop constraint bmsql_oorder_pkey;
drop index bmsql_oorder_idx1;
drop index bmsql_oorder_idx2;
alter table  bmsql_new_order drop constraint bmsql_new_order_pkey;
alter table  bmsql_order_line drop constraint bmsql_order_line_pkey;
alter table  bmsql_stock drop constraint bmsql_stock_pkey;
alter table  bmsql_item drop constraint bmsql_item_pkey;
alter table bmsql_history drop constraint bmsql_history_pkey;
```

4. Update **./runDatabaseBuild.sh**.

Run the following command to open the **runDatabaseBuild.sh** file:

```
vim /home/benchmarksql-5.0/run/runDatabaseBuild.sh
```

Move the cursor to the first character of **foreignKeys** in **AFTER_LOAD="indexCreates foreignKeys extraHistID buildFinish"**, press **x** to delete **foreignKeys** and the space following it, and run **:wq** to save the modification and exit. The string after the deletion is as follows:

```
AFTER_LOAD="indexCreates extraHistID buildFinish"
```

Running BenchmarkSQL

1. Import data.
   ```
   ./runDatabaseBuild.sh props.pg
   ```
2. Perform load testing.
   ```
   ./runBenchmark.sh props.pg
   ```

   CAUTION

   You can tune the involved parameters as required.

3. Delete data.
   ```
   ./runDatabaseDestroy.sh props.pg
   ```


2 Test Data

About IOPS

The input/output operations per second (IOPS) supported by GaussDB depends on the I/O performance of Elastic Volume Service (EVS) disks. For details, see **Disk Types and Performance** in the *EVS Service Overview*.

Test Data

1. Instance type: distributed and centralized
2. Instance specifications: 16 vCPUs | 128 GB and 32 vCPUs | 256 GB
3. Cluster scale: 3 CNs, 3 shards, and 3 replicas for a distributed instance; 1 primary node and 2 standby nodes for a centralized instance
4. Data volume: 1,000 warehouses
5. Stress test duration: 30 minutes (5 minutes for warm-up)

Table 2-1 Performance data

Instance Type	Deployment Model	Instance Specifications	Concurrent Requests	tpmC
Centralized	HA (1 primary + 2 standby)	16 vCPUs | 128 GB	256	70,057
		32 vCPUs | 256 GB	384	132,196
Distributed	Independent	16 vCPUs | 128 GB	1,024	466,930
		32 vCPUs | 256 GB	2,048	658,805

Test Metrics

According to the Transaction Processing Performance Council (TPC), the throughput metric, tpmC, describes the number of new transactions processed per

minute while the system is performing four types of transactions: payment operations, order status queries, deliveries, and stock status queries. The response time of all transactions and the proportion of each type of transactions must meet the requirements of the TPC-C test specifications. In this case, the larger the tpmC value is, the higher the online transaction processing (OLTP) capability of the system is.

