GaussDB

Performance White Paper

Issue	01
Date	2024-07-03

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

- Site: Huawei Cloud.
- Instance type: distributed and primary/standby
- Specifications: 16 vCPUs | 128 GB and 32 vCPUs | 256 GB
- Cluster scale: 3 CNs, 3 shards, and 3 replicas for a distributed instance; 1 primary node and 2 standby nodes for a primary/standby instance

Test Method

1. Change the connection configuration.

   The configuration file is stored in **./run/props.pg**.

2. Change key parameters.

```
//Connection configuration
conn=jdbc:postgresql://127.0.0.1:8000/postgres?autoBalance=true
//Username
user=****
//Password
password=****
//Number of warehouses
warehouses=1000
//Number of concurrent loadWorkers that are used to import data
loadWorkers=10
//Number of concurrent terminals
terminals=2048
//Running duration
runMins=30
```

3. Import data.

cd ~/BenchmarkSQL-5.0/run

./runDatabaseBuild.sh props.pg

4. Run the TPC-C test.

   cd ~/BenchmarkSQL-5.0/run

   ./runBenchmark.sh props.pg

Statements for Creating Tables

```
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);
```


Test Metrics

According to the Transaction Processing Performance Council (TPC), tpmC describes the number of new orders 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) of the system is.

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 primary/standby
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 primary/standby 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	IOPS
Primary/ Standby	HA (1 primary + 2 standby)	16 vCPUs | 128 GB	256	70,057	For details, see **About IOPS**.
		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	

