ALIBABA CLOUD

Alibaba Cloud

ApsaraDB for PolarDB Performance White Paper

Document Version: 20200826

Alibaba Cloud

Legal disclaimer

Alibaba Cloud reminds you to carefully read and fully understand the terms and conditions of this legal disclaimer before you read or use this document. If you have read or used this document, it shall be deemed as your total acceptance of this legal disclaimer.

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.

2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company or individual in any form or by any means without the prior written consent of Alibaba Cloud.

3. The content of this document may be changed because of product version upgrade, adjustment, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and an updated version of this document will be released through Alibaba Cloud-authorized channels from time to time. You should pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.

4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides this document based on the "status quo", "being defective", and "existing functions" of its products and services. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity, applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not take legal responsibility for any errors or lost profits incurred by any organization, company, or individual arising from download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, take responsibility for any indirect, consequential, punitive, contingent, special, or punitive damages, including lost profits arising from the use or trust in this document (even if Alibaba Cloud has been notified of the possibility of such a loss).

5. By law, all the contents in Alibaba Cloud documents, including but not limited to pictures, architecture design, page layout, and text description, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of this document shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates.

6. Please directly contact Alibaba Cloud for any errors of this document.

Document conventions

Style	Description	Example
Danger	A danger notice indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	Danger: Resetting will result in the loss of user configuration data.
Warning	A warning notice indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	Warning: Restarting will cause business interruption. About 10 minutes are required to restart an instance.
Notice	A caution notice indicates warning information, supplementary instructions, and other content that the user must understand.	Notice: If the weight is set to 0, the server no longer receives new requests.
Note	A note indicates supplemental instructions, best practices, tips, and other content.	Note: You can use Ctrl + A to select all files.
>	Closing angle brackets are used to indicate a multi-level menu cascade.	Click **Settings**> **Network**> **Set network type**.
Bold	Bold formatting is used for buttons , menus, page names, and other UI elements.	Click **OK**.
`Courier font`	Courier font is used for commands	Run the `cd /d C:/window` command to enter the Windows system folder.
Italic	Italic formatting is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	This format is used for an optional value, where only one item can be selected.	`ipconfig [-all|-t]`
{} or {a|b}	This format is used for a required value, where only one item can be selected.	`switch {active|stand}`

Table of Contents

1.OLTP performance tests

This topic describes how to test the online transaction processing (OLTP) performance of an ApsaraDB PolarDB MySQL cluster by using SysBench.

Test tool

SysBench is a modular, cross-platform, and multi-threaded benchmark tool used for evaluating the core operating parameters of a system that runs a heavily loaded database. SysBench is designed to test the performance of a database without the need to set complicated benchmark settings or install the database engine.

Test environment

- An Elastic Compute Service (ECS) instance and ApsaraDB PolarDB MySQL clusters are used for testing, and they must be deployed in the same region and zone. In this test, they are both deployed in Zone I of the China (Hangzhou) region.
- The VPC network type is selected for both the ECS instance and the ApsaraDB PolarDB MySQL clusters.

  > **Note** Make sure that the ECS instance and the ApsaraDB PolarDB MySQL clusters are connected to the same VPC network.

- The ApsaraDB PolarDB MySQL clusters for testing are as follows:
  - The node specification is polar.mysql.x4.large (4-core 16 GB).
  - The read-only, write-only, or read-write performance is tested by using a two-node cluster, which contains one primary node and one read-only node. The performance of multi-node clusters is tested by using two- to nine-node clusters, which contain one primary node and one to eight read-only nodes.
  - Use cluster endpoints to connect to the clusters. For information about how to query the cluster endpoints, see View or apply for an endpoint.
- The ECS instance for testing is as follows:
  - The instance type is ecs.c5.4xlarge.
  - The image used by the instance is 64-bit CentOS 7.0.

Test scenario

Test the performance of ApsaraDB PolarDB MySQL clusters with different specifications in the following aspects: read-only performance, write-only performance, read-write performance, and read-only performance in the case of multiple read-only nodes.

The following metrics are used to measure the performance:

- Transactions per second (TPS): the number of transactions executed per second in the ApsaraDB PolarDB MySQL cluster. Only the number of committed transactions is counted.
- Queries per second (QPS): the number of SQL statements executed per second in the ApsaraDB PolarDB MySQL cluster, including INSERT, SELECT, UPDATE, and DELETE statements.

Install SysBench

1. Run the following commands on the ECS instance to install SysBench:

```
yum install gcc gcc-c++ autoconf automake make libtool bzr mysql-devel git mysql

git clone https://github.com/akopytov/sysbench.git
## Download SysBench from GitHub

cd sysbench
## Open the SysBench directory

git checkout 1.0.18
## Switch to the SysBench version 1.0.18

./autogen.sh
## Run autogen.sh

./configure --prefix=/usr --mandir=/usr/share/man

make
## Compile SysBench.

make install
```

2. Run the following commands to configure the SysBench client, so that it can use all available CPU cores to process data with the minimum cross-core context switches. By default, two CPU cores are used.

```
sudo sh -c 'for x in /sys/class/net/eth0/queues/rx-*; do echo ffffffff>$x/rps_cpus; done'
```

> **Note** ffffffff indicates that 32 cores are used for data processing. Modify the command as needed. For example, enter ff if your ECS instance has eight cores.

```
sudo sh -c "echo 32768 > /proc/sys/net/core/rps_sock_flow_entries"
sudo sh -c "echo 4096 > /sys/class/net/eth0/queues/rx-0/rps_flow_cnt"
sudo sh -c "echo 4096 > /sys/class/net/eth0/queues/rx-1/rps_flow_cnt"
```

Test procedure

1. Query the cluster endpoint and port of the ApsaraDB PolarDB MySQL cluster. For more information, see View or apply for an endpoint.
2. Run the following command on your ECS instance to create a database named testdb in the ApsaraDB PolarDB MySQL cluster:

```
mysql -h XXX -P XXX -u XXX -p XXX -e 'create database testdb'
```

> **Note** Replace the four XXXs in this command and the subsequent commands with the cluster endpoint, port number, username, and password of the ApsaraDB PolarDB MySQL cluster, respectively.

Parameter	Description
-h	The cluster endpoint of the ApsaraDB PolarDB MySQL cluster.
-P	The port number of the ApsaraDB PolarDB MySQL cluster.
-u	The username for connecting to the ApsaraDB PolarDB MySQL cluster.
-p	The password of the username.

3. Use SysBench to test the read-only performance of the two-node ApsaraDB PolarDB MySQL cluster. The whole process takes 10 minutes.

```
sysbench --db-driver=mysql --mysql-host=XXX --mysql-port=XXX --mysql-user=XXX --mysql-passw
ord=XXX --mysql-db=sbtest --table_size=25000 --tables=250 --events=0 --time=600 oltp_read_only
prepare
## Prepare test data

sysbench --db-driver=mysql --mysql-host=XXX --mysql-port=XXX --mysql-user=XXX --mysql-passw
ord=XXX --mysql-db=sbtest --table_size=25000 --tables=250 --events=0 --time=600 --threads=XXX
--percentile=95 --range_selects=0 --skip-trx=1 --report-interval=1 oltp_read_only run
## Run workloads

sysbench --db-driver=mysql --mysql-host=XXX --mysql-port=XXX --mysql-user=XXX --mysql-passw
ord=XXX --mysql-db=sbtest --table_size=25000 --tables=250 --events=0 --time=600 --threads=XXX
--percentile=95 --range_selects=0 oltp_read_only cleanup
## Clear temporary data
```

4. Use SysBench to test the write-only performance of the two-node ApsaraDB PolarDB MySQL cluster. The whole process takes 10 minutes.

```
sysbench --db-driver=mysql --mysql-host=XXX --mysql-port=XXX --mysql-user=XXX --mysql-passw
ord=XXX --mysql-db=sbtest --table_size=25000 --tables=250 --events=0 --time=600 oltp_write_onl
y prepare
## Prepare test data

sysbench --db-driver=mysql --mysql-host=XXX --mysql-port=XXX --mysql-user=XXX --mysql-passw
ord=XXX --mysql-db=sbtest --table_size=25000 --tables=250 --events=0 --time=600 --threads=XXX
--percentile=95 --report-interval=1 oltp_write_only run
## Run workloads

sysbench --db-driver=mysql --mysql-host=XXX --mysql-port=XXX --mysql-user=XXX --mysql-passw
ord=XXX --mysql-db=sbtest --table_size=25000 --tables=250 --events=0 --time=600 --threads=XXX
--percentile=95 oltp_write_only cleanup
## Clear temporary data
```

5. Use SysBench to test the read-write performance of the two-node ApsaraDB PolarDB MySQL cluster. The whole process takes 10 minutes.

```
sysbench --db-driver=mysql --mysql-host=XXX --mysql-port=XXX --mysql-user=XXX --mysql-passw
ord=XXX --mysql-db=sbtest --table_size=250000 --tables=25 --events=0 --time=600 oltp_read_writ
e prepare
## Prepare test data

sysbench --db-driver=mysql --mysql-host=XXX --mysql-port=XXX --mysql-user=XXX --mysql-passw
ord=XXX --mysql-db=sbtest --table_size=250000 --tables=25 --events=0 --time=600 --threads=XXX
--percentile=95 --report-interval=1 oltp_read_write run
## Run workloads

sysbench --db-driver=mysql --mysql-host=XXX --mysql-port=XXX --mysql-user=XXX --mysql-passw
ord=XXX --mysql-db=sbtest --table_size=250000 --tables=25 --events=0 --time=600 --threads=XXX
--percentile=95 oltp_read_write cleanup
## Clear temporary data
```

6. Use SysBench to test the read-only performance of 2-node, 3-node, 4-node, 5-node, 6-node, 7-node, 8-node, and 9-node ApsaraDB PolarDB MySQL clusters.

```
sysbench --db-driver=mysql --mysql-host=XXX --mysql-port=XXX --mysql-user=XXX --mysql-passw
ord=XXX --mysql-db=sbtest --table_size=250000 --tables=25 --events=0 --time=600 oltp_read_only
prepare
## Prepare test data

sysbench --db-driver=mysql --mysql-host=XXX --mysql-port=XXX --mysql-user=XXX --mysql-passw
ord=XXX --mysql-db=sbtest --table_size=250000 --tables=25 --events=0 --time=600 --threads=XXX
--percentile=95 --report-interval=1 oltp_read_only --db-ps-mode=disable --skip-trx=1 run
## Run workloads

sysbench --db-driver=mysql --mysql-host=XXX --mysql-port=XXX --mysql-user=XXX --mysql-passw
ord=XXX --mysql-db=sbtest --table_size=250000 --tables=25 --events=0 --time=600 --threads=XXX
--percentile=95 oltp_read_only cleanup
## Clear temporary data
```

What's next

- For information about the test results for ApsaraDB PolarDB MySQL 8.0, see Test results of ApsaraDB PolarDB for MySQL 8.0.
- For information about the test results for ApsaraDB PolarDB MySQL 5.6, see Test results of ApsaraDB PolarDB for MySQL 5.6.
- For information about the test result comparison between ApsaraDB PolarDB MySQL and ApsaraDB RDS MySQL, see Comparison with ApsaraDB RDS for MySQL.

2.Test results of ApsaraDB PolarDB for MySQL 8.0

This topic shows the results of online transaction processing (OLTP) performance tests on ApsaraDB PolarDB for MySQL 5.6.

The PolarDB MySQL 8.0 optimizer supports the followingmodes:

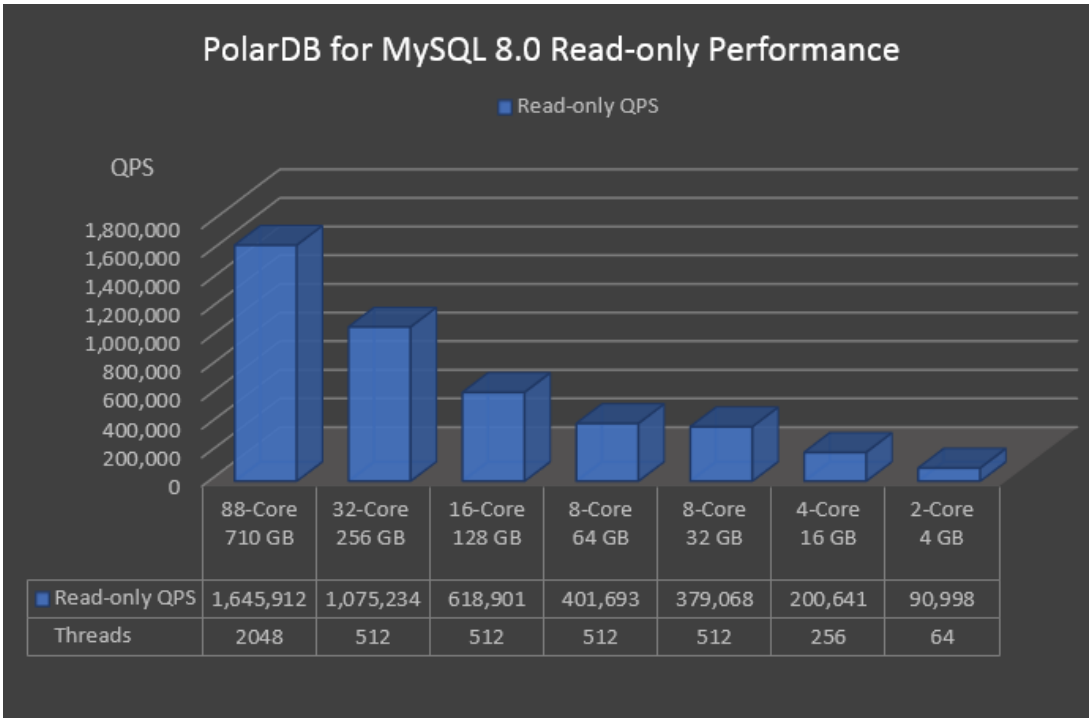
- Cost-Based Optimization (CBO): chooses the mostefficient way to optimize queries by creating a cost-based model base onstatistical information or samples collected in real time.
- Rule-Based Optimization (RBO): optimizes queries based on the preset rules.

The PolarDB for MySQL 8.0 optimizer uses CBO as the primary mode, and also supports RBO under some circumstances. You can run the ANALYZE command to collect the required statistical information, or allow the optimizer to automatically collect the information. The required information includes the table size, number of records, density vector, density histogram,Number of Distinct Value (NDV), and null ratio.

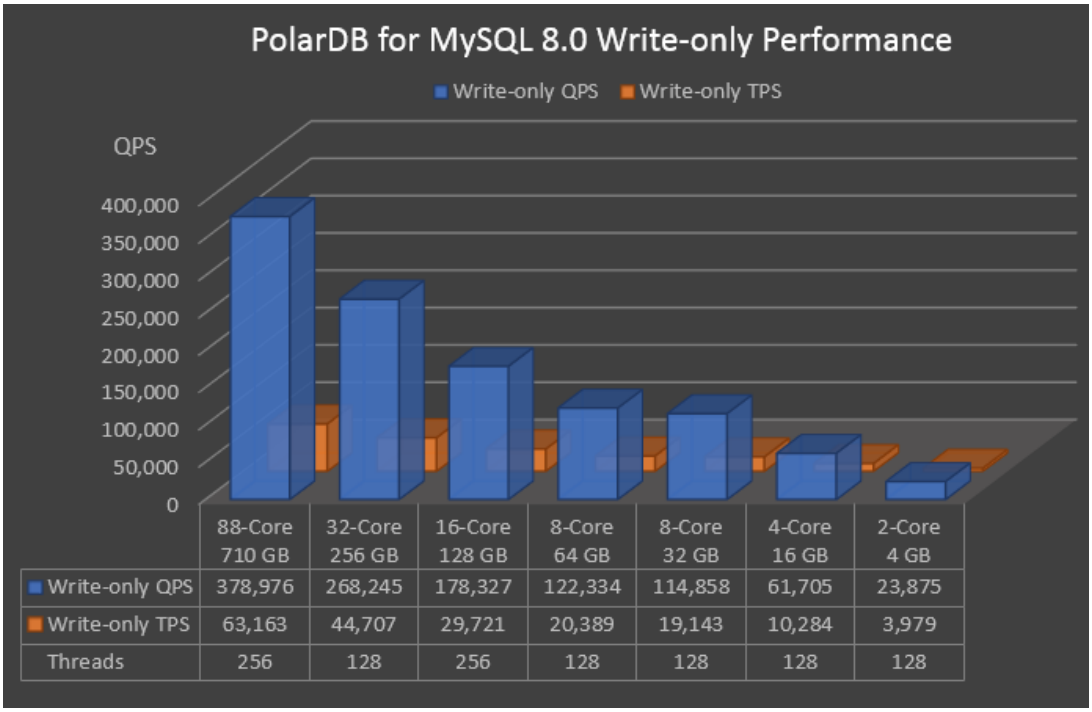
> **Note** For more information about how to test the OLTP performance of ApsaraDB PolarDB for MySQL, see OLTP performance tests.

Test results

Read-only performance of different specifications

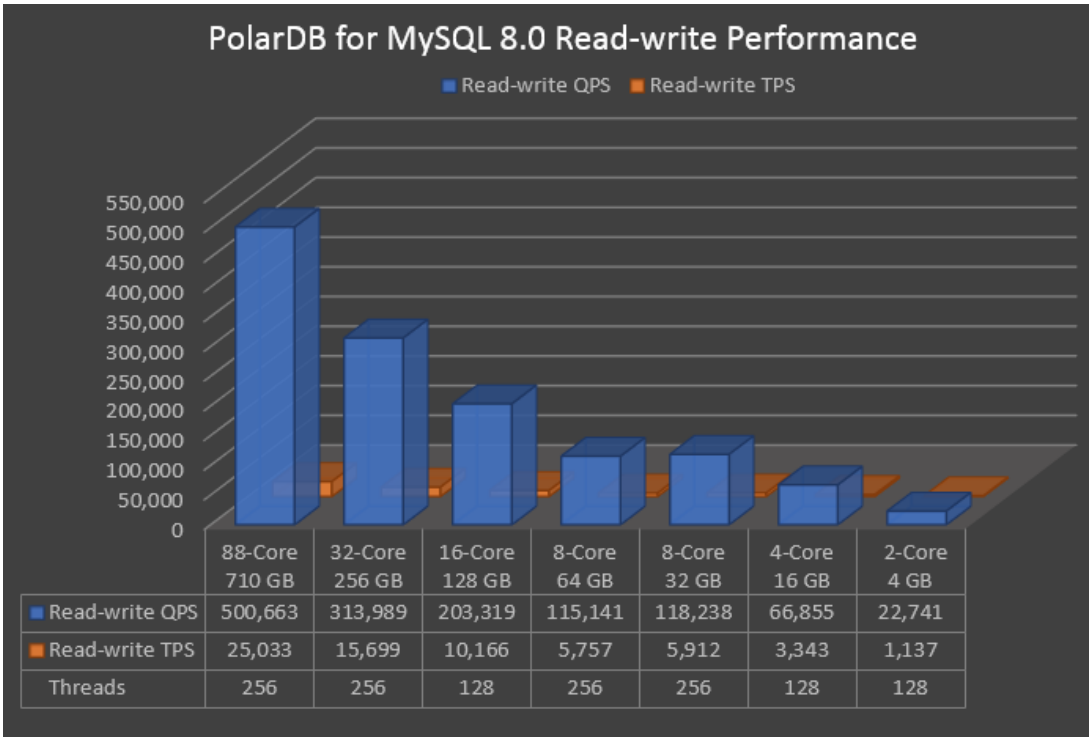


Write-only performance of different specifications



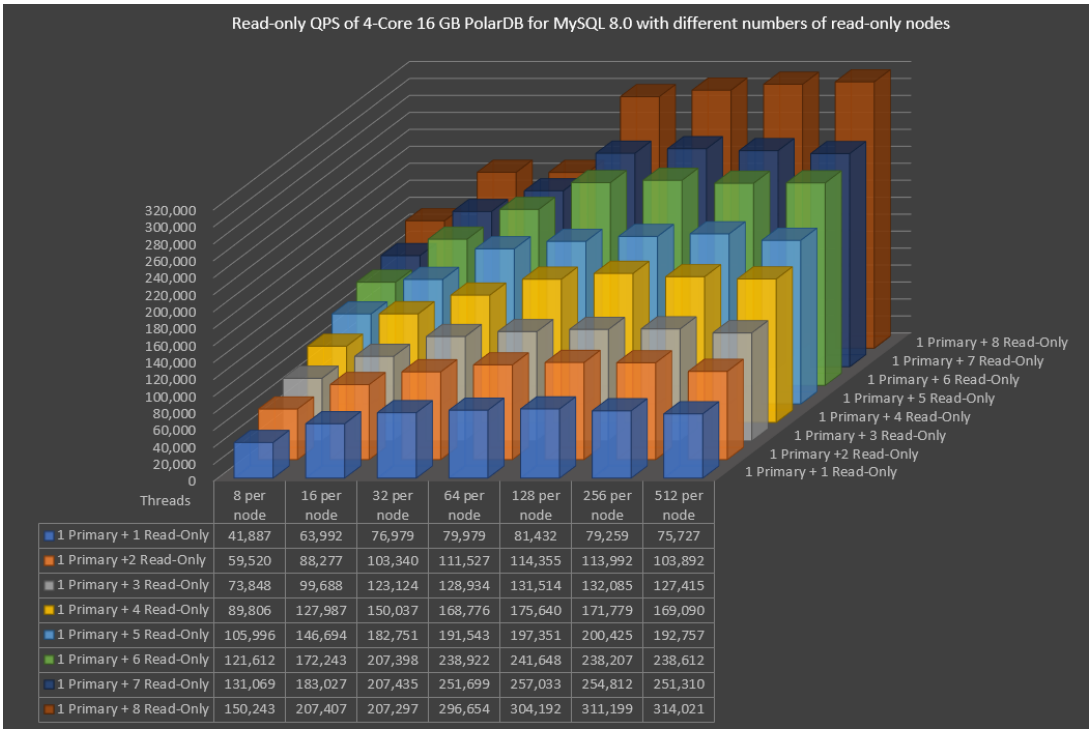
	88-Core 710 GB	32-Core 256 GB	16-Core 128 GB	8-Core 64 GB	8-Core 32 GB	4-Core 16 GB	2-Core 4 GB
Write-only QPS	378,976	268,245	178,327	122,334	114,858	61,705	23,875
Write-only TPS	63,163	44,707	29,721	20,389	19,143	10,284	3,979
Threads	256	128	256	128	128	128	128

Read-write performance of different specifications



	88-Core 710 GB	32-Core 256 GB	16-Core 128 GB	8-Core 64 GB	8-Core 32 GB	4-Core 16 GB	2-Core 4 GB
Read-write QPS	500,663	313,989	203,319	115,141	118,238	66,855	22,741
Read-write TPS	25,033	15,699	10,166	5,757	5,912	3,343	1,137
Threads	256	256	128	256	256	128	128

Use cluster endpoints that are addressed to different numbers of read-only nodes



Threads	8 per node	16 per node	32 per node	64 per node	128 per node	256 per node	512 per node
1 Primary + 1 Read-Only	41,887	63,992	76,979	79,979	81,432	79,259	75,727
1 Primary +2 Read-Only	59,520	88,277	103,340	111,527	114,355	113,992	103,892
1 Primary + 3 Read-Only	73,848	99,688	123,124	128,934	131,514	132,085	127,415
1 Primary + 4 Read-Only	89,806	127,987	150,037	168,776	175,640	171,779	169,090
1 Primary + 5 Read-Only	105,996	146,694	182,751	191,543	197,351	200,425	192,757
1 Primary + 6 Read-Only	121,612	172,243	207,398	238,922	241,648	238,207	238,612
1 Primary + 7 Read-Only	131,069	183,027	207,435	251,699	257,033	254,812	251,310
1 Primary + 8 Read-Only	150,243	207,407	207,297	296,654	304,192	311,199	314,021

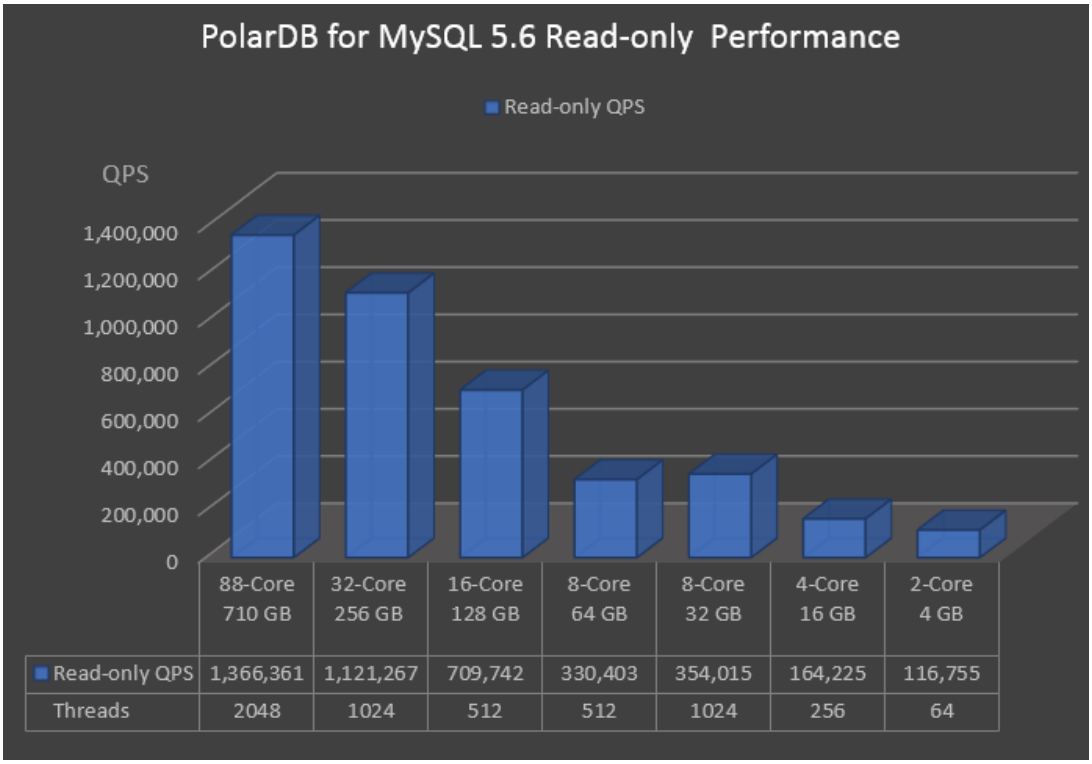
3.Test results of ApsaraDB PolarDB for MySQL 5.6

This topic shows the results of online transaction processing (OLTP) performance tests on ApsaraDB PolarDB for MySQL 5.6.

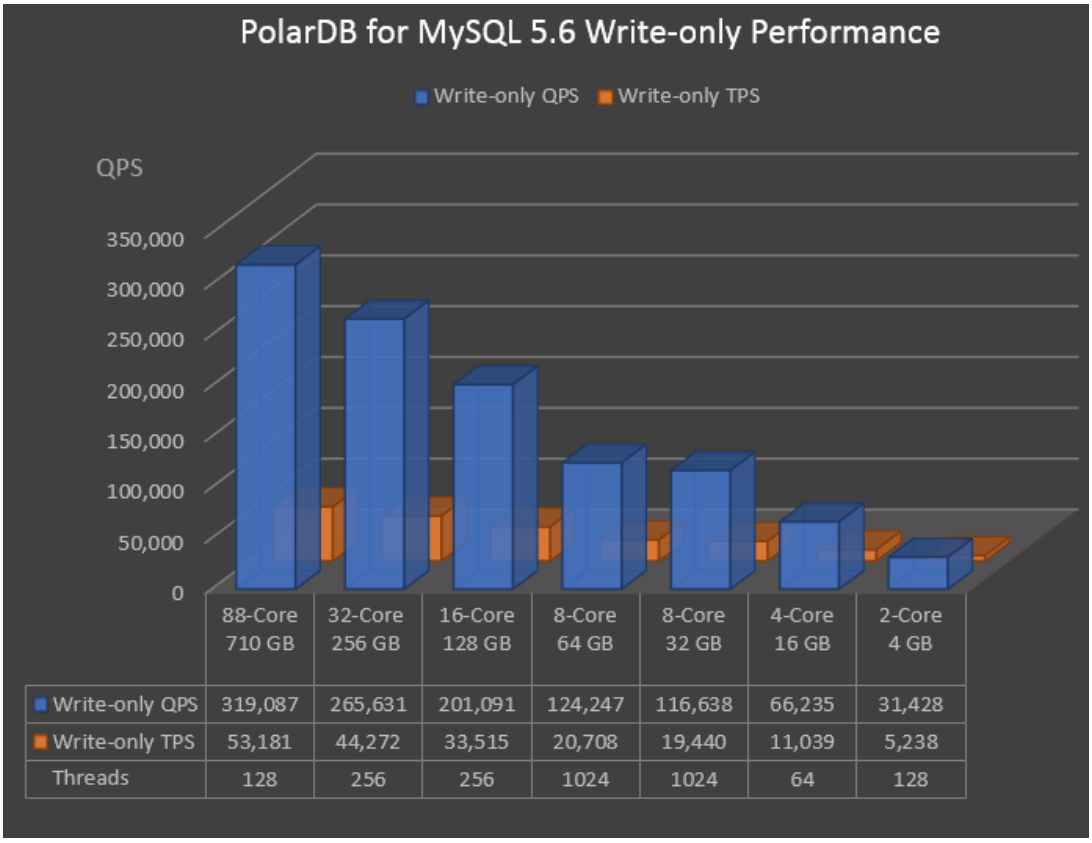
> **Note** For more information about how to test the OLTP performance of ApsaraDB PolarDB for MySQL, see OLTP performance tests.

Test results

Read-only performance of different specifications

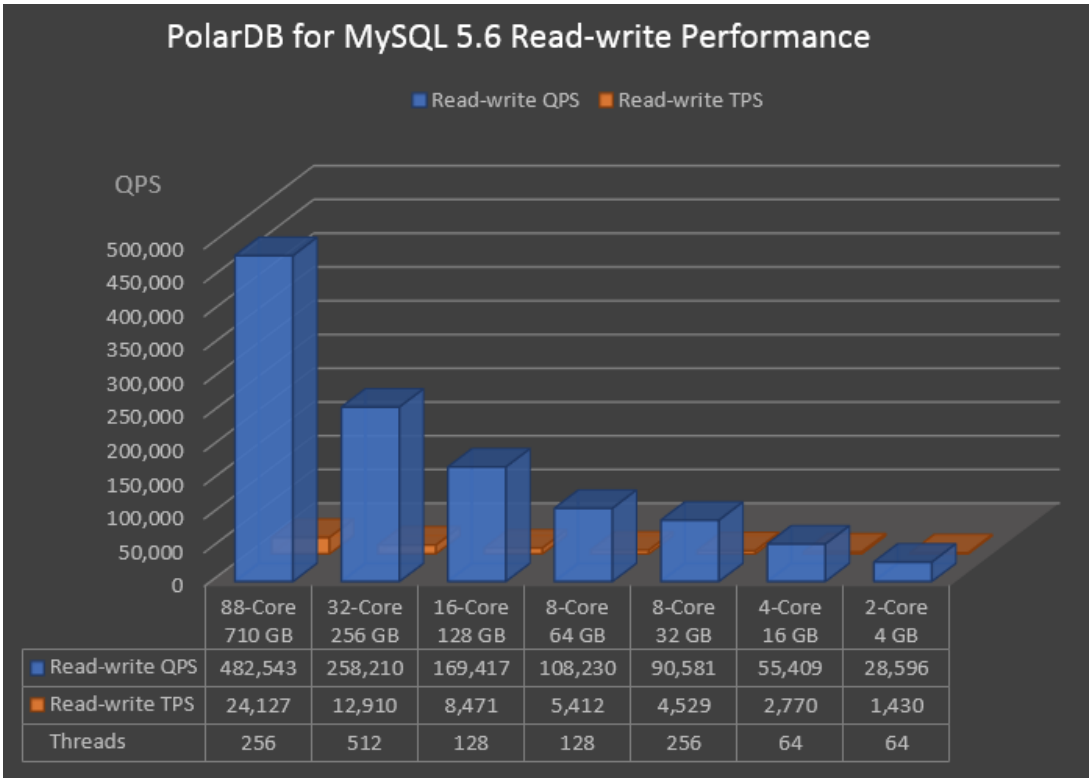


Write-only performance of different specifications



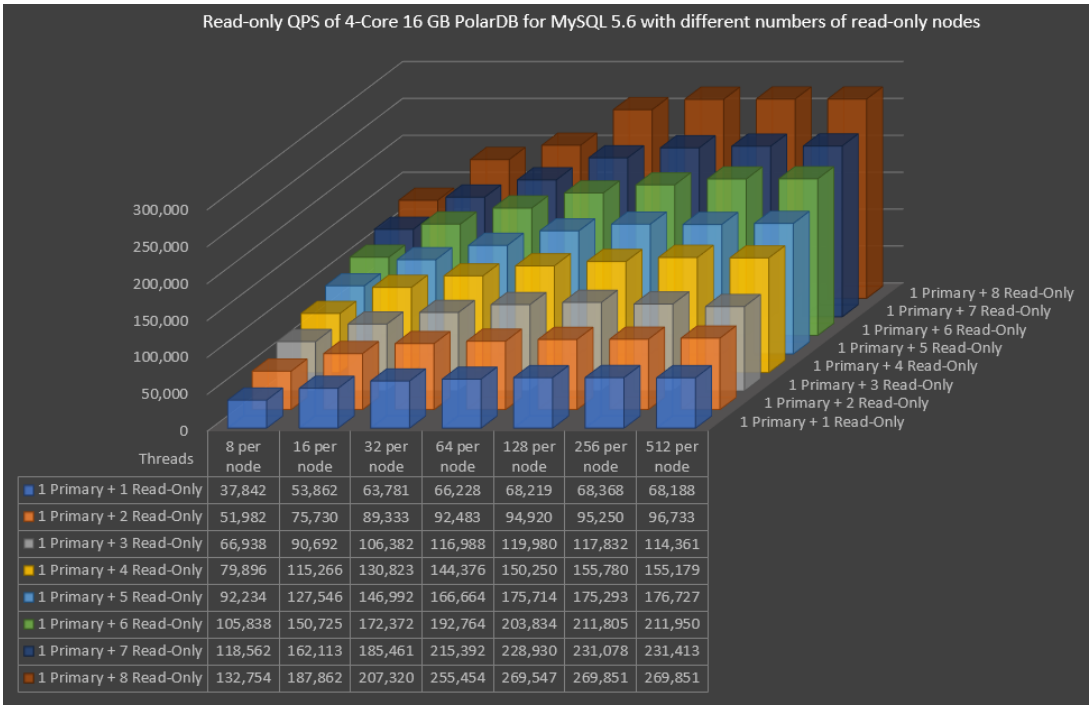
	88-Core 710 GB	32-Core 256 GB	16-Core 128 GB	8-Core 64 GB	8-Core 32 GB	4-Core 16 GB	2-Core 4 GB
Write-only QPS	319,087	265,631	201,091	124,247	116,638	66,235	31,428
Write-only TPS	53,181	44,272	33,515	20,708	19,440	11,039	5,238
Threads	128	256	256	1024	1024	64	128

Read-write performance of different specifications



	88-Core 710 GB	32-Core 256 GB	16-Core 128 GB	8-Core 64 GB	8-Core 32 GB	4-Core 16 GB	2-Core 4 GB
Read-write QPS	482,543	258,210	169,417	108,230	90,581	55,409	28,596
Read-write TPS	24,127	12,910	8,471	5,412	4,529	2,770	1,430
Threads	256	512	128	128	256	64	64

Use cluster endpoints that are addressed to different numbers of read-only nodes



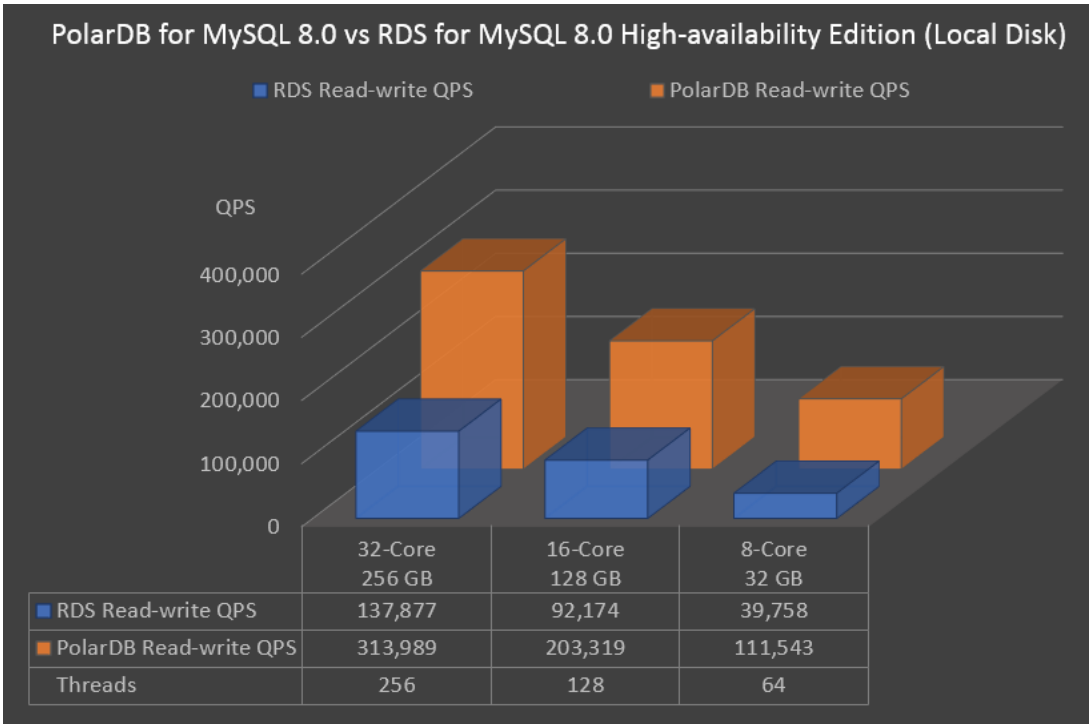
Threads	8 per node	16 per node	32 per node	64 per node	128 per node	256 per node	512 per node
1 Primary + 1 Read-Only	37,842	53,862	63,781	66,228	68,219	68,368	68,188
1 Primary + 2 Read-Only	51,982	75,730	89,333	92,483	94,920	95,250	96,733
1 Primary + 3 Read-Only	66,938	90,692	106,382	116,988	119,980	117,832	114,361
1 Primary + 4 Read-Only	79,896	115,266	130,823	144,376	150,250	155,780	155,179
1 Primary + 5 Read-Only	92,234	127,546	146,992	166,664	175,714	175,293	176,727
1 Primary + 6 Read-Only	105,838	150,725	172,372	192,764	203,834	211,805	211,950
1 Primary + 7 Read-Only	118,562	162,113	185,461	215,392	228,930	231,078	231,413
1 Primary + 8 Read-Only	132,754	187,862	207,320	255,454	269,547	269,851	269,851

4.Comparison with ApsaraDB RDS for MySQL

This topic compares the online transaction processing (OLTP) performance of ApsaraDB PolarDB for MySQL with that of ApsaraDB RDS for MySQL.

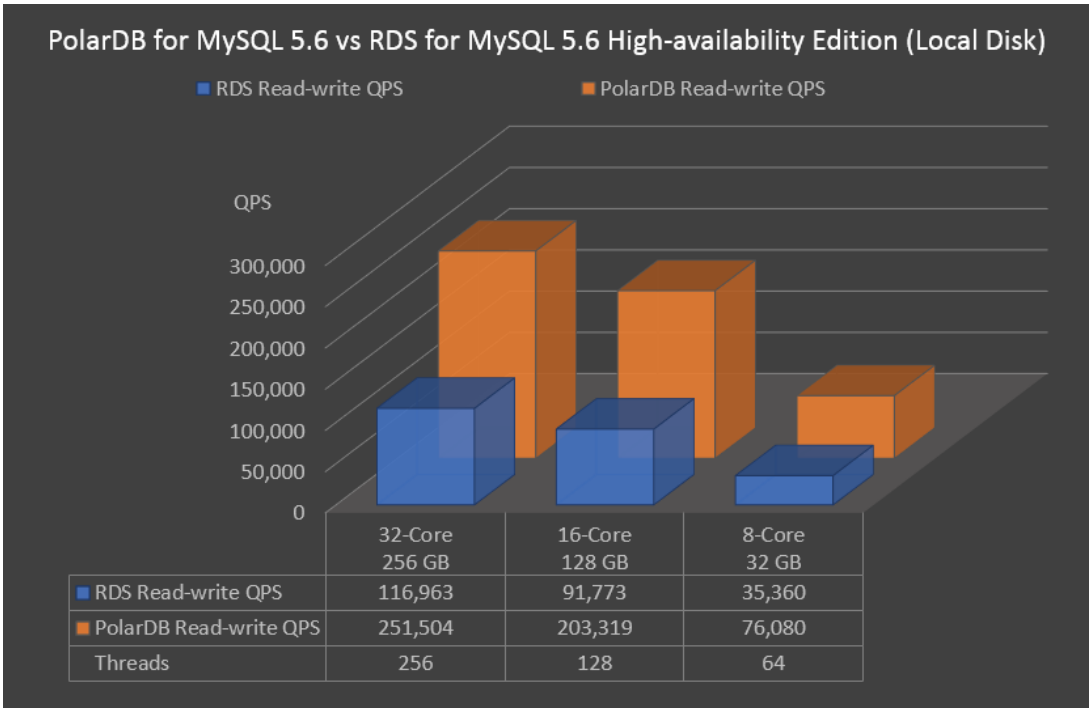
> **Note** For more information about how to test the OLTP performance of ApsaraDB PolarDB for MySQL, see OLTP performance tests.

Compare PolarDB for MySQL 8.0 with RDS for MySQL 8.0



	32-Core 256 GB	16-Core 128 GB	8-Core 32 GB
RDS Read-write QPS	137,877	92,174	39,758
PolarDB Read-write QPS	313,989	203,319	111,543
Threads	256	128	64

Compare PolarDB for MySQL 5.6 with RDS for MySQL 5.6



	32-Core 256 GB	16-Core 128 GB	8-Core 32 GB
RDS Read-write QPS	116,963	91,773	35,360
PolarDB Read-write QPS	251,504	203,319	76,080
Threads	256	128	64

5.OLAP performance tests

This topic describes how to test the online analytical processing (OLAP) performance of an ApsaraDB PolarDB MySQL cluster 8.0 by using TPC-H.

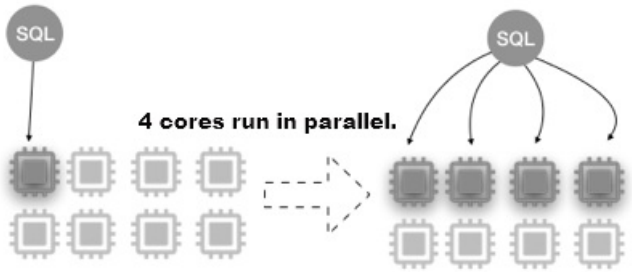
About parallel query

ApsaraDB PolarDB MySQL 8.0 launches a parallel query framework. By default, parallel query is disabled. After parallel query is enabled, the parallel query framework is automatically enabled when the amount of queried data reaches the specified threshold. This helps to reduce the time required to run the query.

> **Note** You can enable parallel query by setting the loose_max_parallel_degree parameter. For more information, see Set cluster parameters.
>
> Follow these rules to set the loose_max_parallel_degree parameter.
>
> - The minimum value is 0, which specifies to disable parallel query.
> - The maximum value is 1024.
> - We recommend that you set the parameter value to 16.

ApsaraDB PolarDB MySQL 8.0 distributes data to different threads at the storage layer. Multiple threads perform parallel computing and return the results to the leader thread. Then, the leader thread merges the results and returns the final result to the user. This helps to improve the query speed and accuracy.

Parallel query is implemented based on the parallel processing capability of multi-core CPUs. The following figure shows how parallel query works on a cluster that has an 8-core CPU and 32-GB memory.



The following section describes how to test the performance of an ApsaraDB PolarDB MySQL cluster under different conditions where the loose_max_parallel_degree parameter is set to 16 and 0, respectively. The test results are also provided.

Test environment

- An Elastic Compute Service (ECS) instance and an ApsaraDB PolarDB MySQL cluster are used for testing, and they must be deployed in the same region and zone. In this test, they are both deployed in Zone I of the China (Hangzhou) region.
- The VPC network type is selected for both the ECS instance and the ApsaraDB PolarDB MySQL cluster.

> **Note** Make sure that the ECS instance and the ApsaraDB PolarDB MySQL cluster are connected to the same VPC network.

- The ApsaraDB PolarDB MySQL cluster for testing is as follows:
  - The node specification is polar.mysql.x8.4xlarge (32-core 256 GB).
  - The database engine is MySQL 8.0.
  - The cluster contains two nodes, one primary node and one read-only node.
  - Use the primary endpoint to connect to the cluster. For information about how to query the primary endpoint, see View or apply for an endpoint.
- The ECS instance for testing is as follows:
  - The instance type is ecs.c5.4xlarge.
  - An ultra disk of 1,000 GiB is attached to the instance.
  - The image used by the instance is 64-bit CentOS 7.0.

Test tool

TPC-H is a standard benchmark. It is developed and released by the Transaction Processing Performance (TPC) Council to evaluate query capabilities of databases. The TPC-H benchmark contains 8 tables and defines 22 complex SQL queries. Most of the queries contain Join operations on several tables, subqueries, and Group by clauses.

Install TPC-H

1. Install TPC-H on the ECS instance.

   > **Note**
   > - The TPC-H version used in this topic is v2.18.0. Download TPC-H v2.18.0.
   > - Before you download TPC-H, make sure that you have performed real-name registration.

   Active Benchmarks

Benchmark/Document	Current Version	Specification	Source Code
TPC-C	5.11.0	pdf	n/a
TPC-DI	1.1.0	pdf	Download TPC-DI_Tools_v1.1.0.zip
TPC-DS	2.11.0	pdf	Download TPC-DS_Tools_v2.11.0.zip
TPC-E	1.14.0	pdf	Download TPC-E_Tools_v1.14.0.zip
TPC-H	2.18.0	pdf	Download TPC-H_Tools_v2.18.0.zip
TPC-VMS	1.2.0	pdf	n/a
TPCX-BB	1.3.1	pdf	Download TPCX-BB_Tools_v1.3.1.zip
TPCX-HCI	1.1.6	pdf	Download TPCx-HCI_Benchmarking_Kit_v1.1.6.zip
TPCX-HS	2.0.3	pdf	Download TPCX-HS_Tools_v2.0.3.zip
TPCX-IOT	1.0.4	pdf	Download TPCX-IOT_Tools_v1.0.4.zip
TPCX-V	2.1.6	pdf	Download TPCx-V_Benchmarking_Kit_v2.1.6.zip

2. Open the *dbgen* directory.

   ```
   cd dbgen
   ```

3. Copy the `makefile` file.

```
cp makefile.suite makefile
```

4. Modify parameters in the `makefile` file, including the CC, DATABASE, MACHINE, and WORKLOAD parameters.
   i. Open the `makefile` file.

   ```
   vim makefile
   ```

   ii. Modify the definitions of the CC, DATABASE, MACHINE, and WORKLOAD parameters.

   ```
   #################
   ## CHANGE NAME OF ANSI COMPILER HERE
   #################
   CC = gcc
   # Current values for DATABASE are: INFORMIX, DB2, ORACLE,
   # SQLSERVER, SYBASE, TDAT (Teradata)
   # Current values for MACHINE are: ATT, DOS, HP, IBM, ICL, MVS,
   # SGI, SUN, U2200, VMS, LINUX, WIN32
   # Current values for WORKLOAD are: TPCH
   DATABASE= MYSQL
   MACHINE = LINUX
   WORKLOAD = TPCH
   ```

   iii. Press Esc and then enter `:wq` to save the file and exit.
5. Modify the `tpcd.h` file and add new macro definitions.
   i. Open the `tpcd.h` file.

   ```
   vim tpcd.h
   ```

   ii. Add the following macro definitions.

   ```
   #ifdef MYSQL
   #define GEN_QUERY_PLAN ""
   #define START_TRAN "START TRANSACTION"
   #define END_TRAN "COMMIT"
   #define SET_OUTPUT ""
   #define SET_ROWCOUNT "limit %d;\n"
   #define SET_DBASE "use %s;\n"
   #endif
   ```

   iii. Press Esc and then enter `:wq` to save the file and exit.
6. Compile the makefile file.

```
make
```

After the file is compiled, two executable files are generated in this directory:

- `dbgen` : the tool to generate data. When you use the InfiniDB test script, you must use this tool to generate TPC-H data.
- `qgen` : the tool to generate SQL queries. The queries generated by different seeds are different. To ensure repeatability, use the 22 queries provided in the attachment.

7. Use TPC-H to generate test data.

```
./dbgen -s 100
```

The `-s` parameter is used to specify the amount of data to be generated.

8. Use TPC-H to generate a query.

> **Note** To ensure the repeatability of test results, you may skip this step and use the 22 queries in the Attachment.

i. Copy `qgen` and `dists.dss` into the *queries* directory.

```
cp qgen queries
cp dists.dss queries
```

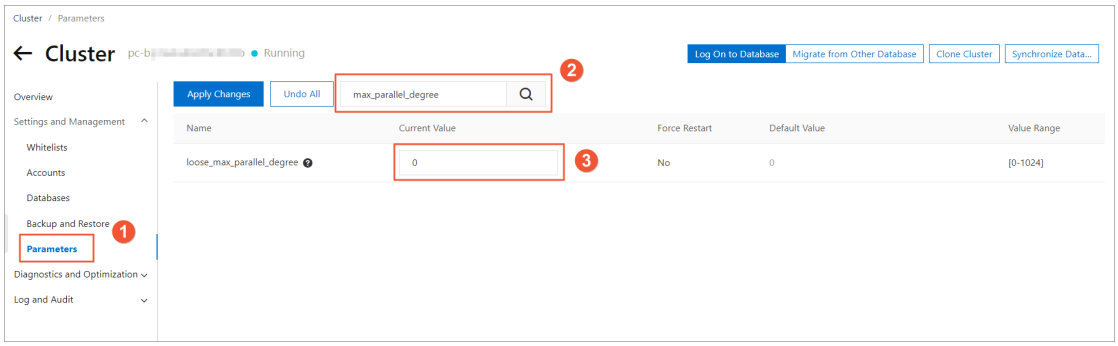
ii. Use the following script to generate a query.

```
#! /usr/bin/bash
for i in {1..22}
do
./qgen -d $i -s 100 > db"$i".sql
done
```

Test procedure

1. Verify that parallel query is enabled on the ApsaraDB PolarDB MySQL cluster.
   i. Log on to the ApsaraDB for PolarDB console.
   ii. In the upper-left corner of the console, select the region where the cluster is deployed.
   iii. Click the ID of the cluster.
   iv. In the left-side navigation pane, choose **Settings and Management** > **Parameters**.

v. Enter loose_max_parallel_degree in the search box and click the search icon.



vi. Set the current value to 16.

> **Note** To clearly understand how parallel query improves the performance of the cluster, perform a comparative test, in which this parameter is set to 16 and then 0.

vii. After the parameter is modified, click **Apply Changes** in the upper-left corner of the page.

viii. In the **Save Changes** dialog box that appears, click **OK**.

2. Connect to an ApsaraDB PolarDB MySQL database on the ECS instance. For more information, see Connect to a database cluster.

3. Create a database.

```
create database tpch100g
```

4. Creates a table.

```
source ./dss.ddl
```

> **Note** `dss.ddl` is under the *dbgen* directory in TPC-H.

5. Load data.

i. Use the following script to create `load.ddl` .

```
load data local INFILE 'customer.tbl' INTO TABLE customer FIELDS TERMINATED BY '|';
load data local INFILE 'region.tbl' INTO TABLE region FIELDS TERMINATED BY '|';
load data local INFILE 'nation.tbl' INTO TABLE nation FIELDS TERMINATED BY '|';
load data local INFILE 'supplier.tbl' INTO TABLE supplier FIELDS TERMINATED BY '|';
load data local INFILE 'part.tbl' INTO TABLE part FIELDS TERMINATED BY '|';
load data local INFILE 'partsupp.tbl' INTO TABLE partsupp FIELDS TERMINATED BY '|';
load data local INFILE 'orders.tbl' INTO TABLE orders FIELDS TERMINATED BY '|';
load data local INFILE 'lineitem.tbl' INTO TABLE lineitem FIELDS TERMINATED BY '|';
```

ii. Load data.

```
source ./load.ddl
```

6. Create a primary key and a foreign key.

```
source ./dss.ri
```

This example uses the `tpch100g` database that was previously created. Replace the content in the `dss.ri` file in TPC-H with the following content.

```
use TPCH100G;
-- ALTER TABLE REGION DROP PRIMARY KEY;
-- ALTER TABLE NATION DROP PRIMARY KEY;
-- ALTER TABLE PART DROP PRIMARY KEY;
-- ALTER TABLE SUPPLIER DROP PRIMARY KEY;
-- ALTER TABLE PARTSUPP DROP PRIMARY KEY;
-- ALTER TABLE ORDERS DROP PRIMARY KEY;
-- ALTER TABLE LINEITEM DROP PRIMARY KEY;
-- ALTER TABLE CUSTOMER DROP PRIMARY KEY;
-- For table REGION
ALTER TABLE REGION
ADD PRIMARY KEY (R_REGIONKEY);
-- For table NATION
ALTER TABLE NATION
ADD PRIMARY KEY (N_NATIONKEY);
ALTER TABLE NATION
ADD FOREIGN KEY NATION_FK1 (N_REGIONKEY) references REGION(R_REGIONKEY);
COMMIT WORK;
-- For table PART
ALTER TABLE PART
ADD PRIMARY KEY (P_PARTKEY);
COMMIT WORK;
-- For table SUPPLIER
ALTER TABLE SUPPLIER
ADD PRIMARY KEY (S_SUPPKEY);
ALTER TABLE SUPPLIER
ADD FOREIGN KEY SUPPLIER_FK1 (S_NATIONKEY) references NATION(N_NATIONKEY);
COMMIT WORK;
-- For table PARTSUPP
ALTER TABLE PARTSUPP
ADD PRIMARY KEY (PS_PARTKEY,PS_SUPPKEY);
COMMIT WORK;
```

```
COMMIT WORK;
-- For table CUSTOMER
ALTER TABLE CUSTOMER
ADD PRIMARY KEY (C_CUSTKEY);
ALTER TABLE CUSTOMER
ADD FOREIGN KEY CUSTOMER_FK1 (C_NATIONKEY) references NATION(N_NATIONKEY);
COMMIT WORK;
-- For table LINEITEM
ALTER TABLE LINEITEM
ADD PRIMARY KEY (L_ORDERKEY,L_LINENUMBER);
COMMIT WORK;
-- For table ORDERS
ALTER TABLE ORDERS
ADD PRIMARY KEY (O_ORDERKEY);
COMMIT WORK;
-- For table PARTSUPP
ALTER TABLE PARTSUPP
ADD FOREIGN KEY PARTSUPP_FK1 (PS_SUPPKEY) references SUPPLIER(S_SUPPKEY);
COMMIT WORK;
ALTER TABLE PARTSUPP
ADD FOREIGN KEY PARTSUPP_FK2 (PS_PARTKEY) references PART(P_PARTKEY);
COMMIT WORK;
-- For table ORDERS
ALTER TABLE ORDERS
ADD FOREIGN KEY ORDERS_FK1 (O_CUSTKEY) references CUSTOMER(C_CUSTKEY);
COMMIT WORK;
-- For table LINEITEM
ALTER TABLE LINEITEM
ADD FOREIGN KEY LINEITEM_FK1 (L_ORDERKEY) references ORDERS(O_ORDERKEY);
COMMIT WORK;
ALTER TABLE LINEITEM
ADD FOREIGN KEY LINEITEM_FK2 (L_PARTKEY,L_SUPPKEY) references
PARTSUPP(PS_PARTKEY,PS_SUPPKEY);
COMMIT WORK;
```

7. Create an index.

```
#! /usr/bin/bash
host=$1
port=$2
user=$3
password=$4
db=$5
sqls=("create index i_s_nationkey on supplier (s_nationkey);"
"create index i_ps_partkey on partsupp (ps_partkey);"
"create index i_ps_suppkey on partsupp (ps_suppkey);"
"create index i_c_nationkey on customer (c_nationkey);"
"create index i_o_custkey on orders (o_custkey);"
"create index i_o_orderdate on orders (o_orderdate);"
"create index i_l_orderkey on lineitem (l_orderkey);"
"create index i_l_partkey on lineitem (l_partkey);"
"create index i_l_suppkey on lineitem (l_suppkey);"
"create index i_l_partkey_suppkey on lineitem (l_partkey, l_suppkey);"
"create index i_l_shipdate on lineitem (l_shipdate);"
"create index i_l_commitdate on lineitem (l_commitdate);"
"create index i_l_receiptdate on lineitem (l_receiptdate);"
"create index i_n_regionkey on nation (n_regionkey);"
"analyze table supplier"
"analyze table part"
"analyze table partsupp"
"analyze table customer"
"analyze table orders"
"analyze table lineitem"
"analyze table nation"
"analyze table region")
for sql in "${sqls[@]}"
do
mysql -h$host -P$port -u$user -p$password -D$db -e "$sql"
done
```

Note To more effectively measure the performance improvement brought by parallel query, you can use the following query to preload the used index data to the memory pool.

```
#! /bin/bash
host=$1
port=$2
user=$3
```

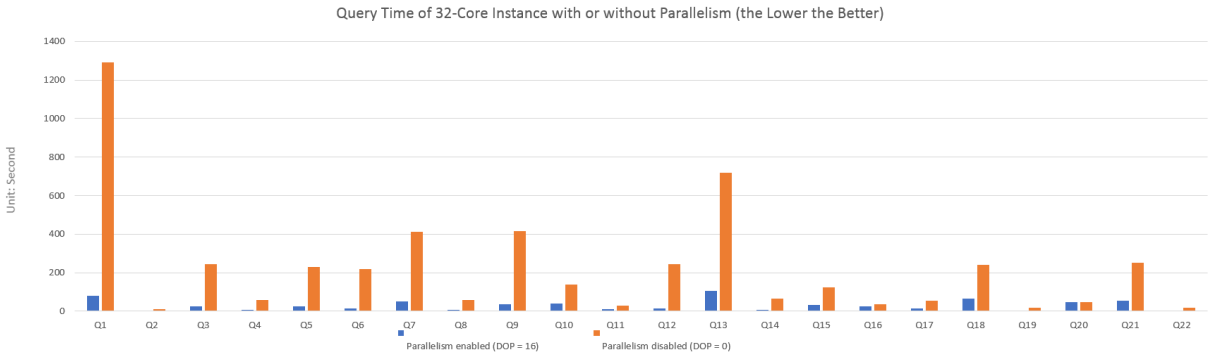
```
password=$4
dbname=$5
MYSQL="mysql -h$host -P$port -u$user -p$password -D$dbname"
if [ -z ${dbname} ]; then
echo "dbname not defined."
exit 1
fi
table_indexes=(
"supplier PRIMARY"
"supplier i_s_nationkey"
"part PRIMARY"
"partsupp PRIMARY"
"partsupp i_ps_partkey"
"partsupp i_ps_suppkey"
"customer PRIMARY"
"customer i_c_nationkey"
"orders PRIMARY"
"orders i_o_custkey"
"orders i_o_orderdate"
"lineitem PRIMARY"
"lineitem i_l_orderkey"
"lineitem i_l_partkey"
"lineitem i_l_suppkey"
"lineitem i_l_partkey_suppkey"
"lineitem i_l_shipdate"
"lineitem i_l_commitdate"
"lineitem i_l_receiptdate"
"nation i_n_regionkey"
"nation PRIMARY"
"region PRIMARY"
)
for table_index in "${table_indexes[@]}"
do
ti=($table_index)
table=${ti[0]}
index=${ti[1]}
SQL="select count(*) from ${table} force index(${index})"
echo "$MYSQL -e '$SQL'"
$MYSQL -e "$SQL"
done
```

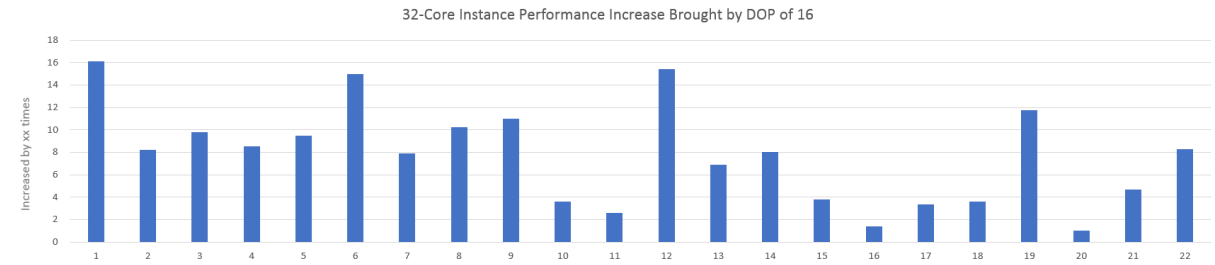
8. Run the query.

```
#! /usr/bin/env bash
host=$1
port=$2
user=$3
password=$4
database=$5
resfile=$6
echo "start test run at"`date "+%Y-%m-%d %H:%M:%S"`|tee -a ${resfile}.out
for (( i=1; i<=22;i=i+1 ))
do
queryfile="Q"${i}".sql"
start_time=`date "+%s. %N"`
echo "run query ${i}"|tee -a ${resfile}.out
mysql -h ${host} -P${port} -u${user} -p${password} $database -e" source $queryfile;" |tee -a ${res
file}.out
end_time=`date "+%s. %N"`
start_s=${start_time%.*}
start_nanos=${start_time#*.}
end_s=${end_time%.*}
end_nanos=${end_time#*.}
if [ "$end_nanos" -lt "$start_nanos" ];then
end_s=$(( 10#$end_s -1 ))
end_nanos=$(( 10#$end_nanos + 10 ** 9))
fi
time=$(( 10#$end_s - 10#$start_s )).`printf "%03d\n" $(( (10#$end_nanos - 10#$start_nanos)/10**
6 ))`
echo ${queryfile} "the "${j}" run cost "${time}" second start at"`date -d @$start_time "+%Y-%m-%d
%H:%M:%S"`" stop at"`date -d @$end_time "+%Y-%m-%d %H:%M:%S"` >> ${resfile}.time
done
```

Result

The following figures illustrate how query performance is improved when parallel query is enabled.





> **Note** In the preceding figures, the letter Q is short for query. For example, Q1 represents the first query.

The test results are as follows.

Query	Time consumed (seconds) Degree of parallelism (DOP) = 16	Time consumed (seconds) Degree of parallelism (DOP) = 0	Performance increases by (DOP 0/DOP 16)
Q1	80.18	1290.6	16
Q2	1.44	11.8	8
Q3	25.05	244.92	10
Q4	6.91	59.61	9
Q5	24.44	231.18	9
Q6	14.51	217.42	15
Q7	51.97	410.59	8
Q8	5.61	57.52	10
Q9	37.84	415.11	11
Q10	38.72	139.73	4
Q11	11.75	30.67	3

Query	Time consumed (seconds) Degree of parallelism (DOP) = 16	Time consumed (seconds) Degree of parallelism (DOP) = 0	Performance increases by (DOP 0/DOP 16)
Q12	15.89	245.19	15
Q13	104.12	718.2	7
Q14	8.31	66.66	8
Q15	32.5	123.79	4
Q16	26.9	37.54	1
Q17	16.2	54.34	3
Q18	66.77	240.28	4
Q19	1.58	18.62	12
Q20	45.88	46.91	1
Q21	53.99	253.27	5
Q22	2.07	17.08	8

6.Guidelines for performance comparison

This topic describes guidelines on how to compare the performance of Apsara PolarDB and Relational Database Service (RDS).

Before you begin, make sure that the following conditions are met for obtaining accurate results of the performance comparison.

- Apsara PolarDB and RDS clusters use the same specifications.
- Apsara PolarDB and RDS clusters are created in the same version.

  Every version of a product provides different implementation mechanisms. For example, MySQL 8.0 is optimized for multi-core CPUs and provides threads such as log_writer, log_fluser, log_checkpoint, and log_write_notifier. However, the performance of MySQL 8.0 is lower than MySQL 5.6/5.7 when the cluster is allocated only a small number of CPU cores. We recommend that you do not compare the performance of PolarDB for MySQL 5.6 and ApsaraDB RDS for MySQL 5.7/8.0, because MySQL 5.7/8.0 uses a new optimizer, which is better than that of MySQL 5.6.
- Perform the performance comparison in a staging environment or use sysbench. This allows you to obtain expected results that meet the requirements of your workloads.
- Do not use a single SQL statement to test the read performance.

  Apsara PolarDB runs in a computing and storage separated architecture. The read performance tested based on a single statement can be affected by the network latency. Consequently, the test result may indicate that the performance of RDS is higher than Apsara PolarDB. 99% queries can hit in the cache of online databases. Only the first read request calls the I/O interface, which degrades the read performance. For subsequent requests, data is directly read from the buffer pool. This means that the read performance is not affected when the database processes these requests.
- Do not use a single SQL statement to test the write performance. To obtain an expected result, perform the performance comparison in a staging environment.

  Make sure that the Apsara PolarDB cluster contains primary and read-only nodes, and the RDS cluster contains master and semi-synchronous read-only instances. By default, Apsara PolarDB uses the Quorum mechanism for writes. If Apsara PolarDB successfully writes data to two or more of the three replicas, the write operation is considered successful. Apsara PolarDB implements data redundancy in the storage plane and ensures high consistency and high availability of the three replicas. We recommend that you use semi-synchronous replication instead of asynchronous replication of ApsaraDB RDS for MySQL to obtain an expected test result.

For more information about performance comparison results, see Comparison with ApsaraDB RDS for MySQL.