ALIBABA CLOUD

阿里云

云数据库 PolarDB
性能白皮书

文档版本：20201201

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.性能测试方法（OLTP）

本文档介绍以SysBench工具测试PolarDB MySQL集群OLTP负载性能，您可以按照本文介绍自行测试对比，快速了解数据库系统的性能。

测试工具

SysBench是一个跨平台且支持多线程的模块化基准测试工具，用于评估系统在运行高负载的数据库时相关核心参数的性能表现。使用SysBench是为了绕过复杂的数据库基准设置，甚至在没有安装数据库的前提下，快速了解数据库系统的性能。

测试环境

- 测试的ECS和PolarDB MySQL均在同一地域、同一可用区。本例中为华东1（杭州）可用区I。
- 网络类型均为VPC网络。

  > **说明** ECS实例和PolarDB MySQL集群需保证在同一个VPC中。

- 测试用PolarDB MySQL集群如下：
  - 节点规格为polar.mysql.x4.large（4核16 GB）。
  - 只读、只写以及读写性能测试使用的是两节点集群（一主一只读）；多个只读节点性能将依次使用一主一只读到一主八只读的集群进行测试。
  - 使用的连接串为集群地址，如何查看PolarDB MySQL集群地址请参见查看或申请连接地址。
- 测试用ECS实例信息如下：
  - 实例规格为ecs.c5.4xlarge。
  - 实例所使用的镜像为CentOS 7.0 64位。

测试场景

对不同规格的PolarDB MySQL进行OLTP的只读、只写、读写以及多个只读节点的只读进行测试。

衡量指标如下：

- TPS（Transactions Per Second）：即数据库每秒执行的事务数，以COMMIT成功次数为准。
- QPS（Queries Per Second）：即数据库每秒执行的SQL数（含INSERT、SELECT、UPDATE、DETELE等）。

安装SysBench

1. 在ECS中执行如下命令安装SysBench。

```
yum install gcc gcc-c++ autoconf automake make libtool bzr mysql-devel git mysql
git clone https://github.com/akopytov/sysbench.git
##从Git中下载sysbench
cd sysbench
##打开sysbench目录
git checkout 1.0.18
##切换到sysbench 1.0.18版本
./autogen.sh
##运行autogen.sh
./configure --prefix=/usr --mandir=/usr/share/man
make
##编译
make install
```

2. 执行如下命令配置SysBench client，使内核可以使用所有的CPU核处理数据包（默认设置为使用2个核），同时减少CPU核之间的上下文切换。

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

> **说明** ffffffff表示使用32个核。请根据实际配置修改，例如ECS为8核，则输入ff。

```
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"
```

测试方法

1. 获取PolarDB MySQL集群地址和端口。具体操作请参见查看或申请连接地址。
2. 关闭PolarDB MySQL集群地址的**主库不接受读**功能，具体操作请参见修改集群地址。
3. 在ECS上执行如下命令，以在PolarDB MySQL集群中创建数据库testdb为例。

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

> **说明** 请将本命令和后续步骤的命令中的 `XXX` 替换为PolarDB MySQL集群的集群地址、端口号、用户名和密码，具体参数说明如下。

参数	说明
-h	PolarDB MySQL集群的集群地址。
-P	PolarDB MySQL集群的端口号。
-u	PolarDB MySQL集群的用户名。
-p	上述用户名对应的密码。

4. 使用SysBench测试PolarDB MySQL集群的一主节点一只读节点的只读性能，整个过程将持续10分钟。

```
sysbench --db-driver=mysql --mysql-host=XXX --mysql-port=XXX --mysql-user=XXX --mysql-password=X
XX --mysql-db=sbtest --table_size=25000 --tables=250 --events=0 --time=600  oltp_read_only prepare
##准备数据
sysbench --db-driver=mysql --mysql-host=XXX --mysql-port=XXX --mysql-user=XXX --mysql-password=X
XX --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
##运行workload
sysbench --db-driver=mysql --mysql-host=XXX --mysql-port=XXX --mysql-user=XXX --mysql-password=X
XX --mysql-db=sbtest --table_size=25000 --tables=250 --events=0 --time=600  --threads=XXX --percentile
=95 --range_selects=0 oltp_read_only cleanup
##清理
```

5. 使用SysBench测试PolarDB MySQL集群的一主节点一只读节点的写入性能，整个过程将持续10分钟。

```
sysbench --db-driver=mysql --mysql-host=XXX --mysql-port=XXX --mysql-user=XXX --mysql-password=X
XX --mysql-db=sbtest --table_size=25000 --tables=250 --events=0 --time=600  oltp_write_only prepare
##准备数据
sysbench --db-driver=mysql --mysql-host=XXX --mysql-port=XXX --mysql-user=XXX --mysql-password=X
XX --mysql-db=sbtest --table_size=25000 --tables=250 --events=0 --time=600  --threads=XXX --percentile
=95 --report-interval=1 oltp_write_only run
##运行workload
sysbench --db-driver=mysql --mysql-host=XXX --mysql-port=XXX --mysql-user=XXX --mysql-password=X
XX --mysql-db=sbtest --table_size=25000 --tables=250 --events=0 --time=600  --threads=XXX --percentile
=95  oltp_write_only cleanup
##清理
```

6. 使用SysBench测试PolarDB MySQL集群一主节点一只读节点的混合读写性能，整个过程将持续10分钟。

```
sysbench --db-driver=mysql --mysql-host=XXX --mysql-port=XXX --mysql-user=XXX --mysql-password=X
XX --mysql-db=sbtest --table_size=250000 --tables=25 --events=0 --time=600  oltp_read_write prepare
##准备数据
sysbench --db-driver=mysql  --mysql-host=XXX --mysql-port=XXX --mysql-user=XXX --mysql-password=X
XX --mysql-db=sbtest --table_size=250000 --tables=25 --events=0 --time=600  --threads=XXX --percentile
=95 --report-interval=1 oltp_read_write run
##运行workload
sysbench --db-driver=mysql  --mysql-host=XXX --mysql-port=XXX --mysql-user=XXX --mysql-password=X
XX --mysql-db=sbtest --table_size=250000 --tables=25 --events=0 --time=600  --threads=XXX --percentile
=95  oltp_read_write cleanup
##清理
```

7. 使用SysBench测试PolarDB MySQL集群的多只读节点的只读性能（对一主节点一只读节点到一主节点八只读节点的集群依次测试）。

```
sysbench --db-driver=mysql --mysql-host=XXX --mysql-port=XXX --mysql-user=XXX --mysql-password=X
XX --mysql-db=sbtest --table_size=250000 --tables=25 --events=0 --time=600  oltp_read_only prepare
##准备数据
sysbench --db-driver=mysql --mysql-host=XXX --mysql-port=XXX --mysql-user=XXX --mysql-password=X
XX --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
##运行workload
sysbench --db-driver=mysql --mysql-host=XXX --mysql-port=XXX --mysql-user=XXX --mysql-password=X
XX --mysql-db=sbtest --table_size=250000 --tables=25 --events=0 --time=600  --threads=XXX --percentile
=95  oltp_read_only cleanup
##清理
```

后续步骤

- PolarDB MySQL 8.0测试结果请参见PolarDB MySQL 8.0性能。
- PolarDB MySQL 5.7测试结果请参见PolarDB MySQL 5.7性能。
- PolarDB MySQL 5.6测试结果请参见PolarDB MySQL 5.6性能。
- PolarDB MySQL对比RDS MySQL测试结果请参见与RDS MySQL的对比。

2.PolarDB MySQL 8.0性能

本文介绍PolarDB MySQL 8.0版OLTP负载性能测试结果。

PolarDB MySQL 8.0优化器支持CBO和RBO两种模式。

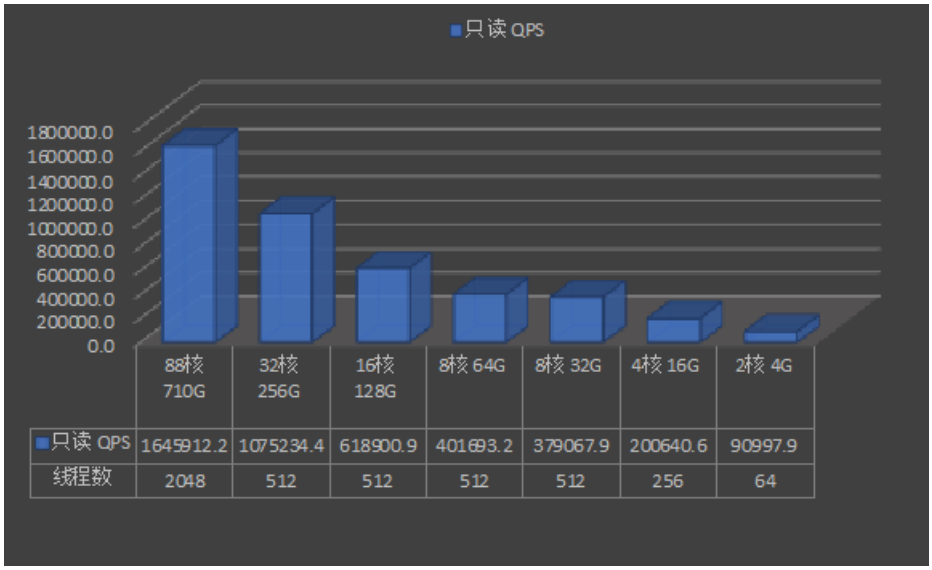
- CBO（Cost-Based Optimization）：根据统计信息或实时采样，结合代价模型评估出代价最小的执行方案。
- RBO（Rule-Based Optimization）：使用预定规则锁定执行方案。

PolarDB MySQL 8.0优化器主要模式为CBO，并支持将RBO作为特定情况下的补充模式。您可以通过ANALYZE命令收集优化器所需要的统计信息，也可以让优化器自动收集统计信息，统计信息包括表大小和记录数、密度向量（density vector）、单列混合直方图（hybrid histogram）、NDV和null ratio。CBO代价模型参数可以通过系统表配置适应软硬件环境，同时可以锁定查询的执行路径或控制CBO搜索范围，防止产出错误的执行方案。

> **说明** 具体测试步骤请参见性能测试方法（OLTP）。

主节点+单只读节点

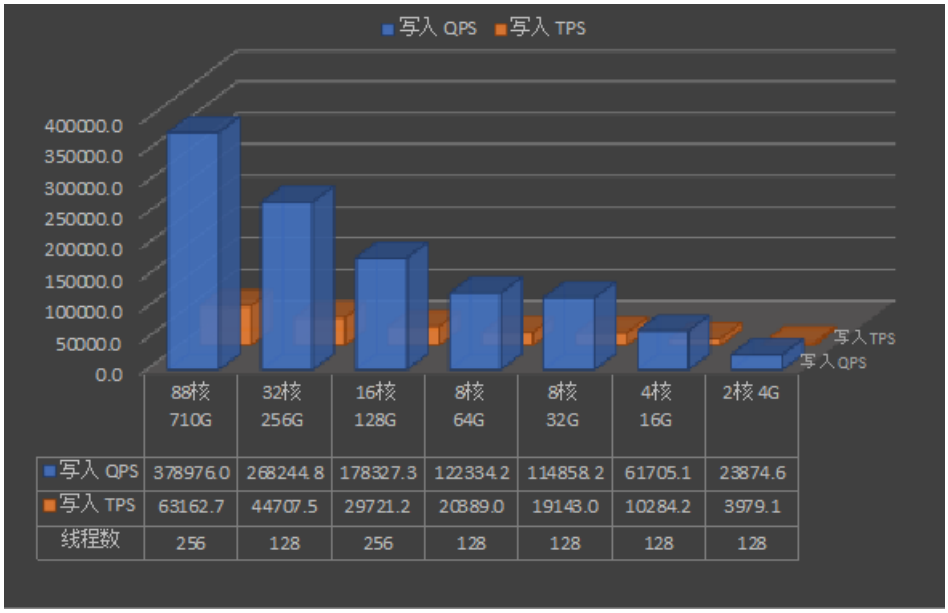
- 各规格只读性能测试结果



	88核 710G	32核 256G	16核 128G	8核 64G	8核 32G	4核 16G	2核 4G
只读 QPS	1645912.2	1075234.4	618900.9	401693.2	379067.9	200640.6	90997.9
线程数	2048	512	512	512	512	256	64

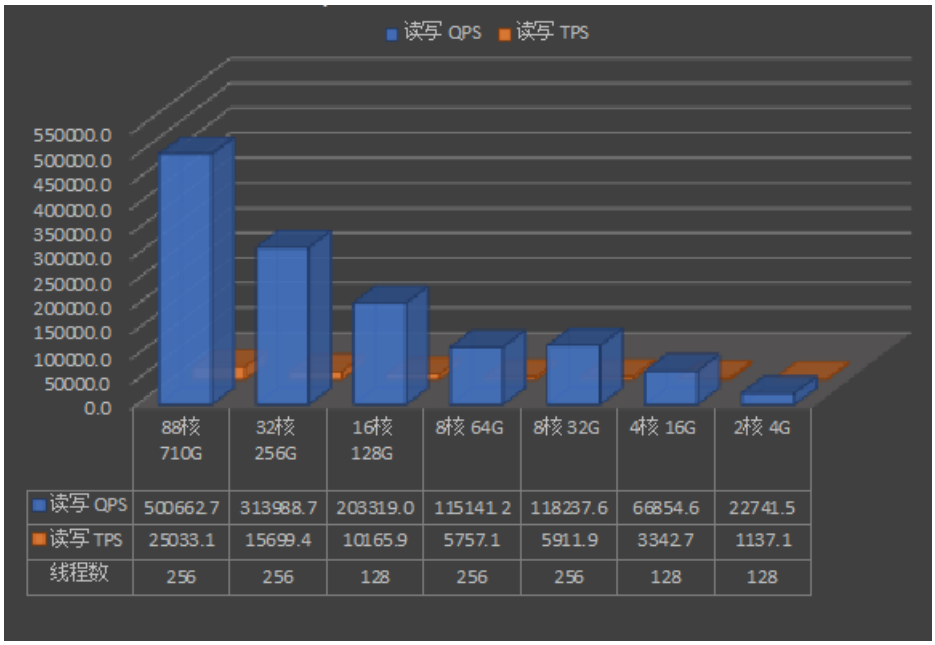
> **说明** 上述测试中未开启所有范围的查询，即设置了range_selects参数为0，故性能较高。

- 各规格写入性能测试结果



	88核 710G	32核 256G	16核 128G	8核 64G	8核 32G	4核 16G	2核 4G
写入 QPS	378976.0	268244.8	178327.3	122334.2	114858.2	61705.1	23874.6
写入 TPS	63162.7	44707.5	29721.2	20389.0	19143.0	10284.2	3979.1
线程数	256	128	256	128	128	128	128

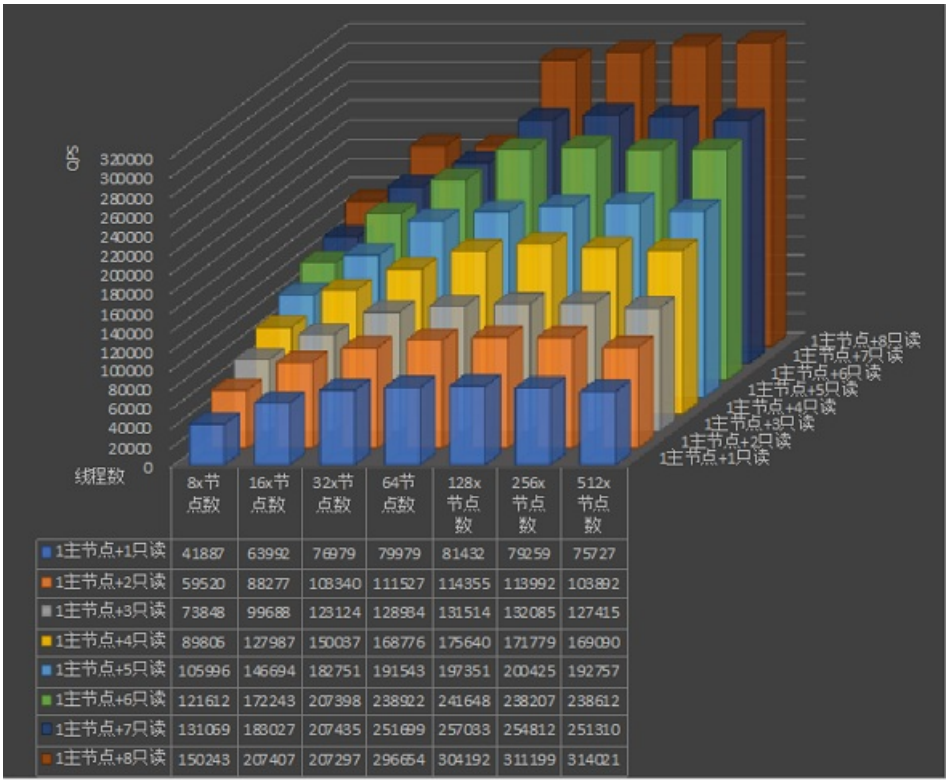
- 各规格混合读写测试结果



	88核 710G	32核 256G	16核 128G	8核 64G	8核 32G	4核 16G	2核 4G
读写 QPS	500662.7	313988.7	203319.0	115141.2	118237.6	66854.6	22741.5
读写 TPS	25033.1	15699.4	10165.9	5757.1	5911.9	3342.7	1137.1
线程数	256	256	128	256	256	128	128

主节点+多只读节点

只读性能测试结果



	8x节点数	16x节点数	32x节点数	64节点数	128x节点数	256x节点数	512x节点数
1主节点+1只读	41887	63992	76979	79979	81432	79259	75727
1主节点+2只读	59520	88277	103340	111527	114355	113992	103892
1主节点+3只读	73848	99688	123124	128984	131514	132085	127415
1主节点+4只读	89806	127987	150037	168776	175640	171779	169090
1主节点+5只读	105996	146694	182751	191543	197351	200425	192757
1主节点+6只读	121612	172243	207398	238922	241648	238207	238612
1主节点+7只读	131059	188027	207435	251699	257033	254812	251310
1主节点+8只读	150243	207407	207297	296654	304192	311199	314021

说明

- 上图显示的是使用集群连接串测试的只读QPS结果。
- 上述测试中未指定range_selects参数，即默认开启了所有范围的查询。
- 上述测试的集群规格为4核16 GB。

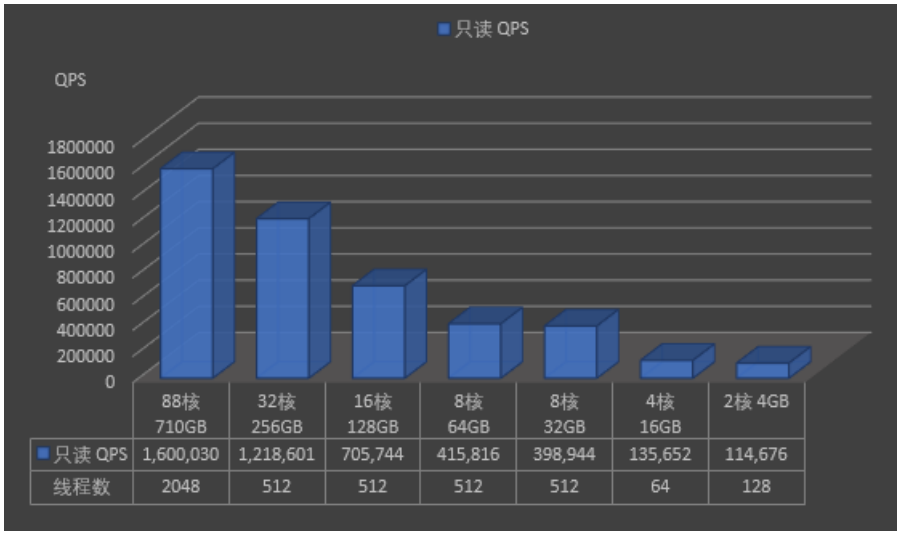
3.PolarDB MySQL 5.7性能

本文介绍PolarDB MySQL 5.7版OLTP负载性能测试结果。

> **说明** 具体测试步骤请参见性能测试方法（OLTP）。

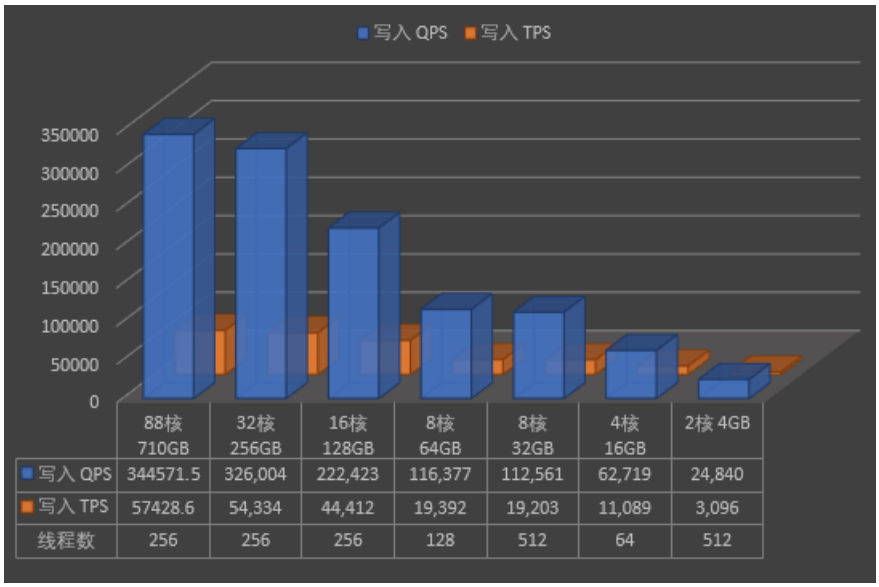
主节点+单只读节点

- 各规格只读性能测试结果



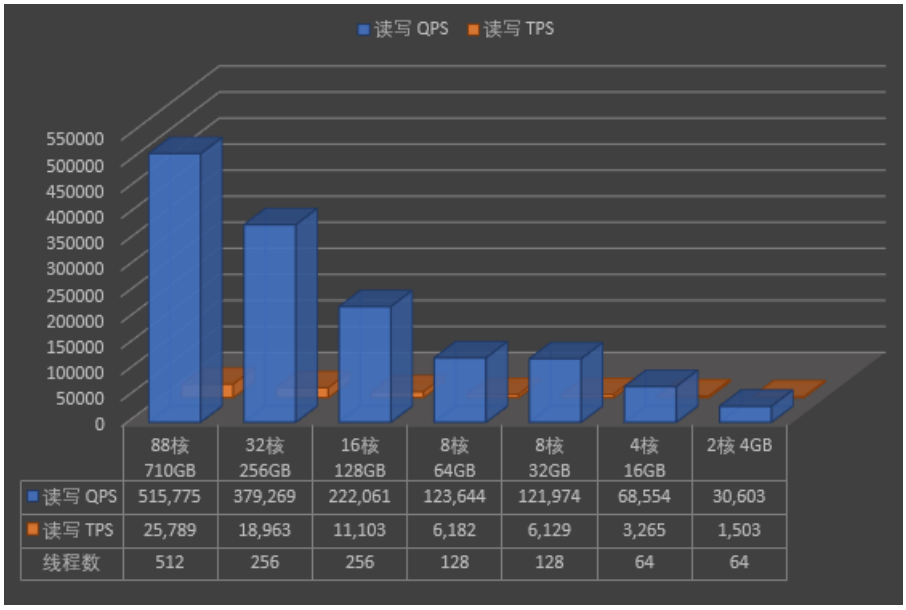
	88核 710GB	32核 256GB	16核 128GB	8核 64GB	8核 32GB	4核 16GB	2核 4GB
只读 QPS	1,600,030	1,218,601	705,744	415,816	398,944	135,652	114,676
线程数	2048	512	512	512	512	64	128

- 各规格写入性能测试结果



	88核 710GB	32核 256GB	16核 128GB	8核 64GB	8核 32GB	4核 16GB	2核 4GB
写入 QPS	344571.5	326,004	222,423	116,377	112,561	62,719	24,840
写入 TPS	57428.6	54,334	44,412	19,392	19,203	11,089	3,096
线程数	256	256	256	128	512	64	512

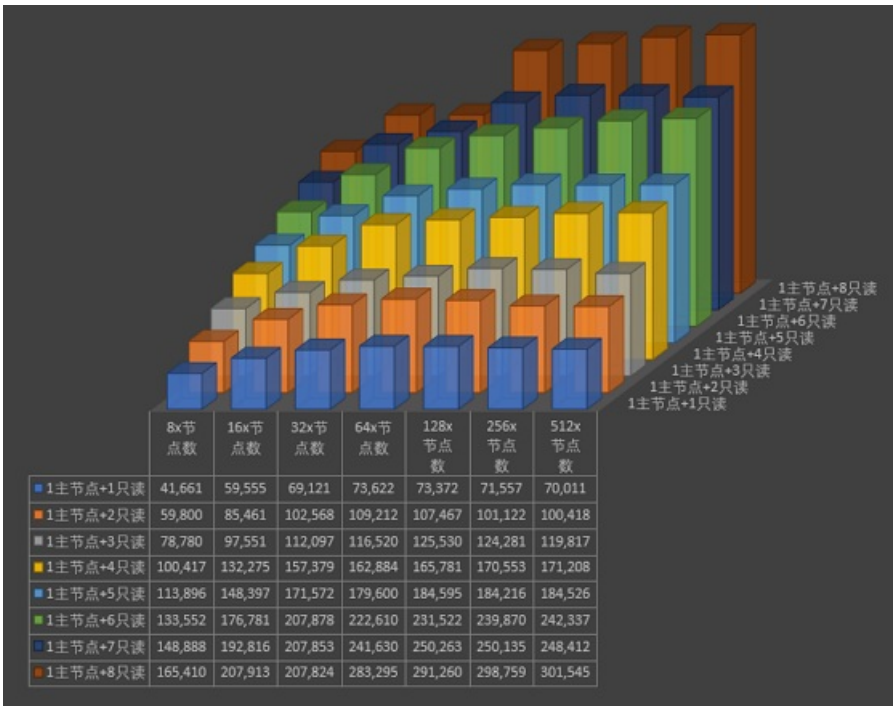
- 各规格混合读写测试结果



	88核 710GB	32核 256GB	16核 128GB	8核 64GB	8核 32GB	4核 16GB	2核 4GB
读写 QPS	515,775	379,269	222,061	123,644	121,974	68,554	30,603
读写 TPS	25,789	18,963	11,103	6,182	6,129	3,265	1,503
线程数	512	256	256	128	128	64	64

主节点+多只读节点

只读性能测试结果



	8x节点数	16x节点数	32x节点数	64x节点数	128x节点数	256x节点数	512x节点数
1主节点+1只读	41,661	59,555	69,121	73,622	73,372	71,557	70,011
1主节点+2只读	59,800	85,461	102,568	109,212	107,467	101,122	100,418
1主节点+3只读	78,780	97,551	112,097	116,520	125,530	124,281	119,817
1主节点+4只读	100,417	132,275	157,379	162,884	165,781	170,553	171,208
1主节点+5只读	113,896	148,397	171,572	179,600	184,595	184,216	184,526
1主节点+6只读	133,552	176,781	207,878	222,610	231,522	239,870	242,337
1主节点+7只读	148,888	192,816	207,853	241,630	250,263	250,135	248,412
1主节点+8只读	165,410	207,913	207,824	283,295	291,260	298,759	301,545

> **说明**
>
> - 上图显示的是使用集群连接串测试的只读QPS结果。
> - 上述测试的集群规格为4核16 GB。

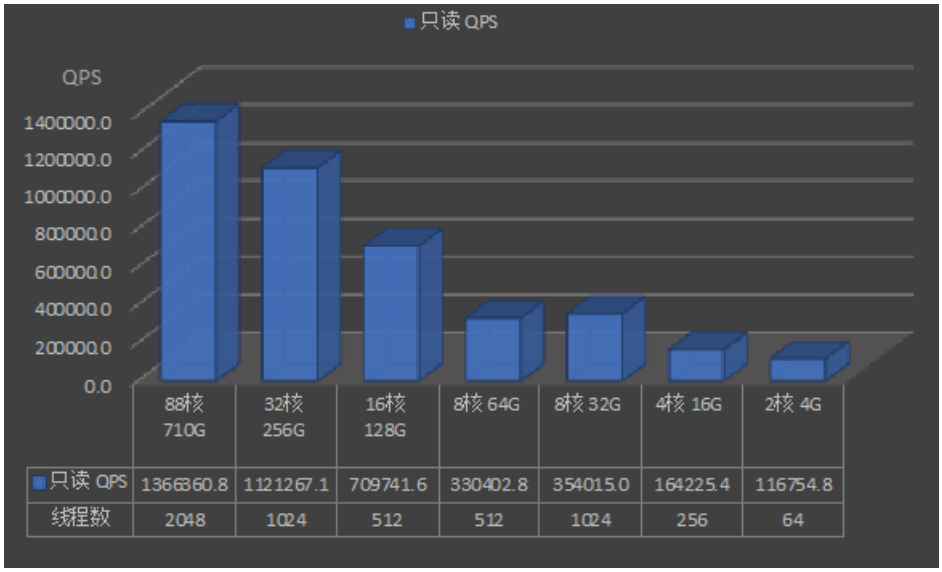
4.PolarDB MySQL 5.6性能

本文介绍PolarDB MySQL 5.6版OLTP负载性能测试结果。

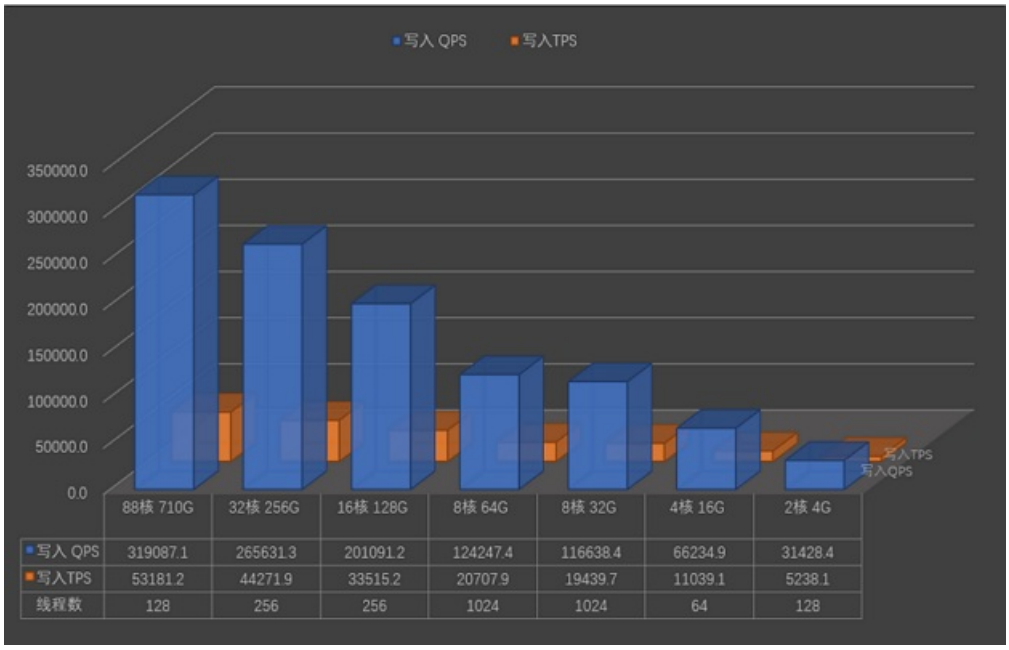
> **说明** 具体测试步骤请参见性能测试方法（OLTP）。

主节点+单只读节点

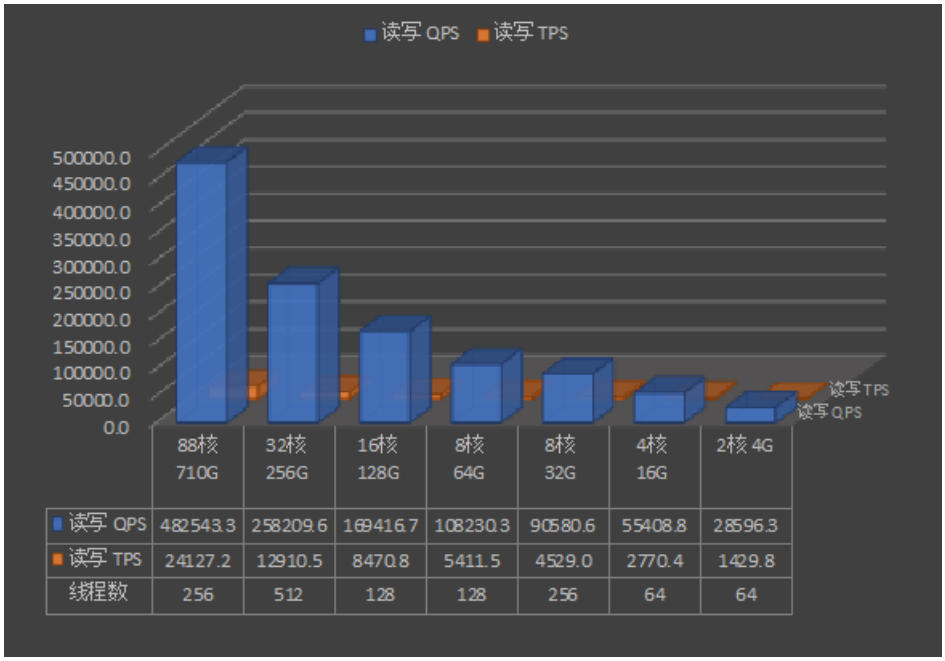
- 各规格只读性能测试结果



- 各规格写入性能测试结果



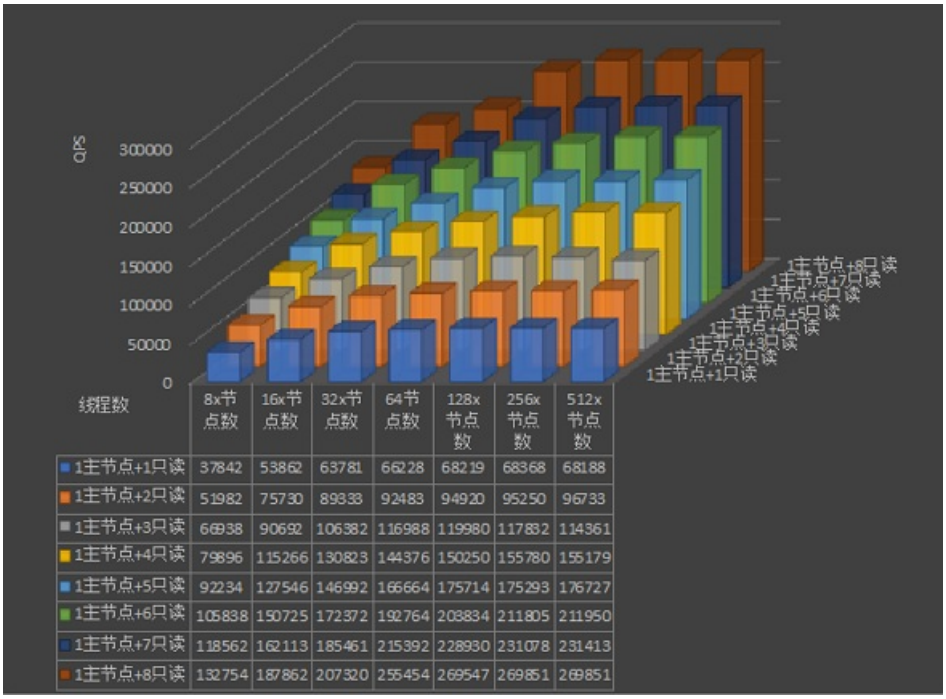
- 各规格混合读写测试结果



	88核 710G	32核 256G	16核 128G	8核 64G	8核 32G	4核 16G	2核 4G
读写 QPS	482543.3	258209.6	169416.7	108230.3	90580.6	55408.8	28596.3
读写 TPS	24127.2	12910.5	8470.8	5411.5	4529.0	2770.4	1429.8
线程数	256	512	128	128	256	64	64

主节点+多只读节点

只读性能测试结果



线程数	8x节点数	16x节点数	32x节点数	64节点数	128x节点数	256x节点数	512x节点数
1主节点+1只读	37842	53862	63781	66228	68219	68368	68188
1主节点+2只读	51982	75730	89333	92483	94920	95250	96733
1主节点+3只读	66938	90692	106382	116988	119980	117832	114361
1主节点+4只读	79896	115266	130823	144376	150250	155780	155179
1主节点+5只读	92234	127546	146992	166664	175714	175293	176727
1主节点+6只读	105838	150725	172372	192764	203834	211805	211950
1主节点+7只读	118562	162113	185461	215392	228930	231078	231413
1主节点+8只读	132754	187862	207320	255454	269547	269851	269851

> **说明**
>
> - 上图显示的是使用集群连接串测试的只读QPS结果。
> - 上述测试的集群规格为4核16 GB。

5.与RDS MySQL的对比

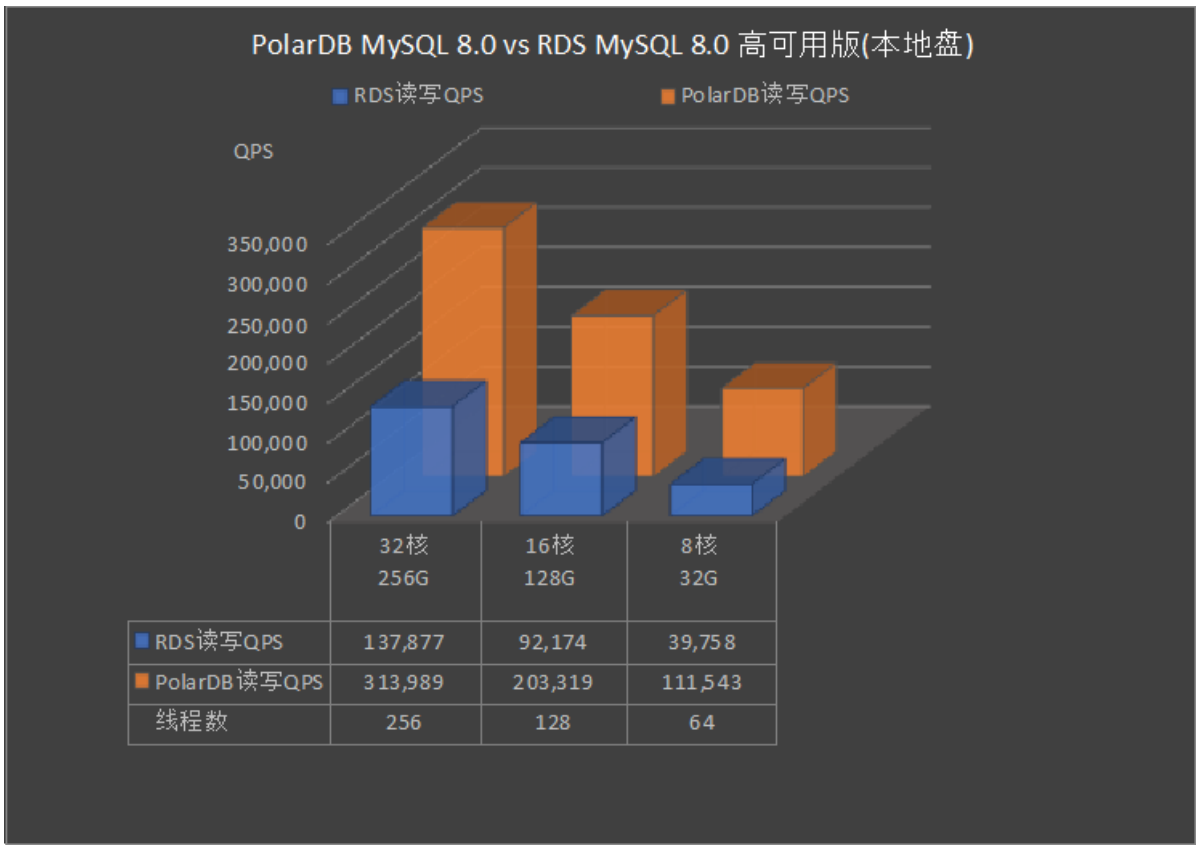
本文介绍PolarDB MySQL与RDS MySQL在相同场景中OLTP负载性能的对比结果。

相较于RDS MySQL，PolarDB MySQL在如下几方面进行了优化，提高了集群整体性能：

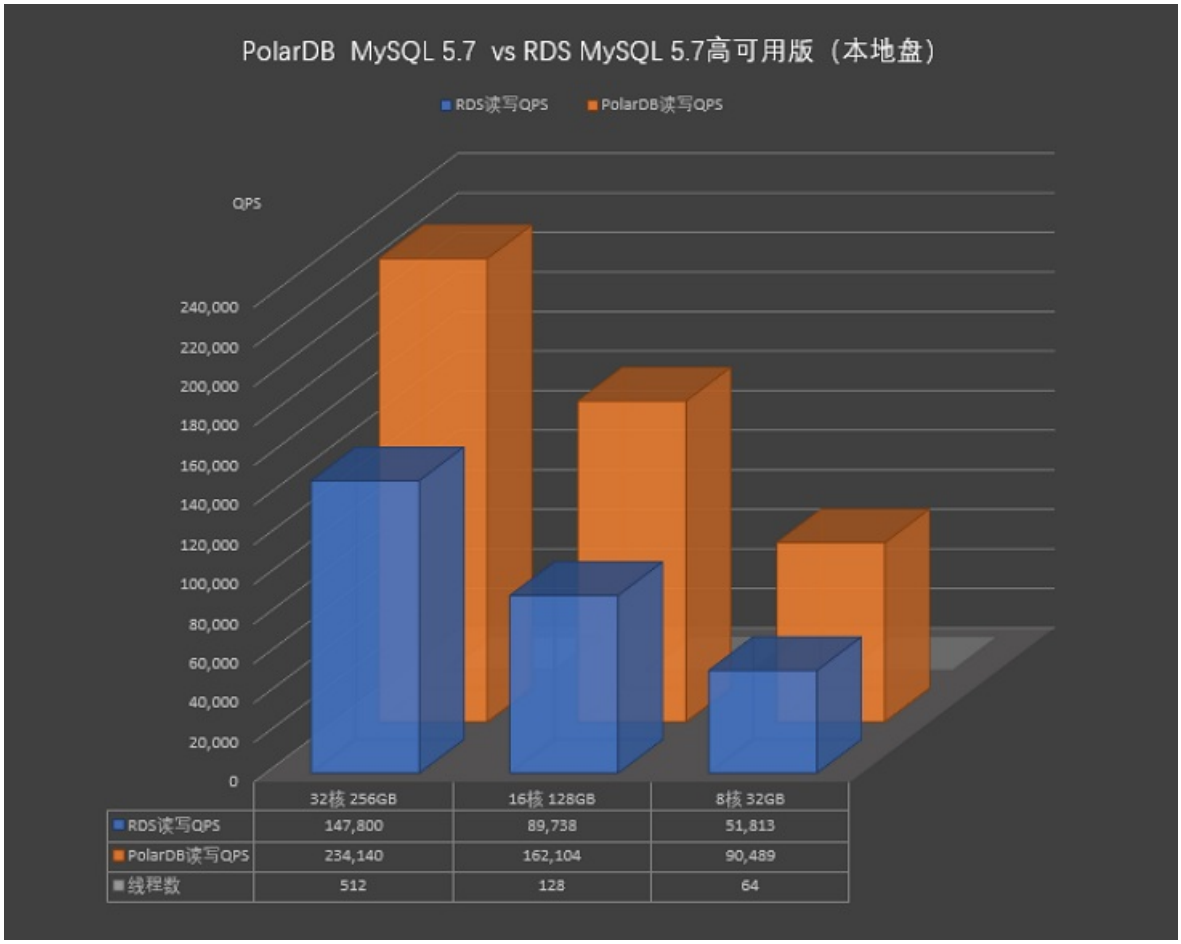
- 采用了领先硬件技术，包括使用3DXpoint存储介质的Optane存储卡、NVMeSSD和RoCE RDMA网络等。
- 基于新的硬件，实现了一整套在用户态运行的IO和网络协议栈，获得了更高的性能、更低的延迟。
- 通过锁优化、IO路径优化、针对大表优化等内核层面的优化，实现了并发场景下的更优性能。

> **说明** 具体测试步骤请参见性能测试方法（OLTP）。

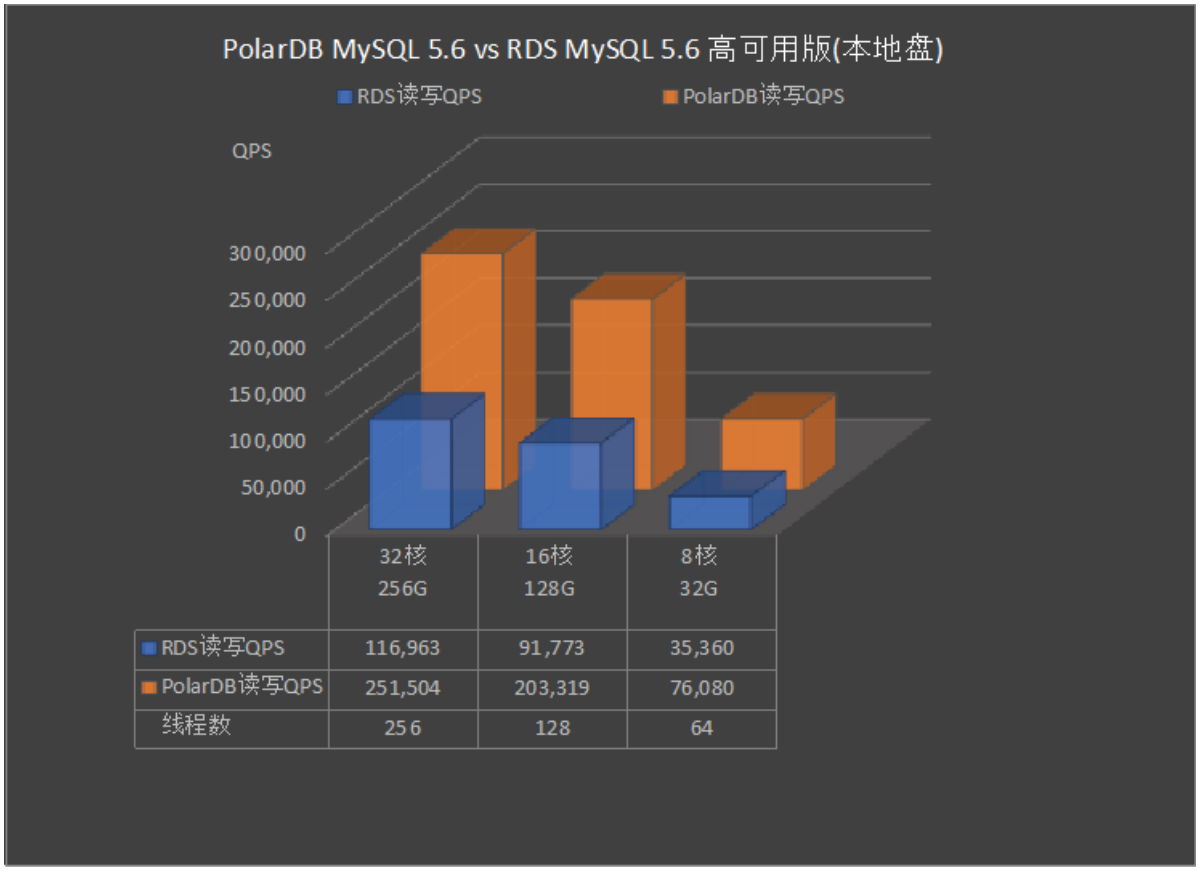
MySQL 8.0对比



MySQL 5.7对比


PolarDB MySQL 5.7 vs RDS MySQL 5.7高可用版（本地盘）
RDS读写QPS
PolarDB读写QPS
QPS
240,000
220,000
200,000
180,000
160,000
140,000
120,000
100,000
80,000
60,000
40,000
20,000
0
32核 256GB
16核 128GB
8核 32GB
RDS读写QPS 147,800 89,738 51,813
PolarDB读写QPS 234,140 162,104 90,489
线程数 512 128 64

MySQL 5.6对比


PolarDB MySQL 5.6 vs RDS MySQL 5.6 高可用版(本地盘)
RDS读写QPS
PolarDB读写QPS
QPS
300,000
250,000
200,000
150,000
100,000
50,000
0
32核 256G
16核 128G
8核 32G
RDS读写QPS 116,963 91,773 35,360
PolarDB读写QPS 251,504 203,319 76,080
线程数 256 128 64

6.并行查询性能（OLAP）

本文档介绍以TPC-H测试PolarDB MySQL 8.0版集群OLAP负载性能，您可以按照本文介绍自行测试对比，快速了解数据库系统的性能。

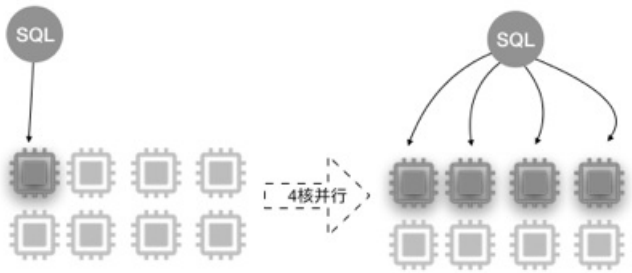
并行查询简介

PolarDB MySQL数据库8.0版推出并行查询（Parallel Query）框架。并行查询默认为关闭状态，当您开启并行查询后，查询数据量到达一定阈值，就会自动启动并行查询框架，从而使查询耗时下降。

> **说明** 您可以通过设置loose_max_parallel_degree参数来开启并行查询，如何设置集群参数请参见设置集群参数。
>
> loose_max_parallel_degree参数说明如下：
>
> - 最小值为0（关闭并行查询）。
> - 最大值为1024。
> - 建议设置并行查询参数为16。

PolarDB MySQL数据库8.0版在存储层将数据分片到不同的线程上，多个线程并行计算，将结果流水线汇聚到总线程，最后由总线程做简单归并将结果返回给用户，提高查询效率。

并行查询利用多核CPU的并行处理能力，以8核32G规格的集群为例，并行查询示意图如下所示。



下文将介绍并行查询参数分别设置为16与0时，PolarDB集群负载性能的测试方式与测试结果。

测试环境

- 测试的ECS和PolarDB均在同一地域、同一可用区。本例中为华东1（杭州）可用区I。
- 网络类型均为VPC网络。

  > **说明** ECS实例和PolarDB集群需保证在同一个VPC中。

- 测试用PolarDB集群如下：
  - 节点规格为polar.mysql.x8.4xlarge（32核256G）。
  - 数据库版本为MySQL 8.0。
  - 节点数量为2个（一个主节点，一个只读节点）。
  - 使用的连接串为主地址，如何查看PolarDB主地址请参见查看或申请连接地址。
- 测试用ECS实例信息如下：
  - 实例规格为ecs.c5.4xlarge。
  - 实例挂载1000G高效云盘。

- 实例所使用的镜像为CentOS 7.0 64位。

测试工具

TPC-H是业界常用的一套基准，由TPC委员会制定发布，用于评测数据库的分析型查询能力。TPC-H查询包含8张数据表、22条复杂的SQL查询，大多数查询包含若干表Join、子查询和Group by聚合等。

安装TPC-H

1. 在ECS上安装TPC-H。

   > 说明
   > - 本文使用的TPC-H版本为v2.18.0。
   > - TPC-H需要完成注册后才可以下载。

   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. 打开*dbgen*目录。

   ```
   cd dbgen
   ```

3. 复制 `makefile` 文件。

   ```
   cp makefile.suite makefile
   ```

4. 修改 `makefile` 文件中的CC、DATABASE、MACHINE、WORKLOAD等参数定义。

   i. 打开 `makefile` 文件。

   ```
   vim makefile
   ```

ii. 修改CC、DATABASE、MACHINE、WORKLOAD参数的定义。

```
################
## 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. 按ECS键，然后输入 `:wq` 退出并保存。

5. 修改 `tpcd.h` 文件，并添加新的宏定义。

i. 打开 `tpcd.h` 文件。

```
vim tpcd.h
```

ii. 添加如下宏定义。

```
#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. 按ECS键，然后输入 `:wq` 退出并保存。

6. 对文件进行编译。

```
make
```

编译完成后该目录下会生成两个可执行文件：

- `dbgen`：数据生成工具。在使用InfiniDB官方测试脚本进行测试时，需要用该工具生成tpch相关表数据。
- `qgen`：SQL生成工具。生成初始化测试查询，由于不同的seed生成的查询不同，为了结果的可重复性，请使用附件提供的22个查询。

7. 使用TPC-H生成测试数据。

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

dbgen参数 `-s` 的作用是指定生成测试数据的仓库数。

8. 使用TPC-H生成查询。

> **说明** 为了测试结果可重复，您可以忽略下面的生成查询的步骤，使用附件的22个查询进行测试。

i. 将 `qgen` 与 `dists.dss` 复制到*queries*目录下。

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

ii. 使用以下脚本生成查询。

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

测试方法

1. 检查并行查询是否开启。

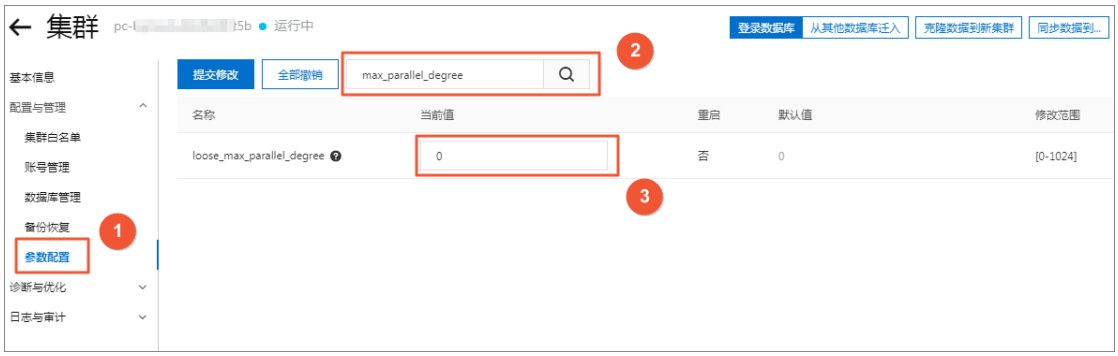
i. 登录PolarDB控制台。

ii. 在控制台左上角，选择集群所在地域。

iii. 单击目标集群ID。

iv. 在左侧导航栏中，选择**配置与管理 > 参数配置**。

v. 在搜索栏输入loose_max_parallel_degree后，单击放大镜图标进行搜索。



vi. 设置**当前值**为16。

> **说明** 本例用于对比开启和关闭并行查询的效果，分别设置为16和0进行对比测试。

vii. 参数设置完成后，单击页面左上方**提交修改**。

viii. 在弹出的**保存改动**对话框中，单击**确定**。

2. 在ECS上连接PolarDB数据库，具体操作请参见连接数据库集群。
3. 创建数据库。

```
create database tpch100g
```

4. 创建表。

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

> 说明 dss.ddl 在TPC-H中*dbgen*目录下。

5. 加载数据。
   i. 创建 load.ddl ，脚本内容如下：

```
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. 加载数据。

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

6. 创建主外键。

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

以创建的数据库 tpch100g 为例，将TPC-H的 dss.ri 文件中的内容替换成如下内容。

```
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
```

```
-- 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;
-- 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. 创建索引。

```
#!/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
```

说明 为了更有效地衡量并行查询带来的性能提升，您可以通过如下查询将使用到的索引数据预载到内存池中。

```
#!/bin/bash
host=$1
port=$2
user=$3
password=$4
```

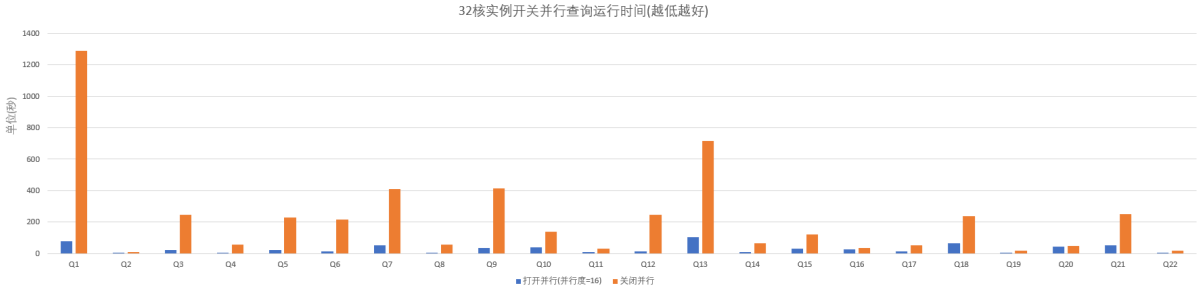
```
dbname=$5
MYSQL="mysql -h$host -P$port -u$user -p$password -D$dbname"
if [ -z ${dbname} ]; then
  echo "dbname not defined."
  exit 1
fi
table_indexes=(
    "supplier PRIMARY"
    "supplier i_s_nationkey"
    "part PRIMARY"
    "partsupp PRIMARY"
    "partsupp i_ps_partkey"
    "partsupp i_ps_suppkey"
    "customer PRIMARY"
    "customer i_c_nationkey"
    "orders PRIMARY"
    "orders i_o_custkey"
    "orders i_o_orderdate"
    "lineitem PRIMARY"
    "lineitem i_l_orderkey"
    "lineitem i_l_partkey"
    "lineitem i_l_suppkey"
    "lineitem i_l_partkey_suppkey"
    "lineitem i_l_shipdate"
    "lineitem i_l_commitdate"
    "lineitem i_l_receiptdate"
    "nation i_n_regionkey"
    "nation PRIMARY"
    "region PRIMARY"
)
for table_index in "${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
```

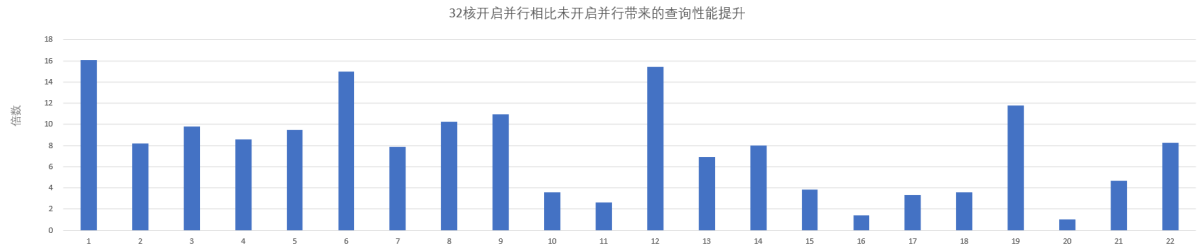
8. 运行查询。

```
#!/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 ${resfile}.o
ut
end_time=`date "+%s.%N"`
start_s=${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
```

执行结果

将并行查询参数分别设置为16与0的对比结果如下图所示。





说明 Q1为第一个查询，Q2为第二个查询，以此类推。

测试结果具体信息如下表所示：

查询	耗时（秒） 并行度=16	耗时（秒） 并行度=0	提高倍数 （并行度=0/并行度=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
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

查询	耗时（秒） 并行度=16	耗时（秒） 并行度=0	提高倍数 （并行度=0/并行度=16）
Q20	45.88	46.91	1
Q21	53.99	253.27	5
Q22	2.07	17.08	8

7.性能对比注意事项

本文介绍PolarDB和RDS进行性能对比的相关注意事项。

在您对PolarDB和RDS进行性能对比前，请了解以下注意事项，以便能获得比较准确、合理的性能对比结果。

- 使用相同规格配置的PolarDB和RDS进行性能对比。
- 使用相同版本的PolarDB和RDS进行性能对比。

  因为不同版本的实现机制不一样，例如MySQL 8.0针对多核数CPU做优化，单独抽象出来Log_writer、log_fluser、log_checkpoint、log_write_notifier等线程，但在CPU核数较少的情况下性能则不如MySQL 5.6/5.7。不推荐使用PolarDB MySQL 5.6和RDS MySQL 5.7/8.0进行对比，因为MySQL 5.6的优化器比较旧，不如新版本。
- 推荐使用模拟线上压力的场景进行实际性能对比，或者使用sysbench进行对比，这样获得的数据更接近线上实际场景。
- 在对比读性能的时候，不推荐您使用单条SQL进行比较。

  因为PolarDB是计算存储分离的架构，所以单条语句有网络延迟的影响，导致读性能不如RDS。但是线上数据库的缓存命中率基本都在99%以上，因此只有第一次的读会调用I/O，因此读取性能会降低，后续数据都在缓存池（Buffer Pool）中，并不需要调用I/O，因此性能是一样的。
- 在对比写性能的时候，同样不推荐您使用单条SQL进行比较，推荐模拟线上环境进行压力测试。

  如果要对比RDS性能，请使用PolarDB（主节点+只读节点）和RDS（主实例+半同步的只读实例）进行对比。这是因为PolarDB的架构在写入数据的时候默认采用Quorum机制，即写入数据时默认写入到三副本里面的大多数（在三个副本中的两个或两个以上写入成功，就认为写操作成功了）。PolarDB已经在存储层面做数据冗余，并保证三副本强同步高可靠，使用RDS MySQL的半同步复制（而不是异步复制）进行对比更合理。

PolarDB MySQL与RDS MySQL性能对比结果请参见与RDS MySQL的对比。