

ALIBABA CLOUD

阿里云

云原生分布式数据库 PolarDB-X
性能白皮书

文档版本：20220406

 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 <code>cd /d C:/window</code> 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	<code>bae log list --instanceid</code> <i>Instance_ID</i>
[] 或者 [a b]	表示可选项，至多选择一个。	<code>ipconfig [-all -t]</code>
{ } 或者 {a b}	表示必选项，至多选择一个。	<code>switch {active stand}</code>

目录


1.TPC-C测试说明	05
2.TPC-H测试说明	08
3.Sysbench测试说明	12
4.Sysbench 使用指南	15

1.TPC-C测试说明

本文介绍如何使用TPC-C工具测试PolarDB-X 1.0数据库的联机交易处理（偏向OLTP能力），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。

背景信息

TPC-C是业界常用的一套Benchmark，由TPC委员会制定发布，用于评测数据库的联机交易处理（偏向OLTP能力）。主要涉及10张表，包含了NewOrder（新订单的生成）、Payment（订单付款）、OrderStatus（最近订单查询）、Delivery（配送）和StockLevel（库存缺货状态分析）等五类业务事务模型。TPC-C使用tpmC值（Transactions per Minute）来衡量系统最大有效吞吐量（MQTh, Max Qualified Throughput），其中Transactions以NewOrder Transaction为准，即最终衡量单位为每分钟处理的新订单数。

 **说明** 本文的TPC-C的实现基于TPC-C的基准测试，并不能与已发布的TPC-C基准测试结果相比较，本文中的测试并不符合TPC-C基准测试的所有要求。

测试设计

● 测试数据量

- 针对普通实例性能测试，TPC-C测试结果主要基于1000 Warehouse，其中主要的表数据量如下：
 - bmsql_order_line 3亿行
 - bmsql_stock 1亿行
 - bmsql_customer、bmsql_history、bmsql_oorder各3000万行
- 为了验证PolarDB-X 1.0的水平扩展能力，还引入了超大规格实例的TPC-C测试，相比于普通实例扩展了近10倍的资源。在超大规格压测设计上，构建了10000 Warehouse，同时TPC-C施压机需要增加到3台32核的ECS机器，避免压测本身成为了瓶颈点。


● 测试所用实例规格

- 企业版测试环境：PolarDB-X 1.0计算资源实例企业版32核128 GB（单节点16核64 GB）、4台RDS MySQL 5.7实例（8核32 GB独享型）。
- 标准版测试环境：PolarDB-X 1.0计算资源实例标准版16核64 GB（单节点8核32 GB）、4台RDS MySQL 5.7实例（4核32 GB独享型）。
- 超大规格测试环境：PolarDB-X 1.0计算资源实例企业版256核1024 GB（单节点16核64 GB）、12台RDS MySQL 5.7实例（32核128 GB独享型）。

测试方法

● 步骤1：准备压力机ECS

需准备一个ECS，后续操作步骤中涉及的数据准备、运行压测等使用的都是这台ECS机器。

 **说明** 建议将测试所用ECS部署在VPC网络内（如选择经典网络可能出现RDS某些规格没有库存的情况）。请记住该VPC的名称和ID，后续的所有实例都将部署在该VPC内。

● 步骤2：准备压测所用PolarDB-X 1.0实例

- 创建PolarDB-X 1.0实例，详细操作步骤请参见[步骤一：购买PolarDB-X 1.0计算层资源创建实例](#)。
- 在实例中创建一个待压测的数据库（本测试中数据库名为 `tpcc`，详细操作步骤请参见[步骤二：在私有定制RDS实例之上构建PolarDB-X 1.0数据库](#)）

 说明 需保证ECS和在同一个VPC中。

● 步骤3：压测数据准备

i. 准备压测工具

说明

- 本文使用开源的BenchmarkSQL 5.0作为TPC-C测试。
- BenchmarkSQL默认不支持MySQL协议，需要进行工具改造适配，具体改动，请参见[BenchmarkSQL 5.0 支持对MySQL的TPC-C测试](#)。

a. 下载改造好的压测包tpcc.tar.gz，并在ECS中执行如下命令将其解压到tpcc目录：

```
mkdir tpcc tar zxvf tpcc.tar.gz -C tpcc
```

b. 按如下说明修改压缩包内的对应文件：

```
src/client/jTPCC.java (增加一个MySQL的type) src/client/jTPCCConnection.java (支持MySQL的语法，加一个别名) src/LoadData/LoadData.java (关闭loader数据时的大事务机制) src/LoadData/LoadDataWorker.java (关闭loader数据时的大事务机制) run/funcs.sh (脚本增加一个MySQL的type) run/runDatabaseBuild.sh (去掉不必要的阶段) run/runBenchmark.sh (调整默认jvm参数) run/runLoader.sh (调整默认jvm参数) run/sql.common/foreignKeys.sql (注释全部外键创建，PolarDB-X 1.0不支持外键) run/sql.common/indexCreates.sql (注释掉全部主键创建，只留2个索引创建，MySQL默认在建表时直接创建索引) run/sql.common/indexDrops.sql (注释全部主键删除) run/sql.common/tableCreates.sql (添加主键和拆分键，PolarDB-X 1.0需要指定拆分键)
```

ii. 准备压测配置

在ECS中执行如下命令，在 tpcc/run 目录下创建 props.mysql 文件：

```
// ----- env config ----- // db=mysql driver=com.mysql.jdbc.Driver conn=jdbc:mysql://drdsxxxx:3306/tpcc? useSSL=false&useServerPrepStmts=false&useConfigs=maxPerformance&rewriteBatchedStatements=true user=tpcc password=tpcc // warehouse 数量 warehouses=1000 // 导入数据的并发数，每100并发预计产生2万TPS，可以结合目标TPS能力调整并发 // runLoader.sh的jvm内存，100并发默认为4 GB，500并发建议设置为16 GB loadWorkers=100 // TPC-C 压测并发数 terminals=1000 // 压测时间，单位分钟 runMins=10 // ----- default config ----- // //To run specified transactions per terminal- runMins must equal zero runTxnsPerTerminal=0 //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%M%S // osCollectorScript=./misc/os_collector_linux.py // osCollectorInterval=1 // osCollectorSSHAddr=user@dbhost // osCollectorDevices=net_eth0 blk_sda
```

说明

- 导入压测数据时，需要关注 `warehouses`（仓库数）和 `loadWorkers`（并发数）。
- TPC-C压测时，需要关注 `terminals`（并发数）和 `runMins`（运行时间）。

iii. 压测执行

a. 在ECS中执行如下命令准备压测数据：

```
cd tpcc/run nohup ./runDatabaseBuild.sh props.mysql &
```

说明 默认按照100并发导入，总共5亿多记录，整体导入时间在小时级别，建议通过nohup推到后台运行，避免ssh命令行断开导致导入中断。

b. 执行如下命令运行TPC-C测试：

```
cd tpcc/run ./runBenchmark.sh props.mysql
```

运行之后可以看到如下测试结果：

```
08:56:16,844 [Thread-883] INFO jTPCC : Term-00, Measured tpmC (NewOrders) = 10423
0.88 08:56:16,844 [Thread-883] INFO jTPCC : Term-00, Measured tpmTOTAL = 231664.4
9 08:56:16,844 [Thread-883] INFO jTPCC : Term-00, Session Start = 2019-09-19 08:5
4:16 08:56:16,845 [Thread-883] INFO jTPCC : Term-00, Session End = 2019-09-19 08:
56:16 08:56:16,845 [Thread-883] INFO jTPCC : Term-00, Transaction Count = 465440
```

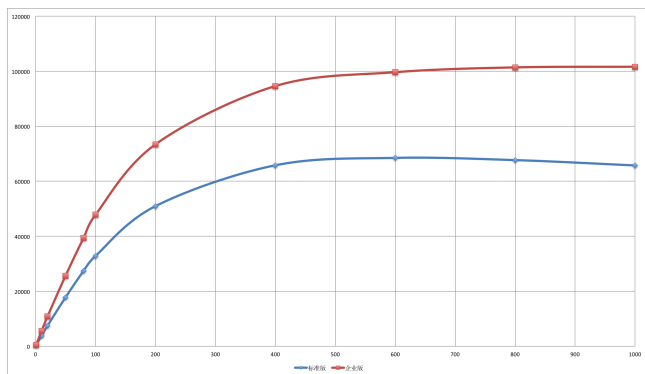
说明 `tpmC` 值即为对应压测场景下的结果。更多测试结果详情，请参见 [测试结果](#)。

c. 执行如下命令清理压测数据：

```
cd tpcc/run ./runDatabaseDestroy.sh props.mysql
```

测试结果

并发度	标准版实例tpmC	企业版实例tpmC	超大规格实例tpmC
1个客户端x 1000并发	65,735.14	101,620.8	/
6个客户端x 1000并发	/	/	821,547.97



2.TPC-H测试说明

本文详细介绍了PolarDB-X 1.0的TPC-H测试设计、测试过程和测试结果。

TPC-H说明

TPC-H是业界常用的一套Benchmark，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。

说明 本文的TPC-H的实现基于TPC-H的基准测试，并不能与已发布的TPC-H基准测试结果相比较，本文中的测试并不符合TPC-H基准测试的所有要求。

测试设计

- 企业版测试环境：PolarDB-X 1.0计算资源实例企业版32C128G（单节点16C64G）、4台RDS MySQL 5.7实例（8C32G独享型）。
- 标准版测试环境：PolarDB-X 1.0计算资源实例标准版16C64G（单节点8C32G）、4台RDS MySQL 5.7实例（4C32G独享型）。

以下测试结果基于50G数据量（Scalar Factor = 50），其中主要表数据量如下：LINEITEM表约3亿行，ORDERS表7500万行，PARSUPP表4000万行。

以Q18为例，包含4张千万到亿级表的Join（含一个子查询Semijoin）和Group-by聚合。在PolarDB-X 1.0计算资源实例上查询执行时间约11秒。

```
select c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice, sum(l_quantity) from CUSTOMER, ORDERS, LINEITEM where o_orderkey in ( select l_orderkey from LINEITEM group by l_orderkey having sum(l_quantity) > 314 ) and c_custkey = o_custkey and o_orderkey = l_orderkey group by c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice order by o_totalprice desc, o_orderdate limit 100;
```

测试过程

说明 以下测试过程依赖了LoadData功能做TPH数据导入，该功能要求内核版本 $\geq 5.4.7-16000638$ 。

1. 准备压力机ECS。这是以下所有操作的基础，准备数据、运行压测等都需在这台ECS上进行。

说明

- 建议选择VPC网络，经典网络有可能遇到RDS某些规格没有库存。请记录VPC的ID和名称，之后所有的实例和数据都部署在这个VPC里面。
- 建议使用最新的Debian或者CentOS镜像，防止编译时缺少依赖库。

2. 创建PolarDB-X 1.0计算资源实例以及相应的RDS实例，注意必须和上一步骤创建的ECS在同一个VPC中。
3. 在PolarDB-X 1.0计算资源实例上创建库和表，注意要指定分库分表方式，建议使用以下表结构：


```

CREATE TABLE `customer` ( `c_custkey` int(11) NOT NULL, `c_name` varchar(25) NOT NULL,
`c_address` varchar(40) NOT NULL, `c_nationkey` int(11) NOT NULL, `c_phone` varchar(15)
NOT NULL, `c_acctbal` decimal(15,2) NOT NULL, `c_mktsegment` varchar(10) NOT NULL, `c_c
omment` varchar(117) NOT NULL, PRIMARY KEY (`c_custkey`) ) ENGINE=InnoDB DEFAULT CHARSE
T=latin1 dbpartition by hash(`c_custkey`) tpartition by hash(`c_custkey`) tpartitions
4; CREATE TABLE `lineitem` ( `l_orderkey` bigint(20) NOT NULL, `l_partkey` int(11) NOT
NULL, `l_suppkey` int(11) NOT NULL, `l_linenum` bigint(20) NOT NULL, `l_quantity` de
cimal(15,2) NOT NULL, `l_extendedprice` decimal(15,2) NOT NULL, `l_discount` decimal(15
,2) NOT NULL, `l_tax` decimal(15,2) NOT NULL, `l_returnflag` varchar(1) NOT NULL, `l_li
nestatus` varchar(1) NOT NULL, `l_shipdate` date NOT NULL, `l_commitdate` date NOT NULL
, `l_receiptdate` date NOT NULL, `l_shipinstruct` varchar(25) NOT NULL, `l_shipmode` va
rchar(10) NOT NULL, `l_comment` varchar(44) NOT NULL, KEY `IDX_LINEITEM_PARTKEY` (`l_pa
rtkey`), PRIMARY KEY (`l_orderkey`,`l_linenum`) ) ENGINE=InnoDB DEFAULT CHARSET=lati
n1 dbpartition by RIGHT_SHIFT(`l_orderkey`,6) tpartition by RIGHT_SHIFT(`l_orderkey`,6
) tpartitions 4; CREATE TABLE `orders` ( `o_orderkey` bigint(20) NOT NULL, `o_custkey`
int(11) NOT NULL, `o_orderstatus` varchar(1) NOT NULL, `o_totalprice` decimal(15,2) NOT
NULL, `o_orderdate` date NOT NULL, `o_orderpriority` varchar(15) NOT NULL, `o_clerk` va
rchar(15) NOT NULL, `o_shippriority` bigint(20) NOT NULL, `o_comment` varchar(79) NOT N
ULL, PRIMARY KEY (`O_ORDERKEY`) ) ENGINE=InnoDB DEFAULT CHARSET=latin1 dbpartition by R
IGHT_SHIFT(`O_ORDERKEY`,6) tpartition by RIGHT_SHIFT(`O_ORDERKEY`,6) tpartitions 4; C
REATE TABLE `part` ( `p_partkey` int(11) NOT NULL, `p_name` varchar(55) NOT NULL, `p_mf
gr` varchar(25) NOT NULL, `p_brand` varchar(10) NOT NULL, `p_type` varchar(25) NOT NULL
, `p_size` int(11) NOT NULL, `p_container` varchar(10) NOT NULL, `p_retailprice` decima
l(15,2) NOT NULL, `p_comment` varchar(23) NOT NULL, PRIMARY KEY (`p_partkey`) ) ENGINE=
InnoDB DEFAULT CHARSET=latin1 dbpartition by hash(`p_partkey`) tpartition by hash(`p_p
artkey`) tpartitions 4; CREATE TABLE `partsupp` ( `ps_partkey` int(11) NOT NULL, `ps_s
uppkey` int(11) NOT NULL, `ps_availqty` int(11) NOT NULL, `ps_supplycost` decimal(15,2)
NOT NULL, `ps_comment` varchar(199) NOT NULL, KEY `IDX_PARTSUPP_SUPPKEY` (`PS_SUPPKEY`
), PRIMARY KEY (`ps_partkey`,`ps_suppkey`) ) ENGINE=InnoDB DEFAULT CHARSET=latin1 dbpart
ition by hash(`ps_partkey`) tpartition by hash(`ps_partkey`) tpartitions 4; CREATE TA
BLE `supplier` ( `s_suppkey` int(11) NOT NULL, `s_name` varchar(25) NOT NULL, `s_addres
s` varchar(40) NOT NULL, `s_nationkey` int(11) NOT NULL, `s_phone` varchar(15) NOT NULL
, `s_acctbal` decimal(15,2) NOT NULL, `s_comment` varchar(101) NOT NULL, PRIMARY KEY (`
s_suppkey`) ) ENGINE=InnoDB DEFAULT CHARSET=latin1 dbpartition by hash(`s_suppkey`) ttp
artition by hash(`s_suppkey`) tpartitions 4; CREATE TABLE `nation` ( `n_nationkey` int
(11) NOT NULL, `n_name` varchar(25) NOT NULL, `n_regionkey` int(11) NOT NULL, `n_commen
t` varchar(152) DEFAULT NULL, PRIMARY KEY (`n_nationkey`) ) ENGINE=InnoDB DEFAULT CHARS
ET=latin1 broadcast; CREATE TABLE `region` ( `r_regionkey` int(11) NOT NULL, `r_name` v
archar(25) NOT NULL, `r_comment` varchar(152) DEFAULT NULL, PRIMARY KEY (`r_regionkey`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 broadcast;

```

4. 准备导入数据到PolarDB-X 1.0计算资源实例。将脚本下载至压力机ECS上，下载链接：[tpchData.zip](#)。

```
#解压tpchData.zip 脚本 unzip tpchData.zip cd tpchData #编辑conf文件 vim param.conf
```

将conf文件中最后一行的连接信息改成真实的连接信息。

```
#!/bin/bash ### remote generating directory export remoteGenDir=./ ### target path expo
rt targetPath=./tpch/tpchRaw export sourcePath=./tpch/tpchRaw ### cores per worker, d
efault value is 1 export coresPerWorker=1 ### threads per worker, default value is 1 ex
port threadsPerWorker=1 export hint="" ###如果需要测试其他规模的数量级，比如100g的话，则数据
库名换成tpch_100g export insertMysql="mysql -hxxxxxxxxx.drds.aliyuncs.com -P3306 -uxxx
-pxxxxxx -Ac --local-infile tpch_50g -e"
```

生成50G的数据。

```
cd workloads #编辑生成各个表数据的文件数量 cp tpch.workload.1.lst tpch.workload.50.lst #进入上层目录
datagen cd datagen #生成数据, 这个步骤只需执行一次, 后续如果有重复准备数据, 可以不再重复执行
sh generateTPCH.sh 50 #将数据载入PolarDB-X 1.0计算资源实例 cd ../loadTpch sh loadTpch.sh 50
```

验证数据正确性。

```
MySQL [tpch_5g]> select (select count(*) from customer) as customer_cnt, -> (select count(*) from lineitem) as lineitem_cnt, -> (select count(*) from nation) as nation_cnt, -> (select count(*) from orders) as order_cnt, -> (select count(*) from part) as part_cnt, -> (select count(*) from partsupp) as partsupp_cnt, -> (select count(*) from region) as region_cnt, -> (select count(*) from supplier) as supplier_cnt;
```

5. 运行TPCH测试前, 使用 ANALYZE 命令采集统计信息。

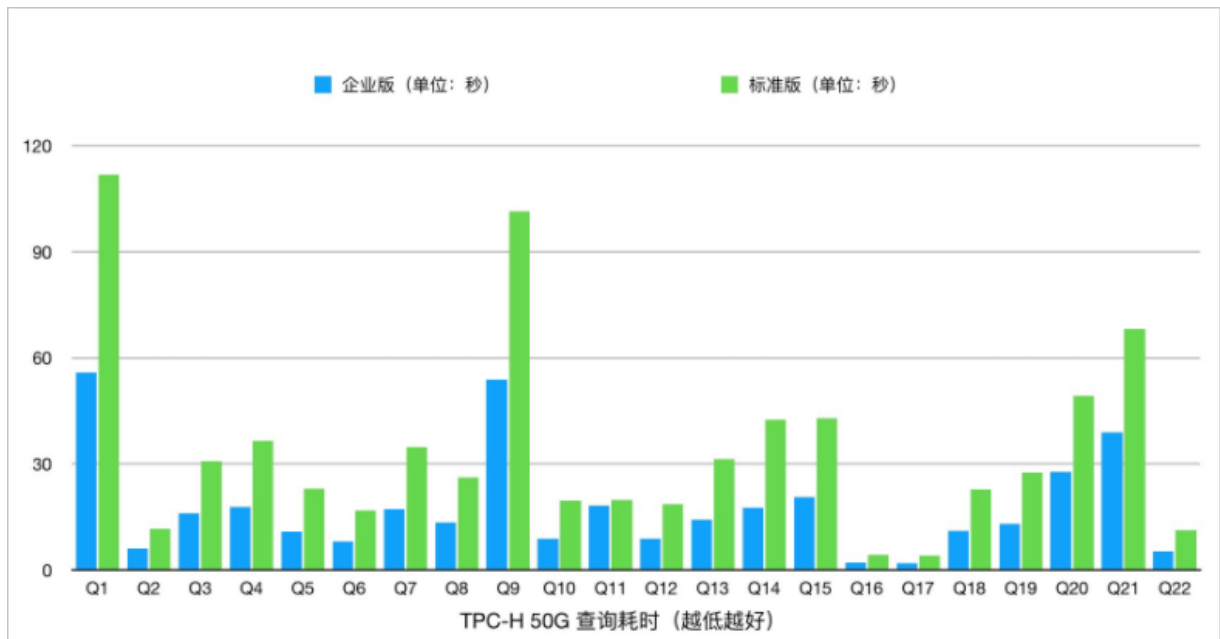
```
analyze table customer; analyze table lineitem; analyze table nation; analyze table orders; analyze table part; analyze table partsupp; analyze table region; analyze table supplier;
```

6. 测试。

```
cd tpchData cd runTpch sh runTpch.sh #运行结束进入result_fixed_mget, 查看各条sql成绩 cd result_fixed_mget
```

测试结果

? 说明 PolarDB-X 1.0计算资源实例入门版不具备Parallel Query能力, 不建议用于执行分析型查询。



Query	企业版 (单位: 秒)	标准版 (单位: 秒)
Q01	55.82	111.84

Query	企业版 (单位: 秒)	标准版 (单位: 秒)
Q02	6.12	11.54
Q03	15.99	30
Q04	17.71	36.56
Q05	10.89	23.01
Q06	8.06	16.76
Q07	17.09	34.80
Q08	13.44	26.09
Q09	53.81	101.51
Q10	8.73	19.67
Q11	18.25	19.74
Q12	8.80	18.60
Q13	14.15	31.33
Q14	17.49	42.43
Q15	20.62	42.79
Q16	2.13	4.15
Q17	1.93	4.07
Q18	11.01	22.82
Q19	12.97	27.61
Q20	27.77	49.25
Q21	38.84	68.08
Q22	5.27	11.29
总计	386.77	754.65

3.Sysbench测试说明

Sysbench是一款开源的、模块化的、跨平台的多线程性能测试工具，可以执行数据库在CPU、内存、线程、IO等方面的性能测试。本文将验证PolarDB-X 1.0在Sysbench OLTP和SELECT场景中的性能表现。

测试设计

- 测试用实例规格如下：
 - PolarDB-X 1.0（4种规格）：
 - 入门版8核32 GB
 - 标准版16核64 GB
 - 企业版32核128 GB
 - 企业版64核256 GB
 - ECS压力机（1台）：32核64 GB，操作系统Aliyun Linux 2.1903 64位，计算网络增强型
 - RDS（12台）：16核64 GB、MySQL 5.7、独享型

 说明 PolarDB-X 1.0、ECS和RDS都处于同一可用区、同一VPC内。

- 在PolarDB-X 1.0控制台按水平拆分模式创建数据库，选择已经购买的12台RDS。
- 在ECS安装Sysbench，并准备1.6亿数据。如何使用Sysbench，请参见[Sysbench 使用指南](#)。

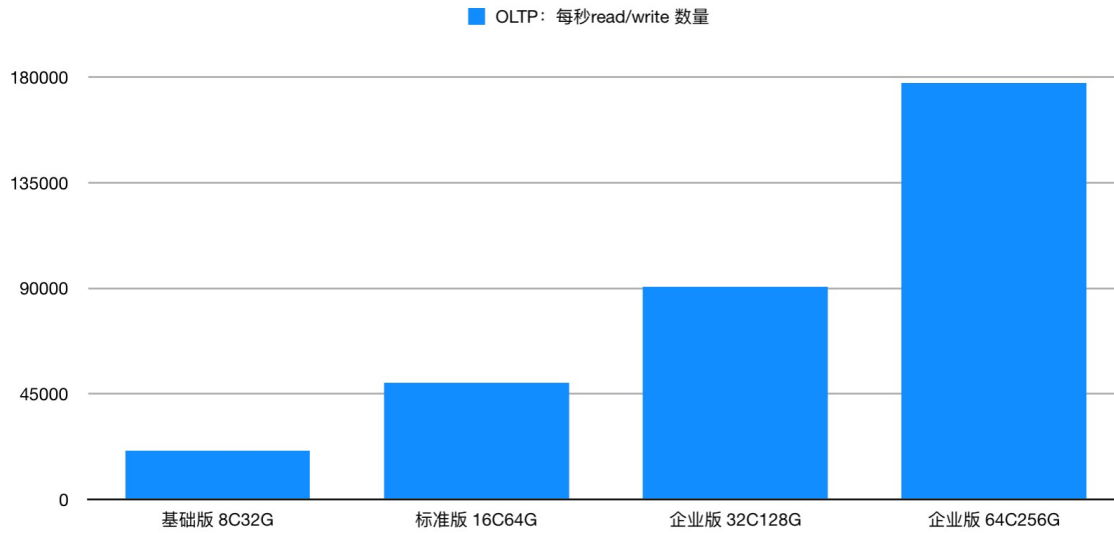
测试参数说明

参数	说明
<code>test='/usr/local/share/sysbench/oltp_drds.lua'</code>	OLTP测试场景数据的路径。
<code>test='/usr/local/share/sysbench/select.lua'</code>	SELECT 场景数据的路径。
<code>mysql_table_options='dbpartition by hash(`id`) tpartition by hash(id) tpartitions 2'</code>	PolarDB-X 1.0分库分表语法，表示每个分库2张分表。
<code>oltp-table-size=160000000</code>	准备1.6亿数据。
<code>oltp_auto_inc=off</code>	关闭自增主键。
<code>oltp_skip_trx=on</code>	开启跳过事务设置。

测试语句范例：

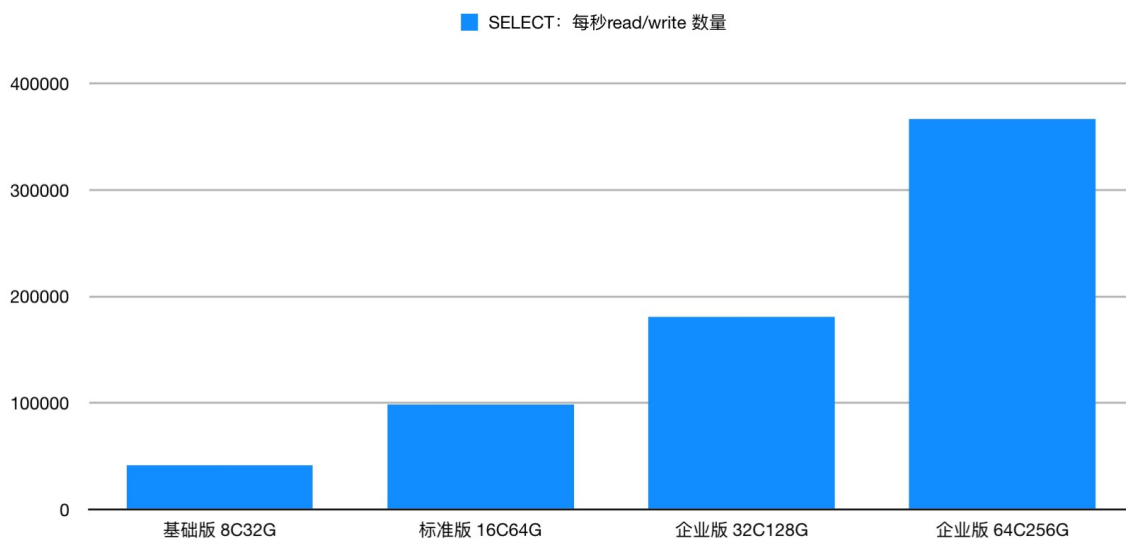
```
sysbench --test='/usr/local/share/sysbench/oltp_drds.lua' --oltp_tables_count=1 --report-interval=5 --oltp-table-size=160000000 --mysql-user=**** --mysql-password=**** --mysql-table-engine=innodb --rand-init=on --mysql-host=**** --mysql-port=3306 --mysql-db=**** --max-requests=0 --oltp_skip_trx=on --oltp_auto_inc=off --oltp_secondary --oltp_range_size=5 --mysql_table_options='dbpartition by hash(`id`) tpartition by hash(id) tpartitions 2' --num-threads=200 --max-time=300 run
```

OLTP测试结果



规格	并发数	每秒 read/write 数量
入门版 8C32G	100	20807.12
标准版 16C64G	230	49667.48
企业版 32C128G	450	90693.70
企业版 64C256G	900	177506.48

SELECT测试结果



规格	并发数	每秒 read/write 数量
入门版 8C32G	200	41401

规格	并发数	每秒 read/write 数量
标准版 16C64G	300	98182.26
企业版 32C128G	600	180500.00
企业版 64C256G	1200	366863.48

OLTP场景测试脚本详情

```

pathstest = string.match(test, "(.*)") if pathstest then dofile(pathstest .. "common.lua") el
se require("common") end function get_range_end(start) return start + oltp_range_size - 1 e
nd function thread_init(thread_id) set_vars() if (((db_driver == "mysql") or (db_driver ==
"attachsql")) and mysql_table_engine == "myisam") then local i local tables = {} for i=1, o
ltp_tables_count do tables[i] = string.format("sbtest%i WRITE", i) end begin_query = "LOCK
TABLES " .. table.concat(tables, " ,") commit_query = "UNLOCK TABLES" else begin_query = "B
EGIN" commit_query = "COMMIT" end end function event(thread_id) local rs local i local tabl
e_name local range_start local c_val local pad_val local query table_name = "sbtest".. sb_r
and_uniform(1, oltp_tables_count) if not oltp_skip_trx then db_query(begin_query) end if no
t oltp_write_only then for i=1, oltp_point_selects do rs = db_query("SELECT c FROM ".. tabl
e_name .." WHERE id=" .. sb_rand(1, oltp_table_size)) end if oltp_range_selects then for i=
1, oltp_simple_ranges do range_start = sb_rand(1, oltp_table_size) rs = db_query("SELECT c
FROM ".. table_name .." WHERE id BETWEEN " .. range_start .. " AND " .. get_range_end(range
_start)) end for i=1, oltp_sum_ranges do range_start = sb_rand(1, oltp_table_size) rs = db_
query("SELECT SUM(K) FROM ".. table_name .." WHERE id BETWEEN " .. range_start .. " AND " .
. get_range_end(range_start)) end for i=1, oltp_order_ranges do range_start = sb_rand(1, ol
tp_table_size) rs = db_query("SELECT c FROM ".. table_name .." WHERE id BETWEEN " .. range
_start .. " AND " .. get_range_end(range_start) .. " ORDER BY c") end for i=1, oltp_distinct
_ranges do range_start = sb_rand(1, oltp_table_size) rs = db_query("SELECT DISTINCT c FROM
".. table_name .." WHERE id BETWEEN " .. range_start .. " AND " .. get_range_end(range_star
t) .. " ORDER BY c") end end end if not oltp_read_only then for i=1, oltp_index_updates do
rs = db_query("UPDATE " .. table_name .. " SET k=k+1 WHERE id=" .. sb_rand(1, oltp_table_si
ze)) end for i=1, oltp_non_index_updates do c_val = sb_rand_str("#####-#####-##
#####-#####-#####-#####-#####-#####-#####-#####-#####-#####-#####-#####
##") query = "UPDATE " .. table_name .. " SET c='" .. c_val .. "' WHERE id=" .. sb_rand(1,
oltp_table_size) rs = db_query(query) if rs then print(query) end end for i=1, oltp_delete_
inserts do i = sb_rand(1, oltp_table_size) rs = db_query("DELETE FROM " .. table_name .. "
WHERE id=" .. i) c_val = sb_rand_str([[ #####-#####-#####-#####-#####-#####-#####
#####-#####-#####-#####-#####-#####-#####]]) pad_val = sb_rand_s
tr([[ #####-#####-#####-#####-#####-#####-#####]]) rs = db_query("INSERT
INTO " .. table_name .. " (id, k, c, pad) VALUES " .. string.format("(%d, %d, '%s', '%s')",
i, sb_rand(1, oltp_table_size) , c_val, pad_val)) end end -- oltp_read_only if not oltp_ski
p_trx then db_query(commit_query) end end

```

4.Sysbench 使用指南

PolarDB-X 1.0性能测试使用Sysbench作为压测工具，本文介绍Sysbench的使用方法。

安装

测试使用的是Sysbench 0.5版本，安装方法如下：

```
git clone https://github.com/akopytov/sysbench.git git checkout 0.5 yum -y install make aut
omake libtool pkgconfig libaio-devel yum -y install mariadb-devel ./autogen.sh ./configure
make -j make install
```

以上是在压测ECS上的安装方法，如果需要安装到其他操作系统，参考Sysbench [官方文档](#)。

安装完毕后，所有自带压测脚本都在 `/usr/local/share/sysbench` 目录下，PolarDB-X 1.0性能测试将使用该目录下的脚本。除此之外，也可以在源码目录 `sysbench/sysbench/tests/db` 下找到对应的压测脚本。

使用简介

常用测试模型

Sysbench通过脚本定义了若干常用的压测模型，以下简单介绍几个常用模型：

压测模型	描述
bulk_insert.lua	批量插入数据
insert.lua	单值插入数据
delete.lua	删除数据
oltp.lua	混合读写测试，读写比例14:4
select.lua	简单的主键查询

表结构

对PolarDB-X 1.0进行测试时，对Sysbench的测试表稍作改造，将其修改为分库分表的形式，测试表的建表语句如下：

```
CREATE TABLE `sbtest1` ( `id` int(10) unsigned NOT NULL, `k` int(10) unsigned NOT NULL DEFA
ULT '0', `c` char(120) NOT NULL DEFAULT '', `pad` char(60) NOT NULL DEFAULT '', KEY `xid` (
`id`), KEY `k_1` (`k`) ) dbpartition by hash(`id`) tpartition by hash(`id`) tpartitions 4
```

以上建表语句中，分库分表数可自行调整，该表会在准备数据阶段自动生成，无需手动创建。

测试命令及参数

使用Sysbench进行压测，通常分为三个步骤：

- prepare: 准备数据；
- run: 运行测试模型；
- cleanup: 清理测试数据。

通常只需准备一次数据，在此数据基础上测试各种模型即可。

常用参数

Sysbench中常用的参数如下：

- `--oltp-tables-count=1` : 表数量。
- `--oltp-table-size=10000000` : 每个表产生的记录行数。
- `--oltp-read-only=off` : 是否生成只读SQL, 默认off, 如果设置为on, 则oltp模型不会生成update, delete, insert的SQL语句。
- `--oltp-skip-trx=[on|off]` : 省略BEGIN/COMMIT语句。默认是off。
- `--rand-init=on` : 是否随机初始化数据, 如果不随机化那么初始好的数据每行内容除了主键不同外其他完全相同。
- `--num-threads=12` : 并发线程数, 可以理解为模拟的客户端并发连接数。
- `--report-interval=10` : 表示每10s输出一次性能数据。
- `--max-requests=0` : 压力测试产生请求的总数, 如果以下面的max-time来记, 这个值设为0即可。
- `--max-time=120` : 测试的持续时间。
- `--oltp_auto_inc=off` : ID是否为自增列。
- `--oltp_secondary=on` : 将ID设置为非主键防止主键冲突。
- `--oltp_range_size=5` : 连续取值5个, 必定走到5个分片。
- `--mysql_table_options='dbpartition by hash(id) tpartition by hash(id) tpartitions 2'` : PolarDB-X 1.0支持拆分表, 在建表的时候需要指定拆分方式。

示例命令

您可以使用如下命令进行对应操作：

准备数据：

```
sysbench --test='/usr/local/share/sysbench/oltp.lua' --oltp_tables_count=1 --report-interval=10 --oltp-table-size=10000000 --mysql-user=*** --mysql-password=*** --mysql-table-engine=innodb --rand-init=on --mysql-host=**** --mysql-port=*** --mysql-db=*** --max-time=300 --max-requests=0 --oltp_skip_trx=on --oltp_auto_inc=off --oltp_secondary=on --oltp_range_size=5 --mysql_table_options='dbpartition by hash(`id`) tpartition by hash(`id`) tpartitions 2' --num-threads=200 prepare
```

执行测试：

```
sysbench --test='/usr/local/share/sysbench/oltp.lua' --oltp_tables_count=1 --report-interval=10 --oltp-table-size=10000000 --mysql-user=*** --mysql-password=*** --mysql-table-engine=innodb --rand-init=on --mysql-host=**** --mysql-port=*** --mysql-db=*** --max-time=300 --max-requests=0 --oltp_skip_trx=on --oltp_auto_inc=off --oltp_secondary --oltp_range_size=5 --mysql_table_options='dbpartition by hash(`id`) tpartition by hash(`id`) tpartitions 2' --num-threads=200 run
```

清理环境：


```
sysbench --test='/usr/local/share/sysbench/oltp.lua' --oltp_tables_count=1 --report-interval=10 --oltp-table-size=10000000 --mysql-user=*** --mysql-password=*** --mysql-table-engine=innodb --rand-init=on --mysql-host=*** --mysql-port=*** --mysql-db=*** --max-time=300 --max-requests=0 --oltp_skip_trx=on --oltp_auto_inc=off --oltp_secondary --oltp_range_size=5 --mysql_table_options='dbpartition by hash(`id`) tpartition by hash(`id`) tpartitions 2' --num-threads=200 cleanup
```