# 背景

让DBA创造更大的价值。

# 概述

## 简介

阿里云自研的云原生关系型数据库PolarDB有三个独立的引擎，分别100%兼容MySQL、100%兼容PostgreSQL、高度兼容Oracle语法，存储容量最高可达100 TB，单库最多可扩展到16个节点，适用于企业多样化的数据库应用场景。

PolarDB既融合了商业数据库稳定可靠、高性能、可扩展的特征，又具有开源云数据库简单开放、自我迭代的优势，例如PolarDB MySQL作为“超级MySQL”，性能最高可以提升至MySQL的6倍，而成本只有商用数据库的1/10，每小时最低只需1.3元即可体验完整的产品功能。

PolarDB MySQL 100%兼容原生MySQL和RDS MySQL，您可以在不修改应用程序任何代码和配置的情况下，将MySQL数据库迁移至PolarDB MySQL。

参考：

<https://help.aliyun.com/document_detail/58764.html?spm=a2c4g.11174283.2.3.20ee6121G2OfxN>

<https://zhuanlan.zhihu.com/p/87742609>

## PolarDB-X

到目前来看PolarDB在架构和性能上都在追求极值。但目前在一些业务开发中，动不动还是能在一个库中搞出来几百张表，SQL编写非常复杂，一个SQL上百，上千行，关联十几张表等情况。在PolarDB 2.0版本引入128个RW节点并行写入，库表写入点秒级切换。对于多节点，并发写入进一步的优化。



PolarDB（这里指PolarDB For MySQL）是一个**基于共享存储技术**的云原生（不是分布式数据库，是分布式存储的关系型数据库）数据库。PolarDB在存储空间上可以做到很强的弹性能力，但一般使用情况下，其计算能力、写入能力依然存在单机的上限。

PolarDB-X 2.0（分布式数据库）这种**share-nothing的架构**，使得包括计算、写入、读取、存储等在内的所有资源，都具备了可水平扩展的能力，因此不会存在单机的瓶颈上限。

但是，share-nothing的架构在单纯的数据容量的弹性上，是不如PolarDB的共享存储架构的。



## 版本

## 专有名词

|  |  |
| --- | --- |
| 名词 | 描述 |
| 地域  （Region） | 数据中心所在的地理位置。 |
| 可用区  （Zone） | 可用区是指在某一地域内，具有独立电力和网络的物理区域。同一可用区内实例之间的网络延时更小。 |
| 集群  （Cluster） | PolarDB采用多节点集群的架构，集群中有一个主节点和多个只读节点。单个PolarDB集群支持跨可用区，但不能跨地域。 |
| 全球数据库网络  （Global Database Network） | 全球数据库网络（GDN）是由分布在全球不同地域的多个PolarDB数据库集群组成的一张网络。网络中所有集群的数据保持同步，完全一致。更多详情，请参见[创建和释放全球数据库网络](https://help.aliyun.com/document_detail/160381.htm" \l "task-2449685" \o "全球数据库网络（Global Database Network，简称GDN）是由分布在全球不同地域的多个集群组成的网络，网络中所有集群的数据保持同步。当您的业务部署在多个地域时，实现应用访问数据库的低延迟和高稳定性。本文介绍如何新建和释放全球数据库网络。)。 |
| 主集群  （Primary Cluster） | 全球数据库网络（GDN）中只有一个集群拥有读写权限，这个可读可写的集群叫做主集群。 |
| 从集群  （Secondary Cluster） | 全球数据库网络（GDN）中从主集群同步数据的从属集群。 |
| 节点  （Node） | PolarDB集群由多个物理节点构成，每个集群中的节点可分为两类，每一类节点关系对等，规格相同。这两类节点分别叫主节点和只读节点。 |
| 主节点  （Primary node） | PolarDB主节点，也叫读写节点，一个集群中只有一个主节点。 |
| 只读节点  （Read-only node） | PolarDB只读节点，一个集群中最多可添加至15个只读节点。 |
| 集群可用区  （Cluster zone） | 集群数据分布的可用区。集群的数据会自动在两个可用区间实现冗余，用于灾备恢复，节点迁移只支持在这两个可用区间进行。 |
| 主可用区  （Primary zone） | PolarDB的主节点所在可用区。 |
| 故障切换  （Failover） | 故障切换（也称主备切换）是指提升一个只读节点为主节点。更多详情，请参见[主备切换](https://help.aliyun.com/document_detail/141229.htm" \l "task-2317735" \o "是一个多节点的集群，其中一个节点是主节点，其他节点为只读节点。除了因系统故障自动进行主备切换外，您也可以手动进行主备切换，指定一个只读节点为新的主节点，适用于高可用演练，或者需要指定某个节点为主节点的场景。)。 |
| 规格  （Class） | 集群规格。PolarDB每个节点的资源配置，例如8核64 GB，更多规格请参见[计算节点规格](https://help.aliyun.com/document_detail/102542.htm" \l "concept-2035312" \o "目前支持、和三种产品系列，每种产品系列所支持的计算节点规格不同。)。 |
| 访问点  （Endpoint） | 访问点定义了数据库的访问入口，访问点也称为接入点。每个集群都提供了多个访问点，每个访问点可以连接1个或多个节点。例如，主访问点永远指向主节点，集群访问点提供了读写分离能力，连接了主节点和多个只读节点。访问点中包含的主要是数据库链路属性，例如读写状态、节点列表、负载均衡、一致性级别等。 |
| 访问地址  （Address） | 访问地址是访问点在不同网络平面中的载体，一个访问点可能包含私网和公网两种访问地址。访问地址中包含了一些网络属性，例如域（Domain）、IP地址、专有网络 （VPC）、交换机（VSwitch）等。 |
| 主地址  （Primary Endpoint） | 主节点的访问点，当发生故障切换（Failover）后，系统会将访问点自动指向新的主节点。 |
| 集群地址  （ Cluster Endpoint） | 整合集群下的多个节点，对外提供一个统一的读写地址，可以设置为只读或读写。集群地址具有自动弹性、读写分离、负载均衡、一致性协调等能力。更多详情，请参见[新增自定义集群地址](https://help.aliyun.com/document_detail/164564.htm" \l "task-2491346" \o "您可以在集群上新增自定义集群地址，通过设置集群地址的读写模式、一致性级别及关联的只读节点等，来满足不同的业务场景，增强业务的灵活性。 本文将介绍如何为新增自定义集群地址。)。 |
| 最终一致性  （Eventual Consistency） | 只读模式下默认选项为最终一致性。最终一致性下PolarDB集群将提供最优的性能。更多详情，请参见[最终一致性](https://help.aliyun.com/document_detail/99093.htm" \l "section-xgc-wzr-aca" \o ")。 |
| 会话一致性  （Session Consistency） | 会话一致性也叫因果一致性，读写模式下的默认选项，提供Session级的读一致性保证，可以满足大部分应用场景。更多详情，请参见[会话一致性](https://help.aliyun.com/document_detail/99093.htm" \l "section-lnv-grf-2gb" \o ")。 |
| 全局一致性  （Global Consistency） | 全局一致性也叫强一致性，跨会话一致性，最高级别的一致性，可以保证跨Session的会话一致性，但会增加主库的负载，当复制延迟高时不适用。更多详情，请参见[全局一致性](https://help.aliyun.com/document_detail/99093.htm" \l "section-ctf-0ez-syo" \o ")。 |
| 事务拆分  （Distributed Transaction） | 集群地址的一个配置项。在保证一致性的前提下，通过拆分一个事务（Transaction）内的读请求（Select Queries）到只读节点，可以在一定程度上降低主节点的负载。更多详情，请参见[功能特性](https://help.aliyun.com/document_detail/85143.htm" \l "section-268-4y3-g1u" \o ")。 |
| 主库不接受读  （Offload Reads from Primary Node） | 集群地址的一个配置项。在确保一致性的前提下，将查询SQL发送到只读节点，来降低主节点的负载，确保主节点稳定。更多详情，请参见[功能特性](https://help.aliyun.com/document_detail/85143.htm" \l "section-268-4y3-g1u" \o ")。 |
| 私有域名  （Private Address） | 为了保留用户原来数据库的连接地址（域名），PolarDB联手PrivateZone，保证PolarDB主地址和集群地址中的每一个内网地址，均可以绑定一个私有域名。该私有域名仅在当前地域内指定的VPC中生效。更多详情，请参见[私有域名](https://help.aliyun.com/document_detail/162306.htm" \l "task-2473542" \o "如果您之前是通过域名连接到数据库，当数据库迁移上云后，想要保留原来的数据库域名，可以通过私有域名功能进行绑定。)。 |
| 快照备份  （Snapshot Backup） | PolarDB数据的备份方式，目前仅支持快照备份。更多详情，请参见[备份数据](https://help.aliyun.com/document_detail/72672.htm" \l "task-1580301" \o "可靠的备份功能可以有效防止数据丢失，支持周期性的自动备份以及即时生效的手动备份。在删除集群时，您还可以选择保留备份数据，避免误操作导致的数据丢失。)。 |
| 一级备份  （Level-1 Backup） | 保存在本地的备份文件叫一级备份。一级备份直接存储在分布式存储集群中，备份和恢复速度最快，但成本高。更多详情，请参见[备份数据](https://help.aliyun.com/document_detail/72672.htm" \l "task-1580301" \o "可靠的备份功能可以有效防止数据丢失，支持周期性的自动备份以及即时生效的手动备份。在删除集群时，您还可以选择保留备份数据，避免误操作导致的数据丢失。)。 |
| 二级备份  （Level-2 Backup） | 保存在其他离线存储介质中的备份文件叫二级备份。二级备份的数据全部来自于一级备份，可以永久保存，成本低但恢复速度较慢。更多详情，请参见[备份数据](https://help.aliyun.com/document_detail/72672.htm" \l "task-1580301" \o "可靠的备份功能可以有效防止数据丢失，支持周期性的自动备份以及即时生效的手动备份。在删除集群时，您还可以选择保留备份数据，避免误操作导致的数据丢失。)。 |
| 日志备份  （Log Backup） | 日志备份是把数据库的Redo日志保存下来，用于按时间点恢复，为了保证最近一段时间数据的安全性，避免误操作导致的数据丢失，因此日志备份最少保留7天。日志备份采用离线存储备份，成本较低。更多详情，请参见[备份数据](https://help.aliyun.com/document_detail/72672.htm" \l "task-1580301" \o "可靠的备份功能可以有效防止数据丢失，支持周期性的自动备份以及即时生效的手动备份。在删除集群时，您还可以选择保留备份数据，避免误操作导致的数据丢失。)。 |
| 计算包  （Compute Plan） | 计算包是PolarDB推出的一款降低集群计算成本的资源包。计算包支持与按量付费集群配合使用。购买计算包后，按量付费集群中的计算节点费用将不再按小时计费，而是由计算包自动抵扣。更多详情，请参见[购买计算包](https://help.aliyun.com/document_detail/183468.htm" \l "task-2035299" \o "本文介绍如何购买计算包。)。 |
| 存储包  （Storage Plan） | 存储包PolarDB推出的一款降低集群存储成本的资源包。PolarDB的存储空间可根据数据量自动伸缩，无需您手动配置，您只需为实际使用的数据量付费。当您的数据量较大时，推荐使用PolarDB存储包以降低存储成本。更多详情，请参见[购买存储包](https://help.aliyun.com/document_detail/127290.htm" \l "task-2038073" \o "本文介绍如何购买存储包。)。 |
| 集群版  （Cluster Edition） | 集群版是PolarDB MySQL中的推荐产品系列，使用了计算与存储分离的架构，计算层的数据库节点可以从2个动态扩展到最多16个。更多详情，请参见[集群版](https://help.aliyun.com/document_detail/183258.htm" \l "section-bnr-2pe-9ga" \o ")。 |
| 单节点  （Single Node） | 单节点是PolarDB MySQL中的入门级产品系列，采用了突发性能型规格，共享计算资源池提升资源利用率，单节点的架构无需Proxy代理节省资源成本。更多详情，请参见[单节点](https://help.aliyun.com/document_detail/183258.htm" \l "section-cjt-e43-12q" \o ")。 |
| 历史库  （Archive Database） | 历史库是PolarDB MySQL中具有较高数据压缩率的产品系列，使用了X-Engine作为默认存储引擎，提供了超大存储容量，满足了归档数据库低存储成本的要求。更多详情，请参见[历史库](https://help.aliyun.com/document_detail/195661.htm" \l "concept-2010571" \o "本文介绍的产品优势、架构及适用场景。)。 |

## 特点

主要技术特点：

1. shared disk架构，存储服务内部，实际上数据被切块成chunk来达到通过多个服务器并发访问I/O的目的
2. 物理replication，物理层面实现redo log replication，可以关闭binlog提升性能；数据共享，只读节点的增加无需再进行数据的安全复制，只需要同步元数据信息，支持基本的MVCC，保证数据读一致性
3. 高速网络下RDMA协议，zero-copy的技术以达到数据在NIC和远端应用内存之间高效低延迟传递，而不用通过中断CPU的方式来进行数据从内核态到应用态的拷贝，极大的降低了性能的抖动。提高了整体系统的处理能力
4. Snapshot物理备份，分布式存储特性
5. parallel-raft算法，存储层面数据一致性
6. 用户态文件系统、docker，性能提升

您可以像使用MySQL、PostgreSQL、Oracle一样使用PolarDB。此外，PolarDB还有传统数据库不具备的优势：

### 大容量

最高100 TB，您不再需要因为单机容量的天花板而去购买多个实例做分片，由此简化应用开发，降低运维负担。

### 低成本

共享存储：计算与存储分离，每增加一个只读节点只收取计算资源的费用，而传统的只读节点同时包含计算和存储资源，每增加一个只读节点需要支付相应的存储费用。

弹性存储：存储空间无需配置，根据数据量自动伸缩，您只需为实际使用的数据量按小时付费。

[存储包](https://help.aliyun.com/document_detail/127290.htm" \l "concept-1375373" \o ")：PolarDB推出了预付费形式的存储包。当您的数据量较大时，推荐您使用存储包，相比按小时付费，预付费购买存储包有折扣，购买的容量越大，折扣力度越大。

[计算包](https://help.aliyun.com/document_detail/183468.htm" \l "task-1949954" \o ")：PolarDB首创计算包，用于抵扣计算节点的费用，计算包兼顾了包年包月付费方式的经济性和按量付费方式的灵活性。您还可以将计算包与[自动扩缩容](https://help.aliyun.com/document_detail/166257.htm" \l "task-2499309" \o "的一键诊断融合了DAS部分功能，您可以通过其中的自治中心开启自治服务。自治服务开启后，DAS会在数据库出现异常时，自动进行根因分析，给出优化或止损建议，并自动进行优化或止损操作（需经您授权后才会开启优化操作）。)配合使用，在业务峰值前后实现自动弹性升降配，轻松应对业务量波动。

### 高性能

大幅提升OLTP性能，支持超过50万次/秒的读请求以及超过15万次/秒的写请求。

### 分钟级扩缩容

存储与计算分离的架构，配合容器虚拟化技术和共享存储，增减节点只需5分钟。存储容量自动在线扩容，无需中断业务。

### 读一致性

集群地址利用LSN（Log Sequence Number）确保读取数据时的全局一致性，避免因为主备延迟引起的不一致。

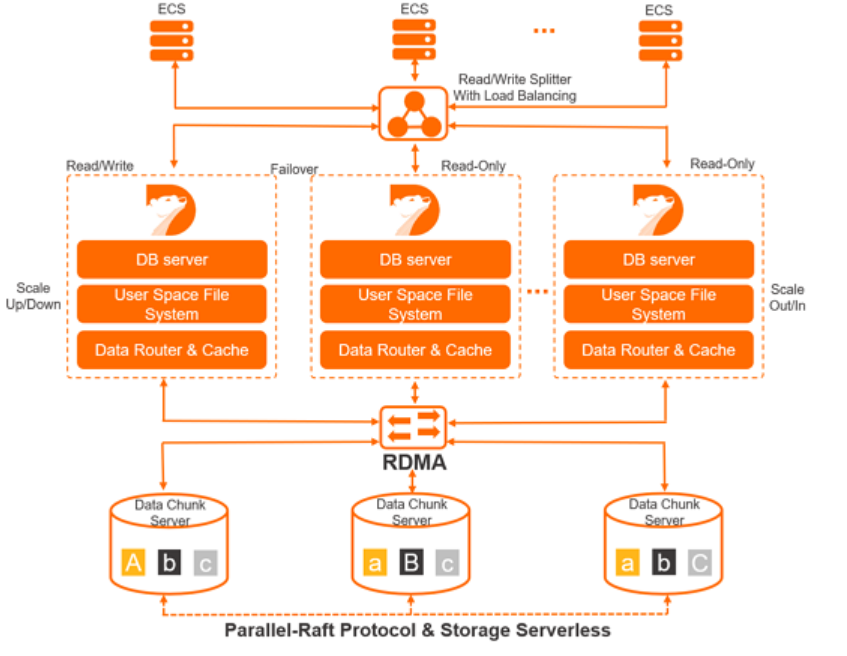
### 毫秒级延迟（物理复制）

利用基于Redo的物理复制代替基于Binlog的逻辑复制，提升主备复制的效率和稳定性。即使对大表进行加索引、加字段等DDL操作，也不会造成数据库的延迟。

### 秒级快速备份

不论多大的数据量，全库备份只需30秒，而且备份过程不会对数据库加锁，对应用程序几乎无影响，全天24小时均可进行备份。

# 架构



PolarDB采用存储和计算分离的架构，所有计算节点共享一份数据，提供分钟级的配置升降级、秒级的故障恢复、全局数据一致性和免费的数据备份容灾服务。

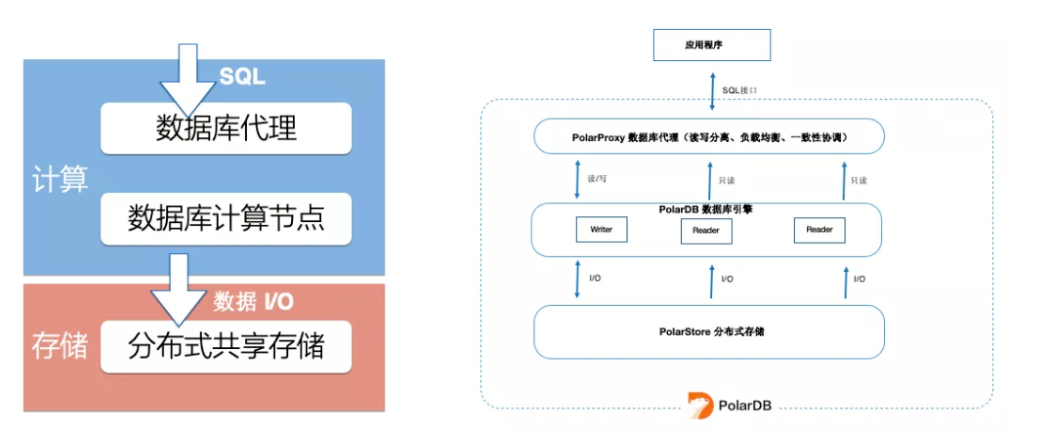
**计算与存储分离，共享分布式存储。**

采用计算与存储分离的设计理念，满足业务弹性扩展的需求。各计算节点通过分布式文件系统（PolarFileSystem）共享底层的存储（PolarStore），极大降低了用户的存储成本。

**一写多读，读写分离。**

PolarDB集群版采用多节点集群的架构，集群中有一个主节点（可读可写）和至少一个只读节点。当应用程序使用集群地址时，PolarDB通过内部的代理层（PolarProxy）对外提供服务，应用程序的请求都先经过代理，然后才访问到数据库节点。代理层不仅可以做安全认证和保护，还可以解析SQL，把写操作发送到主节点，把读操作均衡地分发到多个只读节点，实现自动的读写分离。对于应用程序来说，就像使用一个单点的数据库一样简单。

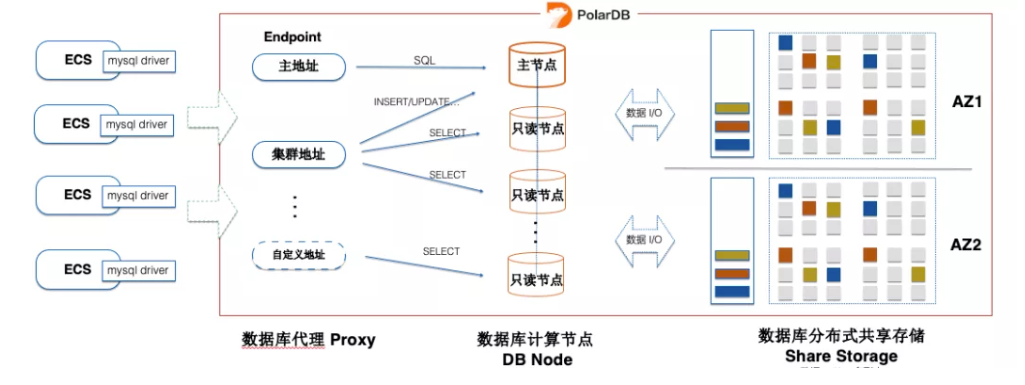
## 整体架构



PolarDB整体架构上也可以分为三个部分：Proxy层（PolarDB的接入层）， PolarDB计算层（支持MySQL，PostgreSQL，Oracle), PolarDB存储层。PolarDB的本身的生态非常丰富包含：

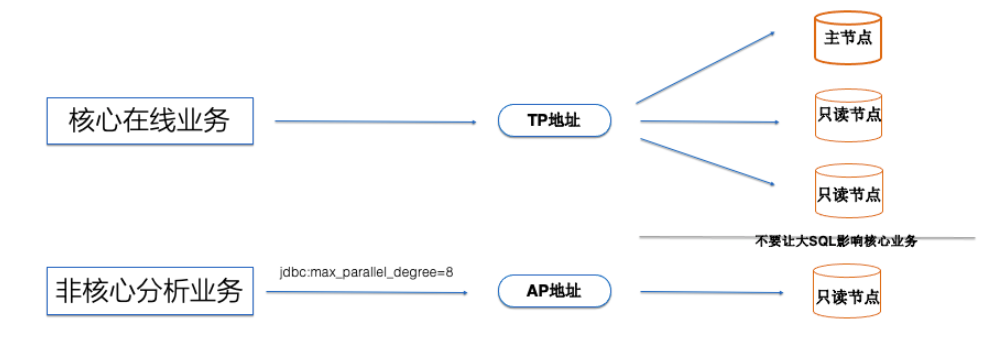


PolarDB MySQL集群整体架构：



### PolarDB Proxy

连接地址，SQL转发，保证事务的一致性，无持久化存储东西。100% MySQL协兼容，多个连接地址，可以起到业务资源隔离，可以一个Proxy访问后面多个计算节点，实现读写分离，多个节点自动负载均衡，应用访问的一致性保障。



通过Proxy层把不同的业务用不同的连接地址，使用不同的数据节点，避免相互影响。并行查询可以在客户端对指定的SQL开启，不会影响核心业务。

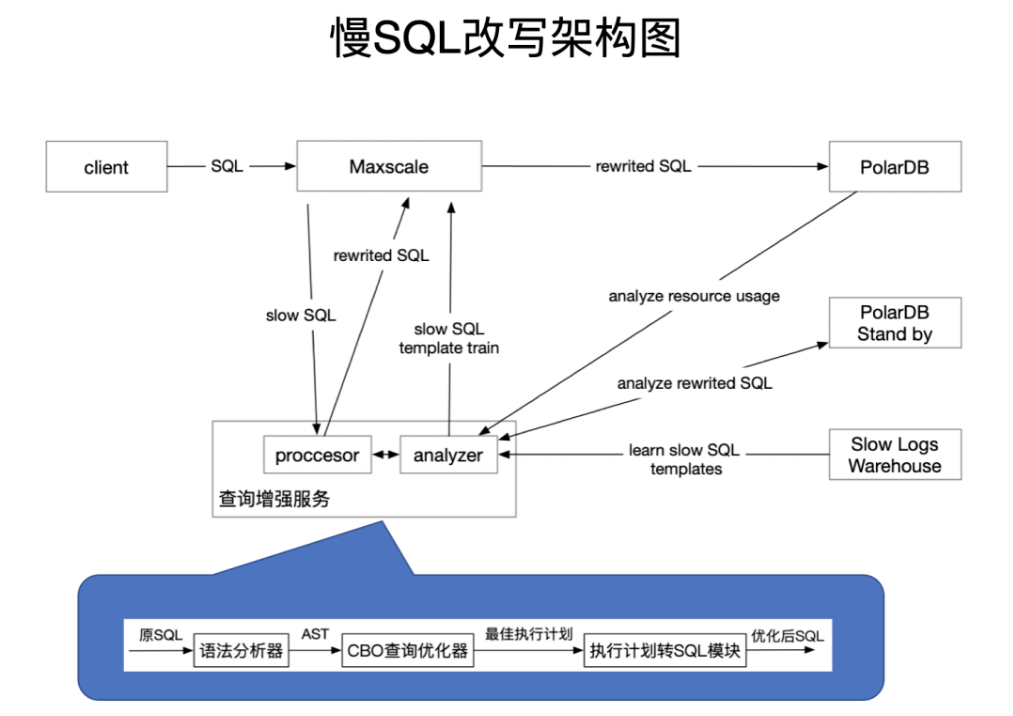
### PolarDB计算节点

在该节点支持MySQL，Pg，Oracle相关语法，无持久化存储数据。 该节点主要用于：SQL解析，事务执行及保证，Query Cache功能等。该节点约束：1个写节点Primary，1-15个只读节点Secondary。所有的节点数据共享，节省成本，物理日志复制，主备延迟<1秒，支持ParallelQuery，充分利用多核CPU的优势，主备切换数据0丢失。其中Primary节点也可以直接对外提供服务。

因为计算机节点无数据持久化，所以支持快速扩容。添加一个只读节点大概在5分钟内快速创建并提供服务。如果开启“自动扩容”，PolarDB会根据系统的负载高低，自动扩容或缩容，从而可以让用户更经济地使用PolarDB。

PolarDB在计算层支持Fast Query Cache在8核64G配置4G的Cache内存查询性能可以提升50%以上。

在事务这块产生的Redo日志均实时写入共享存储，只读节点，利用存储上的数据文件和Redo日志，在内存中恢复最新的数据，对外提供服务。每一次主备故障切换，只读节点均可以从存储中获取最新的事务Redo日志，因此不会出现数据丢现象(RPO=0)在计算节点这块PolarDB还有对应的MaxScale可以实现对慢SQL的自动改写及优化，从而自动提高SQL的执行效率。

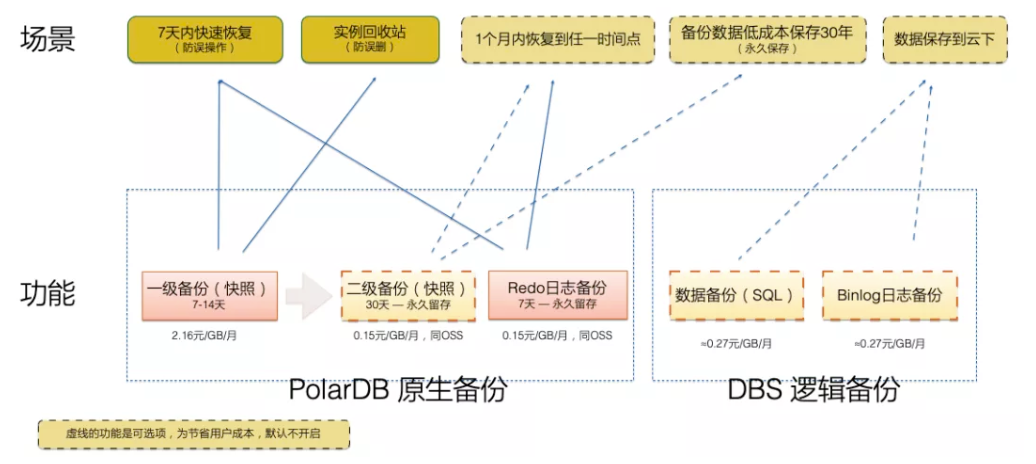


### PolarDB存储层

单可用区3副本，双可用区6副本，ParallelRaft共识算法实现多副本强一致，秒级备份，且无需对数据库加锁，超低的延迟网络IO，利用25Gb的RDMA网络， Bypass Kerner的用用态协议支持。利用RDMA可以避免以太网的性能压力。

在存储层可以实现快速备份：不超过30秒（与容量无关），支持高频备份每个2个小时做一个备份，分钟级恢复：从全量备份到恢复，不超过10分钟。从全备份中可以部分恢复。

在备份中使用了Redirect-on-Write的方式，每次打快照并没真正的Copy数据，只有当数据块后期有修改了(write)时才把历史版本给保留给Snapshot，然后生成新的数据块，被原数据引用(Redirect)，因此，不管数据库的容量是10G还是10TB，每次备备份（快照）均不超过30秒。但需要注意历史快照管理，以防止无限增大。



## [一写多读](https://help.aliyun.com/document_detail/58766.html?spm=a2c4g.11186623.6.551.302f28f023C7NX" \l "title-lto-ilw-pza)

PolarDB采用分布式集群架构，一个集群版集群包含一个主节点和最多15个只读节点（至少一个，用于保障高可用）。主节点处理读写请求，只读节点仅处理读请求。主节点和只读节点之间采用Active-Active的Failover方式，提供数据库的高可用服务。

## [计算与存储分离](https://help.aliyun.com/document_detail/58766.html?spm=a2c4g.11186623.6.551.302f28f023C7NX" \l "title-lqa-rv3-6b7)

PolarDB采用计算与存储分离的设计理念，满足公共云计算环境下根据业务发展弹性扩展集群的刚性需求。**数据库的计算节点（Database Engine Server）仅存储元数据，而将数据文件、Redo Log等存储于远端的存储节点（Database Storage Server）**。各计算节点之间仅需同步Redo Log相关的元数据信息，极大降低了主节点和只读节点间的复制延迟，而且在主节点故障时，只读节点可以快速切换为主节点。

## [读写分离](https://help.aliyun.com/document_detail/58766.html?spm=a2c4g.11186623.6.551.302f28f023C7NX" \l "title-bux-448-r6i)

读写分离是PolarDB集群版默认免费提供的一个透明、高可用、自适应的负载均衡能力。通过集群地址，SQL请求自动转发到PolarDB集群版的各个节点，提供聚合、高吞吐的并发SQL处理能力。

## [高速链路互联](https://help.aliyun.com/document_detail/58766.html?spm=a2c4g.11186623.6.551.302f28f023C7NX" \l "title-wgw-53q-ibv)

数据库的计算节点和存储节点之间采用高速网络互联，并通过RDMA协议进行数据传输，使I/O性能不再成为瓶颈。

## [共享分布式存储](https://help.aliyun.com/document_detail/58766.html?spm=a2c4g.11186623.6.551.302f28f023C7NX" \l "title-zf1-pot-scg)

多个计算节点共享一份数据，而不是每个计算节点都存储一份数据，极大降低了用户的存储成本。基于全新打造的分布式块存储（Distributed Storage）和文件系统（Distributed Filesystem），存储容量可以在线平滑扩展，不会受到单个数据库服务器的存储容量限制，可应对上百TB级别的数据规模。

## [数据多副本、Parallel-Raft协议](https://help.aliyun.com/document_detail/58766.html?spm=a2c4g.11186623.6.551.302f28f023C7NX" \l "title-9td-5cw-6x4)

数据库存储节点的数据采用多副本形式，确保数据的可靠性，并通过Parallel-Raft协议保证数据的一致性。

# 原理

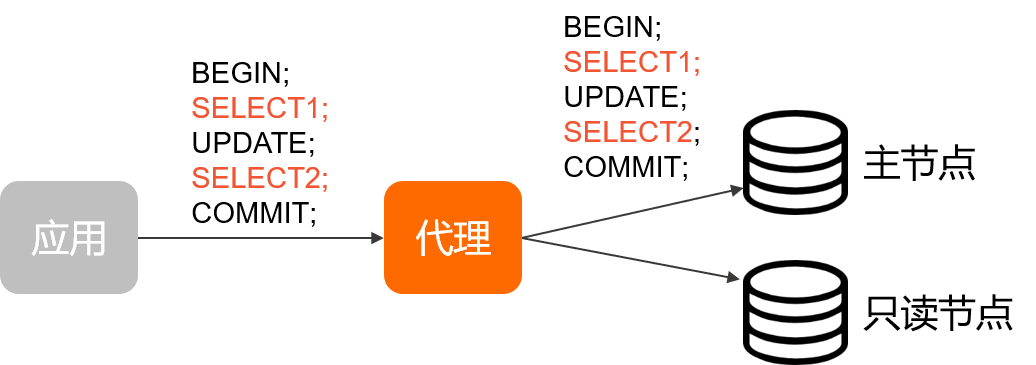
# 操作

# 分布式事务

PolarDB提供了事务拆分功能，旨在保证读写一致性的前提下，将事务中的读请求发送到只读节点，减轻主节点的压力。

## 背景信息

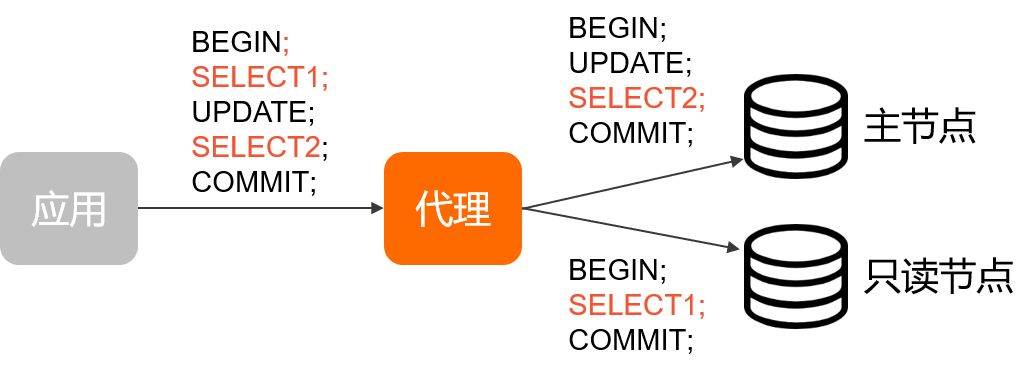
当使用PolarDB可读可写模式集群地址时，读写请求会由数据库代理（Proxy）分发到主节点和只读节点。为了保证一个会话连接中事务读写一致性，代理会将这个会话中所有在事务中的请求都发送到主节点。例如，某些数据库客户端驱动（如JDBC）默认将请求封装在事务中，因此应用的请求都会被发送到主节点，导致主节点压力大，而只读节点几乎没有压力，如下图所示。

[](https://static-aliyun-doc.oss-accelerate.aliyuncs.com/assets/img/zh-CN/6582281161/p232628.png)

为了解决上述问题，PolarDB提供了事务拆分功能，旨在保证业务中读写一致性的前提下，将事务中读请求发送到只读节点，以减轻主节点的压力。

## 功能介绍

Proxy会将事务中第一个写请求前的读请求发送到只读节点，从而减轻主节点的负载，由于事务中未提交（COMMIT）的数据在只读节点上处于不可见的状态，为了保障事务中读写一致性，第一个写请求后的所有读写请求仍路由到主节点。

[](https://static-aliyun-doc.oss-accelerate.aliyuncs.com/assets/img/zh-CN/6582281161/p232629.png)

## 功能优势

不需要改动应用的代码或配置就可以将事务中的读压力从主节点转移到只读节点，从而提高主节点的稳定性。

## 注意事项

仅支持对读已提交（Read Committed）事务隔离级别的事务拆分。

对于事务拆分基础服务，如果一致性级别不是最终一致性，只有只读节点与主节点数据同步成功后，事务中第一次写请求前的读请求才会发送到只读节点，否则依旧发送到主节点。

# 数据分布

不存在分布式数据库那样的分片键（PolarDB-X是存在分片键的），就是单机MySQL的建表语句，然后底层数据采用Raft算法打散后存储在分布式节点上。

# 复制/一致性

## 复制

数据库存储节点的数据采用多副本形式，确保数据的可靠性，并通过Parallel-Raft协议保证数据的一致性。

## 一致性

PolarDB提供了三种一致性级别：最终一致性、会话一致性和全局一致性，满足您在不同场景下对一致性级别的要求。

### 问题与解决方案

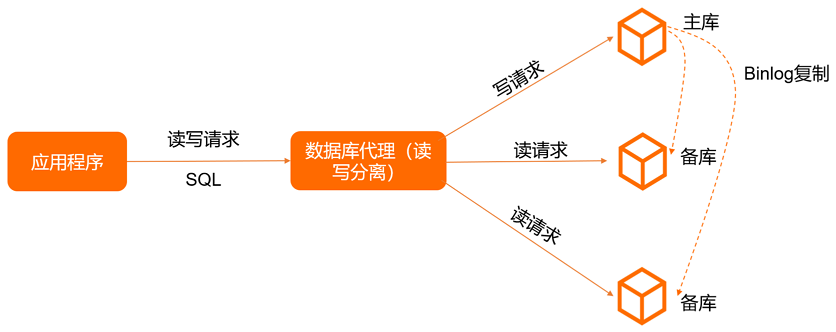
MySQL的主从复制功能，能够将主库的Binlog异步传输到备库并在备库中实时应用，这样既实现了备库可查询，减轻主库的压力，又保证了高可用。

虽然实现了备库可查询，但存在如下问题：

主库和备库一般提供两个不同的访问地址，在访问不同库时，需要在应用程序上修改成对应库的地址，对应用有侵入。

MySQL的复制是异步的，因此备库的数据并不是最新的而是有延迟的，无法保证查询的一致性。

为了解决第一个问题，MySQL引入了读写分离代理功能。一般的实现是，代理会伪造成MySQL与应用程序建立好连接，解析发送进来的每一条SQL，如果是UPDATE、DELETE、INSERT、CREATE等写操作则直接发往主库，如果是SELECT则发送到备库。

[](https://static-aliyun-doc.oss-accelerate.aliyuncs.com/assets/img/zh-CN/1816204061/p34631.png)

但读写分离还是无法解决由于延迟导致的查询不一致问题。当数据库负载很高时，例如对大表执行DDL（如加字段）操作或大批量插入数据的时候，延迟会非常严重，从而导致无法从只读节点中读取最新数据。

PolarDB采用了异步物理复制方式实现了主节点和只读节点间的数据同步。主节点的数据更新后，相关的更新会应用到只读节点，具体的延迟时间与写入压力有关（一般在毫秒级别），通过异步复制的方式确保了主节点和只读节点间数据的最终一致。PolarDB提供了如下三种一致性级别，满足您在不同场景下对一致性级别的要求：

[最终一致性](https://help.aliyun.com/document_detail/99093.html?spm=a2c4g.11186623.6.647.10e51f03eFKmaU" \l "section-xgc-wzr-aca" \o ")

[会话一致性](https://help.aliyun.com/document_detail/99093.html?spm=a2c4g.11186623.6.647.10e51f03eFKmaU" \l "section-lnv-grf-2gb" \o ")

[全局一致性](https://help.aliyun.com/document_detail/99093.html?spm=a2c4g.11186623.6.647.10e51f03eFKmaU" \l "section-ctf-0ez-syo" \o ")

### 最终一致性

#### 功能介绍

PolarDB是读写分离的架构，传统的读写分离只提供最终一致性的保证，主从复制延迟会导致从不同节点查询到的结果不同，例如在一个会话内连续执行如下查询，最后的SELECT结果可能会不同（具体的访问结果由主从复制的延迟决定）。

INSERT INTO t1(id, price) VALUES(111, 96);UPDATE t1 SET price = 100 WHERE id=111;SELECT price FROM t1;

#### 适用场景

若需要减轻主节点压力，让尽量多的读请求路由到只读节点，您可以选择最终一致性。

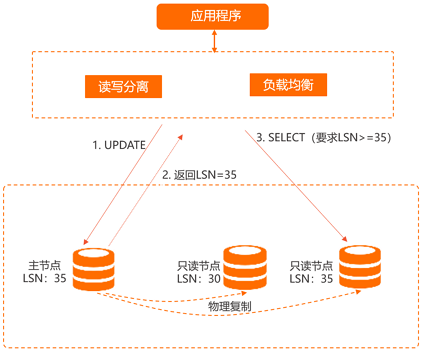
### 会话一致性

#### 功能介绍

针对最终一致性导致查询结果不同的问题，通常需要将业务进行拆分，将一致性要求高的请求直接发往主节点，而可以接受最终一致性的请求则通过读写分离发往只读节点。这既增加了主节点的压力，影响读写分离的效果，又增加了应用开发的负担。

为解决上述问题，PolarDB提供了会话一致性（也称因果一致性）。会话一致性保证了同一个会话内，一定能够查询到读请求执行前已更新的数据，确保了数据单调性。

在PolarDB的链路中间层做读写分离的同时，中间层会追踪各个节点已经应用的Redo日志位点，即日志序号（Log Sequence Number，简称LSN）。同时每次数据更新时PolarDB会记录此次更新的位点为Session LSN。当有新请求到来时，PolarDB会比较Session LSN和当前各个节点的LSN，仅将请求发往LSN大于或等于Session LSN的节点，从而保证了会话一致性。表面上看该方案可能导致主节点压力大，但是因为PolarDB是物理复制，速度极快。

[](https://static-aliyun-doc.oss-accelerate.aliyuncs.com/assets/img/zh-CN/0200685061/p34632.png)

在上述场景中，当更新完成后，返回客户端结果时复制就同步在进行，而当下一个读请求到来时，主节点和只读节点之间的数据复制极有可能已经完成。且大多数应用场景都是读多写少，所以经验证在该机制下既保证了会话一致性，又保证了读写分离负载均衡的效果。

#### 适用场景

PolarDB的一致性级别越高，对主库的压力越大，集群性能也越低。推荐使用会话一致性，该级别对性能影响很小而且能满足绝大多数应用场景的需求。

### 全局一致性

#### 功能介绍

在部分应用场景中，除了会话内部有逻辑上的因果依赖关系，会话之间也存在依赖关系，例如在使用连接池的场景下，同一个线程的请求有可能通过不同连接发送出去。对数据库来说这些请求属于不同会话，但是业务逻辑上这些请求有前后依赖关系，此时会话一致性便无法保证查询结果的一致性。因此PolarDB提供了全局一致性来解决该问题。

[](https://static-aliyun-doc.oss-accelerate.aliyuncs.com/assets/img/zh-CN/0200685061/p88549.png)

每个读请求到达PolarDB数据库代理时，代理都会先去主节点确认当前最新的LSN位点，假设为LSN0（为了减少每次读请求都去获得主节点的最新LSN，内部做了批量优化）， 然后等待所有只读节点的LSN都更新至主节点的LSN0位点后，代理再将读请求发送至只读节点。这样就能保证该读请求能够读到至请求发起时刻为止，任意一条已完成更新的数据。

全局一致性提供如下两个配置参数：

|  |  |
| --- | --- |
| 参数 | 说明 |
| ConsistTimeout | 全局一致性读超时时间，即允许用于只读节点的LSN更新至主节点最新LSN的时间。若超出该时间，PolarDB代理将根据ConsistTimeoutAction参数设置进行对应操作。  取值范围0~300000，默认值为20，单位为毫秒。 |
| ConsistTimeoutAction | 全局一致性读超时策略，若未能在参数ConsistTimeout设置的时间内将只读节点的LSN更新至主节点最新LSN，PolarDB代理将根据ConsistTimeoutAction参数设置进行对应操作。  取值范围如下：  0：将读请求发往主节点（默认）。  1：代理返回一个错误报文wait replication complete timeout, please retry给应用端。 |

#### 适用场景

当主从延迟较高时，使用全局一致性可能会导致更多的请求被路由到主节点，造成主节点压力增大，业务延迟也可能增加。因此建议在读多写少的场景下选择全局一致性。

### 一致性级别选择最佳实践

PolarDB一致性级别越高，集群性能越低。推荐使用会话一致性，该级别对性能影响很小而且能满足绝大多数应用场景的需求。

若对不同会话间的一致性需求较高，可以选择如下方案之一：

使用HINT将特定查询强制发送至主节点执行。

/\*FORCE\_MASTER\*/ select \* from user;

说明：若您需要通过MySQL官方命令行执行上述Hint语句，请在命令行中加上-c参数，否则该Hint会被MySQL官方命令行过滤导致Hint失效，具体请参见[MySQL官方命令行](https://dev.mysql.com/doc/refman/5.6/en/mysql-command-options.html" \l "option_mysql_comments" \t "https://help.aliyun.com/document_detail/_blank)。

Hint的路由优先级最高，不受一致性级别和事务拆分的约束，使用前请进行评估。

Hint语句里不要有改变环境变量的语句，例如/\*FORCE\_SLAVE\*/ set names utf8;等，这类语句可能导致后续的业务出错。

选择全局一致性。

# 备份恢复

## 备份数据

## 恢复数据

## 库表恢复

# 数据迁移

# 高并发

## 连接池

PolarDB支持会话级连接池和事务级连接池，您可以根据业务场景选择合适的连接池，帮助降低因大量连接导致的数据库负载压力。

### 注意事项

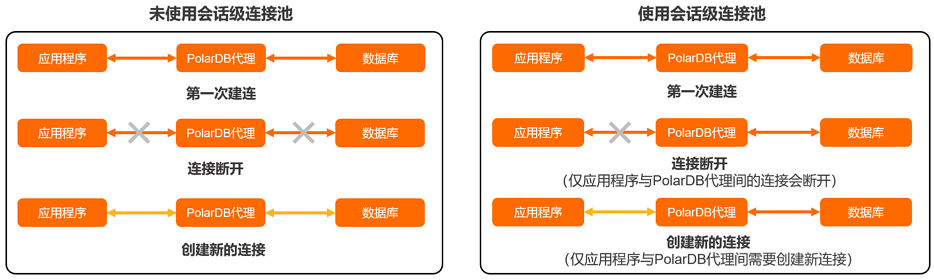
更改连接池设置后，仅对新建连接生效。如何修改连接池设置，请参见[修改集群地址](https://help.aliyun.com/document_detail/115494.htm" \l "section-6qz-cik-z5a" \o ")。

当前连接池功能不支持同一个账号对不同IP有不同的权限。如果您为同一个账号的不同IP设置了不同的库或者表权限，开通连接池可能会导致权限错误问题。例如，user@192.xx.xx.1设置了database\_a的权限，而user@192.xx.xx.2没有database\_a的权限，可能会导致连接复用时权限出错。

本文介绍的功能是PolarDB数据库代理的连接池功能。该功能并不影响客户端的连接池功能，如果客户端已经支持连接池，则可以不使用PolarDB数据库代理的连接池功能。

### 会话级连接池

工作原理：

[](https://static-aliyun-doc.oss-accelerate.aliyuncs.com/assets/img/zh-CN/5545273161/p240048.png)

会话级连接池用于减少短连接业务频繁建立新连接导致MySQL负载高。当您的连接断开时，系统会判断当前的连接是否是一个闲置的连接，如果是闲置连接，系统将会代理该连接并保留在连接池中一小段时间，如果这时新的连接建立的话就会直接从连接池里获得连接（命中的条件包括user、clientip和dbname等），从而减少与数据库的建连开销。如果没有可用的连接，则走正常连接流程，重新与数据库建立一个新的连接。

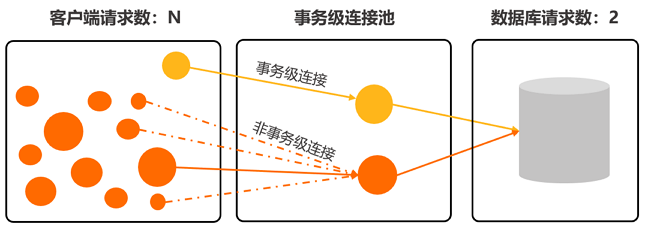
**使用限制：**

会话级连接池并不能减少数据库的并发连接数。该优化只能通过降低应用与数据库的建连速率来减少MySQL主线程的开销，从而更好地处理业务请求，但是连接池里空闲的连接会短暂占您的连接数。

会话级连接池也不能解决由于存在大量慢SQL，导致的连接堆积问题，此类问题的核心是先解决慢SQL问题。

### 事务级连接池

工作原理：

[](https://static-aliyun-doc.oss-accelerate.aliyuncs.com/assets/img/zh-CN/5545273161/p240047.png)

事务级连接池主要用于减少直接连接到数据库的业务连接数，以及减少短连接场景下频繁建连带来的负载。

开启事务级连接池后，客户端与PolarDB代理间可以存在上千个连接，但代理与后端数据库间可能只存在几十或几百个连接。

PolarDB代理本身并没有最大连接数的限制，连接数的限制主要由后端数据库中计算节点的规格决定。未开启事务级连接池时，每条由客户端发起的连接都需要在后端主节点和所有只读节点上各创建一个对应的连接。

开启事务级连接池后，当客户端发送请求时，会先与PolarDB代理建连，代理不会马上将其与后端数据库建连，而是先从事务级连接池里查找是否存在可用的连接（判断是否为可用连接的条件：user、dbname和系统变量这3个参数值是否一致）。若不存在，代理会与数据库创建一个新连接；若存在，则从连接池里直接拿出并使用，并在当前事务结束后将该连接放回事务级连接池，方便下个请求继续使用。

使用限制如下：

当执行以下行为时，锁定连接，直至连接结束，即该连接不会再被放到连接池里供其它用户连接使用。

执行PREPARE语句或命令

创建临时表

修改用户变量

大报文（例如16 MB以上）

使用lock table

多语句

存储过程调用

不支持FOUND\_ROWS、ROW\_COUNT和LAST\_INSERT\_ID函数的调用，这些函数可以调用成功，但是无法保证调用结果的正确性。其中：

1.13.11及以上的数据库代理版本支持在SELECT SQL\_CALC\_FOUND\_ROWS \* FROM t1 LIMIT \*语句后直接使用SELECT FOUND\_ROWS()命令。但MySQL官方已不推荐该用法，建议您将SELECT FOUND\_ROWS()替换为SELECT COUNT(\*) FROM tb1进行查询，详情请参见[FOUND\_ROWS()](https://dev.mysql.com/doc/refman/8.0/en/information-functions.html" \l "function_found-rows" \t "https://help.aliyun.com/document_detail/_blank)。

支持在INSERT后直接使用SELECT LAST\_INSERT\_ID()语句，来保证查询结果正确性。

对于设置了wait\_timeout的连接，wait\_timeout在客户端的表现可能不会生效。因为每次请求都会从连接池中获取连接，当wait\_timeout超时后，只有连接池中的后端连接会断开，而后端连接断开并不会导致客户端连接断开。

除了sql\_mode、character\_set\_server、collation\_server、time\_zone这四个变量以外，如果业务依赖其他会话级别的系统变量，那么需要客户端在建连之后显式进行SET语句执行，否则连接池可能会复用系统变量已经被更改过的连接。

由于连接可能会被复用，所以使用select connection\_id()查询当前连接的thread id可能会变化。

由于连接可能会被复用，所以show processlist中显示的IP地址和端口，可能会与客户端实际的IP地址和端口不一致。

数据库代理会将所有节点上的SHOW PROCESSLIST结果合并后再返回最终结果，而在事务级连接池开启后，前端连接和后端连接的thread id无法对应。这导致KILL命令会可能返回一个错误，但是实际上KILL命令已经正常执行成功，可再通过SHOW PROCESSLIST确定相应的连接是否断开。

### 连接池的选择

您可以根据如下建议评估选择是否开启连接池以及开启何种类型的连接池：

若业务使用的多为长连接且连接数较少，或者业务本身已具备较好的连接池，那么您可以不使用PolarDB的连接池功能。

若业务使用的连接数较多（如连接数需求上万）或使用的是Serverless服务（即连接数会随着业务端服务器的扩容而线性增加），且确认您的业务不涉及上述事务级连接池使用限制的场景，那么您可以选择开启事务级连接池。

若业务使用的纯短连接，且业务使用场景中包含上述事务级连接池使用限制的场景，那么您可以考虑开启会话级连接池。

## 慢SQL改写

PolarDB新增支持慢SQL（本文所指的慢SQL为执行时间超过5秒的SQL）改写功能，能够通过内置的优化引擎自动抓取并改写慢SQL，而无需您手动修改业务端的代码或配置，来提高SQL执行效率。

### 前提条件

集群所在地域需为如下地域之一：

华东1（杭州）、华东2（上海）、华北2（北京）、华南1（深圳）、中国（香港）。

需为PolarDB MySQL5.6、5.7或8.0集群版。

集群创建时间需在2021年03月16日及以后。

说明：针对创建时间在2021年03月16日之前的集群，如需使用该功能，请[提交工单](https://selfservice.console.aliyun.com/ticket/createIndex" \t "https://help.aliyun.com/document_detail/_blank)联系技术支持。

### 适用场景

PolarDB会在如下场景中使用慢SQL改写功能，来优化集群性能：

索引选择不当

一般情况下数据库内置的优化器会优先选择ORDER BY里的字段作为索引的选择参考，但有时ORDER BY里的字段推荐的索引并不是最优索引，从而影响查询效率。为规避该问题，PolarDB支持通过如下两种方式来自动改写SQL：

在SQL语句中添加FORCE INDEX Hint。

例如，将SELECT \* FROM table1改写成SELECT \* FROM table1 FORCE INDEX (index\_name) ...来强制优化器选择目标索引。

将ORDER BY 子句改写成等价的新语句。

例如，将ORDER BY [col1]改写成 ORDER BY [col1] IS NOT NULL, [col1] ，来避免优化器在索引选择时受ORDER BY后的字段影响。

子查询

PolarDB通过尝试将子查询改写成JOIN语句，让数据库在JOIN过程中使用索引来提升性能。

低负载时的慢SQL

在数据库的CPU负载较低的情况下，若某些SQL的执行用时仍然超过几十秒，PolarDB会在这些慢SQL语句中加入PARALLEL Hint，来自动开启并行查询，提高执行效率。

例如，将SELECT \* FROM t1;改写为SELECT /\*+PARALLEL(8)\*/ \* FROM t1;，使得数据库由使用单线程执行改为使用8个线程并行执行。

WHERE语句

PolarDB可以将WHERE语句中的谓词下推到下游的算子中来执行，从而过滤未被使用的数据，减少JOIN的数据量，实现查询效率的提升。

### 工作原理

开启慢SQL改写功能后，PolarDB会自动抓取执行时间超过5秒的SQL（即慢SQL），离线分析其中哪些SQL是可以通过改写来优化的，备用（Standby）计算节点（上面有完整的元数据信息和表统计信息，数据分布与生产环境一样，但不会影响生产）会对这些SQL进行优化验证，对于确实达到优化效果的SQL，在Proxy转发该SQL到后端节点执行之前，PolarDB会自动改写这些SQL，具体改写步骤如下：

将原SQL转化成关系代数。

对关系代数应用公认的等价变换来找到最优的执行计划。

将执行计划转化成关系代数。

将关系代数再转化成SQL。

同时，PolarDB可以从如下两方面保证SQL改写前后逻辑等价性和查询结果的一致性：

逻辑等价

从上述PolarDB改写慢SQL的步骤中，可以看出SQL语句和关系代数间是支持互相转化的，能够保证SQL改写前后的逻辑等价。

结果一致

改写慢SQL前，PolarDB会先离线分析和对比每个查询模板优化前后的结果集行列信息，只有当查询模板优化前后的结果集行列信息完全一致时，PolarDB才会在这些查询模板上应用SQL改写功能，保证了查询结果一致性。

### 如何使用

若集群满足前提条件，您可以在目标集群的基本信息页，单击目标集群地址的编辑配置，在弹出的对话框中打开或关闭慢SQL改写功能的开关。

[](https://static-aliyun-doc.oss-accelerate.aliyuncs.com/assets/img/zh-CN/7531397161/p261327.png)

## 读写分离

## 高并发优化

### Statement Concurrency Control

为了应对突发的数据库请求流量、资源消耗过高的语句访问以及SQL访问模型的变化，保证MySQL实例持续稳定运行，阿里云提供基于语句规则的并发控制CCL（Concurrency Control），并提供了工具包（DBMS\_CCL）便于您快捷使用。

**前提条件**

PolarDB集群版本需为PolarDB MySQL 5.6或PolarDB MySQL 8.0。

**注意事项**

CCL的操作不产生Binlog，所以CCL的操作只影响当前节点。例如主节点进行CCL操作，不会同步到只读节点。

CCL提供超时机制以应对DML导致事务锁死锁，等待中的线程也会响应事务超时和线程KILL操作以应对死锁。

**功能设计**

CCL规则定义了如下三个维度特征：

|  |  |
| --- | --- |
| 维度 | 说明 |
| SQL command | SQL命令类型（如SELECT、UPDATE、INSERT、DELETE等）。 |
| Object | SQL命令操作的对象（如TABLE、VIEW等）。 |
| keywords | SQL命令的关键字。 |

**创建CCL规则表**

PolarDB设计了一个系统表（concurrency\_control）保存CCL规则，系统启动时会自动创建该表，无需您手动创建。该系统表的创建语句如下所示：

CREATE TABLE `concurrency\_control` (

`Id` bigint(20) NOT NULL AUTO\_INCREMENT,

`Type` enum('SELECT','UPDATE','INSERT','DELETE') NOT NULL DEFAULT 'SELECT',

`Schema\_name` varchar(64) COLLATE utf8\_bin DEFAULT NULL,

`Table\_name` varchar(64) COLLATE utf8\_bin DEFAULT NULL,

`Concurrency\_count` bigint(20) DEFAULT NULL,

`Keywords` text COLLATE utf8\_bin,

`State` enum('N','Y') NOT NULL DEFAULT 'Y',

`Ordered` enum('N','Y') NOT NULL DEFAULT 'N',

PRIMARY KEY (`Id`)

) /\*!50100 TABLESPACE `mysql` \*/ ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8\_bin

STATS\_PERSISTENT=0 COMMENT='Concurrency control'

|  |  |
| --- | --- |
| 参数 | 说明 |
| Id | CCL规则ID。  说明 ID越大，规则的优先级越高，关于如何修改优先级，请参见[管理CCL规则](https://help.aliyun.com/document_detail/172926.html?spm=a2c4g.11186623.6.743.303f3d1bMwDReE" \l "section-yzz-zoc-le4" \o ")。 |
| Type | SQL命令类型（如SELECT、UPDATE、INSERT、DELETE等）。 |
| Schema\_name | 数据库名。 |
| Table\_name | 数据库内的表名。 |
| Concurrency\_count | 并发数。 |
| Keywords | 关键字，多个关键字用英文分号（;）分隔。 |
| State | 本规则是否启用，取值范围为N（否）和Y（是），默认为Y。 |
| Ordered | Keywords中多个关键字是否按顺序匹配，取值范围为N（否）和Y（是），默认为N。 |

**管理CCL规则**

为了便捷地管理CCL规则，PolarDB在DBMS\_CCL中定义了四个本地存储规则。详细说明如下：

**add\_ccl\_rule**

增加规则。命令如下：

dbms\_ccl.add\_ccl\_rule('<Type>','<Schema\_name>','<Table\_name>',<Concurrency\_count>,'<Keywords>');

示例：

SELECT语句的并发数为10。

mysql> call dbms\_ccl.add\_ccl\_rule('SELECT', '', '', 10, '');

SELECT语句中出现关键字key1的并发数为20。

mysql> call dbms\_ccl.add\_ccl\_rule('SELECT', '', '', 20, 'key1');

test.t表的SELECT语句的并发数为10。

mysql> call dbms\_ccl.add\_ccl\_rule('SELECT', 'test', 't', 20, '');

**del\_ccl\_rule**

删除规则。命令如下：

dbms\_ccl.del\_ccl\_rule(<Id>);

示例：

删除规则ID为15的CCL规则。

mysql> call dbms\_ccl.del\_ccl\_rule(15);

说明：如果删除的规则不存在，系统会报相应的警告，您可以使用show warnings;查看警告内容。

mysql> call dbms\_ccl.del\_ccl\_rule(100);

Query OK, 0 rows affected, 2 warnings (0.00 sec)

mysql> show warnings;

+---------+------+----------------------------------------------------+

| Level | Code | Message |

+---------+------+----------------------------------------------------+

| Warning | 7514 | Concurrency control rule 100 is not found in table |

| Warning | 7514 | Concurrency control rule 100 is not found in cache |

+---------+------+----------------------------------------------------+

**show\_ccl\_rule**

查看内存中已启用规则。命令如下：

dbms\_ccl.show\_ccl\_rule();

示例：

​mysql> call dbms\_ccl.show\_ccl\_rule();

+------+--------+--------+-------+-------+-------+-------------------+---------+---------+----------+----------+

| ID | TYPE | SCHEMA | TABLE | STATE | ORDER | CONCURRENCY\_COUNT | MATCHED | RUNNING | WAITTING | KEYWORDS |

+------+--------+--------+-------+-------+-------+-------------------+---------+---------+----------+----------+

| 17 | SELECT | test | t | Y | N | 30 | 0 | 0 | 0 | |

| 16 | SELECT | | | Y | N | 20 | 0 | 0 | 0 | key1 |

| 18 | SELECT | | | Y | N | 10 | 0 | 0 | 0 | |

+------+--------+--------+-------+-------+-------+-------------------+---------+---------+----------+----------+​

说明 关于MATCHED、RUNNING和WAITTING参数的说明如下：

MATCHED：规则匹配成功次数。

RUNNING：该规则下正在并发执行的线程数。

WAITTING：该规则下正在等待执行的线程数。

**flush\_ccl\_rule**

如果通过修改表concurrency\_control中的内容来修改规则，您还需要使用如下命令让该规则生效：

dbms\_ccl.flush\_ccl\_rule();

示例：

mysql> update mysql.concurrency\_control set CONCURRENCY\_COUNT = 15 where Id = 18;

Query OK, 1 row affected (0.00 sec)Rows matched: 1 Changed: 1 Warnings: 0

mysql> call dbms\_ccl.flush\_ccl\_rule();

Query OK, 0 rows affected (0.00 sec)​

您可以通过使用UPDATE语句修改规则ID来调整目标规则的优先级。

update mysql.concurrency\_control set ID = xx where ID = xx;

### Inventory Hint

PolarDB提供Inventory Hint，帮助您快速提交、回滚事务。您还可以将Inventory Hint和Statement Queue一起配合使用，有效提高业务吞吐能力。

**前提条件**

PolarDB集群版本需为PolarDB MySQL 8.0且Revision version为8.0.1.1.1及以上，您可以通过[查询版本号](https://help.aliyun.com/document_detail/178903.htm" \l "section-9of-fn2-p5v" \o ")确认集群版本。

**背景信息**

在电商秒杀活动等业务场景中，减少库存是一个常见的需要高并发，同时也需要串行化的任务模型，PolarDB使用排队和事务性Hint来控制并发和快速提交或回滚事务，提高业务吞吐能力。

**语法**

PolarDB提供的Inventory Hint支持SELECT、UPDATE、INSERT、DELETE语句。

Inventory Hint包括如下三个事务语法：

COMMIT\_ON\_SUCCESS：当前语句执行成功就提交事务。

示例：

PolarDB MySQL 5.6

UPDATE COMMIT\_ON\_SUCCESS TSET c = c - 1WHERE id = 1;

PolarDB MySQL 8.0

UPDATE /\*+ COMMIT\_ON\_SUCCESS \*/ TSET c = c - 1WHERE id = 1;

ROLLBACK\_ON\_FAIL：当前语句执行失败就回滚事务。

示例：

PolarDB MySQL 5.6

UPDATE ROLLBACK\_ON\_FAIL TSET c = c - 1WHERE id = 1;

PolarDB MySQL 8.0

UPDATE /\*+ ROLLBACK\_ON\_FAIL \*/ TSET c = c - 1WHERE id = 1;

TARGET\_AFFECT\_ROW number：如果当前语句影响的行数符合设定的行数就成功，否则语句失败。

假设Target Affect Row的值设为1，如果更新语句实际更新到了一条数据则认为成功，如果更新没有命中任何记录则认为失败。

示例：

PolarDB MySQL 5.6

UPDATE TARGET\_AFFECT\_ROW 1 TSET c = c - 1WHERE id = 1;

ERROR HY000: The affected row number does not match that of user specified.

PolarDB MySQL 8.0

UPDATE /\*+ TARGET\_AFFECT\_ROW(1) \*/ TSET c = c - 1WHERE id = 1;

ERROR HY000: The affected row number does not match that of user specified.

配合Statement Queue使用

UPDATE、INSERT、DELETE语句中的COMMIT\_ON\_SUCCESS、ROLLBACK\_ON\_FAIL、TARGET\_AFFECT\_ROW number这三个事务语法可以配合[Statement Queue](https://help.aliyun.com/document_detail/172927.htm" \l "task-2315487" \o "设计了针对语句的排队（Statement Queue）机制，将语句进行分桶排队，尽量把可能具有相同冲突（例如操作相同行）的语句放在一个桶内排队，减少冲突的开销。)进行排队。

示例（以PolarDB MySQL 5.6语法为例）：

UPDATE COMMIT\_ON\_SUCCESS POLARDB\_STATEMENT\_CONCURRENT\_QUEUE id ROLLBACK\_ON\_FAIL TARGET\_AFFECT\_ROW 1 tSET col1 = col1 + 1WHERE id = 1;Query OK, 1 row affected (0.00 sec)Rows matched: 1 Changed: 1 Warnings: 0

UPDATE COMMIT\_ON\_SUCCESS POLARDB\_STATEMENT\_CONCURRENT\_QUEUE 1 ROLLBACK\_ON\_FAIL TARGET\_AFFECT\_ROW 1 tSET col1 = col1 + 1WHERE id = 1;Query OK, 1 row affected (0.00 sec)Rows matched: 1 Changed: 1 Warnings: 0

### Statement Queue

### 热点行优化

### Thread Pool

# 高可用

## 连接保持

PolarDB新增支持连接保持功能，避免由于一些运维操作（如升级配置、主备切换或升级小版本等）或非运维操作故障（如节点所在服务器故障）导致的连接闪断或新建连接短暂失败的问题，进一步提高PolarDB的高可用性。

### 前提条件

集群版本需为PolarDB MySQL5.6、5.7或8.0且产品系列为集群版。

集群创建时间需在2021年03月16日及以后。

说明：若创建时间在2021年03月16日之前集群需使用该功能，请[提交工单](https://selfservice.console.aliyun.com/ticket/createIndex" \t "https://help.aliyun.com/document_detail/_blank)联系技术支持。

### 背景信息

PolarDB已通过高可用组件实现了主节点出现故障时快速进行主备切换，来提供高可用的集群服务。但此类切换过程会对应用程序服务造成影响，导致连接闪断、新建连接短暂失败等问题。引起应用程序服务短暂不可用的场景通常分为如下两种：

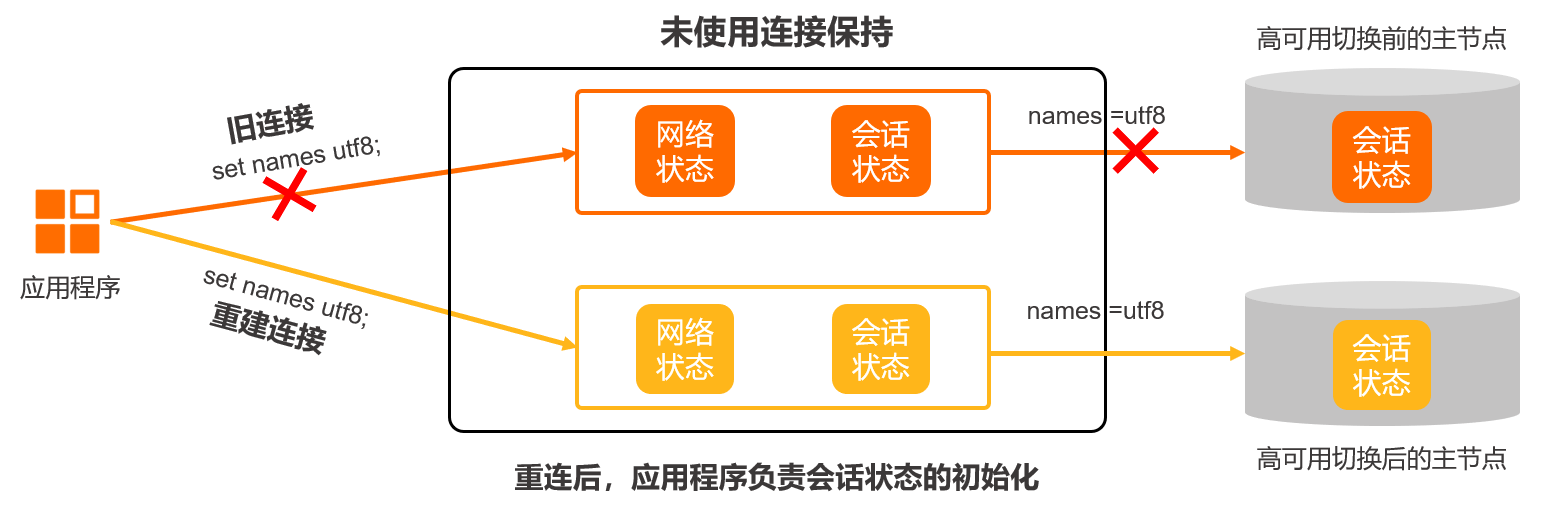
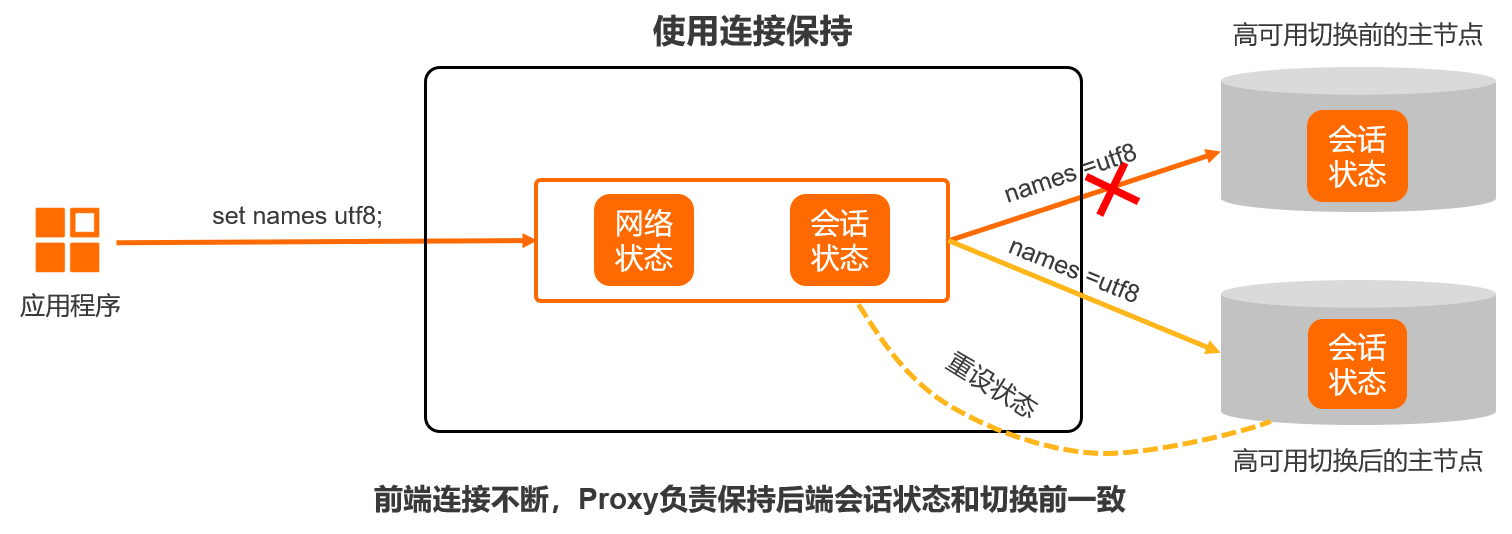
计划内（Switchover）：即由数据库控制台或后台管控发起的各种运维操作，如[升级配置](https://help.aliyun.com/document_detail/72674.htm" \l "task-1580301" \o "您可以根据业务波动变更集群的配置。本文介绍如何升级或降级集群的规格。)、[主备切换](https://help.aliyun.com/document_detail/141229.htm" \l "task-2317735" \o "是一个多节点的集群，其中一个节点是主节点，其他节点为只读节点。除了因系统故障自动进行主备切换外，您也可以手动进行主备切换，指定一个只读节点为新的主节点，适用于高可用演练，或者需要指定某个节点为主节点的场景。)或[升级小版本](https://help.aliyun.com/document_detail/158572.htm" \l "task-2449714" \o "支持手动升级到最新修订版本（Revision version），版本升级涉及性能提升、新功能或问题修复等方面。您还可以选择立即升级或在可运维时间内进行升级版本。)等。

计划外（Failover）：其它非运维操作引起的故障，比如主节点崩溃，节点所在的主机出现故障等。

通常的解决办法是重启应用程序或保证应用程序具备重连机制，但由于开发周期等原因，在开发设计应用程序的前期可能并未考虑到该问题，导致出现大量的非预期行为甚至应用程序的服务不可用。因此PolarDB新增支持连接保持功能，避免由于一些运维操作或非运维操作故障导致的应用程序服务短暂不可用问题，进一步提高PolarDB集群的高可用性。

### 实现原理

从连接建立的角度看，PolarDB中的一个会话（Session）包含了一个前端连接（即应用程序和Proxy的连接）和一个后端连接（即Proxy和后端数据库节点的连接）。开启连接保持功能后，当Proxy与旧主节点（即高可用切换前的主节点）连接断开时，Proxy与前端应用的连接保持不断（即应用程序看到的Session），同时Proxy会与新主节点（即高可用切换后的主节点）重新建立连接并且恢复之前的会话状态，以实现对应用程序端无感知的高可用切换。

[](https://static-aliyun-doc.oss-accelerate.aliyuncs.com/assets/img/zh-CN/2758795161/p251206.png)[](https://static-aliyun-doc.oss-accelerate.aliyuncs.com/assets/img/zh-CN/2758795161/p251207.png)

MySQL的连接（会话）通常包括系统变量、用户变量、临时表、字符集编码、事务状态和的Prepare语句状态信息。本文以字符集编码的状态信息为例介绍开启连接保持功能前后的会话状态。

假设应用程序和Proxy间建立了一个连接，并执行了set names utf8; 命令，此时names=utf8就是这个连接的一个状态。当Proxy在新旧主节点中进行切换时，需要保留这个状态，否则会出现字符集乱码问题。所以连接保持的核心在于保证连接切换前后的会话状态一致。

说明：当Proxy将连接从旧主节点切换到新主节点的过程中，会存在短暂的新老数据库同时不可写不可读的时间（具体时长受数据库压力影响），所以在切换时，Proxy会暂时停止将应用程序的连接请求路由到后端数据库，并根据新数据库可读可写能力的恢复时间决定路由方向：

若新数据库在60s内恢复了可写可读能力，Proxy会将请求路由到新数据库。

若新数据库的可读可写能力未能在60s内恢复，Proxy会断开与应用程序间的连接，应用程序需重新发起连接请求（即此时与未使用连接保持功能时的行为一致）。

### 如何使用

对于创建时间在2021年03月16日及以后的PolarDB MySQL集群，该功能默认开启，无需手动开启。

对于创建时间在2021年03月16日以前的PolarDB MySQL集群，请[提交工单](https://selfservice.console.aliyun.com/ticket/createIndex" \t "https://help.aliyun.com/document_detail/_blank)联系技术支持开启该功能。

### 注意事项

连接保持功能无法保持如下场景中的连接：

当连接切换时，连接（会话）上存在临时表。

当连接切换时，Proxy正在从数据库接收结果报文但只接收了部分结果报文，比如执行SELECT语句后，需要从数据库接收一个包含100 MB数据的结果报文，但切换时数据库只传输了10 MB数据。

当连接切换时，连接（会话）上有正在执行中的事务（如begin;insert into;）。

说明：针对上述场景中的最后两种，若该场景是由于计划内（Switchover）的操作导致的连接切换，Proxy会先保留该请求并等待2s，如果剩余的结果报文能在2s内全部返回或事务能在2s能执行完成，Proxy将在等待时间结束后执行连接切换，从而提高连接保持的概率。

### 性能测试（附）

测试环境

测试用集群如下：

PolarDB MySQL8.0集群（默认包含一主一只读两个节点）。

节点规格为4核16 GB独享规格（polar.mysql.x4.large）。

测试工具：SysBench。

测试数据如下：

20张表，其中每张表包含10000行记录数。

并发度为20。

测试方法

在不同运维场景下，测试PolarDB集群的连接保活率（即执行运维操作前后的连接保持比例）。

测试结果

在如下测试的运维场景中，PolarDB集群均能保持100%的连接保活率。

说明：

当前升级集群规格仅支持在未跳级升级规格（如从4核升级至8核）的场景下能够保持100%的连接保活率，如果是从4核升级到16核（或者更大规格），可能出现连接闪断。

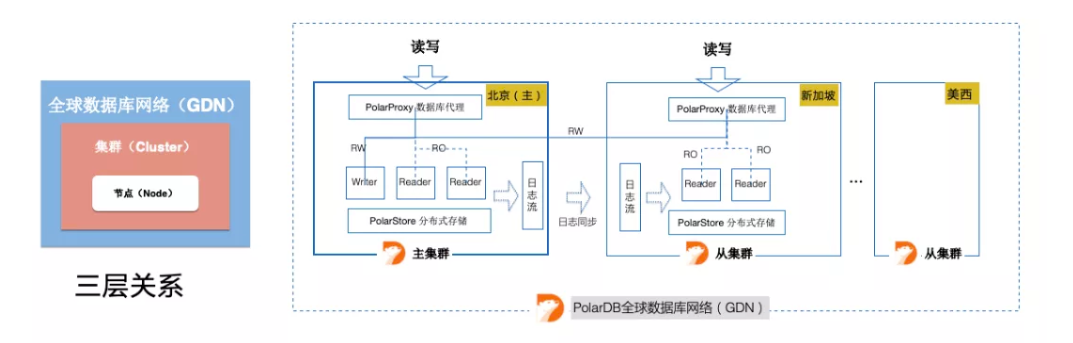
若删除只读节点时触发了数据库代理节点收缩，将导致部分连接的闪断。

内核小版本升级的场景中仅包括数据库内核引擎的小版本升级，而不包含数据库代理的小版本升级，数据库代理的小版本升级会出现连接闪断。

|  |  |
| --- | --- |
| 运维场景 | 保活率 |
| 主备切换 | 100% |
| 内核小版本升级 | 100% |
| 升级集群规格 | 100% |
| 增删节点 | 100% |

## 多可用区部署

## 全球数据库



多个PolarDB对外看起来就是一个集群，但所有集群中只有一个能可以写，其它都是可读。最多支持5个集群，集群间数据同步延迟小于2秒。每个集群均支持读写，从集群会把写请求转给主集群，因此集群上的写请求会跨Region，延迟略高。

所以在PolarDB中对于两地三中心，三地六中心 这种架构支持非常好。多Region之间的流量传输目前是免费的。

# 数据压缩

# 数据安全

# 运维管理

## 监控告警

## 性能监控

### 会话管理

### 实时性能

### 空间分析

### 锁分析

# 性能优化

## DDL优化

### 秒级加字段

使用传统方法执行加列操作时，需要重建整个表数据，占用大量系统资源。PolarDB MySQL新增支持秒级加字段（Instant add column）功能，在加列操作时只需变更表定义信息，无需修改已有数据，帮助您快速完成对任意大小的表的加列操作。本文介绍如何使用秒级加字段功能。

#### 前提条件

集群版本需为以下版本之一：

PolarDB MySQL 5.7且修订版本为5.7.1.0.6或以上，您可以通过[查询版本号](https://help.aliyun.com/document_detail/178903.htm" \l "section-9of-fn2-p5v" \o ")确认集群的修订版本。

说明：您需要先配置innodb\_support\_instant\_add\_column参数才能在PolarDB MySQL 5.7集群上使用该功能。

PolarDB MySQL 8.0。

说明：PolarDB MySQL 8.0集群默认支持秒级加字段功能，无需配置任何参数。

#### 使用限制

新增列只能为表的最后一列。

不支持虚拟列（PolarDB MySQL8.0支持）。

不支持分区表（PolarDB MySQL8.0支持）。

不支持使用了全文索引的表。

不支持开启了Implicit primary key选项且未自定义主键的表。

不支持在同一条SQL中同时执行其它DDL操作和Instant add column操作。

#### 使用方法

**参数说明：**

针对PolarDB MySQL 5.7集群，您需要开启innodb\_support\_instant\_add\_column参数来使用秒级加字段功能。

说明：PolarDB MySQL 8.0集群无需配置该参数即可直接使用秒级加字段功能。

|  |  |  |
| --- | --- | --- |
| 参数 | 级别 | 说明 |
| innodb\_support\_instant\_add\_column | Global | 秒级加字段功能的开关，取值范围如下：  ON：开启秒级加字段功能  OFF：关闭秒级加字段功能（默认值） |

**语句：**

指定ALGORITHM=INSTANT以强制使用秒级加字段功能，语句示例如下：

ALTER TABLE test.t ADD COLUMN test\_column int, ALGORITHM=INSTANT;

使用上述语句时，若返回ERROR 0A000: ALGORITHM=INSTANT is not supported for this operation. Try ALGORITHM=COPY/INPLACE. 的错误，表示当前加列操作不能以Instant算法执行，建议您查看innodb\_support\_instant\_add\_column参数是否已开启，并仔细核对[使用限制](https://help.aliyun.com/document_detail/198913.html?spm=a2c4g.11186623.6.728.5c9c54dckopwoA" \l "section-hkp-4lw-k7w" \o ")。

不指定ALGORITHM或指定ALGORITHM=DEFAULT，PolarDB会自行选择执行速度最快的算法来执行加列操作，语句示例如下：

ALTER TABLE test.t ADD COLUMN test\_column int, ALGORITHM=DEFAULT;ALTER TABLE test.t ADD COLUMN test\_column int;

说明：PolarDB算法选择的优先级为INSTANT > INPLACE > COPY。

**查看通过Instant算法增加的列信息：**

INFORMATION\_SCHEMA数据库中新增了INNODB\_SYS\_INSTANT\_COLUMNS表。该表记录了使用Instant算法增加的列信息，例如列名、列序号和默认值（二进制方式存储）等。您可通过如下语句查看该表详情来确认新增的列信息。

SELECT \* FROM INFORMATION\_SCHEMA.INNODB\_SYS\_INSTANT\_COLUMNS;

说明：对目标表使用了Instant算法增加列后，如果执行了需要重建表的DDL操作（如DROP COLUMN），系统将会删除INNODB\_SYS\_INSTANT\_COLUMNS表中目标表的相关列信息。

### 并行元数据锁同步

PolarDB支持并行元数据锁同步（Async Metadata Lock Replication）功能，用于提高DDL操作的执行效率。

注：GoldenDB不支持。

#### 前提条件

对于PolarDB MySQL 5.6和5.7集群，所有Revision version都支持该功能。

对于PolarDB MySQL8.0集群，Revision version需为8.0.1.1.10或以上，您可以通过[查询版本号](https://help.aliyun.com/document_detail/178903.htm" \l "section-9of-fn2-p5v" \o ")确认集群版本。

#### 功能介绍

对数据库执行DDL操作时，需要在主节点和只读节点间同步元数据锁（Metadata Lock，简称MDL）信息来保证数据定义（Data Definition）的一致性。但主节点上的DDL操作常常会占有MDL，导致只读节点在长时间的等待后才能获取MDL信息进行同步，而在MDL同步成功前，只读节点不再解析Redo Log，这将严重影响DDL操作的整体效率。

通过并行MDL同步功能，PolarDB将MDL同步操作与Redo Log的解析操作解耦，实现了即使在等待MDL锁信息进行同步时，只读节点仍能继续解析并应用Redo Log。

#### 使用方法

该功能默认开启，无需额外操作。

您可以在只读节点上使用如下SQL语句，获取目标只读节点上MDL同步相关的信息：

SELECT \* FROM INFORMATION\_SCHEMA.INNODB\_LOG\_MDL\_SLOT;

说明：关于如何强制指定在某节点执行某查询命令，请参见[Hint语法](https://help.aliyun.com/document_detail/85143.htm" \l "section-268-4y3-g1u" \o ")。

返回结果如下：

+---------+------------+-------------------+----------+-------------------+

| slot\_id | slot\_state | slot\_name | slot\_lsn | thread\_id |

+---------+------------+-------------------+----------+-------------------+

| 0 | SLOT\_NONE | no targeted table | 0 | no running thread |

| 1 | SLOT\_NONE | no targeted table | 0 | no running thread |

| 2 | SLOT\_NONE | no targeted table | 0 | no running thread |

| 3 | SLOT\_NONE | no targeted table | 0 | no running thread |

| 4 | SLOT\_NONE | no targeted table | 0 | no running thread |

+---------+------------+-------------------+----------+-------------------+

该表展示了正在进行同步的MDL信息详情，其中slot\_name展示了相关的数据表信息，slot\_state展示了当前的MDL同步状态信息，包括：

SLOT\_NONE：初始化状态。

SLOT\_RESERVED：只读节点已经接收到MDL获取请求，等待调度系统分配工作线程进行处理。

SLOT\_ACQUIRING：系统已经分配工作线程资源，只读节点正在发送MDL请求。

说明：若只读节点需要的MDL被其它连接所持有，MDL同步状态将保持在该状态。

SLOT\_LOCKED：只读节点上MDL锁已经获取并持有。

SLOT\_RELEASING：只读节点已经接收到MDL释放请求，等待调度系统分配工作线程进行处理。

您可在使用如下SQL语句，获取只读节点上用于同步MDL请求的工作线程状态信息：

SELECT \* FROM INFORMATION\_SCHEMA.INNODB\_LOG\_MDL\_THREAD;

返回结果如下：

+-----------+-----------+------------------+-------------------+----------+

| thread\_id | thr\_state | slot\_state | slot\_name | slot\_lsn |

+-----------+-----------+------------------+-------------------+----------+

| 0 | free | not in acquiring | no targeted table | 0 |

| 1 | free | not in acquiring | no targeted table | 0 |

| 2 | free | not in acquiring | no targeted table | 0 |

| 3 | free | not in acquiring | no targeted table | 0 |

+-----------+-----------+------------------+-------------------+----------+

当INNODB\_LOG\_MDL\_SLOT表中存在状态为SLOT\_ACQUIRING的MDL同步请求时，INNODB\_LOG\_MDL\_THREAD表中将会存在一个对应的工作线程用于处理该请求（即thr\_state值不为free）。该情形说明只读节点上很可能存在MDL等待的情况。您可以通过thr\_state的值是否为free帮助排查是否出现MDL阻塞的问题。

### DDL物理复制优化

PolarDB通过DDL物理复制优化功能，实现了在主节点写物理日志和只读节点应用物理日志的关键路径上的全面优化，大大缩短了主节点上DDL操作的执行时间和只读节点上解析DDL的物理日志复制延迟时间。

#### 前提条件

集群版本需为PolarDB MySQL 8.0集群版且为Revision version为8.0.1.1.10或以上，您可以通过[查询版本号](https://help.aliyun.com/document_detail/178903.htm" \l "section-9of-fn2-p5v" \o ")确认集群版本。

#### 使用限制

目前并行DDL物理复制优化仅支持创建主键或二级索引（不包括全文索引和空间索引）的DDL操作。

对于只需修改元数据的DDL操作（如rename），因其本身执行速度已经很快，无需该优化。

#### 背景信息

PolarDB通过存储计算分离架构，实现了主节点和只读节点共享同一份存储数据，既降低了存储成本，又提高了集群的可用性和可靠性。为实现这一架构，PolarDB采用了业界领先的物理复制技术，不仅实现了共享存储架构上主节点和只读节点间的数据一致性，而且减少了Binlog fsync操作带来的I/O开销。

InnoDB中的数据是通过B-Tree来维护索引的，然而大部分Slow DDL操作（如增加主键或二级索引、optimize table等）往往需要重建或新增B-Tree索引，导致大量物理日志的产生。而针对物理日志进行的操作往往出现在DDL执行的关键路径上，增加了DDL操作的执行时间。此外，物理复制技术要求只读节点解析和应用这些新生成的物理日志，由于DDL操作而产生的大量物理日志可能严重影响只读节点的日志同步进程，甚至导致只读节点不可用等问题。

针对上述问题，PolarDB提供了DDL物理复制优化功能，在主节点写物理日志和只读节点应用物理日志的关键路径上做了全面的优化，使得主节点在执行创建主键DDL操作的执行时间最多约可减少至原来的20.6%，只读节点解析DDL的复制延迟时间最多约可减少至原来的0.4%。

#### 使用方法

您可以通过设置如下参数开启DDL物理复制优化功能。

|  |  |  |
| --- | --- | --- |
| 参数 | 级别 | 说明 |
| innodb\_bulk\_load\_page\_grained\_redo\_enable | Global | DDL物理复制优化功能开关，取值范围如下：  ON：开启DDL物理复制优化  OFF：关闭DDL物理复制优化（默认值） |

### 并行DDL

PolarDB新增支持并行DDL的功能。当数据库硬件资源空闲时，您可以通过并行DDL功能加速DDL执行，避免阻塞后续相关的DML操作，缩短执行DDL操作的窗口期。

#### 前提条件

PolarDB集群版本需满足如下条件之一：

PolarDB MySQL8.0且修订版本为8.0.1.1.7或以上。

PolarDB MySQL5.7且修订版本为5.7.1.0.7或以上

#### 注意事项

开启并行DDL功能后，由于并行线程数的增加，硬件资源（如CPU、内存、IO等）的占用也会随之增加，可能会影响同一时间内执行的其他SQL操作，因此建议在业务低峰或硬件资源充足时使用并行DDL。

#### 使用限制

目前并行DDL加速仅支持创建二级索引（不包括聚簇索引、全文索引、空间索引和虚拟列上的二级索引）的DDL操作。

#### 背景信息

传统的DDL操作基于单核和传统硬盘设计，导致针对大表的DDL操作耗时较久，延迟过高。以创建二级索引为例，过高延迟的DDL操作会阻塞后续依赖新索引的DML查询操作。多核处理器的发展为并行DDL使用更多线程数提供了硬件支持，而固态硬盘（Solid State Disk，简称SSD）的普及使得随机访问延迟与顺序访问延迟相近，使用并行DDL加速大表的索引创建显得尤为重要。

#### 使用方法

1、innodb\_polar\_parallel\_ddl\_threads

您可以通过如下innodb\_polar\_parallel\_ddl\_threads参数开启并行DDL功能：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 级别 | 取值范围 | 说明 |
| innodb\_polar\_parallel\_ddl\_threads | Session | [1~8]  默认值为1。 | 控制每一个DDL操作的并行线程数。默认值为1，即执行单线程DDL。  若该参数值不为1，当执行创建二级索引操作时将自动开启并行DDL。 |

2、Optimization A

若仅开启并行DDL功能仍不能满足您的需求，您还可以通过Optimization A对创建索引过程中的排序或建索引树过程进行进一步优化。

说明：Optimization A功能尚在内测中，暂不支持自定义开启该功能，如需使用，请[提交工单](https://selfservice.console.aliyun.com/ticket/createIndex" \t "https://help.aliyun.com/document_detail/_blank)联系技术支持。

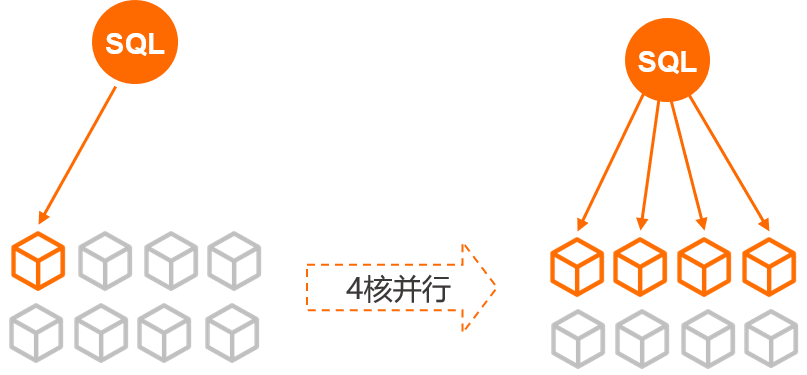
## 大查询优化

### 并行优化

PolarDB MySQL 8.0重磅推出并行查询框架，当您的查询数据量到达一定阈值，就会自动启动并行查询框架，从而使查询耗时指数级下降。

在存储层将数据分片到不同的线程上，多个线程并行计算，将结果流水线汇总到总线程，最后总线程做些简单归并返回给用户，提高查询效率。

并行查询（Parallel Query）利用多核CPU的并行处理能力，以8核32 GB（独享规格）的PolarDB MySQL集群版为例，示意图如下所示。

[](https://static-aliyun-doc.oss-accelerate.aliyuncs.com/assets/img/zh-CN/6999116161/p60483.png)

#### 应用场景

并行查询适用于大部分SELECT语句，例如大表查询、多表连接查询、计算量较大的查询。对于非常短的查询，效果不太显著。

轻分析类业务

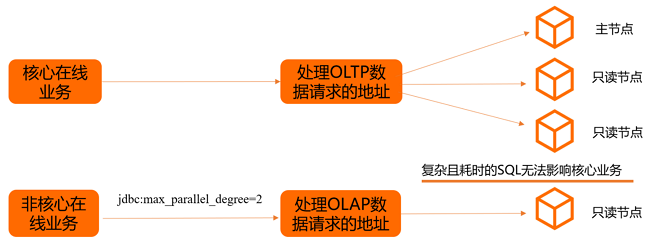
报表查询通常SQL复杂而且比较耗费时间，通过并行查询可以加速单次查询效率。

系统资源相对空闲

并行查询会使用更多的系统资源，只有当系统的CPU较多、IO负载不高、内存够大的时候，才可以充分使用并行查询来提高资源利用率和查询效率。

在离线混合场景

不同的业务用不同的连接地址，使用不同的数据库节点，避免互相影响。如果您为单节点的集群地址开启了并行查询，那么就会通过处理OLAP数据请求的地址对该单节点地址下的对应只读节点进行访问。

[](https://static-aliyun-doc.oss-accelerate.aliyuncs.com/assets/img/zh-CN/5720269951/p97221.png)

#### 并行查询的使用方法

通过系统参数来控制并行查询

PolarDB通过Global全局参数max\_parallel\_degree来控制每一条SQL最多使用多少个线程并行执行，默认值是0，您可以在使用过程中随时修改该参数，无需重启数据库，具体操作请参见[设置集群参数](https://help.aliyun.com/document_detail/72742.htm" \l "task-1580301" \o "PolarDB集群创建完成后，您可以通过控制台修改集群参数，并可以将修改后的集群参数导出为模板，同时同一地域的集群可以通过应用模板功能快速修改集群参数。本节介绍修改集群参数，将修改后参数导出为模板并应用模板的方法。)。

并行查询推荐设置以及相关说明如下：

并行度参数从低到高，逐渐增加，建议不要超过CPU核数的四分之一 。集群的CPU内核数大于等于8才支持开启并行查询，小规格的集群不建议开启该参数。例如，刚开始使用并行查询时，设置max\_parallel\_degree参数为2，试运行一天后，如果CPU压力不大，可以往上增加；如果CPU压力较大，停止增加。

max\_parallel\_degree参数为0时，表示关闭并行查询；max\_parallel\_degree参数为1时，表示开启并行查询，且并行度为1。

为保证并行度参数max\_parallel\_degree也能兼容其它程序版本以及考虑到MySQL配置文件中该参数的默认配置，PolarDB在控制台上增加了前缀loose，因此参数名称是loose\_max\_parallel\_degree，这样保证其他版本接受该参数时也不会报错， 详情请参见[Program Option Modifiers](https://dev.mysql.com/doc/refman/8.0/en/option-modifiers.html" \t "https://help.aliyun.com/document_detail/_blank)。

打开并行查询功能时， 需要设置innodb\_adaptive\_hash\_index参数为OFF，innodb\_adaptive\_hash\_index参数开启会影响并行查询的性能。

除了Global集群级别，您也可以单独调整某Session内SQL查询的并行度。例如，通过Session级环境变量，加到JDBC的连接串配置中，则可以对某个应用程序单独设置并行度。

set max\_parallel\_degree = n

通过Hint来控制并行查询

使用Hint语法可以对单个语句进行控制，例如系统默认关闭并行查询情况下，但需要对某个高频的慢SQL查询进行加速，此时就可以使用Hint对特定SQL进行加速。更多详情，请参见[通过Hint控制](https://help.aliyun.com/document_detail/178155.htm" \l "concept-2567761" \o "Parallel Hints可以指定优化器是否选择并行执行，还支持指定并行度以及需要并行的表。目前支持在并行查询中使用PARALLEL和NO_PARALLEL两种Hints。)。

通过设置阈值来控制优化器是否选择并行执行

PolarDB提供了两个阈值来控制优化器是否选择并行执行，SQL语句只要满足其中任意一个条件，优化器就会考虑并行执行。

records\_threshold\_for\_parallelism

若优化器估算出语句中存在扫描记录数超过该阈值的表，优化器会考虑选择并行执行计划。默认值为10000。若您的业务量较小或复杂查询业务并发较低，您可以选择将该阈值设置为2000或以上。

说明：上文提到的扫描记录数是根据对应表的统计信息进行估算得出的值，可能存在一定的误差。

cost\_threshold\_for\_parallelism

若优化器估算查询的串行执行代价超过该阈值，优化器会考虑选择并行执行计划。默认值为50000。

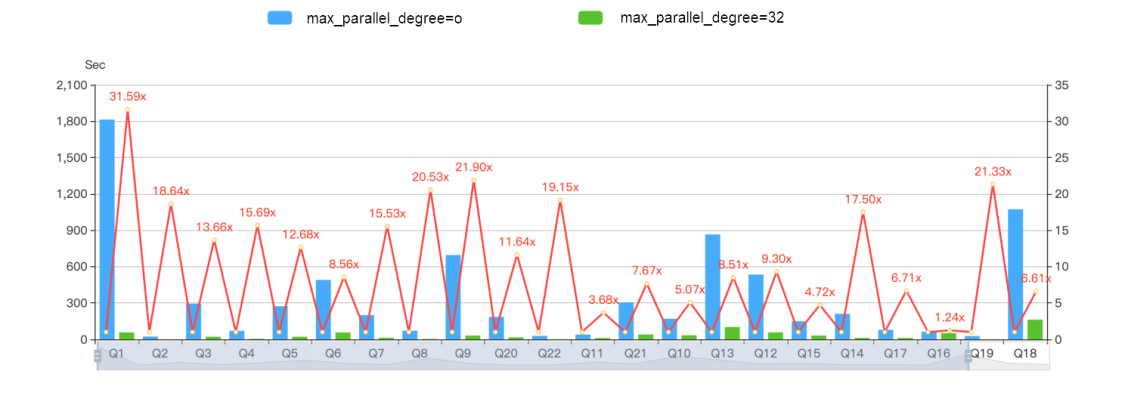
#### 相关参数和变量

|  |  |  |
| --- | --- | --- |
| 表 1. 系统参数 | | |
| 参数名 | 级别 | 描述 |
| max\_parallel\_degree | Global、Session | 单个查询的最大并行度，即最多使用多少个Worker进行并行执行。  取值范围：[0-1024]  默认值：0，表示关闭并行计算。  说明 PolarDB优化器可能会对主查询和子查询分别并行执行，如果同时并行执行，它们的最大Worker数不能超过max\_parallel\_degree的值，整个查询使用的Worker数为主查询和子查询使用的Worker数之和。 |
| parallel\_degree\_policy | Global | 设置单个查询的并行度配置策略，取值范围如下：  TYPICAL：PolarDB选择查询并行度时不会考虑数据库负载（如CPU使用率等），而尽可能与max\_parallel\_degree设置的并行度保持一致。  AUTO：PolarDB会根据数据库负载（如CPU使用率等）来决定是否禁止并行查询计划，并会根据查询代价选择并行度。  REPLICA\_AUTO（默认）：仅只读节点会根据数据库负载（如CPU使用率等）决定是否禁止并行查询计划，并会根据查询代价选择并行度，而主节点不会选择并行执行计划。  说明 更多关于并行度配置策略的详细介绍，请参见[并行度控制策略](https://help.aliyun.com/document_detail/203022.htm" \l "concept-2041772" \o "支持通过parallel_degree_policy参数来设置并行查询中并行度的配置策略。本文将介绍相关参数说明。)。 |
| records\_threshold\_for\_parallelism | Session | 若优化器估算出语句中存在扫描记录数超过该阈值的表，优化器会考虑选择并行执行计划。  取值范围：[0-18446744073709551615]  默认值：10000。  说明 若您的业务量较小或复杂查询业务并发较低，您可以选择将该阈值设置为2000或以上。 |
| cost\_threshold\_for\_parallelism | Session | 若优化器估算查询的串行执行代价超过该阈值，优化器会考虑选择并行执行计划。  取值范围：[0-18446744073709551615]  默认值：50000。 |

|  |  |  |
| --- | --- | --- |
| 表 2. 状态变量 | | |
| 变量名 | 级别 | 描述 |
| Parallel\_workers\_created | Session、Global | 从Session启动开始，生成Parallel Worker的个数。 |
| Gather\_records | Session、Global | Gather记录总数。 |
| PQ\_refused\_over\_memory\_soft\_limit | Session、Global | 由于内存限制没有启用并行的查询数。 |
| PQ\_refused\_over\_total\_workers | Session、Global | 由于总Worker数限制没有启用并行的查询数。 |
| Total\_used\_query\_memory | Global | 当前总的已使用的查询内存（Virtual Memory）。 |
| Total\_running\_parallel\_workers | Global | 当前正在运行的Parallel Worker的数目。 |

#### 性能指标

本次测试将使用TPC-H生成100 GB数据来测试PolarDB MySQL 8.0集群的性能指标。测试用的PolarDB集群规格为32核256 GB（独享规格），并行度max\_parallel\_degree分别设置为32和0，具体测试步骤请参见[并行查询性能（OLAP）](https://help.aliyun.com/document_detail/146099.htm" \l "task-2350322" \o "本文档介绍以TPC-H测试 8.0的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。)。

[](https://static-aliyun-doc.oss-accelerate.aliyuncs.com/assets/img/zh-CN/6068126161/p97292.png)

通过以上测试结果图得出结论，TPC-H中95%的SQL可以被加速，且在被加速的SQL语句中，有70%的SQL语句的执行速度是未使用并行查询的8倍以上。

#### 并行执行EXPLAIN

更多关于EXPLAIN执行计划输出中与并行查询相关的内容，请参见[并行执行EXPLAIN](https://help.aliyun.com/document_detail/208545.htm" \l "concept-2059476" \o "本文介绍如何使用EXPLAIN语句查看执行计划输出中与并行查询相关的内容。)。

#### 相关概念

##### 并行扫描

在并行扫描中，每个Worker并行独立扫描数据表中的数据。Worker扫描产生的中间结果集将会返回给Leader线程，Leader线程通过Gather操作收集产生的中间结果，并将所有结果汇总返回到客户端。

##### 多表并行连接

并行查询会将多表连接操作完整的下推到Worker上去执行。PolarDB优化器只会选择一个自认为最优的表进行并行扫描，而除了该表外，其他表都是一般扫描。每个Worker会将连接结果集返回给Leader线程，Leader线程通过Gather操作进行汇总，最后将结果返回给客户端。

##### 并行排序

PolarDB优化器会根据查询情况，将ORDER BY下推到每个Worker里执行，每个Worker将排序后的结果返回给Leader，Leader通过Gather Merge Sort操作进行归并排序，最后将排序结果返回到客户端。

##### 并行分组

PolarDB优化器会根据查询情况，将GROUP BY下推到Worker上去并行执行。每个Worker负责部分数据的GROUP BY。Worker会将GROUP BY的中间结果返回给Leader，Leader通过Gather操作汇总所有数据。这里PolarDB优化器会根据查询计划情况来自动识别是否需要再次在Leader上进行GROUP BY。例如，如果GROUP BY使用了Loose Index Scan，Leader上将不会进行再次GROUP BY；否则Leader会再次进行GROUP BY操作，然后把最终结果返回到客户端。

##### 并行聚集

并行查询执行聚集函数下推到Worker上并行执行。并行聚集是通过两次聚集来完成的。第一次，参与并行查询部分的每个Worker执行聚集步骤。第二次，Gather或Gather Merge节点将每个Worker产生的结果汇总到Leader。最后，Leader会将所有Worker的结果再次进行聚集得到最终结果。

##### 子查询支持

在并行查询下子查询有四种执行策略：

1、在Leader线程中串行执行

当子查询不可并行执行时，例如在2个表JOIN，在JOIN条件上引用了用户的函数，此时子查询会在Leader线程上进行串行查询。

2、在Leader上并行执行（Leader会启动另一组Worker）

生成并行计划后，在Leader上执行的计划包含有支持并行执行的子查询，但这些子查询不能提前并行执行（即不能采用Shared access)。例如，当前如果子查询中包括window function，子查询就不能采用Shared access策略。

3、Shared access

生成并行计划后，Worker的执行计划引用了可并行执行的子查询，PolarDB优化器会选择先提前并行执行这些子查询，让Worker可以直接访问这些子查询的结果。

4、Pushed down

生成并行计划后，Worker执行计划引用了相关子查询，这些子查询会被整体推送到Worker上执行。

### 并行执行EXPLAIN

#### 查询用表

本文示例中使用pq\_test表进行并行查询测试，其中：

表结构如下：

mysql> SHOW CREATE TABLE pq\_test\G

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Table: pq\_testCreate Table: CREATE TABLE `pq\_test` (

`id` BIGINT(20) NOT NULL AUTO\_INCREMENT,

`help\_topic\_id` INT(10) UNSIGNED NOT NULL,

`name` CHAR(64) NOT NULL,

`help\_category\_id` SMALLINT(5) UNSIGNED NOT NULL,

`description` TEXT NOT NULL,

`example` TEXT NOT NULL,

`url` TEXT NOT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=21495809 DEFAULT CHARSET=utf81 row in set (0.00 sec)

表大小如下：

mysql> SHOW TABLE STATUS\G\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Name: pq\_test

Engine: InnoDB

Version: 10

Row\_format: Dynamic

Rows: 20064988

Avg\_row\_length: 1898

Data\_length: 38085328896Max\_data\_length: 0

Index\_length: 0

Data\_free: 4194304

Auto\_increment: 21495809

Create\_time: 2019-07-30 01:35:27

Update\_time: NULL

Check\_time: NULL

Collation: utf8\_general\_ci

Checksum: NULL

Create\_options:

Comment:1 row in set (0.02 sec)

查询SQL如下：

SELECT COUNT(\*) FROM pq\_test;

#### EXPLAIN查询语句

通过EXPLAIN语句查看不使用并行查询的情况。查询语句如下：

SET max\_parallel\_degree=0; EXPLAIN SELECT COUNT(\*) FROM pq\_test\G

查询结果如下：

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Id: 1

Select\_type: SIMPLE

Table: pq\_test

Partitions: NULL

Type: indexPossible\_keys: NULL

Key: PRIMARY

Key\_len: 8

Ref: NULL

Rows: 20064988

Filtered: 100.00

Extra: Using index1 row in set, 1 warning (0.03 sec)

通过EXPLAIN语句查看使用并行查询的情况，查询语句如下：

EXPLAIN SELECT COUNT(\*) FROM pq\_test\G

查询结果如下：

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Id: 1

Select\_type: SIMPLE

Table: &lt;gather2&gt;

Partitions: NULL

Type: ALLPossible\_keys: NULL

Key: NULL

Key\_len: NULL

Ref: NULL

Rows: 20064988

Filtered: 100.00

Extra: NULL

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 2. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Id: 2

Select\_type: SIMPLE

Table: pq\_test

Partitions: NULL

Type: indexPossible\_keys: NULL

Key: PRIMARY

Key\_len: 8

Ref: NULL

Rows: 10032494

Filtered: 100.00

Extra: Parallel scan (2 workers); Using index2 rows in set, 1 warning (0.00 sec)

从上述结果可以看出：

上述并行计划中包含了Gather操作，该操作负责汇总所有Worker返回的中间结果。

从执行计划输出的Extra信息中可以看到pq\_test表使用了Parallel scan（并行扫描）策略，期望用2个Workers来并行执行。

通过带有子查询的EXPLAIN语句查看使用并行查询的情况，查询语句如下：

EXPLAIN SELECT a, (select sum(t2.b) from t2 where t2.a = t1.b) FROM t1 WHERE (a, b) IN (SELECT b, MAX(a) FROM t2 GROUP BY b)\G

查询结果如下：

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

id: 1

select\_type: PRIMARY

table: <gather1>

partitions: NULL

type: ALLpossible\_keys: NULL

key: NULL

key\_len: NULL

ref: NULL

rows: 2

filtered: 100.00

Extra: NULL

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 2. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

id: 1

select\_type: PRIMARY

table: t1

partitions: NULL

type: ALLpossible\_keys: NULL

key: NULL

key\_len: NULL

ref: NULL

rows: 2

filtered: 100.00

Extra: Parallel scan (1 workers); Using where

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 3. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

id: 2

select\_type: DEPENDENT SUBQUERY

table: t2

partitions: NULL

type: ALLpossible\_keys: NULL

key: NULL

key\_len: NULL

ref: NULL

rows: 2

filtered: 50.00

Extra: Parallel pushdown; Using where

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 4. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

id: 3

select\_type: SUBQUERY

table: <gather3>

partitions: NULL

type: ALLpossible\_keys: NULL

key: NULL

key\_len: NULL

ref: NULL

rows: 1

filtered: 100.00

Extra: Shared access; Using temporary

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 5. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

id: 3

select\_type: SIMPLE

table: t2

partitions: NULL

type: ALLpossible\_keys: NULL

key: NULL

key\_len: NULL

ref: NULL

rows: 2

filtered: 100.00

Extra: Parallel scan (1 workers); Using temporary5 rows in set, 2 warnings (0.02 sec)

从上述结果可以看出：

select\_type为SBUQUERY的子查询中，Extra显示Parallel pushdown，表示该子查询使用了Parallel pushdown策略，即整个子查询被整个下推到Worker去执行。

select\_type为DEPENDENT SUBQUERY的子查询中，Extra显示Shared access，表示该子查询使用了Shared access策略，即PolarDB优化器选择提前并行执行该子查询并将执行结果Share给所有Worker。

### 并行度控制策略

PolarDB支持通过parallel\_degree\_policy参数来设置并行查询中并行度的配置策略。本文将介绍相关参数说明。

#### 前提条件

集群版本需为PolarDB MySQL且修订版本为8.0.1.1.11或以上，详情请参见[查询版本号](https://help.aliyun.com/document_detail/172561.htm" \l "section-9of-fn2-p5v" \o ")。

#### 并行度控制参数

|  |  |  |
| --- | --- | --- |
| 参数 | 级别 | 说明 |
| parallel\_degree\_policy | Global | 设置单个查询的并行度配置策略，取值范围如下：  TYPICAL  AUTO  REPLICA\_AUTO（默认）  说明 更多关于并行查询的使用方式，请参见[并行查询（Parallel Query）](https://help.aliyun.com/document_detail/128615.htm" \l "concept-1563422" \o "8.0重磅推出并行查询框架，当您的查询数据量到达一定阈值，就会自动启动并行查询框架，从而使查询耗时指数级下降。)。 |

##### TYPICAL策略

TYPICAL策略模式下，PolarDB选择查询并行度时不会考虑数据库负载（如CPU使用率等），而尽可能与max\_parallel\_degree设置的并行度保持一致。

##### AUTO策略

AUTO策略下，PolarDB会根据数据库的CPU、内存或IOPS资源的使用率来决定是否禁止并行查询计划，并支持在需要并行执行的前提下，自定义并行查询的并行度选择策略。

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 级别 | 取值 | 说明 |
| loose\_auto\_dop\_cpu\_pct\_hwm | Global | 取值范围：0~100  默认值：70 | CPU使用率阈值。若CPU使用率超过阈值，PolarDB会禁止并行查询计划。 |
| loose\_auto\_dop\_mem\_pct\_hwm | 取值范围：0~100  默认值：90 | 内存使用率阈值。若内存使用率超过阈值，PolarDB会禁止并行查询计划。 |
| loose\_auto\_dop\_iops\_pct\_hwm | 取值范围：0-100  默认值：80 | IOPS使用率阈值。若IOPS使用率超过阈值，PolarDB会禁止并行查询计划。 |
| loose\_auto\_dop\_low\_degree\_cost | 取值范围：0~18446744073709551615  默认值：500000 | 自动并行度选择策略。在确定需要并行执行后，PolarDB会根据如下标准选择查询并行度：  当优化器估算查询的串行执行代价低于该值时，查询并行度为2。  当优化器估算查询的串行执行代价大于或等于该值时，查询并行度将尽可能与max\_parallel\_degree设置的并行度保持一致。  说明 该参数仅用于在PolarDB确定需要并行执行后，选择查询的并行度，而不用于选择是否需要并行执行。 |

##### REPLICA\_AUTO策略

REPLICA\_AUTO策略下，仅只读节点会根据数据库的CPU、内存或IOPS使用率决定是否禁止并行查询计划，而主节点不会选择并行执行计划。支持的配置参数与[AUTO策略](https://help.aliyun.com/document_detail/203022.html?spm=a2c4g.11186623.6.735.45a6442cl1PvuY" \l "section-nn7-ny7-jy8" \o ")的一致。

### Hash Join的并行执行

#### 前提条件

集群版本需为PolarDB MySQL 8.0集群版，且Revision version为8.0.2.1.0或以上，您可以参见[查询版本号](https://help.aliyun.com/document_detail/178903.htm" \l "section-9of-fn2-p5v" \o ")确认集群版本。

#### 使用方法

语句

目前在PolarDB中Hash Join只能通过EXPLAIN FORMAT=TREE语句来显示。

示例

如下示例中创建了2个表，且在表里插入了一些数据：

CREATE TABLE t1 (c1 INT, c2 INT);CREATE TABLE t2 (c1 INT, c2 INT);INSERT INTO t1 VALUES (1,1),(2,2),(3,3),(5,5);INSERT INTO t2 VALUES (1,1),(2,2),(3,3),(5,5);

如下是上述2个表的JOIN的查询计划：

EXPLAIN FORMAT=TREE

EXPLAIN

| -> Gather: 4 worker(s) (cost=10.60 rows=4)

-> Inner hash join (t2.c1 = t1.c1) (cost=0.81 rows=1)

-> Table scan on t2 (cost=0.09 rows=4)

-> Hash

-> Table scan on t1 with 0 parallel partitions (cost=0.16 rows=1)

上述是并行度为4的并行查询计划（即PolarDB会启用4个Worker来执行查询）。其中t1表会执行Parallel Scan，即由4个Worker分扫这个表，每个Worker使用t1的一部分数据建立各自的Hash表，再和整个t2表执行JOIN操作，最后收集（Gather）在Leader，得到整个查询的结果。

### Semi-Join的并行执行

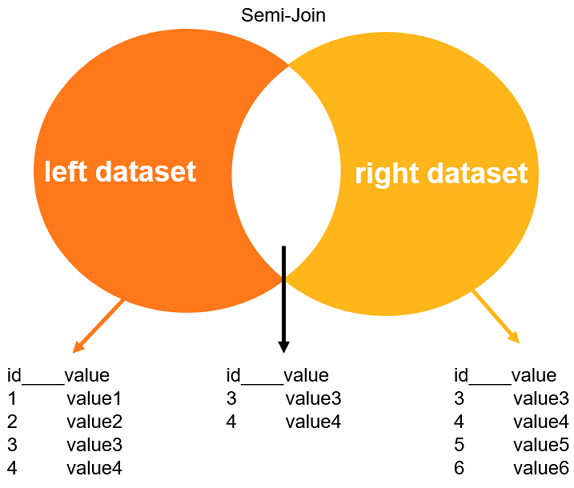
您可以使用Semi-Join半连接优化子查询，减少查询次数，提高查询性能，本文将介绍Semi-Join半连接的基本信息和操作方法。

#### 前提条件

PolarDB集群版本需为PolarDB MySQL 8.0且Revision version为8.0.1.1.2或以上，您可以参见[查询版本号](https://help.aliyun.com/document_detail/178903.htm" \l "section-9of-fn2-p5v" \o ")确认集群版本。

#### 背景信息

MySQL 5.6.5引入了Semi-Join半连接，当外表在内表中找到匹配的记录之后，Semi-Join会返回外表中的记录。但即使在内表中找到多条匹配的记录，外表也只会返回已经存在于外表中的记录。而对于子查询，外表的每个符合条件的元组都要执行一轮子查询，效率比较低下。此时使用半连接操作优化子查询，会减少查询次数，提高查询性能，其主要思路是将子查询上拉到父查询中，这样内表和外表是并列关系，外表的每个符合条件的元组，只需要在内表中找符合条件的元组即可，所以效率会大大提高。

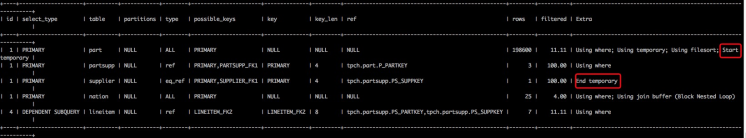
[](https://static-aliyun-doc.oss-accelerate.aliyuncs.com/assets/img/zh-CN/1288900161/p133452.png)

#### 策略

Semi-Join主要使用了如下策略：

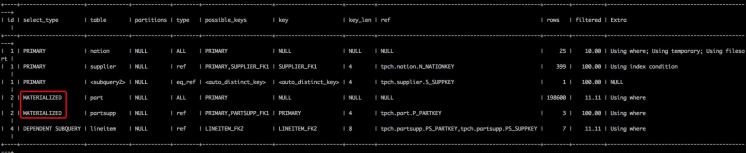
DuplicateWeedout Strategy

该策略创建由row id组成唯一ID的临时表，再通过该唯一ID达到去重目的。

[](https://static-aliyun-doc.oss-accelerate.aliyuncs.com/assets/img/zh-CN/2840359951/p133901.png)

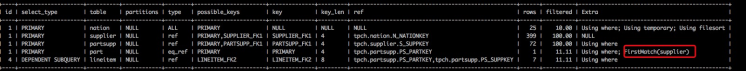
Materialization Strategy

该策略将nested tables物化到临时表中，再通过查找物化表或者遍历物化表查找外表的方法达到去重目的。

[](https://static-aliyun-doc.oss-accelerate.aliyuncs.com/assets/img/zh-CN/2840359951/p133902.png)

Firstmatch Strategy

该策略采用顺序查找表的方式，找到第一个匹配的记录后立即跳转到最后一个外表，并对外表的下一条记录执行JOIN操作，从而达到去重的目的

[](https://static-aliyun-doc.oss-accelerate.aliyuncs.com/assets/img/zh-CN/2840359951/p133903.png)

LooseScan Strategy

该策略对内表基于索引（Index）进行分组，分组后与外表执行JOIN（进行Condition的匹配）操作，如果存在匹配的记录，则提取外表的记录，内表选取下一个分组继续进行计算，从而达到去重目的。

[IMG_260](https://static-aliyun-doc.oss-accelerate.aliyuncs.com/assets/img/zh-CN/2840359951/p133904.png)

#### 语法

Semi-Join通常使用IN或EXISTS作为连接条件。

INSELECT \*FROM EmployeeWHERE DeptName IN (

SELECT DeptName

FROM Dept

)

EXISTSSELECT \*FROM EmployeeWHERE EXISTS (

SELECT 1

FROM Dept

WHERE Employee.DeptName = Dept.DeptName

)

#### 并行Semi-Join性能提升

对于选择Semi-Join策略的查询，PolarDB对Semi-Join所有策略实现了并行加速，通过拆分Semi-Join的任务，多线程模型并行运行任务集，强化去重能力，使查询性能得到了显著的提升，以Q20为例。

SELECT

s\_name,

s\_address FROM

supplier,

nation WHERE

s\_suppkey IN

(

SELECT

ps\_suppkey

FROM

partsupp

WHERE

ps\_partkey IN

(

SELECT

p\_partkey

FROM

part

WHERE

p\_name LIKE '[COLOR]%'

)

AND ps\_availqty > (

SELECT

0.5 \* SUM(l\_quantity)

FROM

lineitem

WHERE

l\_partkey = ps\_partkey

AND l\_suppkey = ps\_suppkey

AND l\_shipdate >= date('[DATE]')

AND l\_shipdate < date('[DATE]') + interval '1' year )

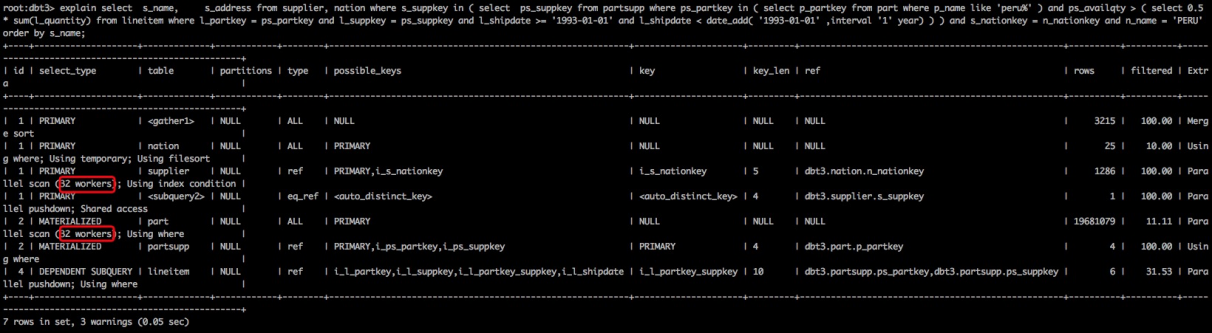
)

AND s\_nationkey = n\_nationkey

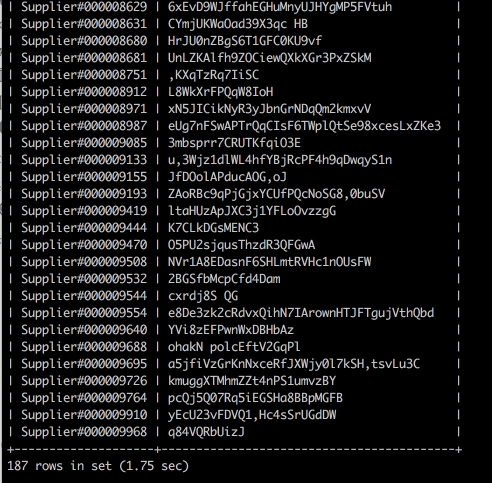
AND n\_name = '[NATION]' ORDER BY

s\_name;

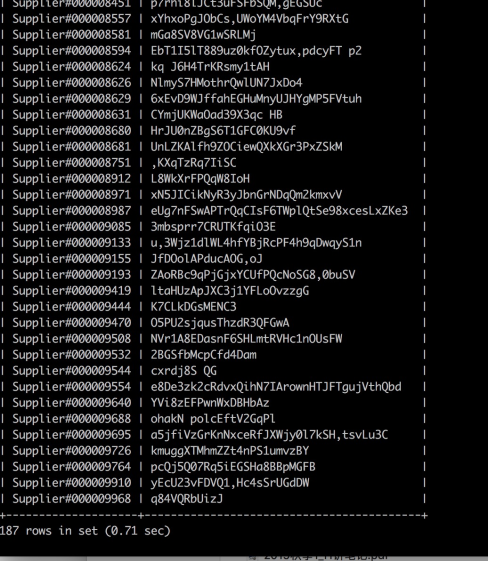
本文例子中将物化处理提前，并且以并行度（DOP）为32执行，后续的处理通过共享之前的物化表，同样充分发挥CPU的处理能力，将主查询的并行能力最大化，在如下的执行计划中，在标准TPCH SCALE为1G的数据量场景下，开启并行后，双层并行处理能力：

[](https://static-aliyun-doc.oss-accelerate.aliyuncs.com/assets/img/zh-CN/2840359951/p133533.png)

在标准TPCH SCALE为1G的数据量场景下，串行的执行时间：

[](https://static-aliyun-doc.oss-accelerate.aliyuncs.com/assets/img/zh-CN/3840359951/p136944.png)

并行开启情况下的执行时间：

[](https://static-aliyun-doc.oss-accelerate.aliyuncs.com/assets/img/zh-CN/3840359951/p136945.png)

在如下自定义SQL并行中，使用了Semi-Join下推的并行方式，在max\_parallel\_degree=32的情况下，执行时间从2.59s减少到0.34s：

mysql> SELECT c1,d1 FROM t1 WHERE c1 IN ( SELECT t2.c1 FROM t2 WHERE t2.c1 = 'f' OR t2.c2 < 'y' ) AND t1.c1 AND d1 > '1900-1-1' LIKE "R1%" ORDER BY t1.c1 DESC, t1.d1 DESC;

Empty set, 1024 warnings (0.34 sec)

mysql> SET max\_parallel\_degree=0;

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT c1,d1 FROM t1 WHERE c1 IN ( SELECT t2.c1 FROM t2 WHERE t2.c1 = 'f' OR t2.c2 < 'y' ) AND t1.c1 AND d1 > '1900-1-1' LIKE "R1%" ORDER BY t1.c1 DESC, t1.d1 DESC;

Empty set, 65535 warnings (2.69 sec)

mysql> EXPLAIN SELECT c1,d1 FROM t1 WHERE c1 IN ( SELECT t2.c1 FROM t2 WHERE t2.c1 = 'f' OR t2.c2 < 'y' ) AND t1.c1 AND d1 > '1900-1-1' LIKE "R1%" ORDER BY t1.c1 DESC, t1.d1 DESC;

+----+--------------+-------------+------------+--------+---------------+------------+---------+----------+--------+----------+---------------------------------------------------------+

| id | select\_type | table | partitions | type | possible\_keys | key | key\_len | ref | rows | filtered | Extra |

+----+--------------+-------------+------------+--------+---------------+------------+---------+----------+--------+----------+---------------------------------------------------------+

| 1 | SIMPLE | <gather1> | NULL | ALL | NULL | NULL | NULL | NULL | 33464 | 100.00 | Merge sort |

| 1 | SIMPLE | t1 | NULL | ALL | NULL | NULL | NULL | NULL | 62802 | 30.00 | Parallel scan (32 workers); Using where; Using filesort |

| 1 | SIMPLE | <subquery2> | NULL | eq\_ref | <auto\_key> | <auto\_key> | 103 | sj.t1.c1 | 1 | 100.00 | NULL |

| 2 | MATERIALIZED | t2 | p0,p1 | ALL | c1,c2 | NULL | NULL | NULL | 100401 | 33.33 | Using where |

+----+--------------+-------------+------------+--------+---------------+------------+------

### ROLLUP性能增强

您可以使用ROLLUP语法实现只借助一条查询语句，就计算出数据在不同维度上划分组之后得到的统计结果以及所有数据的总体值。本文将介绍如何使用ROLLUP语法。

#### 前提条件

PolarDB集群版本需为PolarDB MySQL 8.0且Revision version为8.0.1.1.0或以上，您可以参见[查询版本号](https://help.aliyun.com/document_detail/178903.htm" \l "section-9of-fn2-p5v" \o ")确认集群版本。

#### 语法

ROLLUP语法可以看作是GROUP BY语法的拓展，您只需要在原本的GROUP BY列之后加上WITH ROLLUP即可，例如：

SELECT year, country, product, SUM(profit) AS profit

FROM sales

GROUP BY year, country, product WITH ROLLUP;

除了产生按照GROUP BY所指定的列聚合产生的结果外，ROLLUP还会从右至左依次计算更高层次的聚合结果，直至所有数据的聚合。例如上述语句就会先按照GROUP BY year, country, product计算总收益，再按照GROUP BY year, country以及GROUP BY year计算总收益，最后按照不带任何GROUP BY条件计算整个sales表的总收益。

使用ROLLUP语法主要有如下优势：

方便对数据进行多维度的统计分析，简化原本对不同维度各自进行SQL查询的编程复杂度。

实现更快更高效的查询处理。

ROLLUP可以将所有的聚合工作转移到服务端，只用读一次数据就能够完成原本多次查询才能完成的统计工作，减轻了客户端的处理负载和网络流量。

### 通过Hint控制

Parallel Hints可以指定优化器是否选择并行执行，还支持指定并行度以及需要并行的表。PolarDB目前支持在并行查询中使用PARALLEL和NO\_PARALLEL两种Hints。

#### 开启或关闭并行查询

**开启并行查询**

您可以使用如下任意一种方式开启并行查询：

SELECT /\*+PARALLEL(x)\*/ ... FROM ...;   -- x >0

SELECT /\*+ SET\_VAR(max\_parallel\_degree=n) \*/  \*  FROM ...   // n > 0

**关闭并行查询**

您可以使用如下任意一种方式关闭并行查询：

SELECT /\*+NO\_PARALLEL()\*/ ... FROM ...;

SELECT /\*+ SET\_VAR(max\_parallel\_degree=0) \*/  \*  FROM ...

#### Hint高级用法

并行查询提供了PARALLEL和NO\_PARALLEL两种Hint，其中：

通过PARALLEL Hint可以强制查询并行执行，同时可以指定并行度和并行扫描的表。语法如下所示：

/\*+ PARALLEL [( [query\_block] [table\_name] [degree] )] \*/

通过NO\_PARALLEL Hint可以强制查询串行执行，或者指定不选择某些表作为并行扫描的表。

/\*+ NO\_PARALLEL [( [query\_block] [table\_name][, table\_name] )] \*/

其中参数说明如下所示。

|  |  |
| --- | --- |
| 参数 | 说明 |
| query\_block | 应用Hint的query block名称。 |
| table\_name | 应用Hint的表名称。 |
| degree | 并行度。 |

示例：

SELECT /\*+PARALLEL()\*/ \* FROM t1, t2;

-- 当表记录数小于records\_threshold\_for\_parallelism设置的行数 （ 默认10000行）时，会强制并行，并行度用系统默认max\_parallel\_degree， 如果max\_parallel\_degree > 0, 则打开并行，如果max\_parallel\_degree等于0时，依旧时关闭并行。

SELECT /\*+PARALLEL(8)\*/ \* FROM t1, t2;

-- 强制并行度8并行执行，当表记录数小于records\_threshold\_for\_parallelism设置的行数 （ 默认10000行）时，会强制并行，并行度设置max\_parallel\_degree为8。

SELECT /\*+ SET\_VAR(max\_parallel\_degree=8) \*/ \* FROM ...

-- 设置并行度max\_parallel\_degree为8，当表记录数小于records\_threshold\_for\_parallelism设置的行数（如20000行）时，会自动关闭并行。

SELECT /\*+PARALLEL(t1)\*/ \* FROM t1, t2;

-- 选择t1表并行, 对t1表执行/\*+PARALLEL()\*/ 语法

SELECT /\*+PARALLEL(t1 8)\*/ \* FROM t1, t2;

-- 强制并行度8且选择t1表并行执行，对t1表执行/\*+PARALLEL(8)\*/语法

SELECT /\*+PARALLEL(@subq1)\*/ SUM(t.a) FROM t WHERE t.a = (SELECT /\*+QB\_NAME(subq1)\*/ SUM(t1.a) FROM t1);

--强制subquery并行执行， 并行度用系统默认max\_parallel\_degree，如果max\_parallel\_degree > 0, 则打开并行，max\_parallel\_degree等于0时，依旧时关闭并行

SELECT /\*+PARALLEL(@subq1 8)\*/ SUM(t.a) FROM t WHERE t.a = (SELECT /\*+QB\_NAME(subq1)\*/ SUM(t1.a) FROM t1);

--强制subquery并行执行， 并行度设置max\_parallel\_degree为8

SELECT SUM(t.a) FROM t WHERE t.a = (SELECT /\*+PARALLEL()\*/ SUM(t1.a) FROM t1);

--强制subquery并行执行， 并行度用系统默认max\_parallel\_degree，如果max\_parallel\_degree > 0, 则打开并行，max\_parallel\_degree等于0时，依旧时关闭并行

SELECT SUM(t.a) FROM t WHERE t.a = (SELECT /\*+PARALLEL(8)\*/ SUM(t1.a) FROM t1);

--强制subquery并行执行， 设置并行度max\_parallel\_degree为8

SELECT /\*+NO\_PARALLEL()\*/ \* FROM t1, t2; -- 禁止并行执行

SELECT /\*+NO\_PARALLEL(t1)\*/ \* FROM t1, t2;

-- 只对t1表禁止并行， 当系统打开并行时， 有可能对t2进行并行扫描，并行执行

SELECT /\*+NO\_PARALLEL(t1, t2)\*/ \* FROM t1, t2;

-- 同时对t1和t2表禁止并行

SELECT /\*+NO\_PARALLEL(@subq1)\*/ SUM(t.a) FROM t WHERE t.a =

(SELECT /\*+QB\_NAME(subq1)\*/ SUM(t1.a) FROM t1);

--禁止subquery 并行执行

SELECT SUM(t.a) FROM t WHERE t.a =

(SELECT /\*+NO\_PARALLEL()\*/ SUM(t1.a) FROM t1);

--禁止subquery 并行执行

说明：对于不支持并行的查询或者并行扫描的表，PARALLEL Hint不生效。

并行子查询并行子查询的选择方式（并行子查询详细信息请参见[子查询支持](https://help.aliyun.com/document_detail/208545.htm" \l "section-xth-6e5-5yl" \o ")）也可以通过Hint来进行控制，语法及说明如下：

/\*+ PQ\_PUSHDOWN [( [query\_block])] \*/ 对应的子查询会选择push down的并行子查询执行策略

/\*+ NO\_PQ\_PUSHDOWN [( [query\_block])] \*/ 对应的子查询会选择shared access的并行子查询执行策略

示例：

#子查询选择push down并行策略

EXPLAIN SELECT /\*+ PQ\_PUSHDOWN(@qb1) \*/ \* FROM t2 WHERE t2.a =

(SELECT /\*+ qb\_name(qb1) \*/ a FROM t1);

#子查询选择shared access并行策略

EXPLAIN SELECT /\*+ NO\_PQ\_PUSHDOWN(@qb1) \*/ \* FROM t2 WHERE t2.a = (SELECT /\*+ qb\_name(qb1) \*/ a FROM t1);#不加query block进行控制EXPLAIN SELECT \* FROM t2 WHERE t2.a=(SELECT /\*+ NO\_PQ\_PUSHDOWN() \*/ a FROM t1);

### 并行查询限制与串行执行结果兼容

本文为您介绍并行查询的使用限制以及与串行执行结果可能不兼容的地方，帮助您正确使用并行查询功能。

#### 并行查询的使用限制

PolarDB会持续迭代并行查询的能力，目前以下情况暂时无法享受并行查询带来的性能提升：

查询系统表或非Innodb表。

使用全文索引的查询。

存储过程Procedures。

Recursive CTE。

Windows Functions（函数本身不能并行计算，但整个查询可以并行）。

GIS （函数本身不能并行计算，但整个查询可以并行）。

Index Merge。

串行化隔离级别事务内的查询语句。

#### 与串行执行结果可能不兼容的地方

错误提示次数可能会增多

串行执行中出现错误提示的查询，在并行执行的情况下，可能每个工作线程都会提示错误，导致总体错误提示数增多。

精度问题

并行查询的执行过程中，可能会出现比串行执行多出中间结果的存储，如果中间结果是浮点型，可能会导致浮点部分精度差别，导致最终结果有细微的差别。

网络包或者中间结果长度超出max\_allowed\_packet允许的最大长度

并行查询的执行过程中，相比串行执行可能会多出中间结果。如果中间结果的长度超出了max\_allowed\_packet定义的最大长度，可能出现错误提示，可以通过增加max\_allowed\_packet参数的值来解决。如何修改参数请参见[设置集群参数](https://help.aliyun.com/document_detail/72742.htm" \l "task-1580301" \o "PolarDB集群创建完成后，您可以通过控制台修改集群参数，并可以将修改后的集群参数导出为模板，同时同一地域的集群可以通过应用模板功能快速修改集群参数。本节介绍修改集群参数，将修改后参数导出为模板并应用模板的方法。)。

结果集顺序差别

当并行执行未加ORDER BY关键字的SELECT ... LIMIT n语句时，返回的结果集可能与执行顺序不一致。由于有多个Worker同时执行，每次执行时Worker的执行速度是不确定的，当Leader得到足够的数据后，就会返回结果，因此返回的结果集可能与执行顺序不一致。

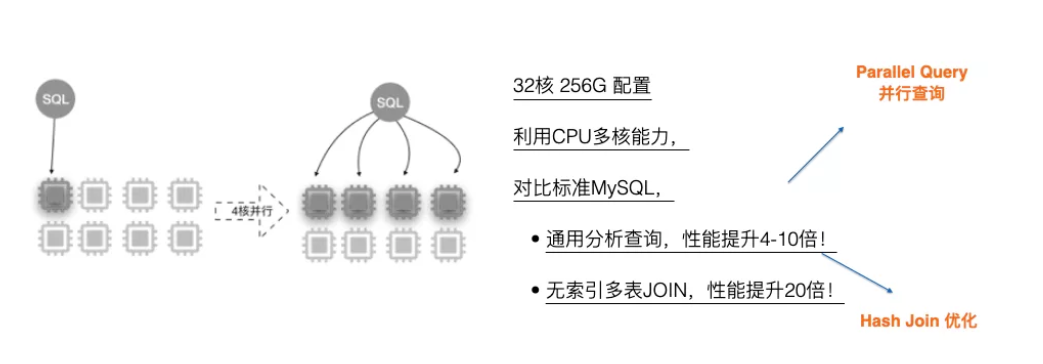
加了行锁的数据记录数增多

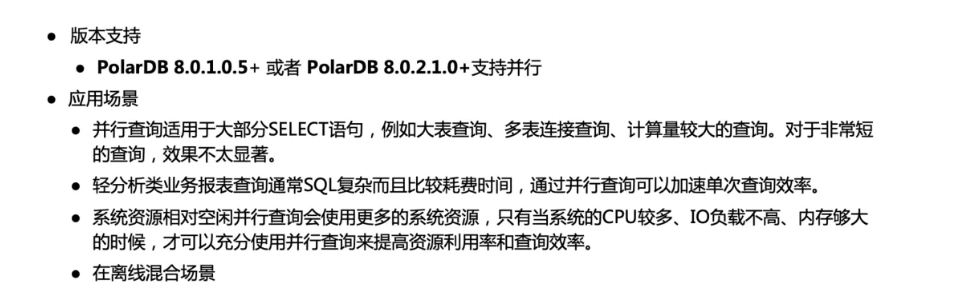
当并行执行SELECT ... FROM ... FOR SHARE语句时，InnoDB会将访问到的每一行数据都加锁，因此加了行锁的记录数可能会比非并行执行的情况下要多，这属于正常现象。

## 高并发优化

# HTAP

在PolarDB中支持Parallel Query，慢SQL Rewrite，In-memory Column Index（全内存列式索引）该方案对于OLAP计算节点，估计需要较大的内存，且MySQL版本要选择MySQL 8.0以上。





# 发展方向

## 硬件红利

## HTAP

## 智能服务