# 概述

## 简介

## 特性

### 多级SCALE-OUT扩展

实现各个系统的扁平化（水平扩展）：

1. 客户端接入层、中间件网关层、DB层三个层次上可横向线性扩展，满足性能及容量的何种处理需求；
2. 中间件网管层支持自动路由分配，根据业务需要可以灵活配置服务网关（proxy）的数量，实现处理能力线性扩展；
3. 网关层内置连接池实现链路共享，数据库连接数线性可扩展；
4. 数据自动切分分片，分片数据在DB根据策略自动重分布，存储容量随着DB节点数线性扩展。

### 强一致分布式事务

### 分布式优化器

内置大量的优化规则，对上百个场景进行优化，复杂SQL语句兼容性和处理性能好，同时支持prepare预处理、执行计划缓存、数据集透传等功能，保证数据一致性条件下实现高性能SQL处理。

支持的典型优化包括：

1. 分片剪枝
2. 合并下压
3. 并行执行
4. 条件下推及条件繁殖
5. 排序下推、limit下推等
6. 聚合函数优化

### 分布式批处理

提供分布式架构下批处理功能，满足金融、政企、运营商等行业日终大数据批处理需求，通过分布式FetchSize和存储过程对数据进行批处理，减少客户端与DB访问次数，批量返回数据集并进行批量处理。

### 读写分离

GoldenDB支持将DQL查询负载均衡到从库，提升系统资源利用和处理效率，读性能可线性扩展。

1. 对应用透明：应用无需改造，在数据库运维界面简单配置即可实现功能；
2. 可视化运维管理：支持在线调整权重，支持在同一个集群中不同业务使用不同的负载均衡模式；
3. SQL Hint提示：支持个别高实时性的SQL通过Hint指定至master执行。

注：

Hint是oracle提供的一种机制，用来告诉优化器按照告诉它的方式生成执行计划。

### 灵活数据切分技术

支持哈希、范围、列表、复制多种数据分片规则，可以根据业务数据特征，选择最合适的分片技术把数据分别存储在多个数据安全组中；通过合理的数据分片规则，发挥分布式数据库的最佳性能。

说明：所谓的切分就是把对应的表建立在多个节点后，下发的SQL语句需要分布在多个节点执行才可以，就需要将一个复杂的SQL拆分到多个节点上。

### 分布式数据备份恢复

支持快照恢复到任意时刻和恢复到快照生成时刻，且可以保证集群数据的强一致性。

### 在线数据重分布

支持数据节点的扩容、缩容，能够高效地将数据均匀分布到数据库集群上，同时保证对在线业务影响小、且可操作性强。

### 分部署数据导入导出

数据导入导出与交易分离，降低对联机交易的影响。

## 产品性能

GoldenDB通过分布式执行缓存、并发事务控制、数据复制优化等手段，对整体性能进行提升。在金融行业客户实测数据中，银行转账核心业务系统TPS能够达到4万（2017年），银行信用卡核心交易TPS10万（2020年），满足银行业务高并发低延时的特性。

# 架构

## 逻辑架构



### 数据库驱动

数据库驱动以集成的方式嵌入到具体应用中，和应用一起部署。GoldenDB支持通用数据库标准协议，并能够提供JDBC、ODBC、Java、Python、C、C++等常见开发语言使用的接口驱动。

从功能角度来看，数据库驱动实现了计算节点的透明接入、负载均衡和故障透明转移（LVS）。

分布式数据库的计算节点负责处理应用的SQL请求，计算节点集群由多个计算节点组成，用户可以规划计算节点和应用之间的对应关系（可以理解为一种负载均衡）。数据库驱动根据配置的规则将应用请求发送给合适的计算节点，并确保负载均衡地分配到这些计算节点上。当这些计算节点发生故障后，驱动层能够实施透明的故障转移，将应用的新请求发送给正常的计算节点，并在故障节点恢复后，能够将应用的请求重新路由到该节点上。

说明：

DRDCCtl：jar包，以数据服务的方式向应用提供数据库功能，并对应用请求或响应进行编解码。

DRDCSrv：将数据服务转换为一系列SQL请求，并发送到合适的目标端口上。

### 计算节点集群(Proxy)

计算节点集群是分布式数据库的核心层，由无状态的计算节点（Proxy）组成。计算节点从驱动层（业务操作）或者管理节点（内部管理命令）接收用户的操作（一般以结构化查询语言描述，即SQL），进行逻辑优化和物理优化，生成满足分布式事务一致性的分布式查询计划。计算节点在执行分布式查询计划时，通过持续地访问数据节点，完成用户的最终操作请求。用户可以根据应用对可靠性、可用性、性能等因素的不同要求，对计算节点进行合理的规划。

### SQL引擎

### 数据节点集群

数据节点集群是应用数据的最终存储组件。所有的数据节点组成一个或多个数据库集群，用户操作的事务尽量不要跨越多个数据库集群（目前已经支持跨集群，但是还是不够完善）。

数据库集群由一个或多个安全组（DBGroup）组成，集群中每个表中的数据按照某种策略进行横向分片后存放到对应的安全组中，分片策略包括复制策略、哈希策略、范围策略和列表策略。

数据按照上述策略（复制策略除外）分片后，每个安全组上的实际数据在理论上只有总数据量的1/N（数据分布的均匀程度依赖于切分策略和真实数据分布的匹配程度）。随着安全组数量的增加，每个安全组承载的数据量和读写负载会相应的减少，从而在数据节点集群内部具备了读写能力的水平拓展。

安全组是由一个或者多个数据节点组成的数据库节点组，组内的数据库节点拥有相同的数据。当安全组存在多个数据节点时，其中一个数据节点为主用节点其他数据节点都是备用节点，数据在主备节点之间实时复制。主用节点具备读写能力，备用节点可以提供度能力。

### 管理节点

管理节点在分布式数据库中负责集群管理流程，不涉及业务的访问流程，无负载压力，一般采用两节点主备方式部署。管理节点按照功能分工，可以有如下几个模块：

#### 统一运维管理(OMM)

OMM是GoldenDB分布式数据库的统一操作维护入口，可以进行用户和权限管理、元数据管理、计算节点管理、数据节点管理、DDL执行、节点扩容、备份恢复、统计及告警管理等。

#### 元数据管理器(MetaDtaServer)

元数据指的是数据的元信息，如库、表、视图、存储过程、触发器、自定义函数等数据模型的定义，元数据管理器存放系统的全量元数据，是整个分布式数据库集群的元数据中心。

此外，GoldenDB元数据管理器还保存了整个集群的拓扑信息，因此是更广义的元数据管理。

为了提高启动和运行的效率，除了元数据管理器存有元数据定义外，计算节点和数据节点也会存放元数据定义，但是计算节点和数据节点只存放本节点所涉及应用的元数据定义，即当计算节点中的元数据和管理节点不一致时，会同步管理节点的元数据到本地。

#### 计算节点管理(ProxyManager)

负责管理计算节点集群。管理工作一般分为两类：

一类为集群的组建管理，包括计算节点的创建、启用、禁用和删除；

另一类为集群的应用管理，包括定义计算节点和应用的对应关系、计算节点异常后的数据恢复调度。

#### 数据节点管理(ClusterManager)

数据节点管理也分为两类：

一类为集群的组建管理，包括数据节点、安全组、数据节点集群的创建、变更和删除；

另一类为集群的任务管理，包括数据节点异常、恢复后的调度管理、数据节点备份恢复的调度、数据重分布等功能的任务调度管理。

#### 全局事务管理节点(GlobalTransactionManager)

全局事务管理器在分布式数据库中维护全局事务的全生命周期，提供申请、释放、查询全局事务的能力，并采用双活方式部署。

## 物理架构

数据库驱动：可以直接使用标准数据库驱动接入GoldenDB集群，如JDBC，可以支持按照最快响应时间和均匀随机方式将SQL请求下发至多个计算节点，当前也可以使用常用的负载均衡设备LVS、F5等接入GoldenDB集群。

**计算节点Proxy：**计算节点理论上可以单机部署，但是通常要求部署2个以上，以便当某些计算节点发生故障后，驱动层能够实施透明的故障转移，将应用的新请求发送给正确的计算节点。

**数据库集群：**数据节点集群可以部署一个或多个，当业务之间的数据要求物理隔离时，可以将不同的业务数据存放在不同的数据节点集群；也可以多个业务的数据共享一个数据节点集群。每个集群内部的安全组数目可以根据业务量、访问性能要求、硬件条件确定；理论上每个安全组内部的副本数目越多，可靠性就越高，但是成本也越高。

**管理节点：**管理节点的各个模块可以单机部署也可以使用HA（现在已经使用ZK）软件进行双击冷备部署。

**全局事务控制节点：**分布式事务涉及节点间数据提交的一致性，该数据的一致性主要依赖于全局事务的状态，所以全局事务节点必须是高可靠的，全局事务控制节点为双活模式，在发生异常时可以进行服务快速接管。

计算节点集群具备横向扩展能力，数据节点集群具备横向和纵向扩展能力。在高性能要求下，分布式数据库集群会涉及大量的计算节点和数据节点。物理部署计算节点和数据节点时可以非常灵活，既可以部署在物理机器上，也可以部署在虚拟化平台上；同时每个节点既可以部署在一个设备上，也可以将多种类型的节点部署在同一设备上。不管采用哪种方式，一般都需要遵守如下原则：

1. 计算节点需要分配更高的计算资源，数据节点需要分配更高的内存和存储资源；
2. 归属于同一个应用的多个计算节点避免部署在同一台物理机器上，更进一步避免部署在同一机框或者数据中心；
3. 归属于同一个安全组的多个数据节点避免部署在同一台物理机器上，更进一步避免部署在同一机框或数据中心。

## 系统数据流向

# 原理

## 协作流程

### 元数据管理流程

#### DDL流程

具体步骤：



1. 用户在web界面查看cluster信息，如果集群状态正常则可实施DDL操作，选择连接实例执行DDL操作；
2. DDL操作被推送至MDS，对于文本类的DDL操作集合，OMM全部读取后，可以分包推送；
3. MDS接收全部DDL操作集合，将DDL操作入操作状态表，标记状态位置位；做简单的逻辑判断后回复用户响应（譬如判断集群是否正常）；同时将消息推送至PM；
4. PM收消息包后，向指定proxy推送DDL；
5. Proxy做语义分析和语法检查，将消息推送至DB上实施落地。对于DDL执行时间较长的情况，需要proxy定时返回中间响应给PM，告知DDL正在执行中，PM将中间响应返回给MDS，MDS更新操作记录表中最近一次响应消息的时间。PM和MDS上都设有定时器，收到中间响应会更新定时器；
6. DDL执行成功后，proxy发送响应给PM，携带DDL执行结果的元数据信息；
7. PM透传proxy的响应消息包给MDS；
8. MDS收到消息后，解析消息，更新数据库中记录；
9. DDL实施完全成功后MDS更新数据库中记录的标志位，同时通知PM去向相关Proxies推送元数据信息，如果推送某个proxy元数据失败，pPM告警；
10. 最终的操作结果由用户在web界面上查询获取。

#### 查询表结构



1. 界面展示集群表列表，用户选择指定表查询结构信息，OMM将消息下发至MDS；
2. MDS收到消息后，查表解析.frm文件；
3. MDS更新操作记录，返回结果给OMM。

### Proxy管理流程

#### 新增proxy

用户在OMM界面进行过添加设备操作后，用户便可以在proxy展示界面看到已经准备好的未分配使用的proxy节点，并可以选择将其加入proxy集群中，步骤如下：



1. proxy启动后向PM发心跳信息，PM不对其做任何处理；
2. 用户在OMM选择将未分配的proxy加入proxy集群；
3. OMM发消息通知MDS添加该proxy到集群中；
4. MDS将操作入库，标记状态位；同时将响应消息回送至OMM；
5. MDS将消息传送至PM；
6. PM查看用户指定IP的proxy是否心跳正常，同时通知proxy做一系列初始化工作；
7. Proxy初步判断准备就绪后，PM回复MDS添加proxy响应消息；
8. MDS同时入库更新本条记录；
9. 用户通过后续操作查看操作的实施结果。

#### Proxy启用和禁用

启用和禁用即proxy的上线和下线，如proxy升级的场景，在进行禁用当前客户端应该不再向本proxy发送新事务，当前正在执行的新事务执行完毕后，禁用操作完成。



具体步骤：

1. 用户在OMM发起将proxy禁用或启用的请求；
2. MDS读取该条proxy及所有连接实例的记录，检查当前是否有操作在进行，如果有则拒绝本地操作；佛足额允许发起操作并更新所有标志位；
3. MDS回复OMM响应消息；
4. PM将请求透传至proxy；
5. Proxy实施操作处理：
6. 禁用操作：禁用所有绑定的连接实例，即proxy不再接受新消息，当前在线事务需要处理结束；
7. 启用操作：核实连接池的状态等。
8. Proxy回送响应；
9. MDS根据操作实施记录：更新落地。

#### Proxy删除

删除proxy是指将该proxy在逻辑关系上从系统中移除。只有处于禁用状态的proxy才可以被删除。

具体步骤：

1. 用户在界面将某proxy删除；
2. OMM将消息发送至MDS；
3. MDS读取该条proxy及所有连接实例的记录，检查当前是否有操作在进行，如果有则拒绝本次操作；检查该proxy是否尚未启用或已经处于禁用状态；满足所有条件后允许发起操作并更新所有标志位；
4. 通知proxy操作实施；
5. MDS根据操作实施记录落地：更新或删除。

#### Proxy状态查询

具体步骤：

1. 每个proxy都实时上报统计和状态信息给PM；
2. PM再上报至MDS。根据不同的信息类别，分别由MDS入库，或者进一步上报OMM入库；
3. 用户在发起当前状态和统计信息查询时，OMMServer读取数据并展示。

### 连接实例管理流程

#### 新增/修改/删除连接实例

用户可以在界面为一个集群新增一个连接实例，即访问某数据库集群的对外端口号和访问用户名、密码信息。供后续的proxy和cluster的关联使用。同时用户也可以在界面修改或删除连接实例，前提是该连接实例未被任何proxy绑定。



具体步骤：

1. 用户在界面为集群增加、修改或删除一个连接实例；
2. MDS对其校验后入库持久化。

#### 绑定连接实例

连接实例的绑定、启用、变更、禁用、解绑的流程图如下所示：



具体步骤：

1. 用户在界面选择绑定proxy的连接实例。用户可以按照集群、连接实例层级展示出端口、用户名、密码、连接数等信息，用户可以指定连接实例的连接池数目以及前端连接数；
2. MDS在数据库里添加一条记录，对其标志位赋值；
3. MDS回复用户OMM响应消息；
4. MDS依次将消息推送至proxy实施；
5. Proxy受理后发送回复消息，依次回送至MDS；
6. MDS根据处理结果处理该条记录：更新落地；
7. 用户通过后续查询操作查看操作的实施结果。

注：连接实例的绑定时，proxy只是建立绑定关系，但不进行实际连接池的创建。

#### 启用连接实例

具体步骤：

1. 用户在界面选择启用响应proxy绑定的某连接实例；
2. MDS在数据库里添加一条记录，对其标志位赋值；
3. MDS回复用户OMM响应消息；
4. MDS依次将消息推送至proxy实施；
5. Proxy受理后发送回复消息，依次回送至MDS；
6. MDS根据处理结果处理该条记录：更新落地；
7. 用户通过后续查询操作查看操作的实施结果。

注：用户选择启用连接实例时，proxy进行连接池的创建。

#### 变更连接实例的连接数

具体步骤：

1. 用户在界面修改proxy的连接实例的连接数目，支持增加和减少。在减少的场景下，用户可以指定proxy处理方式：优雅减少，或强制减少。
2. MDS在数据库里查询出原纪记录，检查其当前状态标志位，如果正在执行更新动作则不处理当前用户请求；如果数据未发生变化则不处理当前用户请求；更新本记录的状态；
3. MDS回复用户OMM响应消息；
4. MDS依次将消息推送至proxy实施；
5. Proxy受理后发送回复消息，依次回送至MDS。对于实例数目减少的情况，proxy的路由模块应该控制不再有新发起的事务推送至该链路，在该链路上已有的事务处理结束后销毁该连接。如果用户指定为优雅方式，proxy等待链路上的消息处理结束后回复响应，同时可配置完成时间，如果在该时间内仍未处理完，则强制销毁连接；指定强制关闭时，不需要等待事务执行完毕就可以关闭；
6. MDS在数据库里做记录更新落地；
7. 用户通过后续查询操作查看操作的实施结果。

#### 禁用连接实例

在发起禁用操作时，客户端不再向该proxy的端口发起新事务。

具体步骤：

1. 用户在界面操作要将某proxy上的某监听实例禁用；
2. OMM将消息下发至MDS，最后落到proxy实施；
3. MDS检查该proxy及连接实例当前是否有操作在进行，如有则拒绝本次操作；否则更新标志位；
4. MDS回复用户OMM响应消息；
5. Proxy拒绝处理来自该端口的新消息，给发送回复消息，依次回送至MDS；
6. MDS根据返回结果做字段更新落地；
7. 用户通过后续查询操作查看操作的实施结果。

#### 解绑连接实例

在删除一个连接实例时应该将其禁用，通过状态查看保证其上面没有激活的全局事务后再将其彻底删除。

具体步骤：

1. 该实例已经被禁用；
2. 用户在界面将其proxy的连接实例删除；
3. OMM将消息下发至MDS；
4. MDS检查proxy及连接实例当前是否有操作在进行，如有则拒绝本次操作；检查此实例当前统计数据的激活事务数目为0则允许发起删除操作，更新记录标志位；
5. 通知proxy操作实施；
6. MDS根据操作实施记过进行该记录的处理：更新或删除；
7. 用户通过后续查询操作查看操作的实施结果；
8. 通过后续查询操作查看操作的实施结果。

### Cluster管理流程

#### 集群新增

用户可以通过界面新增一个集群，系统内部做集群名字和集群ID的映射关系。



具体步骤：

1. 用户新增一个集群，填写集群名称；
2. OMM下发消息至MDS；
3. MDS分配一个内部唯一ID；
4. MDS回复用户响应；
5. MDS将信息推送至CM。

#### 集群组建

集群组建的过程即为将DB添加到集群的过程：



具体步骤为：

1. 安装配置完各个DB节点后，DBAgent自动启动，并向CM发心跳消息；
2. 用户通过OMM进行集群组建，选择处于未分配状态的DB进行cluster组建，这些DB信息是在进行规划组网时录入OMMServer库的。支持group至DB的逐级引导添加，完成后发送消息至MDS；
3. MDS需控制在一次添加操作未结束时不处理新发起的请求；将操作所涉及的所有记录都存入相关表并做状态标记；
4. MDS回复用户中间响应；
5. MDS将消息下发至CM；
6. CM接收到请求后，从待分配列表中找到该DB信息，然后进行必要检查并协调各个DB进行对应的变更；
7. 各DB上的DBAgent接收配置变更及组网规划数据后，进行相应的检查或变更，如协调slave和master之间建立复制通道；处理结束后将结果返回给CM；
8. CM将集群组建结果恢复给MDS；
9. 集群组建成功后MDS将信息入库落地更新信息；
10. MDS通过PM将集群信息推送至各个proxy（这样后续就能获取最新的集群信息，但是这个不是从MDS获取的吗？我们这里应该不需要推送？）
11. 用户可以在界面查询集群组建情况及DB的状态。

#### Group内部主从切换

用户主动发起的主从切换流程如图所示：



具体步骤：

1. 用户在界面查看集群内DB状态正常，则可对某group发起主从切换；
2. MDS收到请求消息，从库查出该group记录，包含但不局限于以下动作：
3. 判断当前状态标志位，如果有其他的操作在进行，则拒绝本次操作；
4. 查看新主的状态，如果为异常，则拒绝此次用户操作；
5. 更新数据库的记录，置状态。
6. MDS回复用户中间响应；
7. MDS向CM发主从切换请求通知；
8. CM做进一步校验，并通知PM至相关proxy停止处理该cluster的新事务；
9. Proxy停止发起涉及该cluster的新事务，处理完所有涉及该cluster的当前在线事务后回复CM响应消息；
10. CM通知相关的DBAgent实施主从切换操作，结束后回送MDS响应消息；
11. MDS根据返回结果将消息持久化入库。

#### Cluster删除

删除只是解除该资源与系统的逻辑关系，不改变DB的运行状态。删除上层资源时，归属其的下层资源同样被处理。



具体步骤：

1. 业务不再发起新事务；
2. 对各proxy至该cluster的连接实例去激活；
3. 删除各proxy至该cluster的连接实例；
4. 用户在界面操作将该cluster删除，接触逻辑关系；
5. 元数据中心做保护，如果当前有proxy和该cluster有绑定关系，则拒绝该操作。

#### Cluster启用和禁用

专门的web界面实现集群启停，可以通过如下操作步骤同样达到目的：

1. 用户确保不再有新请求发送过来，则当前cluster相当于下线；
2. 进一步提高高可靠性，用户可以在界面将该cluster涉及的连接端口实例在proxy层面上全部下线；
3. 可以在proxy层面查看cluster所涉及连接端口实例的在线会话统计信息，确保整个集群无在线会话。



具体步骤如下：

1. 用户在界面对cluster发起启用、禁用请求；
2. MDS收到请求后，检查相关连接实例是否有操作进行，有则拒绝本次操作，没有则更新数据库记录，标记状态；
3. MDS向PM推送相关proxy连接实例进行启用、禁用请求；
4. PM通知相关proxy启用、禁用，proxy进行相关处理后，回复响应消息给PM；
5. MDS收到PM的响应回复后，更新数据库状态。

#### 集群信息查询

用户通过OMM可以实时查询集群的信息，包括集群状态及集群内部各DB信息等。

1. 每个DB都实时上报状态信息给CM；
2. CM再上报至MDS。根据不同的信息类别，分别由MDS入库，或进一步上报OMM入库；
3. 用户在发起当前状态等信息查询时，OMMServer读取数据并展示。

## 数据库客户端DRDC

DRDC和应用部署在一起，分为DRDCClt和DRDCSrv。二者的视图关系如下：



在某个DRDCSrv发生异常时，应不影响APP访问。

### DRDCClt

DRDCClt为jar包，以数据服务的方式向应用提供数据库功能，并对应用请求或响应进行编解码。

1. API均采用同步调用；
2. 使用统一的API类型，DRDCClt做成通用版本；
3. 查询的结果集暂时按照结+结果集的形式；
4. DDL语句均为web界面下发，只有临时表可以在运行态动态从客户端下发；
5. DRDCClt和DRDCSrv之间采用长链接的方式通讯，各连接轮询发送，如果本地DRDCSrv异常再通过其他DRDCSrv发出；
6. 客户端同时支持java和c版本。

### DRDCSrv

DRDCSrv将数据服务转换成一系列SQL请求，并发送到合适的目标端口上。

设计考虑：

1. DRDCSrv在转换SQL语句时考虑配置文件的方式。配置文件里有API名字的段，下面配置该事务执行的SQL语句；DRDCSrc支持启动时加载该配置和动态加载。API调用时将参数传入，DRDCSrv结合配置sql得到最终的执行sql语句。
2. DRDCSrv和proxy之间采用长链接方式进行直连，proxy通过端口号区分要访问的集群及访问使用的用户名和密码。
3. 在一个API内只有一个分布式事务。

## Proxy

Proxy是无状态的代理节点，负责SQL优化、SQL路由、数据节点的负载均衡、分布式事务的调度等；Proxy和上层的数据库客户端以及下层的数据局集群都是采用兼容MySQL的通讯协议。

### 逻辑架构



### 前后端通信协议及MySQL协议层

1. proxy对前端采用兼容mysql的协议，对后端采用mysql协议，需支持mysql的协议处理；
2. Proxy对前端采用TCP长链接，客户端一次性将语句下达至proxy；
3. Proxy对后端采用连接池的方式处理，连接池的数量及用户密码可配置，和端口是一一对应的关系；
4. Proxy根据应用访问的端口号来选择对应的集群以及具体的连接；
5. Proxy连接池耗尽后是否可以动态申请可以配置；
6. Proxy对上做局部流控，如果当前连接数超过配置的最大连接数，要等某连接上的在线事务全部处理结束后再关闭该连接。

### 元数据缓存及持久化

为减少proxy频繁访问元数据给MDS所造成的压力，proxy设置有元数据缓存模块，全量缓存本proxy所涉及的Cluster对应的元数据，具体包括但不局限于：

1. 库表定义以及数据分布信息，存储过程及编译后的语法树；
2. 集群信息；
3. 业务对应的库表统计信息。

元数据缓存模块是能够高效的对元数据更新和查询：

1. 接受推送的元数据变更；
2. 接受proxy内部模块元数据查询。

为了增强系统的健壮性，proxy具备元数据持久化功能，元数据更新时能够实时将元数据信息落地，以便在MDS发生异常时，proxy重启后能使用旧的元数据继续正常对外服务。

### SQL解析

SQL解析模块主要是将文本格式的SQL语句解析成诶不语法树，该步骤只做语法检查。

### SQL处理及优化

SQL优化模块主要是对普通的语法树结合元数据及统计信息进一步分析，生成满足分布式事务场景的分布式语法树。该语法树包括全量信息，SQL执行时不再访问元数据。

考虑的优化包括但不局限于如下实现：

1. 尽可能将条件信息推送到叶子节点；
2. 做条件优化；
3. 子查询优化，将能够满足优化条件的子查询转换为JOIN；
4. Order by、group by优化；
5. Limit优化；
6. 函数优化；
7. 分布式动作；
8. 支持explain查看执行计划。

### SQL执行

根据语法树各个节点的指令动作执行对应的动作，并完成执行结果的分析和处理。

1. 外部交互类：如何GTM的交互，和SQL路由模块的交互；
2. 内部处理类，如排序、分组、join等。

### SQL路由

SQL路由模块负责选择一个连接，将语句发送至合适的DB节点上执行。

### 配置加载及处理模块

配置加载及处理模块蛀牙负责接收统一管理传递过来的配置变更请求，并实施这些变更。

## GTM

全局事务协调中心，用于协助Proxy进行分布式事务管理，主要包括全局事务ID的生成和活跃事务的维护以及当前活跃GTIDs的快照。

### 逻辑架构



消息分发模块：负责将收到的外部请求消息根据clusterid转发至相应的消息处理模块。

消息执行模块：负责处理收到的外部请求消息，并进行GTID的维护，同时将内存消息同步至备机模块。

数据持久化模块：后台定时将激活的GTIDs持久化，同时需要考虑其他手段降低GTIDs丢失的概率，但仍可能丢失一部分最新的GTIDs，如果GTM异常，则需将大于已持久化的GTIDs的所有分布式事务回退，这样可能会误差一些已经结束的分布式事务，但是却是保证数据强一致性的很安全的方案。

### 设计要求

#### 性能要求

为提高GTM全局争抢GTID的性能，GTM的线程结构设计为一个分发线程加若干可配置的执行线程。

1. 消息分发模块根据请求消息中携带的clusterid将其分发至固定的执行线程
2. 每个执行线程对应的clusterid维护一套独立的GTIDs
3. GTM应能根据clusterid和proxyid索引到GTIDs

#### 可靠性设计

GTM采用双活的模式，双机之间有心跳机制，双机内存同步更新。

脑裂问题的解决：引入第三方监控，在主备机之间发生网络故障时，由第三方决定。认为不存在双机之间及双机和第三方之间均出现网络故障的情况。

### 典型流程

对外支持的功能包括：

1. 接收处理proxy的create GTID请求，顺序递增生成全局唯一的事务ID；
2. 接收处理proxy的release GTID的请求，释放对应的全局事务ID；
3. 接收处理proxy的查询归属于特定集群的当前激活的GTIDs请求；
4. 接收处理proxy的查询归属于特定proxy的GTIDs的请求。

GTM仅维护当前激活的GTIDs，其属性值包括但不局限于GTID归属的proxy、clusterid信息。

## MetaDataServer

下文所述元数据是一个广义的概念，它在普通意义上的数据库元数据概念基础上增加了集群的结构管理信息。

元数据管理模块负责对元数据创建、修改、查询、删除、持久化等的管理公国；采用主备方式提供高可靠性和高可用性。

### 逻辑架构

MDS主要有如下的几种类型的数据：

1. 数据字典。

如表定义、索引定义、存储过程定义、函数定义等数据库对象定义信息。

1. 分片信息。

当表需要分布式存储时，就需要定义表的分布策略信息，这些形成分片信息，用于描述表数据分布的算法、分布到哪些节点等。

1. 数据库集群信息。

集群基本信息，如集群ID；集群内部各个DB节点信息。

1. Proxy集群信息。

如proxy和Cluster连接实例的对应关系等。

元数据持久化功能：MDS保存有最全的信息，需要实时持久化，底层挂载mysql数据库。

查询及推送功能：支持PM、CM等模块的元数据查询功能。

变更功能：支持CM、OMMServer等模块发起的元数据变更，并控制变更过程，协调各个模块有序完成元数据的变更。

### 设计要求

#### 可靠性设计

MDS采用主备的部署，每台机器上均部署mysql数据库，主备之间使用半同步方式保持数据一致性。在主机上的进程或数据库发生异常时，切换到备机。

### 典型流程

#### 实施proxy集群管理

主要指实施OMMServer发送过来的用户的proxy集群管理命令，包括proxy的增加、删除、启用及禁用。

#### 实施 连接实例管理

主要是指实施OMMServer发送过来的用户的连接实例管理命令，包括连接实例的增删改、连接实例的绑定、启用、禁用、变更、解绑。

#### 实施cluster集群管理

主要指实施OMMServer发送过来的用户的cluster集群管理命令。

#### 接收数据信息查询

MDS可以接受回复各其他模块发起的元数据查询请求。

## ProxyManager

Proxy集群管理模块，包括proxy的发现、异常、查询，并接收处理MDS传递过来的用户操作。

### 逻辑架构



Proxy命令管理处理模块：负责将接收到的MDS的操作命令进行具体实施。

Proxy心跳维护模块：负责对proxy状态进行监控。

Proxy异常接管处理模块：负责在proxy发生异常时自动触发异常proxy的在线活跃事务回滚。

元数据处理：负责接收外界的元数据增量变更请求；同时可以根据任务定期去MDS获取全量数据。

### 典型流程

#### 异常处理

Proxy发生异常，其和PM的心跳异常，PM发起异常处理流程：

1. PM通知和当前异常proxyA集群上对应proxy发起回滚流程，图中以某proxyB为例；
2. ProxyB去GTM查询该异常proxy对应集群上的活跃的GTIDs忙活去其需要处理的部分，向对应cluster发起回滚流程。



#### 系统全部异常时的恢复机制

PM开放出接口供运维工具使用。当系统全部异常掉电重启后，通过该工具触发，PM能够通知各个proxy发起全局事务回滚。

## ClusterManager

数据库集群管理模块，负责配合实施集群的管理，包括集群发现、集群组建、集群变更、集群查询、集群监控等功能。使用双活机制确保高可用性。

Cluster的管理较proxy管理更加复杂：

1. 设计多层管理（cluster、group、db）
2. 在发生自动主从切换时CM和OMMServer存在同时操作资源的情况：
3. 表设计上需要对DB的状态位和操作标志位细化考量；
4. CM在发起自动主从切换时，发消息至MDS进行操作标志位申请。

### 逻辑架构



Cluster命令管理处理模块：负责将接受到的MDS的操作命令进行具体实施。

DB心跳维护模块：负责对各个DB状态进行监控。

DB异常接管处理模块：负责在DB发生异常时自动触发主从切换流程。

元数据管理：负责接收外界的元数据增量变更请求；同时可以根据任务定期去MDS获取全量数据。

### 典型流程

#### 集群监控

当集群内部某DB发生异常时，CM能够感知DBAgent的心跳异常，触发主备切换，集群里的master访问地址信息发生变化。

基本流程：

1. master出现异常，DBAgent对其尝试重启恢复失败；
2. DBAgent上报CM DB异常；
3. CM进行告警，同时判断若其具备主备切换条件，通知相关的proxy停写，通知MDS执行主备切换；
4. CM通知slave切换为master；
5. Slave完成配置调整和binlog replay后回复CM切换结果；
6. CM将集群信息变化通知给MDS；
7. CM通知PM去向向硅谷那proxy推送集群变更信息。

## DBAgent

部署在DB节点上，负责DB的配置更改、复制变更、状态监控、统计上报等，同时配合实施事务回退、集群管理、数据备份、数据迁移等功能，DBAgent使得这些功能落地。

### 逻辑架构

### 典型流程

#### 告警和统计信息上报

1. DBAgent定时检测DB的状态，如果发生异常则尝试对其进行恢复，几次尝试失败后则上报CM DB异常；
2. DBAgent定时将DB上搜集的统计信息上报CM。

#### 主备切换

#### 已提交事务回滚

已提交事务的回滚是分布式事务方案能保证数据一致性的关键步骤，是DBAgent的最重要功能之一。流程操作如下：

1. proxy通知DBAgent进行某GTID事务的回滚；
2. DBAgent分析读取binlog通过GTID定位到该事务的日志；
3. 根据日志中记录的新旧值构造反向SQL，对已提交事务进行回滚。

说明：

1. 搜索GTID高效；
2. 支持对指定GTID的事务及将大于指定GTID的事务回滚两种模式；
3. 不重复回滚；
4. 回滚失败时能够输出易于对账的日志，当找到该binlog，但无法实施回滚时，包括但不局限于以下场景：
5. 该记录发生了写写冲突，该记录在commit和rollback之间被其他事务修改
6. 该binlog涉及事务是一条delete语句，无法实施回滚

在发生对已提交事务回滚失败的场景时，需要将该事务涉及的表禁用，暂停对这些表的访问，具体分为如下场景：

1. 系统正常运行时，proxy中保留有全局事务的完整信息，可以发起对涉及的表的禁用操作；
2. 在系统发生异常时，如proxy异常，接管发起分布式事务回滚的proxy应能够从DBAgent上报的回滚失败语句中获取该事务所涉及的表，并对其发起禁用流程；



在人工干预恢复后，在界面发起接触表禁用操作，恢复对该表的访问：



## DataManager

数据控制模块，主要负责数据重分布的管理、数据备份与恢复的管理。

### 备份与恢复

优先级提升。

### 数据重分布

在系统扩容，增加新的数据节点或表的原有数据分布策略不合理，导致性能问题、热点数据问题等场景下需要对原有数据进行重分布。

在决策需要对数据重分布时需要数据分布情况及访问情况的统计分析。

## ClusterReplication

集群复制模块，用于集群间数据复制，保证数据一致性。

## 模块间统一通讯

统一消息通信是ZXOS统一支撑的核心组件，完成消息线程的建立、调度、管理、回收等，负责底层的通信链路管理包括端口的侦听、建链、数据的发送和接收、多条并发链路的管理等。

## OMMServer

OMMServer通过web节点提供人机交互功能，具体包括集群规划、集群变更、元数据操作、权限管理、告警与监控、日志分析等功能。

OMMServer与MetaDataServer共用一个数据库，但实行分库管理，各自维护自己的数据；OMMServer会根据需要读取MetaDataServer的库做展示，但是不会对其进行写操作，要做好权限控制。

# 操作

# 分布式事务

分布式事务和单机事务一样，都需要保证事务的ACID四个属性。在分布式环境下，由于网络通信延迟、故障和服务器故障等因素，保证事务一致性尤其具有挑战性。目前业界分布式数据库的数据一致性解决方案为强一致性、弱一致性和最终一致性。GoldenDB实现了强一致性的解决方案，即当用户完成数据更新之后，后续所有读操作都能且只能读到更新以后的值，这是事务一致性的最高级别。

注：强一致性虽然安全性更高，但是也存在性能的问题，即在出现故障时回滚代价非常高。

## 方案

在GoldenDB分布式事务解决方案中，数据节点作为分布式事务参与者要保证对自身数据库操作的本地事务满足ACID属性，计算节点作为分布式事务协调者协调多个本地事务完成整个分布式事务控制，同时将全局事务状态实时记录在全局事务管理器中。

GoldenDB分布式事务是借鉴普通事务实现，与普通事务的区别在于GoldenDB会将所有表中增加gtid隐含列，该列在分布式事务中存放该全局事务ID（每个分布式写事务都会得到一个全局唯一且递增的整型值），在执行完所有SQL后，所有DB节点统一完成commit。如果有DB节点commit提交失败，所有节点的DBAgent通过解析binlog中是否含有该异常事务的全局事务ID来构造反向回滚语句，并向DB下发该反向语句，进行事务回滚，从而保证分布式事务的数据原子性。

基于GTID的分布式事务管理方案架构图：



采用全局事务ID对分布式事务进行控制，其核心思想是全局事务控制和标签数据。前者为每一个分布式写事务分配一个全局唯一的有序事务ID（GTID），并根据事物的存活情况维护对应ID的生命周期。后者在用户表中增加对应用透明的GTID列，并在该列中维护操作本行数据的最近一次分布式事务对应的全局事务ID。

总体在流程交互上为一阶段提交+补偿事务的方式，如果事务在提交阶段有部分节点失败，本方案将采用回滚已成功提交的事务。

## 设计

### 一致性设计

GoldenDB分布式数据库利用数据库普通事务来代替分布式事务，在所有的表中增加隐含列，存放该全局事务ID。在执行完所有的SQL后统一进行commit，各个DB节点的隔离级别需要设置为RC（读已提交）。



create table t1(col1 int,col2 varchar(20));

proxy转化为：create table t1(col1 int,col2 varchar(20),gtid int);

insert into t1 values(xx,’xx’)

proxy转化为：insert into t1 values(xx,’xx’,current\_gtid);

update table t1 set col1=yyy

proxy转化为：update table t1 set col1=yyy,gtid=current\_gtid;

select col1 from t1;

proxy转化为：select col1,gtid from t1;

说明：在创建表中增加隐含列gtid，每一个分布式事务分配一个全局唯一的事务ID，插入或者更新操作会更新gtid列为本地事务ID；为了保证数据一致性读，需要插叙GTID列。

### 可靠性设计

#### 与GTM交互异常

如果是网络异常导致的链路闪断、消息丢失，定时器超时后会重试，超时时间和尝试次数可配，在超时尝试时间内网络恢复可以保证事务不失败。

GTM采用双机双活高可用机制，备机异常不影响业务，主机异常后，HA主备切换，Proxy会定时尝试备机一定次数，尝试时间间隔和尝试次数可配，尝试间隔\*尝试次数>HA切换时间可以保证事务不失败。

如果GTM双机异常，为整个分布式集群的数据强一致性，需将GTM持久化数据里的活跃事务列表及比最大GTID还大的所有事务全部回滚掉。

#### 部分节点提交失败数据一致性保证

当所有SQL执行结束后，proxy再向各个DB发起commit操作，此时如果有的节点提交失败有的提交成功，会造成数据不一致的情况；这种情况下proxy触发提交成功节点的DBAgent进行已提交事务回滚，回滚成功后才能向GTM释放GTID。

#### DB节点异常数据一致性保证

DB节点发生crash时，DB进行主备切换，在新主完成binlog replay的这个时间段该DB节点无法对外提供服务。对于DB异常有如下三种场景需要考虑：

1. 所有涉及该DB节点的新事务无法执行；
2. 所有涉及该DB节点已经执行还未提交的事务会给本事务涉及的DB节点发起回滚；
3. 事务正在提交节点DB异常，提交成功的节点会触发已提交事务回滚；

DB长时间无响应，OS会检测链路长时间无消息会自动断链，默认是7小时。

#### Proxy节点异常数据一致性保证

分布式事务正常运行时，proxy发生异常crash，此时proxy与所有db集群和客户端连接都会中断，客户端可以通过与集群下其他proxy建立连接进行数据库集群访问，不会影响发起的业务；但是未结束的残留事务有如下两种场景需要考虑：

1. DB上未提交的事务，由于与数据节点断链，MySQL会自动回滚；
2. Proxy异常宕机时，部分DB节点提交成功个，部分节点未提交，由于GTID不会释放，其他事务访问异常数据会挂住，知道这些数据被回滚掉。

## 隔离级别

### 单机事务隔离级别

### 分布式事务隔离级别

## 特点

GoldenDB的分布式事务实现方案最大的特点即为其在获得事务强一致性的同时还能具备较高的性能，可以与常用的两（三）阶段提交做对比：

两（三）阶段提交的核心思想是通过前期的多次准备和协调，尽可能的让最后的提交操作能够成功。而在实际场景中SQL执行成功，最后commit阶段失败的概率极低。本方案认为大部分事务可以一次提交成功，因此采用一阶段提交+补偿事务的方式，与两（三）阶段提交相比，本方案在大部分情况下减少了与数据节点的交互次数，降低了锁冲突的概率，提升了事务处理效率。

虽然如此，分布式事务相比非分布式事务还是会占用更多的资源，GoldenDB提供了灵活的分布式事务控制，应用可以根据实际情况控制是否启动分布式事务。控制参数具备多粒度能力，可以在系统级、会话级、事务级三个维度控制，同时也提供了多种读隔离级别：

UR（uncommited read）：未提交读，即计算节点不做任何分布式事务控制，业务要么允许脏读，要么不存在读的时候同时写；

CR（consistency read）：强一致性读，在高并发读写时，严格杜绝脏读的可能性。

## 待优化

# 数据分布

集群中每个表中的数据按照某种策略进行横向分片后存放到对应的安全组中，分片策略包括复制策略、哈希策略、范围策略和列表策略。

注：也可以通过配置文件，采用默认hash或复制的分发策略。

## 复制策略

复制策略（duplcate）：适用于不经常修改（因为修改需要同步多个节点，比较耗时），且频繁出现在关联或子查询中的小表。复制策略下，复制表的数据保存在每一个节点都需要这个表中数据的情况下，可以减少节点之间网络数据（RPC）的传输，提高查询性能。

## 哈希策略

哈希策略（Hash）：适用于将数据均匀的分布到预定义的各个安全组中，保证各个安全组的数据量大致一致。一般用户不需要关心分发字段的取值范围和具体含义，且对该表的SQL操作基本都是等值操作的场景。

## 范围策略

范围策略（range）：适用于指定一个给定的列值或列值集合应该保存在哪个安全组上，常用于时间、日期、数值等类型的字段上，如数据按照自然月或者自然天分布存储。

## 列表策略

列表策略（list）：适用于含有一定列限定性或枚举性的字段上，如数据按照机构代码、国家代码、地区代码分布存储。

数据按照上述策略（复制策略除外）分片后，每个安全组上的实际数据在理论上只有总数据量的1/N（数据分布的均匀程度依赖于切分策略和真实数据分布的匹配程度）。随着安全组数量的增加，每个安全组承载的数据量和读写负载会相应的减少，从而在数据节点集群内部具备了读写能力的水平拓展。

安全组是由一个或者多个数据节点组成的数据库节点组，组内的数据库节点拥有相同的数据。当安全组存在多个数据节点时，其中一个数据节点为主用节点其他数据节点都是备用节点，数据在主备节点之间实时复制。主用节点具备读写能力，备用节点可以提供度能力。

# 主从复制/一致性

## 主从复制

## 一致性检查

# 备份恢复

## 备份

GoldenDB的备份属于物理备份，备份的数据包括每个分片的数据文件、每个分片的binlog日志及每个时刻的全局事务列表快照，后两种数据在进行集群的一致性数据恢复以及数据恢复到任意时刻时使用。

对GoldenDB的数据备份功能概括如下：

1. 备份任务管理。

GoldenDB支持在OMM上进行备份任务的管理，包括备份任务的发起、备份历史操作的查询等。

备份任务的发起一般有两种模式：

一种为实时备份，即用户配置好后立即会发起一次备份任务；另一种为定时备份，用户可以设置定时备份策略，典型的策略为每周备份全量数据，其他时间分别在周日全量备份的基础上做一次增量备份，如此，可以通过全量数据和其他任一增量备份数据，快速恢复出想要的那天的数据。

1. 备份文件存储。

GoldenDB的备份文件可以在分片节点本地挂载NFS共享目录，将备份结果文件统一存放；同时GoldenDB也支持与IBM TSM系统对接，可以通过TSM直接将备份结果灌入到磁带库。

## 恢复

GoldenDB的数据恢复功能概括如下：

1. 可恢复到任意时刻。

由于GoldenDB有数据节点的全量及增量备份文件，同时有其运行过程中的binlog日志，借助这些数据可以将系统恢复到任意需要的时刻，但需要注意全量及增量备份文件恢复的速度要大大快于binlog日志的回放速度，因此无特殊要求，建议选择恢复到某次备份的结束时刻，以便更快的完成数据恢复。

1. 一致性的数据恢复。

由于GOldenDB备份了运行过程中的binlog日志及每个时刻的全局事务列表快照，因此可以根据这些信息恢复到一个全局一致性的数据副本。

1. 恢复任务管理。

GoldenDB支持在OMM运维平台上进行节点的一键恢复操作，用户直接在OMM上选定要恢复的节点以及要恢复到的时间点，即可以进行自动化的数据恢复。

# 兼容性

## MySQL兼容性

## Oracle兼容性

### sequence

#### 背景

Sequence用来在多用户环境下产生唯一整数的数据库对象。序列产生器生成数字，它可用于自动生成主键值，并能协调多行或者多表的主键操作。没有sequence，顺序的值只能靠编写程序来生成。先找出最近产生的值然后自增长。这种方法要求一个事务级别的锁，这将导致多用户并发操作的环境下，必须育人等待下一个主键值的产生，而且这种方法很容易产生主键冲突。

还有一个问题，那就是完成生成主键的程序（一般情况包含PLSQL块）本身对于并发调用也是一个瓶颈，因此这样的程序段往往是提供给好多用户调用，如果代码写的不够优化（比如没有使用绑定变量等），或者此代码存在问题，那么它所影响的是系统全局。我们应该提倡开发人员使用sequence，它笑出了序列化问题，而且改善了应用的并发能力。

#### 概述

在oracle中sequence就是序号，每次取的时候它会自动增加。Sequence与表没有关系。

##### 表

Sequence的功能就是产生分布式系统单调变化的序列：为了分布式系统能够像单机系统一样产生可靠的全局单调序列，因此选择系统中控制全局的网元GTM承担sequence用户数据表的创建、维护和消耗功能，但是不维护sequence的元数据。

GTM使用一张表来保存系统用户创建的所有sequence实体，表结构如下：

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| clusterId | SeqPrefix | Seqname | starval | currval | nextval | step | minval | maval | cache | cycle | order |
| 1 | Prefix1 | Seq1 | 10 | 12 | 14 | 2 | 1 | 65535 | 20 | 1 | 0 |
| 2 | Prefix2 | Seq2 | 1 | 2 | 3 | 1 | 1 | 65535 | 100 | 1 | 0 |

注；这里涉及cache可以提高并发，但是这样就不能严格递增。

##### 功能

Sequence的作用主要有两个方面：

1. 作为代理主键，唯一识别；
2. 用于记录数据库中最新动作的语句，只要语句有动作（insert/update/delete等）sequence号就会随着更新，所以我们可以根据sequence号来select出更新的语句

##### 特点

将sequence的用户查询数据放到GTM的优点：

1. 复用GTM设计的初衷思想，不需要再引入其他网元；
2. 可以保证整个分布式系统sequence值全局唯一和申请有序。

#### 操作

##### CREATE

创建语句：

CREATE SEQUENCE seq

INCREMENT BY 1 --每次加几个

START WITH 1 --从1开始计数

NOMAX VALUE --不设置最大值

NOCYCLE --一直增加，不循环

CACHE 10; --设置缓存cache个序列，如果系统down掉了或者其他情况将会导致序列不连续，也可以设置为NOCACHE

说明：

Cache参数是为了应对并发访问的。Cache参数高速oracle预先分配一个sequence number的集合，并且保留在内存中，以便sequence number能够被快速的访问。这个内存的大小就是cache所指定的大小，当多个用户同时访问一个sequence的时候，是在oracle SGA中读取sequence当前的合理数值，如果并发访问太大，cache的大小不够，那么就会产生sequence cache相关的等待，影响系统性能。既然cache涉及到了内存，那么就会想到oracle实例恢复的问题。

**如果数据库shutdown abort，sequence会如何？**

当然会有问题，sequence number保存在内存里的但是没有被应用到表中的会丢失！

如果指定cache值，oracle就会预先在内存中设置一些sequence，这样存储比较快。Cache里面的取完了，oracle会自动再取一组到cache。使用cache或许会跳号，比如数据库突然不正常down掉（shutdown abort），cache中的sequence就会丢失，所以可以在sequence的时候用nocache防止这种情况。

注：在GoldenDB中对于sequence是设置为1~255，然后循环。

##### 获取值

##### ALTER

##### DROP

### synonym

### hint

#### 背景

Hint是oracle数据库的特色功能，是很多DBA优化中经常采用的一个手段。那么oracle为什么会考虑引入优化器呢？

基于代价的优化器是很聪明的，在绝大多数情况下会选择正确的优化器，减轻DBA的负担。但是，有时候它聪明反被聪明误，选择了很差的执行计划，使得某个语句的执行变得奇慢无比。此时，就需要DBA进行人为的干预，高速优化器使用指定的存取路径或连接类型生成执行计划，从而使语句高效地运行。

#### 概述

Hint是oracle提供的一种机制，用来告诉优化器按照告诉它的方式生成执行计划。

##### 注释

提示是oracle为了不破坏和其他数据库引擎之间对SQL语句的兼容性而提供的一种扩展功能。Oracle决定把提示作为一种特殊的注释来添加。它的特殊性表现在提示必须紧跟着DELETE、INSERT、UPDATE或者MERGE关键字。

换句话说，提示不想普通注释那样在SQL语句中随处添加。且在注释分隔符之后的第一个字符必须是加号。

注：OceanBase采用注释的形式加hint信息，GoldenDB采用结尾关键字的形式。

##### 功能

Hint提供的功能非常丰富，可以很灵活地调整语句的执行计划。通过hint信息，我们可以调整：

1. 优化器类型
2. 优化器优化目标
3. 数据读取方式（访问路径）
4. 类型转换类型
5. 表间关联顺序
6. 表间关联类型
7. 并行特性

##### 弊端

1. Hint是比较“暴力”的一种解决方式，不是很优雅，需要开发人员手动修改代码；
2. Hint不会去适应新的变化。比如数据结构、数据规模发生了重大变化，但是使用hint的语句是感知不到变化的，无法获取更优的执行计划；
3. Hint随着数据库版本的变化，可能会有一些差异、甚至废弃的情况。此时，语句本身是无感知的，必须人工测试并修改。

注：比如我们采用storagedb g1，如果数据分布改为g2，那么这个SQL就有问题了。

#### 使用

当遇到SQL执行计划不好的情况，应优先考虑统计信息等问题，而不是直接加hint了事。

如果统计信息无误，应该考虑物理结构是否合理，即没有合适的索引。只有在最后仍然態SQL按照优化的执行计划执行的时候，才考虑使用hint。

毕竟使用hint，需要应用系统修改代码，hint只能解决一条SQL的问题，并且由于数据分布的变化或其他云因（如索引名变更）等，会导致SQL再次出现性能问题。

#### GoldenDB Hint

GoldenDB支持的hint包括：

1. storagedb
2. storagecluster
3. samedb
4. MULTI\_STEP\_QUERY
5. WRITE\_SAMEDB
6. CHGDISKEY
7. NOGTID
8. KEEPALIVE
9. REDISTRIBUTING
10. write\_consistency\_option：SW、CW
11. read\_consistency\_option：UR、CR、Semi-CR
12. read\_write\_split\_option：READMASTER、READSLAVE、READBALANCE

### 函数

# 拓展性

## 多级扩展

常见的系统扩展方式有Scale up和Scale out两种；

1. scale up（纵向扩展）主要是利用现有的存储系统，通过不断增加存储容量来满足数据增长的需求。

但是这种方式只增加了容量，而带宽和计算能力并没有相应的增加。所以，整个存储系统很快就会达到性能瓶颈，需要继续扩展。

这个时候有两种方法：

1. 采用更强性能的存储引擎（EMC的Clarion系统和NetAPP的FAS系列，采用增加控制器CPU或内存的方式来提供更强的性能），但是价格昂贵；
2. 额外购买独立的存储系统，这样会增加管理的复杂度

注：OceanBase采用的就是使用单独分布式文件系统，这样不需要采用划分分片，只需要使用分区打散数据，使用paxos做数据分布。

1. Scale out（横向扩展）通常以节点为单位，每个节点往往将包含容量、处理能力和I/O带宽。一个节点被添加到存储系统，系统中的三种资源将同时升级。

Scale-out架构的存储系统在扩展之后，从用户的视角看起来仍然是一个单一的系统，这一点与我们将多个相互独立的存储系统简单的叠加在一个机柜中是完全不同的。

所以，scale-out方式使得存储系统升级工作大大简化，用户能够真正实现按需购买，降低TCO。

GoldenDB的架构设计使得其具有良好的扩展性，可以从如下几个角度分析：

1. Proxy计算节点是无状态的，可以动态任意扩展；
2. 数据库集群可以根据需要进行如下维度的扩展：
3. 在安全组内部添加数据节点增加数据副本数。一方面可以获得更高的可靠性，同时也可以通过读写分离功能达到读能力的线性扩展；
4. 增加安全组数目。获得数据节点存储和计算能力的线性扩展，在增加数据的分片时需要考虑如何将数据重新迁移到新分片上，即表数据的重分布；
5. 增加数据节点集群数目，扩展组建新的集群。该方法可以在得到处理能力扩展的同时做到不同业务间数据的隔离性，用户可以根据要求选择。

GoldenDB计算节点和数据分片的数目配比推荐为1:1，性能可以随计算节点和分片数目的增加线性扩展，损耗控制在10%内。若业务为CPU密集型，则计算节点书目要适当增加，若为IO密集型，计算节点数目可以适当减少。

## 数据重分布

业务的增长不可避免的需要对资源进行扩容，由于使用了分片技术，数据被切分橙细小的分片分布在数据节点集群中。集群扩容后，原有的数据分片就面临着被打散重新分配的过程，这个过程就是数据重分布（resharding）。

GoldenDB数据重分布功能的基本原理是对重分布表，建一个和原表字段结构相同但是分发策略为目标分发策略的新表，对于需要重分布的数据分片，将其上的数据导出，然后将该部分数据导入到新表，导入过程会使用新表的分发策略，最后将新表改名为原表。其中数据的导出导入过程又是个多次反复渐进的过程，每次导出的数据均是在前一次导出导入过程中产生的增量数据，直至最后仅剩少量差距时对应用进行短暂禁写，最终完成切换。

GoldenDB的数据重分布对在线业务影响小、且课操作性强，具备以下特点：

1. 执行时间可控制，对业务影响秒级。重分布过程中数据的迁出迁入是循序渐进的过程，仅在原分片和目标分片数据存在少量差距时才会禁写，因此对在线业务影响秒级，并不影响读服务；
2. 支持多表并发重分布。在分布式数据库的数据模型中，会将有关联关系的表采用相同的分片策略。在数据重分布过程中，这些相关表可以选择同步进行处理，以避免在数据重分布过程中由于相关表的分片策略临时不同而导致该SQL语句无法直接下压，影响SQL执行性能；
3. 重分布数据智能识别。数据重分布过程中会涉及到数据从源分片至目的分片的移动，开销很可观，GoldenDB支持对表的源分片信息和目标分片信息进行详细的分析比对，最大程度的减少数据在分片间的移动，提升重分布效率（采用更加细粒度的bucketid，作为隐藏列存入表中）；
4. 可操作性好。GoldenDB在统一运维界面OMM上提供重分布操作界面，可全程可视化操作，包括执行、暂停、继续、取消、异常情况下的重试等；
5. 产品通用性好。对比需要预先做好分片规则的预sharding方案，GOldenDB的重分布方案对设计人员的要求相对较低，GoldenDB支持任意分片策略间的重分布，彻底与业务模型解耦，无需在系统设计初期就要精确规划好未来的数据分布情况。

## 异构数据库

采用slot、o2g（oracle to goledendb）、g2o（goldendb to oracle）实现异构数据库之间的在线业务和离线业务对接。

# 高并发

## LVS

## Proxy

### 池化技术

#### 线程池

#### 连接池

Proxy对前端采用TCP长链接，客户端一次性将语句下达至proxy；

Proxy对后端采用连接池的方式处理，连接池的数量及用户密码可配置，和端口是一一对应的关系；

Proxy根据应用访问的端口号来选择对应的集群以及具体的连接；

Proxy连接池耗尽后是否可以动态申请可以配置。

### 缓存

GoldenDB通过构建执行计划缓存、SQL缓存，提高SQL解析效率、提高数据读取效率，在高频度读写系统中可以减少磁盘I/O负担，提升整体系统效率。

#### 元数据缓存

#### 执行计划缓存

#### 结果集缓存

### 并发

GoldenDB在事务处理上通过采用成熟的主流技术来实现高效的事务管理，这些技术包括：以行级为主表级锁为辅的锁技术、多版本并发控制技术、全乐观锁+自动补偿机制。这些技术在保证事务ACID特征的前提下大大提高了事务的并发处理能力。

### 限流

根据消息积压数进行计算，设置流量限定阈值。

Proxy对上做局部流控，如果当前连接数超过配置的最大连接数，要等某连接上的在线事务全部处理结束后再关闭该连接。

### 热点数据

使用重分布解决热点数据问题。

### 结果集透传

### 分包/分布式批处理

对于大结果集进行分包处理（结果集在内存中缓存），防止一次性处理卡顿。

分布式批处理。提供分布式架构下批处理功能，满足金融、政企、运营商等行业日终大数据批处理的要求，通过分布式FetchSize和存储过程功能对数据进行批处理，减少客户端与DB的交互次数，批量返回数据集并进行批量处理。

### 读写分离

读写分离是指利用数据节点集群安全组多副本，将部分读请求发往备节点，提升系统的读能力。

在启动读写分离时，GoldenDB的计算节点在受到应用SQL请求时，根据当前的语句类型和负载策略选择SQL下发的数据节点，将写操作发往主节点，将读操作发往备节点。

注意：GoldenDB的读写分离是事务间的读写分离，如果一个事务内同时存在读写请求，该事务的所有SQL都会发往主节点。

在多个应用接入一个数据节点集群时，为了满足不同应用的需求，GoldenDB支持对同一集群不同的连接实例，设置不同的读写分离模式，包括以下三种：

1. 读主节点。读操作默认发往主节点，当应用强制指定将读请求发往备节点时，读请求即在备机间做负载均衡；
2. 读备节点。读操作仅在备机间根据配置的权重做读负载均衡。

除了上述连接实例级别的读写分离模式设置，GoldenDB还支持SQL级别的读写分离模式指定，应用可以在SQL语句后面添加hint信息强制发往主节点或备节点，SQL级别的优先级高于连接实例级别。常见的使用场景如下：

1. 由于数据在安全组内部的节点间同步存在时延，因此对实时性要求比较高的SQL请求，应用希望将其发往主节点；
2. 对于一些SQL如分析聚合类SQL，应用希望将其发往备节点，减少对主节点的影响。

结合上述连接实例和SQL级别的读写分离模式设置，应用可以根据自己的希望设计合理的读写分离策略。

### 负载均衡

可以设置不同group的优先级，同城机房高于异地灾备机房。

## SQL引擎

### 优化器

### 全局(唯一)索引

### 锁

采用乐观锁（重试）和悲观锁（select for update）两种控制策略，针对不同场景设置不同锁类型。

### MPP

采用presto大数据组件，对于复杂的SQL进行计算。

### force index

为了避免update、delete中where条件索引失效，造成全表锁（对于悲观锁，proxy会先下发select for update where锁住对应数据，然后执行更新），采用在where条件中增加force\_index。

### Hint

#### MULTI\_STEP

#### NOGTID

#### READMASTER

#### READSLAVE

#### READBALANCE

#### storagedb

不需要计算分片，直接到对应的group。

#### samedb

如果可以确定某一个操作全部是对一个group的操作，前面第一个计算分片信息后，后面的全部添加samedb的hint信息，这样就可以直接用前面缓存的group信息了。

## DB

### 快同步

分布式数据库通过增加副本数来提高系统的可用性，为了数据的安全可靠，数据必须在满足拥有一定数量的副本之后，才返回处理结果给客户端。GoldenDB采用自研的gSync复制技术，在配置的副本同步策略满足后，主库返回操作结果给客户端；通过线程池、非阻塞式同步、并行复制等关键技术可以实现在确保数据RPO为0的同时保证系统的吞吐量。

### 大容量数据处理

GoldenDB具备完备的海量数据管理功能，提供PB级的数据容量支持，具有高效索引和查询优化技术，具备海量数据处理能力并支持大数据量用户的并发访问。

## GTM

### 多线程

Proxy分别采用三个线程线程分别执行申请、提交、回滚GTID的操作。

GTM线程结构设计为一个分发线程加若干可配置的执行线程。

## 待优化

# 高可用

## 方案

## 组件高可用

## 部署

### 机房内高可用

即组件高可用

### 同城双活

### 两地三中心

# 数据压缩

GoldenDB支持表数据压缩和备份压缩。表数据压缩时通过对表及分区表进行数据压缩来减少磁盘存储空间，在进行压缩操作前，可以对压缩率进行提前估算，从而做到有针对性的有效压缩，压缩算法采用zlib算法，压缩比在2~11之间。

备份压缩是指备份出来的数据会进行要说来减少空间占用，GoldenDB采用quicklz算法对备份文件进行压缩，压缩比在2~10之间。

GoldenDB同时支持对表的数据空间进行归并整理，可以实现对通过数据压缩和归并整理优化出来的表空间进行释放到表空间，从而实现最大限度的存储优化能力。

# 数据安全

## 备份恢复

## 加密审计

为了确保数据的安全性，数据库需要对数据的全生命周期进行数据安全保证。分布式数据库系统支持数据安全功能包括：

1. 访问鉴权

分布式数据库需要对来访用户进行鉴权，通过IP白名单的方式过滤连接请求，支持通过国密算法对用户名密码进行解密校验。

1. 通道加密

分布式数据库对客户端和数据库的连接进行SSL通道加密，防止通过抓包等手段查看业务数据。

1. 数据加密

分布式数据库支持表级别和列级别的加密。通过列级别的加密，表中仅存储加密后的字符串，在不知道加密串的情况下，即使查询出来也会是乱吗；表级别的加密可以加密整张表，在没有秘钥文件的情况下，即使拷贝走表数据文件，也无法破解表中数据。

1. SQL日志审计

分布式数据库支持所有的业务SQL写入审计日志文件，以便对SQL语句进行统计和审计，找出潜在的威胁。

# 数据迁移

## 重分布

对于少量数据的迁移可以采用重分布。

## 导入导出

数据导入导出一般用于系统间的数据迁移，小到导入导出一部分数据，大到数据割接、数据库迁移式升级、数据的分库、海量数据的迁移等，使用场景十分丰富。

GoldenDB支持将集群中符合查询条件的记录导出到指定的数据文件中，也支持将外部数据文件导入到集群合适的数据分片中。

导入导出功能：

1. 外部接口。GoldenDB导入导出的外部接口为文件，待导入的格式要符合要求，一般为字段间采用分隔符隔离的文本文件，导出结果也同样为文本文件。
2. 工具插件。导入导出可以认为是系统的一个功能插件，部署灵活，与在线交易流分离，可以降低对联机交易的影响。
3. 原始文件自动导入数据库集群。GoldenDB的导入导出工具，能自动根据导入表的分片规则，自动对原始数据文件记录分拣，将数据下发至对应数据节点，最终完成导入。这其中的各环节均支持并发处理，提升整体导入效率。工具也支持带条件导入，即仅将满足条件的记录导入至系统中。
4. 多分片数据统一出口导出。GoldenDB的导入导出工具，能够自动控制各个分片并发进行数据导出，并可以选择将导出文件上传至同一个服务器，进一步的，也可以将多个导出文件汇总成单个大文件，用户可以灵活选择控制导出过程。
5. 容错处理。导入操作耗时长，对系统资源占用多，且通常是将外部系统的数据文件导入到分布式系统中，而外部的文件往往会有异常数据。因此，导入操作经常会遇到错误，GoldenDB充分考虑导入异常的错误处理，举例如下：
6. 导入过程中遇到异常数据时能够将异常数据分拣出来，待后续处理；
7. 流程处理的各个阶段都有重试机制，最大限度保证导入成功；
8. 系统异常时能够从异常位置继续执行导入流程；
9. 支持对原始文件记录末尾的空值进行处理。

**说明：**

使用loadserver或datatransfer进行数据导入导出，走的是后端链路（DB），导入导出成功后，需要在前端连接proxy设置一下sequence自增列的初始值（因为后端链路导入后sequence的值proxy并不知道）。

使用goldendumper走proxy执行导入导出，这个性能可能会消耗比较大（因为是采用insert+delete的方案），可以采用一个专门数转的proxy做这个操作，配置也做相应的优化。

## 日切卸数

在金融领域，针对应用使用传统数据库存在批前卸数丢失跨天事务，分析报告不准确的问题，需要事后进行手工修复数据。GoldenDB提供日切、卸数插件，可以获得准备的当日业务数据，更快得到准确的分析报告，无需在事后手动修复数据。

以导出日切时刻的数据快照为例，使用GoldenDB日切卸数功能导出数据无需经过ODS处理，既能满足事务全局一致性，又能满足具有业务含义的数据一致性（在日切前发生的事务都保留在快照数据中，日切后发生的事务操作都不体现在快照数据上）。

# 分布式查询优化器

分布式查询优化器是GoldenDB计算节点的核心功能，计算节点在接收到SQL语句后，首先由解析模块转成等价的可识别的语法树，然后由优化模块转成等价的执行计划树，最后由执行模块按照步骤执行该计划树，从而完成整个SQL语句的执行。

## 影响因素

优化器的优化工作主要体现在计划树的生成上，GoldenDB的查询优化器设计实现主要考虑以下两个方面：

1. 代价模型的选择。

GoldenDB采用分布式系统代价估算模型，考虑节点间传输数据的代价，以减少数据传输的次数和数据量作为查询优化的目标，提高数据节点之间计算的并行度、减少计算节点的计算量。这主要考虑在分布式数据库系统环境中，表结构被水平或垂直拆分到多个数据节点，因此需要考虑语句如何分拆、分片之间数据如何移动、结果如何计算与合并的问题，网络通信开销不可忽视。

1. 考虑数据一致性开销。

在分布式数据库系统中，数据全局一致性机制相较于单机数据库需要更为复杂的控制。因此，如何降低数据全局一致性保证的开销，也是GoldenDB查询优化器的设计要求。

总体来讲，GOldenDB的分布式查询优化器遵循了上述的设计原则，以基于规则的优化为主，基于成本的优化为辅，在提升系统的灵活性的同时控制系统实现的复杂性。优化器内部内置大量的优化规则，通过查询重写的方式进行经验性优化。在优化规则的选择上，重点分析分片剪枝、并行执行、合并下压、条件下推、条件繁殖、排序消除、去重消除、排序下推等。

内置大量的优化规则，对上百个场景进行优化，复杂SQL语句兼容性和处理性能好，同时支持prepare预处理、执行计划缓存、数据集透传等功能，保证数据一致性条件下实现高性能SQL处理。

支持的典型优化包括：

1. 分片剪枝
2. 合并下压
3. 并行执行
4. 条件下推及条件繁殖
5. 排序下推、limit下推等
6. 聚合函数优化

得益于完善的优化器设计，使得GoldenDB对单节点、跨节点的复杂SQL的兼容支持程度很高，包括跨节点SUM、COUNT、AVG等汇聚类操作，跨节点WHERE、FROM等子查询，跨节点JOIN，跨节点GROUP BY、ORDER BY、LIMIT等。这是GoldenDB将Proxy命名为计算节点而非中间件的原因之一，也是其和很多分布式数据库产品中间件的主要区别。

## 优化策略

# 运维

## OMM

GoldenDB支持的运维能力包括：

1. 一键式安装、升级。
2. 配置可视化。
3. 统计监控。
4. 故障自动检测和告警。
5. 日志。
6. Explain查看执行计划。
7. Dbtoll管理工具。
8. OMM统一运维平台。提供网络拓扑、集群管理、元数据管理等功能。

## Insight

OMM的升级版本，提供了更加详细的监控信息。

## 巡检脚本