# 概述

# 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。

### 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。

# 操作

# 分布式事务

## 方案

## 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。

## 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

## 数据备份恢复

## 元数据备份恢复

# 兼容性

# 拓展性

## 数据节点扩容

## 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

### 线程池

### 连接池

### 缓存

#### 元数据缓存

#### 执行计划缓存

#### 结果集缓存

### 限流

根据消息积压数进行计算，设置流量限定阈值。

### 分包

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

读操作在备机，写操作在主机。

### 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

### 全局(唯一)索引

### 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。

### 指定group

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

## 待优化

# 高可用

## 方案

## 组件高可用

## 部署

### 同城双活

### 两地三中心

# 数据压缩

# 数据安全

# 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

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。

# 运维

## OMM

## Insight

## 巡检脚本